Formalization of Forcing in Isabelle/ZF

Emmanuel Gunther* Miguel Pagano* Pedro Sánchez Terraf*†

September 14, 2021

Abstract

We formalize the theory of forcing in the set theory framework of Isabelle/ZF. Under the assumption of the existence of a countable transitive model of *ZFC*, we construct a proper generic extension and show that the latter also satisfies *ZFC*.

Contents

1 Introduction **6**

2 Forcing notions **6**
2.1 Basic concepts 6
2.2 Towards Rasiowa-Sikorski Lemma (RSL) 12

3 A pointed version of DC **15**

4 The general Rasiowa-Sikorski lemma **18**

5 Auxiliary results on arithmetic **19**
5.1 Some results in ordinal arithmetic 22

6 Various results missing from ZF. **25**

7 Some enhanced theorems on recursion **29**

8 Automatic synthesis of formulas **36**

9 Aids to internalize formulas **45**

*Universidad Nacional de Córdoba. Facultad de Matemática, Astronomía, Física y Computación.

†Centro de Investigación y Estudios de Matemática (CIEM-FaMAF), Conicet. Córdoba. Argentina. Supported by Secyt-UNC project 33620180100465CB.

10 The binder *Least* **47**
10.1 Uniqueness, absoluteness and closure under *Least* 49

11 Fully relational versions of higher order construct **51**

12 Automatic relativization of terms and formulas. **53**
12.1 Discipline for *Pow* 78
12.2 Discipline for *PiP* 80
12.3 Discipline for *Pi* 82
12.4 Auxiliary ported results on *Pi_rel*, now unused 85

13 Arities of internalized formulas **86**
13.1 Discipline for $\lambda A\ B.\ A \to B$ 96
13.2 Discipline for *Collect* terms. 100
13.3 Discipline for *inj* 100
13.4 Discipline for *surj* 104
13.5 Discipline for *bij* 108
13.6 Discipline for $(\approx)$ 109
13.7 Discipline for $(\lesssim)$ 111
13.8 Discipline for $(\prec)$ 112

14 Relativization of the cumulative hierarchy **112**
14.1 Formula synthesis 114
14.2 Absoluteness results 115

15 Renaming of variables in internalized formulas **120**
15.1 Renaming of free variables 121
15.2 Renaming of formulas 129

16 Interface between set models and Constructibility **134**
16.1 Interface with *M_trivial* 136
16.2 Interface with *M_basic* 136
16.3 Interface with *M_trancl* 146
16.4 Interface with *M_eclose* 150
16.5 Interface for proving Collects and Replace in M. 161

17 Transitive set models of ZF **167**
17.1 A forcing locale and generic filters 167

18 Names and generic extensions **170**
18.1 The well-founded relation *ed* 171
18.2 Values and check-names 176

19 Well-founded relation on names **191**

20 Replacements using Lambdas **212**
20.1 Replacement instances obtained through Powerset 218
20.2 Particular instances 238

21 Relative, Choice-less Cardinal Numbers **243**
21.1 The Schroeder-Bernstein Theorem 245
21.2 lesspoll_rel: contributions by Krzysztof Grabczewski 250

22 Porting from *ZF.Cardinal* **253**

23 Relative, Choice-less Cardinal Arithmetic **270**
23.1 Cardinal addition 275
23.1.1 Cardinal addition is commutative 275
23.1.2 Cardinal addition is associative 275
23.1.3 0 is the identity for addition 276
23.1.4 Addition by another cardinal 276
23.1.5 Monotonicity of addition 277
23.1.6 Addition of finite cardinals is "ordinary" addition . . . 277
23.2 Cardinal multiplication 278
23.2.1 Cardinal multiplication is commutative 278
23.2.2 Cardinal multiplication is associative 279
23.2.3 Cardinal multiplication distributes over addition . . . 279
23.2.4 Multiplication by 0 yields 0 280
23.2.5 1 is the identity for multiplication 280
23.3 Some inequalities for multiplication 280
23.3.1 Multiplication by a non-zero cardinal 281
23.3.2 Monotonicity of multiplication 281
23.4 Multiplication of finite cardinals is "ordinary" multiplication . 281
23.5 Infinite Cardinals are Limit Ordinals 282
23.5.1 Toward's Kunen's Corollary 10.13 (1) 298
23.6 For Every Cardinal Number There Exists A Greater One . . 300
23.7 Basic Properties of Successor Cardinals 303
23.7.1 Theorems by Krzysztof Grabczewski, proofs by lcp . . 304

24 Cohen forcing notions **314**
24.1 MOVE THIS to an appropriate place 317
24.2 Combinatorial results on Cohen posets 317

25 Relativization of Finite Functions **318**
25.1 The set of finite binary sequences 318
25.2 Representation of finite functions 319

26 Relative, Cardinal Arithmetic Using AC **327**
26.1 Strengthened Forms of Existing Theorems on Cardinals . . . 329
26.2 The relationship between cardinality and le-pollence 330
26.3 Other Applications of AC 331

27 Library of basic *ZF* results **335**

28 Cardinal Arithmetic under Choice **349**
28.1 More Instances of Separation 360
28.2 More Instances of Replacement 390

29 Separative notions and proper extensions **408**

30 A poset of successions **410**
30.1 Cohen extension is proper 415

31 The ZFC axioms, internalized **416**
31.1 The Axiom of Separation, internalized 418
31.2 The Axiom of Replacement, internalized 422

32 The definition of *forces* **427**
32.1 The relation *frecrel* 427
32.2 Definition of *forces* for equality and membership 429
32.3 The well-founded relation *forcerel* 431
32.4 *frc_at*, forcing for atomic formulas 432
32.5 Recursive expression of *frc_at* 448
32.6 Absoluteness of *frc_at* 448
32.7 Forcing for general formulas 452
32.7.1 The primitive recursion 455
32.8 Forcing for atomic formulas in context 456
32.9 The arity of *forces* 458

33 The Forcing Theorems **460**
33.1 The forcing relation in context 461
33.2 Kunen 2013, Lemma IV.2.37(a) 461
33.3 Kunen 2013, Lemma IV.2.37(a) 461
33.4 Kunen 2013, Lemma IV.2.37(b) 462
33.5 Kunen 2013, Lemma IV.2.38 463
33.6 The relation of forcing and atomic formulas 465
33.7 The relation of forcing and connectives 465
33.8 Kunen 2013, Lemma IV.2.29 466
33.9 Auxiliary results for Lemma IV.2.40(a) 467
33.10Induction on names 470
33.11Lemma IV.2.40(a), in full 472
33.12Lemma IV.2.40(b) 473

33.13The Strenghtening Lemma 479
33.14The Density Lemma 480
33.15The Truth Lemma 482
33.16The "Definition of forcing" 491

34 Auxiliary renamings for Separation **493**

35 The Axiom of Separation in $M[G]$ **504**

36 The Axiom of Pairing in $M[G]$ **512**

37 The Axiom of Unions in $M[G]$ **513**

38 The Powerset Axiom in $M[G]$ **517**

39 The Axiom of Extensionality in $M[G]$ **523**

40 The Axiom of Foundation in $M[G]$ **523**

41 The Axiom of Replacement in $M[G]$ **524**

42 The Axiom of Infinity in $M[G]$ **536**

43 The Axiom of Choice in $M[G]$ **537**
43.1 $M[G]$ is a transitive model of ZF 542

44 Ordinals in generic extensions **545**

45 The main theorem **546**
45.1 The generic extension is countable 547
45.2 The main result 547

46 Cardinal Arithmetic under Choice **549**
46.1 Miscellaneous 549
46.2 Countable and uncountable sets 556
46.3 Results on Aleph_rels 559
46.4 Applications of transfinite recursive constructions 564

47 The Delta System Lemma, Relativized **574**

48 Cohen forcing notions **583**

49 From M to V **636**
49.1 Locales of a class M hold in $\mathcal{V}$ 636

50 Main definitions of the development **639**
50.1 ZF 639
50.2 Relative concepts 641
50.3 Forcing 646

1 Introduction

We formalize the theory of forcing. We work on top of the Isabelle/ZF framework developed by Paulson and Grabczewski [4]. Our mechanization is described in more detail in our papers [1] (LSFA 2018), [2], and [3] (IJCAR 2020).

Release notes

We have improved several aspects of our development before submitting it to the AFP:

1. Our session `Forcing` depends on the new release of `ZF-Constructible`.
2. We streamlined the commands for synthesizing renames and formulas.
3. The command that synthesizes formulas produces the lemmas for them (the synthesized term is a formula and the equivalence between the satisfaction of the synthesized term and the relativized term).
4. Consistently use of structured proofs using Isar (except for one coming from a schematic goal command).

A cross-linked HTML version of the development can be found at https://cs.famaf.unc.edu.ar/~pedro/forcing/.

2 Forcing notions

This theory defines a locale for forcing notions, that is, preorders with a distinguished maximum element.

theory *Forcing_Notions*
  imports *ZF-Constructible.Relative*
begin

2.1 Basic concepts

We say that two elements p, q are *compatible* if they have a lower bound in P

definition *compat_in* :: $i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow o$ **where**

compat_in(*A*,*r*,*p*,*q*) ≡ ∃ *d*∈*A* . ⟨*d*,*p*⟩∈*r* ∧ ⟨*d*,*q*⟩∈*r*

definition
is_compat_in :: [*i*⇒*o*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_compat_in(*M*,*A*,*r*,*p*,*q*) ≡ ∃ *d*[*M*]. *d*∈*A* ∧ (∃ *dp*[*M*]. *pair*(*M*,*d*,*p*,*dp*) ∧ *dp*∈*r* ∧
(∃ *dq*[*M*]. *pair*(*M*,*d*,*q*,*dq*) ∧ *dq*∈*r*))

lemma *compat_inI* :
⟦ *d*∈*A* ; ⟨*d*,*p*⟩∈*r* ; ⟨*d*,*g*⟩∈*r* ⟧ ⟹ *compat_in*(*A*,*r*,*p*,*g*)
by (*auto simp add*: *compat_in_def*)

lemma *refl_compat*:
⟦ *refl*(*A*,*r*) ; ⟨*p*,*q*⟩ ∈ *r* | *p*=*q* | ⟨*q*,*p*⟩ ∈ *r* ; *p*∈*A* ; *q*∈*A*⟧ ⟹ *compat_in*(*A*,*r*,*p*,*q*)
by (*auto simp add*: *refl_def compat_inI*)

lemma *chain_compat*:
refl(*A*,*r*) ⟹ *linear*(*A*,*r*) ⟹ (∀ *p*∈*A*.∀ *q*∈*A*. *compat_in*(*A*,*r*,*p*,*q*))
by (*simp add*: *refl_compat linear_def*)

lemma *subset_fun_image*: *f*:*N*→*P* ⟹ *f*‘‘*N*⊆*P*
by (*auto simp add*: *image_fun apply_funtype*)

lemma *refl_monot_domain*: *refl*(*B*,*r*) ⟹ *A*⊆*B* ⟹ *refl*(*A*,*r*)
unfolding *refl_def* **by** *blast*

locale *forcing_notion* =
fixes *P leq one*
assumes *one_in_P*: *one* ∈ *P*
and *leq_preord*: *preorder_on*(*P*,*leq*)
and *one_max*: ∀ *p*∈*P*. ⟨*p*,*one*⟩∈*leq*
begin

abbreviation *Leq* :: [*i*, *i*] ⇒ *o* (**infixl** ≼ *50*)
where *x* ≼ *y* ≡ ⟨*x*,*y*⟩∈*leq*

lemma *refl_leq*:
r∈*P* ⟹ *r*≼*r*
using *leq_preord* **unfolding** *preorder_on_def refl_def* **by** *simp*

A set D is *dense* if every element $p \in P$ has a lower bound in D.

definition
dense :: *i*⇒*o* **where**
dense(*D*) ≡ ∀ *p*∈*P*. ∃ *d*∈*D* . *d*≼*p*

There is also a weaker definition which asks for a lower bound in D only for the elements below some fixed element q.

definition
dense_below :: *i*⇒*i*⇒*o* **where**
dense_below(*D*,*q*) ≡ ∀ *p*∈*P*. *p*≼*q* ⟶ (∃ *d*∈*D*. *d*∈*P* ∧ *d*≼*p*)

lemma *P_dense*: *dense*(P)
by (*insert leq_preord*, *auto simp add*: *preorder_on_def refl_def dense_def*)

definition
increasing :: $i \Rightarrow o$ **where**
increasing(F) $\equiv \forall\, x{\in}F.\ \forall\ p \in P\ .\ x{\preceq}p \longrightarrow p{\in}F$

definition
compat :: $i \Rightarrow i \Rightarrow o$ **where**
compat(p,q) $\equiv$ *compat_in*(P,*leq*,p,q)

lemma *leq_transD*: $a{\preceq}b \Longrightarrow b{\preceq}c \Longrightarrow a \in P \Longrightarrow b \in P \Longrightarrow c \in P \Longrightarrow a{\preceq}c$
using *leq_preord trans_onD* **unfolding** *preorder_on_def* **by** *blast*

lemma *leq_transD'*: $A{\subseteq}P \Longrightarrow a{\preceq}b \Longrightarrow b{\preceq}c \Longrightarrow a \in A \Longrightarrow b \in P \Longrightarrow c \in P \Longrightarrow a{\preceq}c$
using *leq_preord trans_onD subsetD* **unfolding** *preorder_on_def* **by** *blast*

lemma *compatD*[*dest*!]: *compat*(p,q) $\Longrightarrow \exists\, d{\in}P.\ d{\preceq}p \wedge d{\preceq}q$
unfolding *compat_def compat_in_def* **.**

abbreviation *Incompatible* :: [i, i] $\Rightarrow$ o (**infixl** $\perp$ *50*)
where $p \perp q \equiv \neg$ *compat*(p,q)

lemma *compatI*[*intro*!]: $d{\in}P \Longrightarrow d{\preceq}p \Longrightarrow d{\preceq}q \Longrightarrow$ *compat*(p,q)
unfolding *compat_def compat_in_def* **by** *blast*

lemma *denseD* [*dest*]: *dense*(D) $\Longrightarrow p{\in}P \Longrightarrow\ \exists\, d{\in}D.\ d{\preceq}\ p$
unfolding *dense_def* **by** *blast*

lemma *denseI* [*intro*!]: $[\![\ \bigwedge p.\ p{\in}P \Longrightarrow \exists\, d{\in}D.\ d{\preceq}\ p\ ]\!] \Longrightarrow$ *dense*(D)
unfolding *dense_def* **by** *blast*

lemma *dense_belowD* [*dest*]:
assumes *dense_below*(D,p) $q{\in}P$ $q{\preceq}p$
shows $\exists\, d{\in}D.\ d{\in}P \wedge d{\preceq}q$
using *assms* **unfolding** *dense_below_def* **by** *simp*

lemma *dense_belowI* [*intro*!]:
assumes $\bigwedge q.\ q{\in}P \Longrightarrow q{\preceq}p \Longrightarrow \exists\, d{\in}D.\ d{\in}P \wedge d{\preceq}q$
shows *dense_below*(D,p)
using *assms* **unfolding** *dense_below_def* **by** *simp*

lemma *dense_below_cong*: $p{\in}P \Longrightarrow D = D' \Longrightarrow$ *dense_below*(D,p) $\longleftrightarrow$ *dense_below*(D',p)
by *blast*

lemma *dense_below_cong'*: $p{\in}P \Longrightarrow [\![\bigwedge x.\ x{\in}P \Longrightarrow Q(x) \longleftrightarrow Q'(x)]\!] \Longrightarrow$
dense_below($\{q{\in}P.\ Q(q)\}$,p) $\longleftrightarrow$ *dense_below*($\{q{\in}P.\ Q'(q)\}$,p)

by *blast*

lemma *dense_below_mono*: $p{\in}P \Longrightarrow D \subseteq D' \Longrightarrow$ *dense_below*$(D,p) \Longrightarrow$ *dense_below*(D',p)
by *blast*

lemma *dense_below_under*:
assumes *dense_below*(D,p) $p{\in}P$ $q{\in}P$ $q{\preceq}p$
shows *dense_below*(D,q)
using *assms leq_transD* **by** *blast*

lemma *ideal_dense_below*:
assumes $\bigwedge q.\ q{\in}P \Longrightarrow q{\preceq}p \Longrightarrow q{\in}D$
shows *dense_below*(D,p)
using *assms refl_leq* **by** *blast*

lemma *dense_below_dense_below*:
assumes *dense_below*$(\{q{\in}P.\ $ *dense_below*$(D,q)\},p)$ $p{\in}P$
shows *dense_below*(D,p)
using *assms leq_transD refl_leq* **by** *blast*

A filter is an increasing set G with all its elements being compatible in G.

definition
filter :: $i{\Rightarrow}o$ **where**
filter$(G) \equiv G{\subseteq}P \wedge$ *increasing*$(G) \wedge (\forall\, p{\in}G.\ \forall\, q{\in}G.\ $ *compat_in*$(G,leq,p,q))$

lemma *filterD* : *filter*$(G) \Longrightarrow x \in G \Longrightarrow x \in P$
by (*auto simp add* : *subsetD filter_def*)

lemma *filter_leqD* : *filter*$(G) \Longrightarrow x \in G \Longrightarrow y \in P \Longrightarrow x{\preceq}y \Longrightarrow y \in G$
by (*simp add*: *filter_def increasing_def*)

lemma *filter_imp_compat*: *filter*$(G) \Longrightarrow p{\in}G \Longrightarrow q{\in}G \Longrightarrow$ *compat*(p,q)
unfolding *filter_def compat_in_def compat_def* **by** *blast*

lemma *low_bound_filter*: — says the compatibility is attained inside G
assumes *filter*(G) **and** $p{\in}G$ **and** $q{\in}G$
shows $\exists\, r{\in}G.\ r{\preceq}p \wedge r{\preceq}q$
using *assms*
unfolding *compat_in_def filter_def* **by** *blast*

We finally introduce the upward closure of a set and prove that the closure of A is a filter if its elements are compatible in A.

definition
upclosure :: $i{\Rightarrow}i$ **where**
upclosure$(A) \equiv \{p{\in}P.{\exists}\, a{\in}A.\ a{\preceq}p\}$

lemma *upclosureI* [*intro*] : $p{\in}P \Longrightarrow a{\in}A \Longrightarrow a{\preceq}p \Longrightarrow p{\in}$*upclosure*$(A)$
by (*simp add*:*upclosure_def*, *auto*)

lemma *upclosureE* [*elim*] :
p∈*upclosure*(*A*) ⟹ (⋀*x a*. *x*∈*P* ⟹ *a*∈*A* ⟹ *a*≼*x* ⟹ *R*) ⟹ *R*
by (*auto simp add*:*upclosure_def*)

lemma *upclosureD* [*dest*] :
p∈*upclosure*(*A*) ⟹ ∃ *a*∈*A*.(*a*≼*p*) ∧ *p*∈*P*
by (*simp add*:*upclosure_def*)

lemma *upclosure_increasing* :
assumes *A*⊆*P*
shows *increasing*(*upclosure*(*A*))
unfolding *increasing_def upclosure_def*
using *leq_transD'*[*OF* ‹*A*⊆*P*›] **by** *auto*

lemma *upclosure_in_P*: *A* ⊆ *P* ⟹ *upclosure*(*A*) ⊆ *P*
using *subsetI upclosure_def* **by** *simp*

lemma *A_sub_upclosure*: *A* ⊆ *P* ⟹ *A*⊆*upclosure*(*A*)
using *subsetI leq_preord*
unfolding *upclosure_def preorder_on_def refl_def* **by** *auto*

lemma *elem_upclosure*: *A*⊆*P* ⟹ *x*∈*A* ⟹ *x*∈*upclosure*(*A*)
by (*blast dest*:*A_sub_upclosure*)

lemma *closure_compat_filter*:
assumes *A*⊆*P* (∀ *p*∈*A*.∀ *q*∈*A*. *compat_in*(*A*,*leq*,*p*,*q*))
shows *filter*(*upclosure*(*A*))
unfolding *filter_def*
proof(*auto*)
show *increasing*(*upclosure*(*A*))
using *assms upclosure_increasing* **by** *simp*
next
let *?UA*=*upclosure*(*A*)
show *compat_in*(*upclosure*(*A*), *leq*, *p*, *q*) **if** *p*∈*?UA q*∈*?UA* **for** *p q*
proof -
from *that*
obtain *a b* **where** *1*:*a*∈*A b*∈*A a*≼*p b*≼*q p*∈*P q*∈*P*
using *upclosureD*[*OF* ‹*p*∈*?UA*›] *upclosureD*[*OF* ‹*q*∈*?UA*›] **by** *auto*
with *assms*(*2*)
obtain *d* **where** *d*∈*A d*≼*a d*≼*b*
unfolding *compat_in_def* **by** *auto*
with *1*
have *d*≼*p d*≼*q d*∈*?UA*
using *A_sub_upclosure*[*THEN subsetD*] ‹*A*⊆*P*›
leq_transD'[*of A d a*] *leq_transD'*[*of A d b*] **by** *auto*
then
show *?thesis* **unfolding** *compat_in_def* **by** *auto*
qed
qed

```
lemma aux_RS1: f ∈ N → P ⟹ n∈N ⟹ f‘n ∈ upclosure(f ‘‘N)
  using elem_upclosure[OF subset_fun_image] image_fun
  by (simp, blast)

lemma decr_succ_decr:
  assumes f ∈ nat → P preorder_on(P,leq)
    ∀ n∈nat. ⟨f ‘ succ(n), f ‘ n⟩ ∈ leq
    m∈nat
  shows n∈nat ⟹ n≤m ⟹ ⟨f ‘ m, f ‘ n⟩ ∈ leq
  using ‹m∈_›
proof(induct m)
  case 0
  then show ?case using assms refl_leq by simp
next
  case (succ x)
  then
  have 1:f‘succ(x) ≼ f‘x f‘n∈P f‘x∈P f‘succ(x)∈P
    using assms by simp_all
  consider (lt) n<succ(x) | (eq) n=succ(x)
    using succ le_succ_iff by auto
  then
  show ?case
  proof(cases)
    case lt
    with 1 show ?thesis using leI succ leq_transD by auto
  next
    case eq
    with 1 show ?thesis using refl_leq by simp
  qed
qed

lemma decr_seq_linear:
  assumes refl(P,leq) f ∈ nat → P
    ∀ n∈nat. ⟨f ‘ succ(n), f ‘ n⟩ ∈ leq
    trans[P](leq)
  shows linear(f ‘‘ nat, leq)
proof -
  have preorder_on(P,leq)
    unfolding preorder_on_def using assms by simp
  {
    fix n m
    assume n∈nat m∈nat
    then
    have f‘m ≼ f‘n ∨ f‘n ≼ f‘m
    proof(cases m≤n)
      case True
      with ‹n∈_› ‹m∈_›
      show ?thesis
```

using *decr_succ_decr*[*of f n m*] *assms leI* ‹*preorder_on*(*P*,*leq*)› **by** *simp*
next
case *False*
with ‹*n*∈_› ‹*m*∈_›
show *?thesis*
using *decr_succ_decr*[*of f m n*] *assms leI not_le_iff_lt* ‹*preorder_on*(*P*,*leq*)›
by *simp*
qed
}
then
show *?thesis*
unfolding *linear_def* **using** *ball_image_simp assms* **by** *auto*
qed

end

2.2 Towards Rasiowa-Sikorski Lemma (RSL)

locale *countable_generic* = *forcing_notion* +
fixes $\mathcal{D}$
assumes *countable_subs_of_P*: $\mathcal{D} \in nat \to Pow(P)$
and *seq_of_denses*: $\forall\, n \in nat.\ dense(\mathcal{D}\,`n)$

begin

definition
D_generic :: *i*⇒*o* **where**
$D\_generic(G) \equiv filter(G) \wedge (\forall\, n{\in}nat.(\mathcal{D}\,`n){\cap}G{\neq}0)$

The next lemma identifies a sufficient condition for obtaining RSL.

lemma *RS_sequence_imp_rasiowa_sikorski*:
assumes
$p{\in}P\ f : nat \to P\ f\ `\ 0 = p$
$\bigwedge n.\ n{\in}nat \Longrightarrow f\ `\ succ(n) \preceq f\ `\ n \wedge f\ `\ succ(n) \in \mathcal{D}\ `\ n$
shows
$\exists\, G.\ p{\in}G \wedge D\_generic(G)$
proof -
note *assms*
moreover from *this*
have *f‘‘nat* ⊆ *P*
by (*simp add*:*subset_fun_image*)
moreover from *calculation*
have *refl*(*f‘‘nat*, *leq*) ∧ *trans*[*P*](*leq*)
using *leq_preord* **unfolding** *preorder_on_def* **by** (*blast intro*:*refl_monot_domain*)
moreover from *calculation*
have ∀ *n*∈*nat*. *f* ‘ *succ*(*n*)≼ *f* ‘ *n* **by** (*simp*)
moreover from *calculation*
have *linear*(*f‘‘nat*, *leq*)
using *leq_preord* **and** *decr_seq_linear* **unfolding** *preorder_on_def* **by** (*blast*)

moreover from *calculation*
have $(\forall\, p{\in}f``nat.\forall\, q{\in}f``nat.\ compat\_in(f``nat,leq,p,q))$
using *chain_compat* **by** (*auto*)
ultimately
have *filter*(*upclosure*($f``nat$)) (**is** *filter*(*?G*))
using *closure_compat_filter* **by** *simp*
moreover
have $\forall\, n{\in}nat.\ \mathcal{D}\ `\ n \cap ?G \neq 0$
proof
fix *n*
assume $n{\in}nat$
with *assms*
have $f`succ(n) \in ?G \wedge f`succ(n) \in \mathcal{D}\ `\ n$
using *aux_RS1* **by** *simp*
then
show $\mathcal{D}\ `\ n \cap ?G \neq 0$ **by** *blast*
qed
moreover from *assms*
have $p \in ?G$
using *aux_RS1* **by** *auto*
ultimately
show *?thesis* **unfolding** *D_generic_def* **by** *auto*
qed

end

— TODO: already in ZF Library
lemma *Pi_rangeD*:
assumes $f{\in}Pi(A,B)$ $b \in range(f)$
shows $\exists\, a{\in}A.\ f`a = b$
using *assms apply_equality*[*OF _ assms*(*1*), *of _ b*] *domain_type*[*OF _ assms*(*1*)]
by *auto*

Now, the following recursive definition will fulfill the requirements of lemma *RS_sequence_imp_rasiowa_sikorski*

consts $RS\_seq :: [i,i,i,i,i,i] \Rightarrow i$
primrec
$RS\_seq(0,P,leq,p,enum,\mathcal{D}) = p$
$RS\_seq(succ(n),P,leq,p,enum,\mathcal{D}) =$
$enum`(\mu\ m.\ \langle enum`m,\ RS\_seq(n,P,leq,p,enum,\mathcal{D})\rangle \in leq \wedge enum`m \in \mathcal{D}\ `\ n)$

context *countable_generic*
begin

lemma *countable_RS_sequence_aux*:
fixes *p enum*
defines $f(n) \equiv RS\_seq(n,P,leq,p,enum,\mathcal{D})$
and $Q(q,k,m) \equiv enum`m \preceq q \wedge enum`m \in \mathcal{D}\ `\ k$
assumes $n{\in}nat\ p{\in}P\ P \subseteq range(enum)\ enum{:}nat{\to}M$

```
    ⋀x k. x∈P ⟹ k∈nat ⟹  ∃ q∈P. q≼ x ∧ q ∈ 𝒟 ‘ k
  shows
    f(succ(n)) ∈ P ∧ f(succ(n))≼ f(n) ∧ f(succ(n)) ∈ 𝒟 ‘ n
  using ‹n∈nat›
proof (induct)
  case 0
  from assms
  obtain q where q∈P q≼ p q ∈ 𝒟 ‘ 0 by blast
  moreover from this and ‹P ⊆ range(enum)›
  obtain m where m∈nat enum‘m = q
    using Pi_rangeD[OF ‹enum:nat→M›] by blast
  moreover
  have 𝒟‘0 ⊆ P
    using apply_funtype[OF countable_subs_of_P] by simp
  moreover note ‹p∈P›
  ultimately
  show ?case
    using LeastI[of Q(p,0) m] unfolding Q_def f_def by auto
next
  case (succ n)
  with assms
  obtain q where q∈P q≼ f(succ(n)) q ∈ 𝒟 ‘ succ(n) by blast
  moreover from this and ‹P ⊆ range(enum)›
  obtain m where m∈nat enum‘m≼ f(succ(n)) enum‘m ∈ 𝒟 ‘ succ(n)
    using Pi_rangeD[OF ‹enum:nat→M›] by blast
  moreover note succ
  moreover from calculation
  have 𝒟‘succ(n) ⊆ P
    using apply_funtype[OF countable_subs_of_P] by auto
  ultimately
  show ?case
    using LeastI[of Q(f(succ(n)),succ(n)) m] unfolding Q_def f_def by auto
qed

lemma countable_RS_sequence:
  fixes p enum
  defines f ≡ λn∈nat. RS_seq(n,P,leq,p,enum,𝒟)
    and   Q(q,k,m) ≡ enum‘m≼ q ∧ enum‘m ∈ 𝒟 ‘ k
  assumes n∈nat p∈P P ⊆ range(enum) enum:nat→M
  shows
    f‘0 = p f‘succ(n)≼ f‘n ∧ f‘succ(n) ∈ 𝒟 ‘ n f‘succ(n) ∈ P
proof -
  from assms
  show f‘0 = p by simp
  {
    fix x k
    assume x∈P k∈nat
    then
    have ∃ q∈P. q≼ x ∧ q ∈ 𝒟 ‘ k
```

using *seq_of_denses apply_funtype*[*OF countable_subs_of_P*]
unfolding *dense_def* **by** *blast*
}
with *assms*
show *f‘succ(n)≼ f‘n ∧ f‘succ(n) ∈ 𝒟 ‘ n f‘succ(n)∈P*
unfolding *f_def* **using** *countable_RS_sequence_aux* **by** *simp_all*
qed

lemma *RS_seq_type*:
assumes *n ∈ nat p∈P P ⊆ range(enum) enum:nat→M*
shows *RS_seq(n,P,leq,p,enum,𝒟) ∈ P*
using *assms countable_RS_sequence(1,3)*
by (*induct*;*simp*)

lemma *RS_seq_funtype*:
assumes *p∈P P ⊆ range(enum) enum:nat→M*
shows (*λn∈nat. RS_seq(n,P,leq,p,enum,𝒟)*): *nat → P*
using *assms lam_type RS_seq_type* **by** *auto*

lemmas *countable_rasiowa_sikorski =*
RS_sequence_imp_rasiowa_sikorski[*OF _ RS_seq_funtype countable_RS_sequence*(*1,2*)]

end

end

3 A pointed version of DC

theory *Pointed_DC* **imports** *ZF.AC*

begin

This proof of DC is from Moschovakis "Notes on Set Theory"

consts *dc_witness* :: *i ⇒ i ⇒ i ⇒ i ⇒ i ⇒ i*
primrec
wit0 : *dc_witness*(*0*,*A*,*a*,*s*,*R*) = *a*
witrec :*dc_witness*(*succ*(*n*),*A*,*a*,*s*,*R*) = *s‘*{*x∈A. ⟨dc_witness*(*n*,*A*,*a*,*s*,*R*),*x⟩∈R* }

lemma *witness_into_A* [*TC*]:
assumes *a∈A*
(∀ *X* . *X≠0 ∧ X⊆A ⟶ s‘X∈X*)
∀ *y∈A*. {*x∈A*. ⟨*y*,*x⟩∈R* } ≠ *0 n∈nat*
shows *dc_witness*(*n*, *A*, *a*, *s*, *R*)∈*A*
using ‹*n∈nat*›
proof(*induct n*)
case *0*
then show *?case* **using** ‹*a∈A*› **by** *simp*
next
case (*succ x*)

then
show *?case* **using** *assms* **by** *auto*
qed

lemma *witness_related* :
assumes $a \in A$
$(\forall X . X \neq 0 \wedge X \subseteq A \longrightarrow s`X \in X)$
$\forall y \in A. \{x \in A. \langle y,x\rangle \in R\} \neq 0$ $n \in nat$
shows $\langle dc\_witness(n, A, a, s, R), dc\_witness(succ(n), A, a, s, R)\rangle \in R$
proof -
from *assms*
have $dc\_witness(n, A, a, s, R) \in A$ (**is** $?x \in A$)
using *witness_into_A*[*of* _ _ *s R n*] **by** *simp*
with *assms*
show *?thesis* **by** *auto*
qed

lemma *witness_funtype*:
assumes $a \in A$
$(\forall X . X \neq 0 \wedge X \subseteq A \longrightarrow s`X \in X)$
$\forall y \in A. \{x \in A. \langle y,x\rangle \in R\} \neq 0$
shows $(\lambda n \in nat. dc\_witness(n, A, a, s, R)) \in nat \to A$ (**is** $?f \in \_ \to \_$)
proof -
have $?f \in nat \to \{dc\_witness(n, A, a, s, R). n \in nat\}$ (**is** $\_ \in \_ \to ?B$)
using *lam_funtype assms* **by** *simp*
then
have $?B \subseteq A$
using *witness_into_A assms* **by** *auto*
with ‹*?f* ∈ _›
show *?thesis*
using *fun_weaken_type*
by *simp*
qed

lemma *witness_to_fun*: **assumes** $a \in A$
$(\forall X . X \neq 0 \wedge X \subseteq A \longrightarrow s`X \in X)$
$\forall y \in A. \{x \in A. \langle y,x\rangle \in R\} \neq 0$
shows $\exists f \in nat \to A. \forall n \in nat. f`n = dc\_witness(n,A,a,s,R)$
using *assms bexI*[*of* _ $\lambda n \in nat. dc\_witness(n,A,a,s,R)$] *witness_funtype*
by *simp*

theorem *pointed_DC* :
assumes $(\forall x \in A. \exists y \in A. \langle x,y\rangle \in R)$
shows $\forall a \in A. (\exists f \in nat \to A. f`0 = a \wedge (\forall n \in nat. \langle f`n, f`succ(n)\rangle \in R))$
proof -
have *0*:$\forall y \in A. \{x \in A . \langle y, x\rangle \in R\} \neq 0$
using *assms* **by** *auto*
from *AC_func_Pow*[*of A*]
obtain *g*

```
    where 1: g ∈ Pow(A) - {0} → A
      ∀X. X ≠ 0 ∧ X ⊆ A ⟶ g ‘ X ∈ X
    by auto
  let ?f =λa.λn∈nat. dc_witness(n,A,a,g,R)
  {
    fix a
    assume a∈A
    from ‹a∈A›
    have f0: ?f(a)‘0 = a by simp
    with ‹a∈A›
    have ⟨?f(a) ‘ n, ?f(a) ‘ succ(n)⟩ ∈ R if n∈nat for n
      using witness_related[OF ‹a∈A› 1(2) 0] beta that by simp
    then
    have ∃f∈nat → A. f ‘ 0 = a ∧ (∀n∈nat. ⟨f ‘ n, f ‘ succ(n)⟩ ∈ R) (is ∃x∈_
.?P(x))
      using f0 witness_funtype 0 1 ‹a∈_› by blast
  }
  then show ?thesis by auto
qed

lemma aux_DC_on_AxNat2 : ∀x∈A×nat. ∃y∈A. ⟨x,⟨y,succ(snd(x))⟩⟩ ∈ R ⟹
                ∀x∈A×nat. ∃y∈A×nat. ⟨x,y⟩ ∈ {⟨a,b⟩∈R. snd(b) = succ(snd(a))}
  by (rule ballI, erule_tac x=x in ballE, simp_all)

lemma infer_snd : c∈ A×B ⟹ snd(c) = k ⟹ c=⟨fst(c),k⟩
  by auto

corollary DC_on_A_x_nat :
  assumes (∀x∈A×nat. ∃y∈A. ⟨x,⟨y,succ(snd(x))⟩⟩ ∈ R) a∈A
  shows ∃f ∈ nat→A. f‘0 = a ∧ (∀n ∈ nat. ⟨⟨f‘n,n⟩,⟨f‘succ(n),succ(n)⟩⟩∈R) (is
∃x∈_.?P(x))
proof -
  let ?R′={⟨a,b⟩∈R. snd(b) = succ(snd(a))}
  from assms(1)
  have ∀x∈A×nat. ∃y∈A×nat. ⟨x,y⟩ ∈ ?R′
    using aux_DC_on_AxNat2 by simp
  with ‹a∈_›
  obtain f where
    F:f∈nat→A×nat f ‘ 0 = ⟨a,0⟩  ∀n∈nat. ⟨f ‘ n, f ‘ succ(n)⟩ ∈ ?R′
    using pointed_DC[of A×nat ?R′] by blast
  let ?f=λx∈nat. fst(f‘x)
  from F
  have ?f∈nat→A ?f ‘ 0 = a by auto
  have 1:n∈ nat ⟹ f‘n= ⟨?f‘n, n⟩ for n
  proof(induct n set:nat)
    case 0
    then show ?case using F by simp
  next
    case (succ x)
```

then
have $\langle f`x,\ f`succ(x)\rangle \in ?R'\ f`x \in A\times nat\ f`succ(x)\in A\times nat$
using *F* **by** *simp_all*
then
have $snd(f`succ(x)) = succ(snd(f`x))$ **by** *simp*
with *succ* ‹$f`x\in\_$›
show *?case* **using** *infer_snd*[*OF* ‹$f`succ(\_)\in\_$›] **by** *auto*
qed
have $\langle\langle ?f`n,n\rangle,\langle ?f`succ(n),succ(n)\rangle\rangle \in R$ **if** $n\in nat$ **for** *n*
using *that 1*[*of* $succ(n)$] *1*[*OF* ‹$n\in\_$›] *F*(*3*) **by** *simp*
with ‹$f`0=\langle a,0\rangle$›
show *?thesis* **using** *rev_bexI*[*OF* ‹$?f\in\_$›] **by** *simp*
qed

lemma *aux_sequence_DC* :
assumes $\forall\, x\in A.\ \forall\, n\in nat.\ \exists\, y\in A.\ \langle x,y\rangle \in S`n$
$R=\{\langle\langle x,n\rangle,\langle y,m\rangle\rangle \in (A\times nat)\times(A\times nat).\ \langle x,y\rangle\in S`m\ \}$
shows $\forall\ x\in A\times nat\ .\ \exists\, y\in A.\ \langle x,\langle y,succ(snd(x))\rangle\rangle \in R$
using *assms Pair_fst_snd_eq* **by** *auto*

lemma *aux_sequence_DC2* : $\forall\, x\in A.\ \forall\, n\in nat.\ \exists\, y\in A.\ \langle x,y\rangle \in S`n \Longrightarrow$
$\forall\, x\in A\times nat.\ \exists\, y\in A.\ \langle x,\langle y,succ(snd(x))\rangle\rangle \in \{\langle\langle x,n\rangle,\langle y,m\rangle\rangle\in(A\times nat)\times(A\times nat).$
$\langle x,y\rangle\in S`m\ \}$
by *auto*

lemma *sequence_DC*:
assumes $\forall\, x\in A.\ \forall\, n\in nat.\ \exists\, y\in A.\ \langle x,y\rangle \in S`n$
shows $\forall\, a\in A.\ (\exists\, f \in nat\to A.\ f`0 = a \wedge (\forall\, n \in nat.\ \langle f`n,f`succ(n)\rangle\in S`succ(n)))$
by (*rule ballI*,*insert assms*,*drule aux_sequence_DC2*, *drule DC_on_A_x_nat*, *auto*)

end

4 The general Rasiowa-Sikorski lemma

theory *Rasiowa_Sikorski* **imports** *Forcing_Notions Pointed_DC* **begin**

context *countable_generic*
begin

lemma *RS_relation*:
assumes $p\in P\ n\in nat$
shows $\exists\, y\in P.\ \langle p,y\rangle \in (\lambda m\in nat.\ \{\langle x,y\rangle\in P\times P.\ y\preceq x\ \wedge\ y\in\mathcal{D}`(pred(m))\})`n$
proof -
from *seq_of_denses* ‹$n\in nat$›
have $dense(\mathcal{D}\ `\ pred(n))$ **by** *simp*
with ‹$p\in P$›
have $\exists\, d\in\mathcal{D}\ `\ Arith.pred(n).\ d\preceq p$
unfolding *dense_def* **by** *simp*

then obtain d **where** *3*: $d \in \mathcal{D}$ ` *Arith.pred*$(n) \wedge d \preceq p$
by *blast*
from *countable_subs_of_P* ‹$n \in nat$›
have $\mathcal{D}$ ` *Arith.pred*$(n) \in Pow(P)$
by (*blast dest*:*apply_funtype intro*:*pred_type*)
then
have $\mathcal{D}$ ` *Arith.pred*$(n) \subseteq P$
by (*rule PowD*)
with *3*
have $d \in P \wedge d \preceq p \wedge d \in \mathcal{D}$ ` *Arith.pred*(n)
by *auto*
with ‹$p \in P$› ‹$n \in nat$›
show *?thesis* **by** *auto*
qed

lemma *DC_imp_RS_sequence*:
assumes $p \in P$
shows $\exists f.\ f: nat \to P \wedge f$ ` $0 = p \wedge$
$(\forall n \in nat.\ f$ ` $succ(n) \preceq f$ ` $n \wedge f$ ` $succ(n) \in \mathcal{D}$ ` $n)$
proof -
let *?S*$= (\lambda m \in nat.\ \{\langle x,y\rangle \in P \times P.\ y \preceq x \wedge y \in \mathcal{D}$`$(pred(m))\})$
have $\forall x \in P.\ \forall n \in nat.\ \exists y \in P.\ \langle x,y\rangle \in$ *?S*`n
using *RS_relation* **by** (*auto*)
then
have $\forall a \in P.\ (\exists f \in nat \to P.\ f$`$0 = a \wedge (\forall n \in nat.\ \langle f$`$n, f$`$succ(n)\rangle \in$ *?S*`$succ(n)))$
using *sequence_DC* **by** (*blast*)
with ‹$p \in P$›
show *?thesis* **by** *auto*
qed

theorem *rasiowa_sikorski*:
$p \in P \Longrightarrow \exists G.\ p \in G \wedge D\_generic(G)$
using *RS_sequence_imp_rasiowa_sikorski* **by** (*auto dest*:*DC_imp_RS_sequence*)

end

end

5 Auxiliary results on arithmetic

theory *Nat_Miscellanea* **imports** *ZF* **begin**

Most of these results will get used at some point for the calculation of arities.

lemmas *nat_succI* = *Ord_succ_mem_iff* [*THEN iffD2*,*OF nat_into_Ord*]

lemma *nat_succD* : $m \in nat \Longrightarrow succ(n) \in succ(m) \Longrightarrow n \in m$
by (*drule_tac j*=$succ(m)$ **in** *ltI*,*auto elim*:*ltD*)

lemmas *zero_in_succ* = *ltD* [*OF nat_0_le*]

lemma *in_n_in_nat* : $m \in nat \Longrightarrow n \in m \Longrightarrow n \in nat$
by(*drule ltI*[*of n*],*auto simp add*: *lt_nat_in_nat*)

lemma *in_succ_in_nat* : $m \in nat \Longrightarrow n \in succ(m) \Longrightarrow n \in nat$
by(*auto simp add*:*in_n_in_nat*)

lemma *ltI_neg* : $x \in nat \Longrightarrow j \leq x \Longrightarrow j \neq x \Longrightarrow j < x$
by (*simp add*: *le_iff*)

lemma *succ_pred_eq* : $m \in nat \Longrightarrow m \neq 0 \Longrightarrow succ(pred(m)) = m$
by (*auto elim*: *natE*)

lemma *succ_ltI* : $succ(j) < n \Longrightarrow j < n$
by (*simp add*: *succ_leE*[*OF leI*])

lemma *succ_In* : $n \in nat \Longrightarrow succ(j) \in n \Longrightarrow j \in n$
by (*rule succ_ltI*[*THEN ltD*], *auto intro*: *ltI*)

lemmas *succ_leD* = *succ_leE*[*OF leI*]

lemma *succpred_leI* : $n \in nat \Longrightarrow n \leq succ(pred(n))$
by (*auto elim*: *natE*)

lemma *succpred_n0* : $succ(n) \in p \Longrightarrow p \neq 0$
by (*auto*)

lemmas *natEin* = *natE* [*OF lt_nat_in_nat*]

lemma *succ_in* : $succ(x) \leq y \Longrightarrow x \in y$
by (*auto dest*:*ltD*)

lemmas *Un_least_lt_iffn* = *Un_least_lt_iff* [*OF nat_into_Ord nat_into_Ord*]

lemma *pred_type* : $m \in nat \Longrightarrow n \leq m \Longrightarrow n \in nat$
by (*rule leE*,*auto simp*:*in_n_in_nat ltD*)

lemma *pred_le* : $m \in nat \Longrightarrow n \leq succ(m) \Longrightarrow pred(n) \leq m$
by(*rule_tac n*=*n* **in** *natE*,*auto simp add*:*pred_type*[*of succ*(*m*)])

lemma *pred_le2* : $n \in nat \Longrightarrow m \in nat \Longrightarrow pred(n) \leq m \Longrightarrow n \leq succ(m)$
by(*subgoal_tac n*∈*nat*,*rule_tac n*=*n* **in** *natE*,*auto*)

lemma *Un_leD1* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow Ord(k) \Longrightarrow i \cup j \leq k \Longrightarrow i \leq k$
by (*rule Un_least_lt_iff*[*THEN iffD1*[*THEN conjunct1*]],*simp_all*)

lemma *Un_leD2* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow Ord(k) \Longrightarrow i \cup j \leq k \Longrightarrow j \leq k$
by (*rule Un_least_lt_iff*[*THEN iffD1*[*THEN conjunct2*]],*simp_all*)

lemma *gt1* : $n \in nat \Longrightarrow i \in n \Longrightarrow i \neq 0 \Longrightarrow i \neq 1 \Longrightarrow 1<i$
by(*rule_tac n=i* **in** *natE*,*erule in_n_in_nat*,*auto intro*: *Ord_0_lt*)

lemma *pred_mono* : $m \in nat \Longrightarrow n \leq m \Longrightarrow pred(n) \leq pred(m)$
by(*rule_tac n=n* **in** *natE*,*auto simp add*:*le_in_nat*,*erule_tac n=m* **in** *natE*,*auto*)

lemma *succ_mono* : $m \in nat \Longrightarrow n \leq m \Longrightarrow succ(n) \leq succ(m)$
by *auto*

lemma *union_abs1* :
$[\![i \leq j]\!] \Longrightarrow i \cup j = j$
by (*rule Un_absorb1*,*erule le_imp_subset*)

lemma *union_abs2* :
$[\![i \leq j]\!] \Longrightarrow j \cup i = j$
by (*rule Un_absorb2*,*erule le_imp_subset*)

lemma *ord_un_max* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow i \cup j = max(i,j)$
using *max_def union_abs1 not_lt_iff_le leI union_abs2*
by *auto*

lemma *ord_max_ty* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow Ord(max(i,j))$
unfolding *max_def* **by** *simp*

lemmas *ord_simp_union* = *ord_un_max ord_max_ty max_def*

lemma *le_succ* : $x \in nat \Longrightarrow x \leq succ(x)$ **by** *simp*

lemma *le_pred* : $x \in nat \Longrightarrow pred(x) \leq x$
using *pred_le*[*OF _ le_succ*] *pred_succ_eq*
by *simp*

lemma *Un_le_compat* : $o \leq p \Longrightarrow q \leq r \Longrightarrow Ord(o) \Longrightarrow Ord(p) \Longrightarrow Ord(q) \Longrightarrow Ord(r) \Longrightarrow o \cup q \leq p \cup r$
using *le_trans*[*of q r p∪r*,*OF _ Un_upper2_le*] *le_trans*[*of o p p∪r*,*OF _ Un_upper1_le*]
ord_simp_union
by *auto*

lemma *Un_le* : $p \leq r \Longrightarrow q \leq r \Longrightarrow$
$Ord(p) \Longrightarrow Ord(q) \Longrightarrow Ord(r) \Longrightarrow$
$p \cup q \leq r$
using *ord_simp_union* **by** *auto*

lemma *Un_leI3* : $o \leq r \Longrightarrow p \leq r \Longrightarrow q \leq r \Longrightarrow$
$Ord(o) \Longrightarrow Ord(p) \Longrightarrow Ord(q) \Longrightarrow Ord(r) \Longrightarrow$
$o \cup p \cup q \leq r$
using *ord_simp_union* **by** *auto*

lemma *diff_mono* :
  assumes $m \in nat$ $n \in nat$ $p \in nat$ $m < n$ $p \leq m$
  shows $m\#\text{-}p < n\#\text{-}p$
proof -
  from *assms*
  have $m\#\text{-}p \in nat$ $m\#\text{-}p \ \#+ p = m$
    using *add_diff_inverse2* **by** *simp_all*
  with *assms*
  show *?thesis*
    using *less_diff_conv*[*of n p m #- p*,*THEN iffD2*] **by** *simp*
qed

lemma *pred_Un*:
  $x \in nat \Longrightarrow y \in nat \Longrightarrow Arith.pred(succ(x) \cup y) = x \cup Arith.pred(y)$
  $x \in nat \Longrightarrow y \in nat \Longrightarrow Arith.pred(x \cup succ(y)) = Arith.pred(x) \cup y$
  using *pred_Un_distrib pred_succ_eq* **by** *simp_all*

lemma *le_natI* : $j \leq n \Longrightarrow n \in nat \Longrightarrow j \in nat$
  by(*drule ltD*,*rule in_n_in_nat*,*rule nat_succ_iff*[*THEN iffD2*,*of n*],*simp_all*)

lemma *le_natE* : $n \in nat \Longrightarrow j < n \Longrightarrow \ j \in n$
  by(*rule ltE*[*of j n*],*simp+*)

lemma *leD* : **assumes** $n \in nat$ $j \leq n$
  shows $j < n \mid j = n$
  using *leE*[*OF* ‹$j \leq n$›,*of* $j < n \mid j = n$] **by** *auto*

5.1 Some results in ordinal arithmetic

The following results are auxiliary to the proof of wellfoundedness of the relation *frecR*

lemma *max_cong* :
  assumes $x \leq y$ $Ord(y)$ $Ord(z)$
  shows $max(x,y) \leq max(y,z)$
proof (*cases* $y \leq z$)
  case *True*
  then show *?thesis*
    unfolding *max_def* **using** *assms* **by** *simp*
next
  case *False*
  then have $z \leq y$ **using** *assms not_le_iff_lt leI* **by** *simp*
  then show *?thesis*
    unfolding *max_def* **using** *assms* **by** *simp*
qed

lemma *max_commutes* :
  assumes $Ord(x)$ $Ord(y)$
  shows $max(x,y) = max(y,x)$

using *assms Un_commute ord_simp_union(1) ord_simp_union(1)*[*symmetric*]
by *auto*

lemma *max_cong2* :
assumes $x \leq y$ $Ord(y)$ $Ord(z)$ $Ord(x)$
shows $max(x,z) \leq max(y,z)$
proof -
from *assms*
have $x \cup z \leq y \cup z$
using *lt_Ord Ord_Un Un_mono*[*OF le_imp_subset*[*OF* ‹$x \leq y$›]] *subset_imp_le*
by *auto*
then show *?thesis*
using *ord_simp_union* ‹$Ord(x)$› ‹$Ord(z)$› ‹$Ord(y)$› **by** *simp*
qed

lemma *max_D1* :
assumes $x = y$ $w < z$ $Ord(x)$ $Ord(w)$ $Ord(z)$ $max(x,w) = max(y,z)$
shows $z \leq y$
proof -
from *assms*
have $w < x \cup w$ **using** *Un_upper2_lt*[*OF* ‹$w<z$›] *assms ord_simp_union* **by** *simp*
then
have $w < x$ **using** *assms lt_Un_iff*[*of x w w*] *lt_not_refl* **by** *auto*
then
have $y = y \cup z$ **using** *assms max_commutes ord_simp_union assms leI* **by** *simp*
then
show *?thesis* **using** *Un_leD2 assms* **by** *simp*
qed

lemma *max_D2* :
assumes $w = y \vee w = z$ $x < y$ $Ord(x)$ $Ord(w)$ $Ord(y)$ $Ord(z)$ $max(x,w) = max(y,z)$
shows $x < w$
proof -
from *assms*
have $x < z \cup y$ **using** *Un_upper2_lt*[*OF* ‹$x<y$›] **by** *simp*
then
consider (*a*) $x < y$ | (*b*) $x < w$
using *assms ord_simp_union* **by** *simp*
then show *?thesis* **proof** (*cases*)
case *a*
consider (*c*) $w = y$ | (*d*) $w = z$
using *assms* **by** *auto*
then show *?thesis* **proof** (*cases*)
case *c*
with *a* **show** *?thesis* **by** *simp*
next

```
      case d
      with a
      show ?thesis
      proof (cases y <w)
        case True
        then show ?thesis using lt_trans[OF ‹x<y›] by simp
      next
        case False
        then
        have w ≤ y
          using not_lt_iff_le[OF assms(5) assms(4)] by simp
        with ‹w=z›
        have max(z,y) = y  unfolding max_def using assms by simp
        with assms
        have ... = x ∪ w using ord_simp_union max_commutes  by simp
        then show ?thesis using le_Un_iff assms by blast
      qed
    qed
  next
    case b
    then show ?thesis .
  qed
qed

lemma oadd_lt_mono2 :
  assumes  Ord(n) Ord(α) Ord(β) α < β x < n y < n 0 <n
  shows n ** α ++ x < n **β ++ y
proof -
  consider (0) β=0 | (s) γ where  Ord(γ) β = succ(γ) | (l) Limit(β)
    using Ord_cases[OF ‹Ord(β)›,of ?thesis] by force
  then show ?thesis
  proof cases
    case 0
    then show ?thesis using ‹α<β› by auto
  next
    case s
    then
    have α≤γ using ‹α<β› using leI by auto
    then
    have n ** α ≤ n ** γ using omult_le_mono[OF _ ‹α≤γ›] ‹Ord(n)› by simp
    then
   have n ** α ++ x < n ** γ ++ n using oadd_lt_mono[OF _ ‹x<n›] by simp
    also
    have ... = n ** β using ‹β=succ(_)› omult_succ ‹Ord(β)› ‹Ord(n)› by simp
    finally
    have n ** α ++ x < n ** β by auto
    then
    show ?thesis using oadd_le_self ‹Ord(β)› lt_trans2 ‹Ord(n)› by auto
  next
```

case *l*
have *Ord*(*x*) **using** ‹*x*<*n*› *lt_Ord* **by** *simp*
with *l*
have *succ*(*α*) < *β* **using** *Limit_has_succ* ‹*α*<*β*› **by** *simp*
have *n* ** *α* ++ *x* < *n* ** *α* ++ *n*
using *oadd_lt_mono*[*OF le_refl*[*OF Ord_omult*[*OF* _ ‹*Ord*(*α*)›]] ‹*x*<*n*›]
‹*Ord*(*n*)› **by** *simp*
also
have ... = *n* ** *succ*(*α*) **using** *omult_succ* ‹*Ord*(*α*)› ‹*Ord*(*n*)› **by** *simp*
finally
have *n* ** *α* ++ *x* < *n* ** *succ*(*α*) **by** *simp*
with ‹*succ*(*α*) < *β*›
have *n* ** *α* ++ *x* < *n* ** *β* **using** *lt_trans omult_lt_mono* ‹*Ord*(*n*)› ‹*0*<*n*›
by *auto*
then show *?thesis* **using** *oadd_le_self* ‹*Ord*(*β*)› *lt_trans2* ‹*Ord*(*n*)› **by** *auto*
qed
qed
end

6 Various results missing from ZF.

theory *ZF_Miscellanea*
imports
ZF
Nat_Miscellanea
begin

definition
SepReplace :: [*i*, *i*⇒*i*, *i*⇒ *o*] ⇒ *i* **where**
SepReplace(*A*,*b*,*Q*) ≡ {*y* . *x*∈*A*, *y*=*b*(*x*) ∧ *Q*(*x*)}

syntax
SepReplace :: [*i*, *pttrn*, *i*, *o*] ⇒ *i* ((*1*{ ../ _ ∈ _, _}))
translations
{*b* .. *x*∈*A*, *Q*} => *CONST SepReplace*(*A*, λ*x*. *b*, λ*x*. *Q*)

lemma *Sep_and_Replace*: {*b*(*x*) .. *x*∈*A*, *P*(*x*) } = {*b*(*x*) . *x*∈{*y*∈*A*. *P*(*y*)}}
by (*auto simp add*:*SepReplace_def*)

lemma *SepReplace_subset* : *A*⊆*A*′⟹ {*b* .. *x*∈*A*, *Q*}⊆{*b* .. *x*∈*A*′, *Q*}
by (*auto simp add*:*SepReplace_def*)

lemma *SepReplace_iff* [*simp*]: *y*∈{*b*(*x*) .. *x*∈*A*, *P*(*x*)} ⟷ (∃ *x*∈*A*. *y*=*b*(*x*) & *P*(*x*))
by (*auto simp add*:*SepReplace_def*)

lemma *SepReplace_dom_implies* :
(⋀ *x* . *x* ∈*A* ⟹ *b*(*x*) = *b*′(*x*))⟹ {*b*(*x*) .. *x*∈*A*, *Q*(*x*)}={*b*′(*x*) .. *x*∈*A*, *Q*(*x*)}
by (*simp add*:*SepReplace_def*)

lemma *SepReplace_pred_implies* :
$\forall x.\ Q(x) \longrightarrow b(x) = b'(x) \Longrightarrow \{b(x)\ ..\ x{\in}A,\ Q(x)\} = \{b'(x)\ ..\ x{\in}A,\ Q(x)\}$
by (*force simp add*:*SepReplace_def*)

lemma *funcI* : $f \in A \to B \Longrightarrow a \in A \Longrightarrow b = f\ `\ a \Longrightarrow \langle a, b\rangle \in f$
by(*simp_all add*: *apply_Pair*)

lemma *vimage_fun_sing*:
assumes $f{\in}A{\to}B\ b{\in}B$
shows $\{a{\in}A\ .\ f`a{=}b\} = f\text{-}``\{b\}$
using *assms vimage_singleton_iff function_apply_equality Pi_iff funcI* **by** *auto*

lemma *image_fun_subset*: $S{\in}A{\to}B \Longrightarrow C{\subseteq}A \Longrightarrow \{S\ `\ x\ .\ x{\in}\ C\} = S``C$
using *image_function*[*symmetric*,*of S C*] *domain_of_fun Pi_iff* **by** *auto*

lemma *subset_Diff_Un*: $X \subseteq A \Longrightarrow A = (A\ -\ X) \cup X$ **by** *auto*

lemma *Diff_bij*:
assumes $\forall\, A{\in}F.\ X \subseteq A$ **shows** $(\lambda A{\in}F.\ A\text{-}X) \in bij(F, \{A\text{-}X.\ A{\in}F\})$
using *assms* **unfolding** *bij_def inj_def surj_def*
by (*auto intro*:*lam_type*, *subst subset_Diff_Un*[*of X*]) *auto*

lemma *function_space_nonempty*:
assumes $b{\in}B$
shows $(\lambda x{\in}A.\ b) : A \to B$
using *assms lam_type* **by** *force*

lemma *vimage_lam*: $(\lambda x{\in}A.\ f(x))\ \text{-}``\ B = \{\ x{\in}A\ .\ f(x) \in B\ \}$
using *lam_funtype*[*of A f*, *THEN* [*2*] *domain_type*]
lam_funtype[*of A f*, *THEN* [*2*] *apply_equality*] *lamI*[*of _ A f*]
by *auto blast*

lemma *range_fun_subset_codomain*:
assumes $h{:}B \to C$
shows $range(h) \subseteq C$
unfolding *range_def domain_def converse_def* **using** *range_type*[*OF _ assms*]
by *auto*

lemma *Pi_rangeD*:
assumes $f{\in}Pi(A,B)\ b \in range(f)$
shows $\exists\, a{\in}A.\ f`a = b$
using *assms apply_equality*[*OF _ assms*(*1*), *of _ b*]
domain_type[*OF _ assms*(*1*)] **by** *auto*

lemma *Pi_range_eq*: $f \in Pi(A,B) \Longrightarrow range(f) = \{f\ `\ x\ .\ x \in A\}$
using *Pi_rangeD*[*of f A B*] *apply_rangeI*[*of f A B*]
by *blast*

lemma *Pi_vimage_subset* : $f \in Pi(A,B) \Longrightarrow f\text{-}``C \subseteq A$

unfolding *Pi_def* **by** *auto*

definition
minimum :: $i \Rightarrow i \Rightarrow i$ **where**
$minimum(r,B) \equiv THE\ b.\ first(b,B,r)$

lemma *minimum_in*: $[\![\ well\_ord(A,r);\ B{\subseteq}A;\ B{\neq}0\ ]\!] \Longrightarrow minimum(r,B) \in B$
using *the_first_in* **unfolding** *minimum_def* **by** *simp*

lemma *well_ord_surj_imp_inj_inverse*:
assumes $well\_ord(A,r)$ $h \in surj(A,B)$
shows $(\lambda b{\in}B.\ minimum(r,\ \{a{\in}A.\ h`a{=}b\})) \in inj(B,A)$
proof -
let *?f*$=\lambda b{\in}B.\ minimum(r,\ \{a{\in}A.\ h`a{=}b\})$
have $minimum(r,\ \{a \in A\ .\ h\ `\ a = b\}) \in \{a{\in}A.\ h`a{=}b\}$ **if** $b{\in}B$ **for** b
proof -
from ‹$h \in surj(A,B)$› *that*
have $\{a{\in}A.\ h`a{=}b\} \neq 0$
unfolding *surj_def* **by** *blast*
with ‹*well_ord*(A,r)›
show $minimum(r,\{a{\in}A.\ h`a{=}b\}) \in \{a{\in}A.\ h`a{=}b\}$
using *minimum_in* **by** *blast*
qed
moreover from *this*
have *?f* : $B \to A$
using *lam_type*[*of* B _ $\lambda\_.A$] **by** *simp*
moreover
have *?f* ` $w$ = *?f* ` $x \Longrightarrow w = x$ **if** $w{\in}B$ $x{\in}B$ **for** w x
proof -
from *calculation that*
have $w = h\ `\ minimum(r,\{a{\in}A.\ h`a{=}w\})$
$x = h\ `\ minimum(r,\{a{\in}A.\ h`a{=}x\})$
by *simp_all*
moreover
assume *?f* ` $w$ = *?f* ` x
moreover from *this* **and** *that*
have $minimum(r,\ \{a \in A\ .\ h\ `\ a = w\}) = minimum(r,\ \{a \in A\ .\ h\ `\ a = x\})$
unfolding *minimum_def* **by** *simp_all*
moreover from *calculation*(*1*,*2*,*4*)
show $w{=}x$ **by** *simp*
qed
ultimately
show *?thesis*
unfolding *inj_def* **by** *blast*
qed

lemma *well_ord_surj_imp_lepoll*:
assumes $well\_ord(A,r)$ $h \in surj(A,B)$

shows $B \lesssim A$
unfolding *lepoll_def* **using** *well_ord_surj_imp_inj_inverse*[*OF assms*]
by *blast*

— New result
lemma *surj_imp_well_ord*:
assumes *well_ord*(*A*,*r*) $h \in surj(A,B)$
shows $\exists\, s.\ well\_ord(B,s)$
using *assms lepoll_well_ord*[*OF well_ord_surj_imp_lepoll*]
by *force*

lemma $Pow\_sing : Pow(\{a\}) = \{0,\{a\}\}$
proof(*intro equalityI*,*simp_all*)
have $z \in \{0,\{a\}\}$ **if** $z \subseteq \{a\}$ **for** z
using *that* **by** *auto*
then
show $Pow(\{a\}) \subseteq \{0, \{a\}\}$ **by** *auto*
qed

lemma *Pow_cons*:
shows $Pow(cons(a,A)) = Pow(A) \cup \{\{a\} \cup X\ .\ X{:}\ Pow(A)\}$
using *Un_Pow_subset Pow_sing*
proof(*intro equalityI*,*auto simp add*:*Un_Pow_subset*)
{
fix *C D*
assume $\bigwedge B\ .\ B{\in}Pow(A) \Longrightarrow C \neq \{a\} \cup B\ C \subseteq \{a\} \cup A\ D \in C$
moreover from *this*
have $\forall\, x{\in}C\ .\ x{=}a \vee x{\in}A$ **by** *auto*
moreover from *calculation*
consider $(a)\ D{=}a \mid (b)\ D{\in}A$ **by** *auto*
from *this*
have $D{\in}A$
proof(*cases*)
case *a*
with *calculation* **show** *?thesis* **by** *auto*
next
case *b*
then show *?thesis* **by** *simp*
qed
}
then show $\bigwedge x\ xa.\ (\forall\, xa{\in}Pow(A).\ x \neq \{a\} \cup xa) \Longrightarrow x \subseteq cons(a,\ A) \Longrightarrow xa \in x \Longrightarrow xa \in A$
by *auto*
qed

lemma *app_nm* :
assumes $n{\in}nat\ m{\in}nat\ f{\in}n{\rightarrow}m\ x \in nat$
shows $f`x \in nat$
proof(*cases* $x{\in}n$)

case *True*
then show *?thesis* **using** *assms in_n_in_nat apply_type* **by** *simp*
next
case *False*
then show *?thesis* **using** *assms apply_0 domain_of_fun* **by** *simp*
qed

lemma *Upair_eq_cons*: $Upair(a,b) = \{a,b\}$
unfolding *cons_def* **by** *auto*

lemma *converse_apply_eq* : $converse(f)\ `x = \bigcup(f\text{-}``\{x\})$
unfolding *apply_def vimage_def* **by** *simp*

end

7 Some enhanced theorems on recursion

theory *Recursion_Thms*
imports *ZF.Epsilon ZF-Constructible.Datatype_absolute*

begin

We prove results concerning definitions by well-founded recursion on some relation R and its transitive closure $R^{\widehat{}}*$

lemma *fld_restrict_eq* : $a \in A \Longrightarrow (r \cap A{\times}A)\text{-}``\{a\} = (r\text{-}``\{a\} \cap A)$
by(*force*)

lemma *fld_restrict_mono* : $relation(r) \Longrightarrow A \subseteq B \Longrightarrow r \cap A{\times}A \subseteq r \cap B{\times}B$
by(*auto*)

lemma *fld_restrict_dom* :
assumes $relation(r)$ $domain(r) \subseteq A$ $range(r) \subseteq A$
shows $r \cap A{\times}A = r$
proof (*rule equalityI*,*blast*,*rule subsetI*)
{ fix x
assume *xr*: $x \in r$
from *xr assms* **have** $\exists\ a\ b\ .\ x = \langle a,b\rangle$ **by** (*simp add*: *relation_def*)
then obtain $a\ b$ **where** $\langle a,b\rangle \in r$ $\langle a,b\rangle \in r \cap A{\times}A$ $x \in r \cap A{\times}A$
using *assms xr*
by *force*
then have $x \in r \cap A{\times}A$ **by** *simp*
}
then show $x \in r \Longrightarrow x \in r \cap A{\times}A$ **for** x **.**
qed

definition *tr_down* :: $[i,i] \Rightarrow i$
where $tr\_down(r,a) = (r^{\widehat{}}+)\text{-}``\{a\}$

lemma *tr_downD* : $x \in tr\_down(r,a) \Longrightarrow \langle x,a\rangle \in r^{\widehat{}}+$

by (*simp add*: *tr_down_def vimage_singleton_iff*)

lemma *pred_down* : *relation*(r) $\Longrightarrow$ r-‘‘$\{a\} \subseteq$ *tr_down*(r,a)
by(*simp add*: *tr_down_def vimage_mono r_subset_trancl*)

lemma *tr_down_mono* : *relation*(r) $\Longrightarrow$ $x \in r$-‘‘$\{a\} \Longrightarrow$ *tr_down*(r,x) $\subseteq$ *tr_down*(r,a)
by(*rule subsetI*,*simp add*:*tr_down_def*,*auto dest*: *underD*,*force simp add*: *underI r_into_trancl trancl_trans*)

lemma *rest_eq* :
assumes *relation*(r) **and** r-‘‘$\{a\} \subseteq B$ **and** $a \in B$
shows r-‘‘$\{a\} = (r \cap B \times B)$-‘‘$\{a\}$
proof (*intro equalityI subsetI*)
fix x
assume $x \in r$-‘‘$\{a\}$
then
have $x \in B$ **using** *assms* **by** (*simp add*: *subsetD*)
from ‹$x \in r$-‘‘$\{a\}$›
have $\langle x,a\rangle \in r$ **using** *underD* **by** *simp*
then
show $x \in (r \cap B \times B)$-‘‘$\{a\}$ **using** ‹$x \in B$› ‹$a \in B$› *underI* **by** *simp*
next
from *assms*
show $x \in r$ -‘‘ $\{a\}$ **if** $x \in (r \cap B \times B)$ -‘‘ $\{a\}$ **for** x
using *vimage_mono that* **by** *auto*
qed

lemma *wfrec_restr_eq* : $r' = r \cap A \times A \Longrightarrow$ *wfrec*[A](r,a,H) = *wfrec*(r',a,H)
by(*simp add*:*wfrec_on_def*)

lemma *wfrec_restr* :
assumes *rr*: *relation*(r) **and** *wfr*:*wf*(r)
shows $a \in A \Longrightarrow$ *tr_down*(r,a) $\subseteq A \Longrightarrow$ *wfrec*(r,a,H) = *wfrec*[A](r,a,H)
proof (*induct a arbitrary*:*A rule*:*wf_induct_raw*[*OF wfr*])
case (*1 a*)
have *wfRa* : *wf*[A](r)
using *wf_subset wfr wf_on_def Int_lower1* **by** *simp*
from *pred_down rr*
have r -‘‘ $\{a\} \subseteq$ *tr_down*(r, a) **.**
with *1*
have r-‘‘$\{a\} \subseteq A$ **by** (*force simp add*: *subset_trans*)
{
fix x
assume *x_a* : $x \in r$-‘‘$\{a\}$
with ‹r-‘‘$\{a\} \subseteq A$›
have $x \in A$ **..**
from *pred_down rr*
have *b* : r -‘‘$\{x\} \subseteq$ *tr_down*(r,x) **.**
then

```
    have tr_down(r,x) ⊆ tr_down(r,a)
      using tr_down_mono x_a rr by simp
    with 1
    have tr_down(r,x) ⊆ A using subset_trans by force
    have ⟨x,a⟩ ∈ r using x_a  underD by simp
    with 1 ‹tr_down(r,x) ⊆ A› ‹x ∈ A›
    have wfrec(r,x,H) = wfrec[A](r,x,H) by simp
  }
  then
  have x∈ r-‘‘{a} ⟹ wfrec(r,x,H) =  wfrec[A](r,x,H) for x  .
  then
  have Eq1 :(λ x ∈ r-‘‘{a} . wfrec(r,x,H)) = (λ x ∈ r-‘‘{a} . wfrec[A](r,x,H))
    using lam_cong by simp

  from assms
  have wfrec(r,a,H) = H(a,λ x ∈ r-‘‘{a} . wfrec(r,x,H)) by (simp add:wfrec)
  also
  have ... = H(a,λ x ∈ r-‘‘{a} . wfrec[A](r,x,H))
    using assms Eq1 by simp
  also from 1 ‹r-‘‘{a} ⊆ A›
  have ... = H(a,λ x ∈ (r∩A×A)-‘‘{a} . wfrec[A](r,x,H))
    using assms rest_eq  by simp
  also from ‹a∈A›
  have ... = H(a,λ x ∈ (r-‘‘{a})∩A . wfrec[A](r,x,H))
    using fld_restrict_eq by simp
  also from ‹a∈A› ‹wf[A](r)›
  have ... = wfrec[A](r,a,H) using wfrec_on by simp
  finally show ?case .
qed

lemmas wfrec_tr_down = wfrec_restr[OF _ _ _ subset_refl]

lemma wfrec_trans_restr : relation(r) ⟹ wf(r) ⟹ trans(r) ⟹ r-‘‘{a}⊆A ⟹
a ∈ A ⟹
  wfrec(r, a, H) = wfrec[A](r, a, H)
   by(subgoal_tac tr_down(r,a) ⊆ A,auto simp add : wfrec_restr tr_down_def
trancl_eq_r)


lemma field_trancl : field(r^+) = field(r)
  by (blast intro: r_into_trancl dest!: trancl_type [THEN subsetD])

definition
  Rrel :: [i⇒i⇒o,i] ⇒ i where
  Rrel(R,A) ≡ {z∈A×A. ∃ x y. z = ⟨x, y⟩ ∧ R(x,y)}

lemma RrelI : x ∈ A ⟹ y ∈ A ⟹ R(x,y) ⟹ ⟨x,y⟩ ∈ Rrel(R,A)
  unfolding Rrel_def by simp
```

lemma *Rrel_mem*: *Rrel*(*mem*,*x*) = *Memrel*(*x*)
unfolding *Rrel_def Memrel_def* **..**

lemma *relation_Rrel*: *relation*(*Rrel*(*R*,*d*))
unfolding *Rrel_def relation_def* **by** *simp*

lemma *field_Rrel*: *field*(*Rrel*(*R*,*d*)) ⊆ *d*
unfolding *Rrel_def* **by** *auto*

lemma *Rrel_mono* : *A* ⊆ *B* ⟹ *Rrel*(*R*,*A*) ⊆ *Rrel*(*R*,*B*)
unfolding *Rrel_def* **by** *blast*

lemma *Rrel_restr_eq* : *Rrel*(*R*,*A*) ∩ *B*×*B* = *Rrel*(*R*,*A*∩*B*)
unfolding *Rrel_def* **by** *blast*

lemma *field_Memrel* : *field*(*Memrel*(*A*)) ⊆ *A*

using *Rrel_mem field_Rrel* **by** *blast*

lemma *restrict_trancl_Rrel*:
assumes *R*(*w*,*y*)
shows *restrict*(*f*,*Rrel*(*R*,*d*)-‘‘{*y*})‘*w*
= *restrict*(*f*,(*Rrel*(*R*,*d*)^+)-‘‘{*y*})‘*w*
proof (*cases y*∈*d*)
let *?r*=*Rrel*(*R*,*d*) **and** *?s*=(*Rrel*(*R*,*d*))^+
case *True*
show *?thesis*
proof (*cases w*∈*d*)
case *True*
with ‹*y*∈*d*› *assms*
have ⟨*w*,*y*⟩∈*?r*
unfolding *Rrel_def* **by** *blast*
then
have ⟨*w*,*y*⟩∈*?s*
using *r_subset_trancl*[*of ?r*] *relation_Rrel*[*of R d*] **by** *blast*
with ‹⟨*w*,*y*⟩∈*?r*›
have *w*∈*?r*-‘‘{*y*} *w*∈*?s*-‘‘{*y*}
using *vimage_singleton_iff* **by** *simp_all*
then
show *?thesis* **by** *simp*
next
case *False*
then
have *w*∉*domain*(*restrict*(*f*,*?r*-‘‘{*y*}))
using *subsetD*[*OF field_Rrel*[*of R d*]] **by** *auto*
moreover from ‹*w*∉*d*›
have *w*∉*domain*(*restrict*(*f*,*?s*-‘‘{*y*}))
using *subsetD*[*OF field_Rrel*[*of R d*], *of w*] *field_trancl*[*of ?r*]

```
      fieldI1[of w y ?s] by auto
    ultimately
    have restrict(f,?r-‘‘{y})‘w = 0 restrict(f,?s-‘‘{y})‘w = 0
      unfolding apply_def by auto
    then show ?thesis by simp
  qed
next
  let ?r=Rrel(R,d)
  let ?s=?r^+
  case False
  then
  have ?r-‘‘{y}=0
    unfolding Rrel_def by blast
  then
  have w∉?r-‘‘{y} by simp
  with ‹y∉d› assms
  have y∉field(?s)
    using field_trancl subsetD[OF field_Rrel[of R d]] by force
  then
  have w∉?s-‘‘{y}
    using vimage_singleton_iff by blast
  with ‹w∉?r-‘‘{y}›
  show ?thesis by simp
qed

lemma restrict_trans_eq:
  assumes w ∈ y
  shows restrict(f,Memrel(eclose({x}))-‘‘{y})‘w
      = restrict(f,(Memrel(eclose({x}))^+)-‘‘{y})‘w
  using assms restrict_trancl_Rrel[of mem ] Rrel_mem by (simp)

lemma wf_eq_trancl:
  assumes ⋀ f y . H(y,restrict(f,R-‘‘{y})) = H(y,restrict(f,R^+-‘‘{y}))
  shows  wfrec(R, x, H) = wfrec(R^+, x, H) (is wfrec(?r,_,_) = wfrec(?r',_,_))
proof -
  have wfrec(R, x, H) = wftrec(?r^+, x, λy f. H(y, restrict(f,?r-‘‘{y})))
    unfolding wfrec_def ..
  also
  have  ... = wftrec(?r^+, x, λy f. H(y, restrict(f,(?r^+)-‘‘{y})))
    using assms by simp
  also
  have  ... =  wfrec(?r^+, x, H)
    unfolding wfrec_def using trancl_eq_r[OF relation_trancl trans_trancl] by
simp
  finally
  show ?thesis .
qed

lemma transrec_equal_on_Ord:
```

assumes
$\bigwedge x\ f\ .\ Ord(x) \Longrightarrow foo(x,f) = bar(x,f)$
$Ord(\alpha)$
shows
$transrec(\alpha,\ foo) = transrec(\alpha,\ bar)$
proof -
have $transrec(\beta,foo) = transrec(\beta,bar)$ **if** $Ord(\beta)$ **for** β
using *that*
proof (*induct rule*:*trans_induct*)
case (*step* β)
have $transrec(\beta,\ foo) = foo(\beta,\ \lambda x{\in}\beta.\ transrec(x,\ foo))$
using *def_transrec*[*of* $\lambda x.\ transrec(x,\ foo)$ *foo*] **by** *blast*
also from *assms* **and** *step*
have $\ldots = bar(\beta,\ \lambda x{\in}\beta.\ transrec(x,\ foo))$
by *simp*
also from *step*
have $\ldots = bar(\beta,\ \lambda x{\in}\beta.\ transrec(x,\ bar))$
by (*auto*)
also
have $\ldots = transrec(\beta,\ bar)$
using *def_transrec*[*of* $\lambda x.\ transrec(x,\ bar)$ *bar*, *symmetric*]
by *blast*
finally
show $transrec(\beta,\ foo) = transrec(\beta,\ bar)$ **.**
qed
with *assms*
show *?thesis* **by** *simp*
qed

— Next theorem is very similar to $[\![\bigwedge x\ f.\ Ord(x) \Longrightarrow ?foo(x,\ f) = ?bar(x,\ f);\ Ord(?\alpha)]\!] \Longrightarrow transrec(?\alpha,\ ?foo) = transrec(?\alpha,\ ?bar)$
lemma (**in** *M_eclose*) *transrec_equal_on_M*:
assumes
$\bigwedge x\ f\ .\ M(x) \Longrightarrow M(f) \Longrightarrow foo(x,f) = bar(x,f)$
$\bigwedge \beta.\ M(\beta) \Longrightarrow transrec\_replacement(M,is\_foo,\beta)\ relation2(M,is\_foo,foo)$
$strong\_replacement(M,\ \lambda x\ y.\ y = \langle x,\ transrec(x,\ foo)\rangle)$
$\forall\, x[M].\ \forall\, g[M].\ function(g) \longrightarrow M(foo(x,g))$
$M(\alpha)\ Ord(\alpha)$
shows
$transrec(\alpha,\ foo) = transrec(\alpha,\ bar)$
proof -
have $M(transrec(x,\ foo))$ **if** $Ord(x)$ **and** $M(x)$ **for** x
using *that assms transrec_closed*[*of is_foo*]
by *simp*
have $transrec(\beta,foo) = transrec(\beta,bar)\ M(transrec(\beta,foo))$ **if** $Ord(\beta)\ M(\beta)$ **for** β
using *that*
proof (*induct rule*:*trans_induct*)
case (*step* β)
moreover

```
    assume M(β)
    moreover
    note ‹Ord(β)⟹ M(β) ⟹ M(transrec(β, foo))›
    ultimately
    show M(transrec(β, foo)) by blast
    with step ‹M(β)› ‹⋀x. Ord(x)⟹ M(x) ⟹ M(transrec(x, foo))›
      ‹strong_replacement(M, λx y. y = ⟨x, transrec(x, foo)⟩)›
    have M(λx∈β. transrec(x, foo))
      using Ord_in_Ord transM[of _ β]
      by (rule_tac lam_closed) auto
    have transrec(β, foo) = foo(β, λx∈β. transrec(x, foo))
      using def_transrec[of λx. transrec(x, foo) foo] by blast
    also from assms and ‹M(λx∈β. transrec(x, foo))› ‹M(β)›
    have  ... = bar(β, λx∈β. transrec(x, foo))
      by simp
    also from step and ‹M(β)›
    have  ... = bar(β, λx∈β. transrec(x, bar))
      using transM[of _ β] by (auto)
    also
    have  ... = transrec(β, bar)
      using def_transrec[of λx. transrec(x, bar) bar, symmetric]
      by blast
    finally
    show transrec(β, foo) = transrec(β, bar) .
  qed
  with assms
  show ?thesis by simp
qed


lemma ordermap_restr_eq:
  assumes well_ord(X,r)
  shows ordermap(X, r) = ordermap(X, r ∩ X×X)
proof -
  let ?A=λx . Order.pred(X, x, r)
  let ?B=λx . Order.pred(X, x, r ∩ X × X)
  let ?F=λx f. f `` ?A(x)
  let ?G=λx f. f `` ?B(x)
  let ?P=λ z. z∈X ⟶ wfrec(r ∩ X × X,z,λx f. f `` ?A(x)) = wfrec(r ∩ X ×
X,z,λx f. f `` ?B(x))
  have pred_eq:
    Order.pred(X, x, r ∩ X × X) = Order.pred(X, x, r) if x∈X for x
    unfolding Order.pred_def using that by auto
  from assms
  have wf_onX:wf(r ∩ X × X) unfolding well_ord_def wf_on_def by simp
  {
    have ?P(z) for z
    proof(induct rule:wf_induct[where P=?P,OF wf_onX])
      case (1 x)
```

```
    {
      assume x∈X
      from 1
      have lam_eq:
          (λw∈(r ∩ X × X) -‘‘ {x}. wfrec(r ∩ X × X, w, ?F)) =
           (λw∈(r ∩ X × X) -‘‘ {x}. wfrec(r ∩ X × X, w, ?G)) (is ?L=?R)
      proof -
            have wfrec(r ∩ X × X, w, ?F) = wfrec(r ∩ X × X, w, ?G) if
w∈(r∩X×X)-‘‘{x} for w
          using 1 that by auto
        then show ?thesis using lam_cong[OF refl] by simp
      qed
      then
      have wfrec(r ∩ X × X, x, ?F) = ?L ‘‘ ?A(x)
        using wfrec[OF wf_onX,of x ?F] by simp
      also have ... =  ?R ‘‘ ?B(x)
        using lam_eq pred_eq[OF ‹x∈_›] by simp
      also
      have ... = wfrec(r ∩ X × X, x, ?G)
        using wfrec[OF wf_onX,of x ?G] by simp
      finally
      have wfrec(r ∩ X × X, x, ?F) = wfrec(r ∩ X × X, x, ?G) by simp
    }
    then
    show ?case by simp
  qed
 }
 then
 show ?thesis
   unfolding ordermap_def wfrec_on_def using Int_ac by simp
qed

end
theory Utils
 imports ZF-Constructible.Formula
begin
```

This theory encapsulates some ML utilities

```
ML_file‹Utils.ml›

end
```

8 Automatic synthesis of formulas

```
theory Synthetic_Definition
 imports ZF-Constructible.Formula Utils
 keywords
   synthesize :: thy_decl % ML
   and
```

synthesize_notc :: *thy_decl* % *ML*
and
generate_schematic :: *thy_decl* % *ML*
and
arity_theorem :: *thy_decl* % *ML*
and
manual_schematic :: *thy_goal_stmt* % *ML*
and
manual_arity :: *thy_goal_stmt* % *ML*
and
from_schematic
and
for
and
from_definition
and
assuming
and
intermediate

begin

named_theorems *fm_definitions Definitions of synthetized formulas.*

named_theorems *iff_sats Theorems for synthetising formulas.*

named_theorems *arity Theorems for arity of formulas.*

ML‹
```
val $‘ = curry ((op $) o swap)
infix $‘

infix 6 &&&
val op &&& = Utils.&&&

infix 6 ***
val op *** = Utils.***

fun arity_goal intermediate def_name lthy =
  let
    val thm = Proof_Context.get_thm lthy (def_name ^ "_def")
    val (_, tm, _) = Utils.thm_concl_tm lthy (def_name ^ "_def")
    val (def, tm) = tm |> Utils.dest_eq_tms'
    fun first_lambdas (Abs (body as (_, ty, _))) =
        if ty = @{typ i}
          then (op ::) (Term.dest_abs body |>> Utils.var_i ||> first_lambdas)
          else Term.dest_abs body |> first_lambdas o #2
      | first_lambdas _ = []
    val (def, vars) = Term.strip_comb def
```

```
    val vs = vars @ first_lambdas tm
    val def = fold (op $`) vs def
    val hyps = map (fn v => Utils.mem_ v Utils.nat_ |> Utils.tp) vs
    val concl = @{const IFOL.eq(i)} $ (@{const arity} $ def) $ Var ((ar, 0), @{typ
i})
    val g_iff = Logic.list_implies (hyps, Utils.tp concl)
    val attribs = if intermediate then [] else @{attributes [arity]}
  in
    (g_iff, arity_ ^ def_name ^ (if intermediate then ' else ), attribs, thm, vs)
  end

fun manual_arity intermediate def_name pos lthy =
  let
    val (goal, thm_name, attribs, _, _) = arity_goal intermediate def_name lthy
  in
    Proof.theorem NONE (fn thmss => Utils.display theorem pos
                                  o Local_Theory.note ((Binding.name thm_name,
attribs), hd thmss))
    [[(goal, [])]] lthy
  end

fun prove_arity thms goal ctxt =
  let
    val rules = Named_Theorems.get ctxt named_theorems ‹arity›
  in
    Goal.prove ctxt [] [] goal
   (K (rewrite_goal_tac ctxt thms 1 THEN Method.insert_tac ctxt rules 1 THEN
asm_simp_tac ctxt 1))
  end

fun auto_arity intermediate def_name pos lthy =
  let
    val (goal, thm_name, attribs, def_thm, vs) = arity_goal intermediate def_name
lthy
    val intermediate_text = if intermediate then intermediate else
    val help = You can manually prove the arity_theorem by typing:\n
          ^ manual_arity  ^ intermediate_text ^ for \ ^ def_name ^ \\n
    val thm = prove_arity [def_thm] goal lthy
          handle ERROR s => help ^ \n\n ^ s |> Exn.reraise o ERROR
    val thm = Utils.fix_vars thm (map Utils.freeName vs) lthy
  in
   Local_Theory.note ((Binding.name thm_name, attribs), [thm]) lthy |> Utils.display
theorem pos
  end

fun prove_tc_form goal thms ctxt =
  Goal.prove ctxt [] [] goal (K (rewrite_goal_tac ctxt thms 1 THEN TypeCheck.typecheck_tac
ctxt))
```

```
fun prove_sats_tm thm_auto thms goal ctxt =
  let
    val ctxt' = ctxt |> Simplifier.add_simp (hd thm_auto)
  in
    Goal.prove ctxt [] [] goal
   (K (rewrite_goal_tac ctxt thms 1 THEN PARALLEL_ALLGOALS (asm_simp_tac
ctxt')))
  end

fun prove_sats_iff goal ctxt = Goal.prove ctxt [] [] goal (K (asm_simp_tac ctxt
1))

fun is_mem (@{const mem} $ _ $  _) = true
  | is_mem _ = false

fun pre_synth_thm_sats term set env vars vs lthy =
  let
    val (_, tm, ctxt1) = Utils.thm_concl_tm lthy term
   val (thm_refs, ctxt2) = Variable.import true [Proof_Context.get_thm lthy term]
ctxt1 |>> #2
    val vs' = map (Thm.term_of o #2) vs
    val vars' = map (Thm.term_of o #2) vars
    val r_tm = tm |> Utils.dest_lhs_def |> fold (op $`) vs'
    val sats = @{const apply} $ (@{const satisfies} $ set $ r_tm) $ env
    val sats' = @{const IFOL.eq(i)} $ sats $ (@{const succ} $ @{const zero})
  in
    { vars = vars'
    , vs = vs'
    , sats = sats'
    , thm_refs = thm_refs
    , lthy = ctxt2
    , env = env
    , set = set
    }
  end

fun synth_thm_sats_gen name lhs hyps pos attribs aux_funs environment lthy =
  let
    val ctxt = (#prepare_ctxt aux_funs) lthy
    val ctxt = Utils.add_to_context (Utils.freeName (#set environment)) ctxt
    val (new_vs, ctxt') = (#create_variables aux_funs) (#vs environment, ctxt)
    val new_hyps = (#create_hyps aux_funs) (#vs environment, new_vs)
    val concl = (#make_concl aux_funs) (lhs, #sats environment, new_vs)
    val g_iff = Logic.list_implies (new_hyps @ hyps, Utils.tp concl)
    val thm = (#prover aux_funs) g_iff ctxt'
     val thm = Utils.fix_vars thm (map Utils.freeName ((#vars environment) @
new_vs)) lthy
  in
    Local_Theory.note ((name, attribs), [thm]) lthy |> Utils.display theorem pos
```

```
  end

fun synth_thm_sats_iff def_name lhs hyps pos environment =
  let
    val subst = Utils.zip_with (I *** I) (#vs environment)
    fun subst_nth (@{const nth} $ v $ _) new_vs = AList.lookup (op =) (subst
new_vs) v |> the
    | subst_nth (t1 $ t2) new_vs = (subst_nth t1 new_vs) $ (subst_nth t2 new_vs)
      | subst_nth (Abs (v, ty, t)) new_vs = Abs (v, ty, subst_nth t new_vs)
      | subst_nth t _ = t
    val name = Binding.name (def_name ^ "_iff_sats")
    val iff_sats_attrib = @{attributes [iff_sats]}
   val aux_funs = { prepare_ctxt = fold Utils.add_to_context (map Utils.freeName
(#vs environment))
                  , create_variables = fn (vs, ctxt) => Variable.variant_fixes (map
Utils.freeName vs) ctxt |>> map Utils.var_i
                 , create_hyps = fn (vs, new_vs) => Utils.zip_with (fn (v, nv) =>
Utils.eq_ (Utils.nth_ v (#env environment)) nv) vs new_vs |> map Utils.tp
                   , make_concl = fn (lhs, rhs, new_vs) => @{const IFOL.iff} $
(subst_nth lhs new_vs) $ rhs
                  , prover = prove_sats_iff
                  }
  in
    synth_thm_sats_gen name lhs hyps pos iff_sats_attrib aux_funs environment
  end

fun synth_thm_sats_fm def_name lhs hyps pos thm_auto environment =
  let
    val name = Binding.name ("sats_" ^ def_name ^ "_fm")
    val simp_attrib = @{attributes [simp]}
    val aux_funs = { prepare_ctxt = I
                  , create_variables = K [] *** I
                  , create_hyps = K []
                  , make_concl = fn (rhs, lhs, _) => @{const IFOL.iff} $ lhs $ rhs
                  , prover = prove_sats_tm thm_auto (#thm_refs environment)
                  }
  in
    synth_thm_sats_gen name lhs hyps pos simp_attrib aux_funs environment
  end

fun synth_thm_tc def_name term hyps vars pos lthy =
  let
    val (_,tm,ctxt1) = Utils.thm_concl_tm lthy term
   val (thm_refs,ctxt2) = Variable.import true [Proof_Context.get_thm lthy term]
ctxt1 |>> #2
    val vars' = map (Thm.term_of o #2) vars
    val tc_attrib = @{attributes [TC]}
    val r_tm = tm |> Utils.dest_lhs_def |> fold (op $`) vars'
    val concl = @{const mem} $ r_tm $ @{const formula}
```

```
    val g = Logic.list_implies(hyps, Utils.tp concl)
    val thm = prove_tc_form g thm_refs ctxt2
    val name = Binding.name (def_name ^ "_fm_type")
    val thm = Utils.fix_vars thm (map Utils.freeName vars') ctxt2
  in
    Local_Theory.note ((name, tc_attrib), [thm]) lthy |> Utils.display theorem pos
  end

fun synthetic_def def_name thm_ref pos tc auto thy =
  let
   val (((_,vars),thm_tms),_) = Variable.import true [Proof_Context.get_thm thy
thm_ref] thy
    val (tm,hyps) = thm_tms |> hd |> Thm.concl_of &&& Thm.prems_of
    val (lhs,rhs) = tm |> Utils.dest_iff_tms o Utils.dest_trueprop
    val ((set,t),env) = rhs |> Utils.dest_sats_frm
    fun olist t = Ord_List.make String.compare (Term.add_free_names t [])
    fun relevant ts (@{const mem} $ t $ _) = (not (t = @{term 0})) andalso (not
(Term.is_Free t) orelse
        Ord_List.member String.compare ts (t |> Term.dest_Free |> #1))
      | relevant _ _ = false
    val t_vars = olist t
    val vs = List.filter (Utils.inList t_vars o #1 o #1 o #1) vars
    val at = List.foldr (fn ((_,var),t') => lambda (Thm.term_of var) t') t vs
    val hyps' = List.filter (relevant t_vars o Utils.dest_trueprop) hyps
    val def_attrs = @{attributes [fm_definitions]}
  in
    Local_Theory.define ((Binding.name (def_name ^ "_fm"), NoSyn),
                         ((Binding.name (def_name ^ "_fm_def"), def_attrs), at)) thy
    |>> (#2 #> I *** single) |> Utils.display theorem pos |>
    (if tc then synth_thm_tc def_name (def_name ^ "_fm_def") hyps' vs pos else
I) |>
    (if auto then
      pre_synth_thm_sats (def_name ^ "_fm_def") set env vars vs
      #> I &&& #lthy
     #> #1 &&& uncurry (synth_thm_sats_fm def_name lhs hyps pos thm_tms)
      #> uncurry (synth_thm_sats_iff def_name lhs hyps pos)
    else I)
  end

fun prove_schematic thms goal ctxt =
  let
    val rules = Named_Theorems.get ctxt named_theorems‹iff_sats›
  in
    Goal.prove ctxt [] [] goal
    (K (rewrite_goal_tac ctxt thms 1 THEN REPEAT1 (CHANGED (resolve_tac
ctxt rules 1 ORELSE asm_simp_tac ctxt 1))))
  end

val valid_assumptions = [ (nonempty, Utils.mem_ @{term 0})
```

```
                    ]

fun pre_schematic_def target assuming lthy =
let
    val thm = Proof_Context.get_thm lthy (target ^ "_def")
    val (vars, tm, ctxt1) = Utils.thm_concl_tm lthy (target ^ "_def")
     val (const, tm) = tm |> Utils.dest_eq_tms' o Utils.dest_trueprop |>> #1 o
strip_comb
    val t_vars = Term.add_free_names tm []
    val vs = List.filter (#1 #> #1 #> #1 #> Utils.inList t_vars) vars
            |> List.filter ((curry op = @{typ i}) o #2 o #1)
            |> List.map (Utils.var_i o #1 o #1 o #1)
    fun first_lambdas (Abs (body as (_, ty, _))) =
        if ty = @{typ i}
          then (op ::) (Term.dest_abs body |>> Utils.var_i ||> first_lambdas)
          else Term.dest_abs body |> first_lambdas o #2
      | first_lambdas _ = []
    val vs = vs @ (first_lambdas tm)
    val ctxt1' = fold Utils.add_to_context (map Utils.freeName vs) ctxt1
    val (set, ctxt2) = Variable.variant_fixes ["A"] ctxt1' |>> Utils.var_i o hd
    val class = @{const setclass} $ set
    val (env, ctxt3) = Variable.variant_fixes ["env"] ctxt2 |>> Utils.var_i o hd
     val assumptions = filter (Utils.inList assuming o #1) valid_assumptions |>
map #2
    val hyps = (List.map (fn v => Utils.tp (Utils.mem_ v Utils.nat_)) vs)
              @ [Utils.tp (Utils.mem_ env (Utils.list_ set))]
                @ Utils.zip_with (fn (f,x) => Utils.tp (f x)) assumptions (replicate
(length assumptions) set)
    val args = class :: map (fn v => Utils.nth_ v env) vs
    val (fm_name, ctxt4) = Variable.variant_fixes ["fm"] ctxt3 |>> hd
    val fm_type = fold (K (fn acc => Type ("fun", [@{typ i}, acc]))) vs @{typ i}
    val fm = Var (("fm_name", 0), fm_type)
    val lhs = fold (op $`) args const
    val fm_app = fold (op $`) vs fm
    val sats = @{const apply} $ (@{const satisfies} $ set $ fm_app) $ env
    val rhs = @{const IFOL.eq(i)} $ sats $ (@{const succ} $ @{const zero})
    val concl = @{const IFOL.iff} $ lhs $ rhs
    val schematic = Logic.list_implies (hyps, Utils.tp concl)
  in
    (schematic, ctxt4, thm, set, env, vs)
  end

fun str_join _ [] =
  | str_join _ [s] = s
  | str_join c (s :: ss) = s ^ c ^ (str_join c ss)

fun schematic_def def_name target assuming pos lthy =
  let
     val (schematic, ctxt, thm, set, env, vs) = pre_schematic_def target assuming
```

```
lthy
    val assuming_text = if null assuming then  else assuming  ^ (map (fn s => \
^ s ^ \) assuming |> str_join  )
    val help = You can manually prove the schematic_goal by typing:\n
          ^ manual_schematic [sch_name] for \ ^ target ^ \ ^ assuming_text ^ \n
          ^ And then complete the synthesis with:\n
          ^ synthesize \ ^ target ^ \ from_schematic [sch_name]\n
          ^ In both commands, «sch_name» must be the same and, if ommited,
will be defaulted to sats_ ^ target ^ _fm_auto.\n
          ^ You can also try adding new assumptions and/or synthetizing definitions
of sub-terms.
    val thm = prove_schematic [thm] schematic ctxt
          handle ERROR s => help ^ \n\n ^ s |> Exn.reraise o ERROR
    val thm = Utils.fix_vars thm (map Utils.freeName (set :: env :: vs)) lthy
  in
    Local_Theory.note ((Binding.name def_name, []), [thm]) lthy |> Utils.display
theorem pos
  end

fun schematic_synthetic_def def_name target assuming pos tc auto =
    (synthetic_def def_name (sats_ ^ def_name ^ _fm_auto) pos tc auto)
    o (schematic_def (sats_ ^ def_name ^ _fm_auto) target assuming pos)

fun manual_schematic def_name target assuming pos lthy =
  let
    val (schematic, _, _, _, _, _) = pre_schematic_def target assuming lthy
  in
    Proof.theorem NONE (fn thmss => Utils.display theorem pos
                                o Local_Theory.note ((Binding.name def_name, []),
hd thmss))
    [[(schematic, [])]] lthy
  end
⟩

ML⟨
local
  val simple_from_schematic_synth_constdecl =
      Parse.string --| (Parse.$$$ from_schematic)
      >> (fn bndg => synthetic_def bndg (sats_ ^ bndg ^ _fm_auto))

  val full_from_schematic_synth_constdecl =
      Parse.string -- ((Parse.$$$ from_schematic |-- Parse.thm))
     >> (fn (bndg,thm) => synthetic_def bndg (#1 (thm |>> Facts.ref_name)))

  val full_from_definition_synth_constdecl =
     Parse.string -- ((Parse.$$$ from_definition |-- Parse.string)) -- (Scan.optional
(Parse.$$$ assuming |-- Scan.repeat Parse.string) [])
      >> (fn ((bndg,target), assuming) => schematic_synthetic_def bndg target
assuming)
```

```
  val simple_from_definition_synth_constdecl =
    Parse.string -- (Parse.$$$ from_definition |-- (Scan.optional (Parse.$$$ assuming
|-- Scan.repeat Parse.string)) [])
      >> (fn (bndg, assuming) => schematic_synthetic_def bndg bndg assuming)

  val synth_constdecl =
    Parse.position (full_from_schematic_synth_constdecl || simple_from_schematic_synth_constdecl
                    || full_from_definition_synth_constdecl
                    || simple_from_definition_synth_constdecl)

  val full_schematic_decl =
      Parse.string -- ((Parse.$$$ for |-- Parse.string)) -- (Scan.optional (Parse.$$$
assuming |-- Scan.repeat Parse.string) [])

  val simple_schematic_decl =
      (Parse.$$$ for |-- Parse.string >> (fn name => sats_ ^ name ^ _fm_auto)
&&& I) -- (Scan.optional (Parse.$$$ assuming |-- Scan.repeat Parse.string) [])

  val schematic_decl = Parse.position (full_schematic_decl || simple_schematic_decl)

  val _ =
      Outer_Syntax.local_theory command_keyword⟨synthesize⟩ ML setup for
synthetic definitions
      (synth_constdecl >> (fn (f,p) => f p true true))

  val _ =
    Outer_Syntax.local_theory command_keyword⟨synthesize_notc⟩ ML setup
for synthetic definitions
      (synth_constdecl >> (fn (f,p) => f p false false))

  val _ = Outer_Syntax.local_theory command_keyword⟨generate_schematic⟩
ML setup for schematic goals
       (schematic_decl >> (fn (((bndg,target), assuming),p) => schematic_def
bndg target assuming p))

  val _ = Outer_Syntax.local_theory_to_proof command_keyword⟨manual_schematic⟩
ML setup for schematic goals
      (schematic_decl >> (fn (((bndg,target), assuming),p) => manual_schematic
bndg target assuming p))

   val arity_parser = Parse.position ((Scan.option (Parse.$$$ intermediate) >>
is_some) -- (Parse.$$$ for |-- Parse.string))

  val _ = Outer_Syntax.local_theory_to_proof command_keyword⟨manual_arity⟩
ML setup for arities
         (arity_parser >> (fn ((intermediate, target), pos) => manual_arity
intermediate target pos))
```

val _ = Outer_Syntax.local_theory ***command_keyword***‹*arity_theorem*› *ML setup for arities*
(*arity_parser* >> (*fn* ((*intermediate*, *target*), *pos*) => *auto_arity intermediate target pos*))

in

end
›

The `synthetic_def` function extracts definitions from schematic goals. A new definition is added to the context.

end

9 Aids to internalize formulas

theory *Internalizations*
imports
ZF-Constructible.DPow_absolute
Synthetic_Definition
begin

notation *Member* (‹·_ ∈/ _·›)
notation *Equal* (‹·_ =/ _·›)
notation *Nand* (‹·¬'(_ ∧/ _')·›)
notation *And* (‹·_ ∧/ _·›)
notation *Or* (‹·_ ∨/ _·›)
notation *Iff* (‹·_ ↔/ _·›)
notation *Implies* (‹·_ →/ _·›)
notation *Neg* (‹·¬_·›)
notation *Forall* (‹'(·∀(/_)·')›)
notation *Exists* (‹'(·∃(/_)·')›)

notation *subset_fm* (‹·_ ⊆/ _·›)
notation *succ_fm* (‹·*succ*'(_') *is* _·›)
notation *empty_fm* (‹·_ *is empty*·›)
notation *fun_apply_fm* (‹·_`_ *is* _·›)
notation *big_union_fm* (‹·⋃_ *is* _·›)
notation *upair_fm* (‹·{_,_} *is* _ ·›)
notation *ordinal_fm* (‹·_ *is ordinal*·›)

abbreviation
fm_surjection :: [*i*,*i*,*i*] ⇒ *i* (‹·_ *surjects* _ *to* _·›) **where**
fm_surjection(*f*,*A*,*B*) ≡ *surjection_fm*(*A*,*B*,*f*)

abbreviation
fm_typedfun :: [*i*,*i*,*i*] ⇒ *i* (‹·_ : _ → _·›) **where**
fm_typedfun(*f*,*A*,*B*) ≡ *typed_function_fm*(*A*,*B*,*f*)

We found it useful to have slightly different versions of some results in ZF-Constructible:

lemma *nth_closed* :
  assumes *env∈list(A) 0∈A*
  shows *nth(n,env)∈A*
  using *assms* **unfolding** *nth_def* **by** (*induct env*; *simp*)

lemma *mem_model_iff_sats* [*iff_sats*]:
      [| *0 ∈ A*; *nth(i,env) = x*; *env ∈ list(A)*|]
       ==> (*x∈A*) ⟷ *sats*(*A*, *Exists*(*Equal*(*0*,*0*)), *env*)
  using *nth_closed*[*of env A i*]
  by *auto*

lemma *not_mem_model_iff_sats* [*iff_sats*]:
      [| *0 ∈ A*; *nth(i,env) = x*; *env ∈ list(A)*|]
       ==> (∀ *x* . *x* ∉ *A*) ⟷ *sats*(*A*, *Neg*(*Exists*(*Equal*(*0*,*0*))), *env*)
  by *auto*

lemma *top_iff_sats* [*iff_sats*]:
  env ∈ list(A) ⟹ *0 ∈ A* ⟹ *sats*(*A*, *Exists*(*Equal*(*0*,*0*)), *env*)
  by *auto*

lemma *prefix1_iff_sats*[*iff_sats*]:
  assumes
    x ∈ nat env ∈ list(A) 0 ∈ A a ∈ A
  shows
    a = nth(x,env) ⟷ *sats*(*A*, *Equal*(*0*,*x#+1*), *Cons*(*a*,*env*))
    nth(x,env) = a ⟷ *sats*(*A*, *Equal*(*x#+1*,*0*), *Cons*(*a*,*env*))
    a ∈ nth(x,env) ⟷ *sats*(*A*, *Member*(*0*,*x#+1*), *Cons*(*a*,*env*))
    nth(x,env) ∈ a ⟷ *sats*(*A*, *Member*(*x#+1*,*0*), *Cons*(*a*,*env*))
  using *assms nth_closed*
  by *simp_all*

lemma *prefix2_iff_sats*[*iff_sats*]:
  assumes
    x ∈ nat env ∈ list(A) 0 ∈ A a ∈ A b ∈ A
  shows
    b = nth(x,env) ⟷ *sats*(*A*, *Equal*(*1*,*x#+2*), *Cons*(*a*,*Cons*(*b*,*env*)))
    nth(x,env) = b ⟷ *sats*(*A*, *Equal*(*x#+2*,*1*), *Cons*(*a*,*Cons*(*b*,*env*)))
    b ∈ nth(x,env) ⟷ *sats*(*A*, *Member*(*1*,*x#+2*), *Cons*(*a*,*Cons*(*b*,*env*)))
    nth(x,env) ∈ b ⟷ *sats*(*A*, *Member*(*x#+2*,*1*), *Cons*(*a*,*Cons*(*b*,*env*)))
  using *assms nth_closed*
  by *simp_all*

lemma *prefix3_iff_sats*[*iff_sats*]:
  assumes
    x ∈ nat env ∈ list(A) 0 ∈ A a ∈ A b ∈ A c ∈ A
  shows
    c = nth(x,env) ⟷ *sats*(*A*, *Equal*(*2*,*x#+3*), *Cons*(*a*,*Cons*(*b*,*Cons*(*c*,*env*))))

$nth(x,env) = c \longleftrightarrow sats(A,\ Equal(x\#+3,2),\ Cons(a,Cons(b,Cons(c,env))))$
$c \in nth(x,env) \longleftrightarrow sats(A,\ Member(2,x\#+3),\ Cons(a,Cons(b,Cons(c,env))))$
$nth(x,env) \in c \longleftrightarrow sats(A,\ Member(x\#+3,2),\ Cons(a,Cons(b,Cons(c,env))))$
using *assms nth_closed*
by *simp_all*

lemmas *FOL_sats_iff = sats_Nand_iff sats_Forall_iff sats_Neg_iff sats_And_iff sats_Or_iff sats_Implies_iff sats_Iff_iff sats_Exists_iff*

lemma *nth_ConsI*: $[\![nth(n,l) = x;\ n \in nat]\!] \Longrightarrow nth(succ(n),\ Cons(a,l)) = x$
by *simp*

lemmas *nth_rules = nth_0 nth_ConsI nat_0I nat_succI*
lemmas *sep_rules = nth_0 nth_ConsI FOL_iff_sats function_iff_sats fun_plus_iff_sats successor_iff_sats omega_iff_sats FOL_sats_iff Replace_iff_sats*

Also a different compilation of lemmas (term*sep_rules*) used in formula synthesis

lemmas *fm_defs =*
omega_fm_def limit_ordinal_fm_def empty_fm_def typed_function_fm_def pair_fm_def upair_fm_def domain_fm_def function_fm_def succ_fm_def cons_fm_def fun_apply_fm_def image_fm_def big_union_fm_def union_fm_def relation_fm_def composition_fm_def field_fm_def ordinal_fm_def range_fm_def transset_fm_def subset_fm_def Replace_fm_def

lemmas *formulas_def [fm_definitions] = fm_defs*
is_iterates_fm_def iterates_MH_fm_def is_wfrec_fm_def is_recfun_fm_def is_transrec_fm_def
is_nat_case_fm_def quasinat_fm_def number1_fm_def ordinal_fm_def finite_ordinal_fm_def cartprod_fm_def sum_fm_def Inr_fm_def Inl_fm_def formula_functor_fm_def
Memrel_fm_def transset_fm_def subset_fm_def pre_image_fm_def restriction_fm_def list_functor_fm_def tl_fm_def quasilist_fm_def Cons_fm_def Nil_fm_def

lemmas *sep_rules' [iff_sats] = nth_0 nth_ConsI FOL_iff_sats function_iff_sats fun_plus_iff_sats omega_iff_sats FOL_sats_iff*

end

10 The binder *Least*

theory *Least*
imports
Internalizations

begin

We have some basic results on the least ordinal satisfying a predicate.

lemma *Least_Ord*: $(\mu\ \alpha.\ R(\alpha)) = (\mu\ \alpha.\ Ord(\alpha) \wedge R(\alpha))$
unfolding *Least_def* **by** (*simp add:lt_Ord*)

lemma *Ord_Least_cong*:
assumes $\bigwedge y.\ Ord(y) \Longrightarrow R(y) \longleftrightarrow Q(y)$
shows $(\mu\ \alpha.\ R(\alpha)) = (\mu\ \alpha.\ Q(\alpha))$
proof -
from *assms*
have $(\mu\ \alpha.\ Ord(\alpha) \wedge R(\alpha)) = (\mu\ \alpha.\ Ord(\alpha) \wedge Q(\alpha))$
by *simp*
then
show *?thesis* **using** *Least_Ord* **by** *simp*
qed

definition
$least :: [i \Rightarrow o, i \Rightarrow o, i] \Rightarrow o$ **where**
$least(M,Q,i) \equiv ordinal(M,i) \wedge ($
$(empty(M,i) \wedge (\forall\, b[M].\ ordinal(M,b) \longrightarrow \neg Q(b)))$
$\vee\ (Q(i) \wedge (\forall\, b[M].\ ordinal(M,b) \wedge b \in i \longrightarrow \neg Q(b))))$

definition
$least\_fm :: [i,i] \Rightarrow i$ **where**
$least\_fm(q,i) \equiv And(ordinal\_fm(i),$
$Or(And(empty\_fm(i), Forall(Implies(ordinal\_fm(0), Neg(q)))),$
$And(Exists(And(q, Equal(0, succ(i)))),$
$Forall(Implies(And(ordinal\_fm(0), Member(0, succ(i))), Neg(q))))))$

lemma *least_fm_type*[*TC*] :$i \in nat \Longrightarrow q \in formula \Longrightarrow least\_fm(q,i) \in formula$
unfolding *least_fm_def*
by *simp*

lemmas *basic_fm_simps* = *sats_subset_fm'* *sats_transset_fm'* *sats_ordinal_fm'*

lemma *sats_least_fm* :
assumes *p_iff_sats*:
$\bigwedge a.\ a \in A \Longrightarrow P(a) \longleftrightarrow sats(A,\ p,\ Cons(a,\ env))$
shows
$[\![y \in nat;\ env \in list(A)\ ;\ 0 \in A]\!]$
$\Longrightarrow sats(A,\ least\_fm(p,y),\ env) \longleftrightarrow$
$least(\#\#A,\ P,\ nth(y,env))$
using *nth_closed p_iff_sats* **unfolding** *least_def least_fm_def*
by (*simp add:basic_fm_simps*)

lemma *least_iff_sats* [*iff_sats*]:
assumes *is_Q_iff_sats*:
$\bigwedge a.\ a \in A \Longrightarrow is\_Q(a) \longleftrightarrow sats(A,\ q,\ Cons(a,env))$
shows
$[\![nth(j,env) = y;\ j \in nat;\ env \in list(A);\ 0 \in A]\!]$

$\Longrightarrow$ *least*(*##A*, *is_Q*, *y*) $\longleftrightarrow$ *sats*(*A*, *least_fm*(*q*,*j*), *env*)
using *sats_least_fm* [*OF is_Q_iff_sats*, *of j* , *symmetric*]
by *simp*

lemma *least_conj*: *a*∈*M* $\Longrightarrow$ *least*(*##M*, $\lambda x.\ x$∈*M* $\wedge$ *Q*(*x*),*a*) $\longleftrightarrow$ *least*(*##M*,*Q*,*a*)
unfolding *least_def* **by** *simp*

context *M_trivial*
begin

10.1 Uniqueness, absoluteness and closure under *Least*

lemma *unique_least*:
assumes *M*(*a*) *M*(*b*) *least*(*M*,*Q*,*a*) *least*(*M*,*Q*,*b*)
shows *a*=*b*
proof -
from *assms*
have *Ord*(*a*) *Ord*(*b*)
unfolding *least_def*
by *simp_all*
then
consider (*le*) *a*∈*b* | *a*=*b* | (*ge*) *b*∈*a*
using *Ord_linear*[*OF* ‹*Ord*(*a*)› ‹*Ord*(*b*)›] **by** *auto*
then
show *?thesis*
proof(*cases*)
case *le*
then show *?thesis* **using** *assms* **unfolding** *least_def* **by** *auto*
next
case *ge*
then show *?thesis* **using** *assms* **unfolding** *least_def* **by** *auto*
qed
qed

lemma *least_abs*:
assumes $\bigwedge x.\ Q(x) \Longrightarrow Ord(x) \Longrightarrow \exists\, y[M].\ Q(y) \wedge Ord(y)\ M(a)$
shows *least*(*M*,*Q*,*a*) $\longleftrightarrow$ *a* = (μ *x*. *Q*(*x*))
unfolding *least_def*
proof (*cases* $\forall\, b[M].\ Ord(b) \longrightarrow \neg\ Q(b)$; *intro iffI*; *simp add*:*assms*)
case *True*
with *assms*
have $\neg\ (\exists\, i.\ Ord(i) \wedge Q(i))$ **by** *blast*
then
show *0* =(μ *x*. *Q*(*x*)) **using** *Least_0* **by** *simp*
then
show *ordinal*(*M*, μ *x*. *Q*(*x*)) $\wedge$ (*empty*(*M*, *Least*(*Q*)) $\vee$ *Q*(*Least*(*Q*)))
by *simp*
next

assume $\exists\, b[M].\ Ord(b) \wedge Q(b)$
then
obtain i **where** $M(i)$ $Ord(i)$ $Q(i)$ **by** *blast*
assume $a = (\mu\ x.\ Q(x))$
moreover
note ‹$M(a)$›
moreover from ‹$Q(i)$› ‹$Ord(i)$›
have $Q(\mu\ x.\ Q(x))$ (**is** *?G*)
by (*blast intro*:*LeastI*)
moreover
have $(\forall\, b[M].\ Ord(b) \wedge b \in (\mu\ x.\ Q(x)) \longrightarrow \neg\ Q(b))$ (**is** *?H*)
using *less_LeastE*[*of Q _ False*]
by (*auto*, *drule_tac ltI*, *simp*, *blast*)
ultimately
show $ordinal(M,\ \mu\ x.\ Q(x)) \wedge (empty(M,\ \mu\ x.\ Q(x)) \wedge (\forall\, b[M].\ Ord(b) \longrightarrow \neg\ Q(b)) \vee ?G \wedge ?H)$
by *simp*
next
assume *1*:$\exists\, b[M].\ Ord(b) \wedge Q(b)$
then
obtain i **where** $M(i)$ $Ord(i)$ $Q(i)$ **by** *blast*
assume $Ord(a) \wedge (a = 0 \wedge (\forall\, b[M].\ Ord(b) \longrightarrow \neg\ Q(b)) \vee Q(a) \wedge (\forall\, b[M].\ Ord(b) \wedge b \in a \longrightarrow \neg\ Q(b)))$
with *1*
have $Ord(a)$ $Q(a)$ $\forall\, b[M].\ Ord(b) \wedge b \in a \longrightarrow \neg\ Q(b)$
by *blast*+
moreover from *this* **and** *assms*
have $Ord(b) \Longrightarrow b \in a \Longrightarrow \neg\ Q(b)$ **for** b
by (*auto dest*:*transM*)
moreover from *this* **and** ‹$Ord(a)$›
have $b < a \Longrightarrow \neg\ Q(b)$ **for** b
unfolding *lt_def* **using** *Ord_in_Ord* **by** *blast*
ultimately
show $a = (\mu\ x.\ Q(x))$
using *Least_equality* **by** *simp*
qed

lemma *Least_closed*:
assumes $\bigwedge x.\ Q(x) \Longrightarrow Ord(x) \Longrightarrow \exists\, y[M].\ Q(y) \wedge Ord(y)$
shows $M(\mu\ x.\ Q(x))$
using *assms Least_le*[*of Q*] *Least_0*[*of Q*]
by (*cases* $(\exists\, i[M].\ Ord(i) \wedge Q(i))$) (*force dest*:*transM ltD*)+

Older, easier to apply versions (with a simpler assumption on Q).

lemma *least_abs'*:
assumes $\bigwedge x.\ Q(x) \Longrightarrow M(x)$ $M(a)$
shows $least(M,Q,a) \longleftrightarrow a = (\mu\ x.\ Q(x))$
using *assms least_abs*[*of Q*] **by** *auto*

lemma *Least_closed'*:
  assumes $\bigwedge x.\ Q(x) \Longrightarrow M(x)$
  shows $M(\mu\ x.\ Q(x))$
  using *assms Least_closed*[*of Q*] **by** *auto*

end

end

11 Fully relational versions of higher order construct

theory *Higher_Order_Constructs*
  imports
    ZF-Constructible.Relative
    ZF-Constructible.Datatype_absolute
    Recursion_Thms
    Least
begin

syntax
  sats :: [*i*, *i*, *i*] ⇒ *o* ((, _ ⊨ _) [*36*,*36*,*36*] *25*)
translations
  (*M*,*env* ⊨ *φ*) ⇌ *CONST sats*(*M*,*φ*,*env*)

definition
  is_If :: [*i*⇒*o*,*o*,*i*,*i*,*i*] ⇒ *o* **where**
  is_If(*M*,*b*,*t*,*f*,*r*) ≡ (*b* ⟶ *r*=*t*) ∧ (¬*b* ⟶ *r*=*f*)

lemma (**in** *M_trans*) *If_abs*:
    is_If(*M*,*b*,*t*,*f*,*r*) ⟷ *r* = *If*(*b*,*t*,*f*)
by (*simp add*: *is_If_def*)

definition
  is_If_fm :: [*i*,*i*,*i*,*i*] ⇒ *i* **where**
  is_If_fm(*φ*,*t*,*f*,*r*) ≡ *Or*(*And*(*φ*,*Equal*(*t*,*r*)),*And*(*Neg*(*φ*),*Equal*(*f*,*r*)))

lemma *is_If_fm_type* [*TC*]: *φ* ∈ *formula* ⟹ *t* ∈ *nat* ⟹ *f* ∈ *nat* ⟹ *r* ∈ *nat*
⟹
  is_If_fm(*φ*,*t*,*f*,*r*) ∈ *formula*
  unfolding *is_If_fm_def* **by** *auto*

lemma *sats_is_If_fm*:
  assumes *Qsats*: *Q* ⟷ *A*, *env* ⊨ *φ* *env* ∈ *list*(*A*)
   shows *is_If*(##*A*, *Q*, *nth*(*t*, *env*), *nth*(*f*, *env*), *nth*(*r*, *env*)) ⟷ *A*, *env* ⊨
is_If_fm(*φ*,*t*,*f*,*r*)
  using *assms* **unfolding** *is_If_def is_If_fm_def* **by** *auto*

lemma *is_If_fm_iff_sats* [*iff_sats*]:

assumes *Qsats*: $Q \longleftrightarrow A$, $env \models \varphi$ **and**
$nth(t,\ env) = ta$ $nth(f,\ env) = fa$ $nth(r,\ env) = ra$
$t \in nat$ $f \in nat$ $r \in nat$ $env \in list(A)$
shows $is\_If(\#\#A,Q,ta,fa,ra) \longleftrightarrow A,\ env \models is\_If\_fm(\varphi,t,f,r)$
using *assms sats_is_If_fm*[*of Q A* φ *env t f r*] **by** *simp*

lemma *arity_is_If_fm* [*arity*]:
$\varphi \in formula \Longrightarrow t \in nat \Longrightarrow f \in nat \Longrightarrow r \in nat \Longrightarrow$
$arity(is\_If\_fm(\varphi,\ t,\ f,\ r)) = arity(\varphi) \cup succ(t) \cup succ(r) \cup succ(f)$
unfolding *is_If_fm_def*
by *auto*

definition
$is\_The :: [i \Rightarrow o, i \Rightarrow o, i] \Rightarrow o$ **where**
$is\_The(M,Q,i) \equiv (Q(i) \wedge (\exists\, x[M].\ Q(x) \wedge (\forall\, y[M].\ Q(y) \longrightarrow y = x))) \vee$
$(\neg(\exists\, x[M].\ Q(x) \wedge (\forall\, y[M].\ Q(y) \longrightarrow y = x))) \wedge empty(M,i)$

lemma (**in** *M_trans*) *The_abs*:
assumes $\bigwedge x.\ Q(x) \Longrightarrow M(x)\ M(a)$
shows $is\_The(M,Q,a) \longleftrightarrow a = (THE\ x.\ Q(x))$
proof (*cases* $\exists\, x[M].\ Q(x) \wedge (\forall\, y[M].\ Q(y) \longrightarrow y = x)$)
case *True*
with *assms*
show *?thesis*
unfolding *is_The_def*
by (*intro iffI the_equality*[*symmetric*])
(*auto*, *blast intro*:*theI*)
next
case *False*
with ‹$\bigwedge x.\ Q(x) \Longrightarrow M(x)$›
have $\neg\ (\exists\, x.\ Q(x) \wedge (\forall\, y.\ Q(y) \longrightarrow y = x))$
by *auto*
then
have $The(Q) = 0$
by (*intro the_0*) *auto*
with *assms* **and** *False*
show *?thesis*
unfolding *is_The_def*
by *auto*
qed

definition
$is\_recursor :: [i \Rightarrow o, i, [i,i,i] \Rightarrow o, i, i] \Rightarrow o$ **where**
$is\_recursor(M,a,is\_b,k,r) \equiv is\_transrec(M,\ \lambda n\ f\ ntc.\ is\_nat\_case(M,a,$
$\lambda m\ bmfm.$

$\exists fm[M].\ fun\_apply(M,f,m,fm) \wedge is\_b(m,fm,bmfm),n,ntc),k,r)$

lemma (**in** *M_eclose*) *recursor_abs*:
assumes *Ord*(*k*) **and**
types: *M*(*a*) *M*(*k*) *M*(*r*) **and**
b_iff: $\bigwedge m\ f\ bmf.\ M(m) \Longrightarrow M(f) \Longrightarrow M(bmf) \Longrightarrow is\_b(m,f,bmf) \longleftrightarrow bmf = b(m,f)$ **and**
b_closed: $\bigwedge m\ f\ bmf.\ M(m) \Longrightarrow M(f) \Longrightarrow M(b(m,f))$ **and**
repl: *transrec_replacement*(*M*, $\lambda n\ f\ ntc.$ *is_nat_case*(*M*, *a*,
$\lambda m\ bmfm.\ \exists fm[M].\ fun\_apply(M,\ f,\ m,\ fm) \wedge is\_b(\ m,\ fm,\ bmfm),\ n,\ ntc),\ k)$
shows
$is\_recursor(M,a,is\_b,k,r) \longleftrightarrow r = recursor(a,b,k)$
unfolding *is_recursor_def recursor_def*
using *assms*
apply (*rule_tac transrec_abs*)
apply (*auto simp*:*relation2_def*)
apply (*rule nat_case_abs*[*THEN iffD1*, **where** *is_b1*=$\lambda m\ bmfm.$
$\exists fm[M].\ fun\_apply(M,\_,m,fm) \wedge is\_b(m,fm,bmfm)$])
apply (*auto simp*:*relation1_def*)
apply (*rule nat_case_abs*[*THEN iffD2*, **where** *is_b1*=$\lambda m\ bmfm.$
$\exists fm[M].\ fun\_apply(M,\_,m,fm) \wedge is\_b(m,fm,bmfm)$])
apply (*auto simp*:*relation1_def*)
done

definition
$is\_wfrec\_on :: [i{=>}o,[i,i,i]{=>}o,i,i,i,\ i]\ {=>}\ o$ **where**
$is\_wfrec\_on(M,MH,A,r,a,z) == is\_wfrec(M,MH,r,a,z)$

lemma (**in** *M_trancl*) *trans_wfrec_on_abs*:
$[|wf(r);\ trans(r);\ relation(r);\ M(r);\ M(a);\ M(z);$
$wfrec\_replacement(M,MH,r);\ relation2(M,MH,H);$
$\forall x[M].\ \forall g[M].\ function(g) \longrightarrow M(H(x,g));$
$r\text{-}``\{a\} \subseteq A;\ a \in A|]$
$==> is\_wfrec\_on(M,MH,A,r,a,z) \longleftrightarrow z=wfrec[A](r,a,H)$
using *trans_wfrec_abs wfrec_trans_restr*
unfolding *is_wfrec_on_def* **by** *simp*

end

12 Automatic relativization of terms and formulas.

Relativization of terms and formulas. Relativization of formulas shares relativized terms as far as possible; assuming that the witnesses for the relativized terms are always unique.

theory *Relativization*
imports *ZF-Constructible.Formula*
ZF-Constructible.Relative

ZF-Constructible.Datatype_absolute
Higher_Order_Constructs
keywords
relativize :: *thy_decl* % *ML*
and
relativize_tm :: *thy_decl* % *ML*
and
reldb_add :: *thy_decl* % *ML*
and
reldb_rem :: *thy_decl* % *ML*
and
relationalize :: *thy_decl* % *ML*
and
rel_closed :: *thy_goal_stmt* % *ML*
and
is_iff_rel :: *thy_goal_stmt* % *ML*
and
univalent :: *thy_goal_stmt* % *ML*
and
absolute
and
functional
and
relational
and
external
and
for

begin

ML_file‹*Relativization_Database.ml*›

ML‹
structure Absoluteness = Named_Thms
(*val name = @{binding absolut}*
val description = Theorems of absoulte terms and predicates.)
›
setup‹*Absoluteness.setup*›

lemmas *relative_abs* =
M_trans.empty_abs
M_trans.pair_abs
M_trivial.cartprod_abs
M_trans.union_abs
M_trans.inter_abs
M_trans.setdiff_abs
M_trans.Union_abs
M_trivial.cons_abs

M_trivial.successor_abs
M_trans.Collect_abs
M_trans.Replace_abs
M_trivial.lambda_abs2
M_trans.image_abs

M_trivial.nat_case_abs

M_trivial.omega_abs
M_basic.sum_abs
M_trivial.Inl_abs
M_trivial.Inr_abs
M_basic.converse_abs
M_basic.vimage_abs
M_trans.domain_abs
M_trans.range_abs
M_basic.field_abs

M_basic.composition_abs
M_trans.restriction_abs
M_trans.Inter_abs
M_trivial.bool_of_o_abs
M_trivial.not_abs
M_trivial.and_abs
M_trivial.or_abs
M_trivial.Nil_abs
M_trivial.Cons_abs

M_trivial.list_case_abs
M_trivial.hd_abs
M_trivial.tl_abs
M_trivial.least_abs'
M_eclose.transrec_abs
M_trans.If_abs
M_trans.The_abs
M_eclose.recursor_abs
M_trancl.trans_wfrec_abs
M_trancl.trans_wfrec_on_abs

lemmas *datatype_abs* =
M_datatypes.list_N_abs
M_datatypes.list_abs
M_datatypes.formula_N_abs
M_datatypes.formula_abs
M_eclose.is_eclose_n_abs
M_eclose.eclose_abs
M_datatypes.length_abs

M_datatypes.nth_abs
M_trivial.Member_abs
M_trivial.Equal_abs
M_trivial.Nand_abs
M_trivial.Forall_abs
M_datatypes.depth_abs
M_datatypes.formula_case_abs

declare *relative_abs*[*absolut*]
declare *datatype_abs*[*absolut*]

ML‹
signature Relativization =
sig
structure Data: *GENERIC_DATA*
val Rel_add: *attribute*
val Rel_del: *attribute*
val add_rel_const : *Database.mode -> term -> term -> Data.T -> Data.T*
val add_constant : *Database.mode -> string -> string -> Proof.context -> Proof.context*
val rem_constant : (*term -> Data.T -> Data.T*) *-> string -> Proof.context -> Proof.context*
val db: *Data.T*
val init_db : *Data.T -> theory -> theory*
val get_db : *Proof.context -> Data.T*
val relativ_fm: *bool -> bool -> term -> Data.T -> (term * (term * term)) list * Proof.context * term list * bool -> term -> term * ((term * (term * term)) list * term list * term list * Proof.context)*
val relativ_tm: *bool -> bool -> term option -> term -> Data.T -> (term * (term * term)) list * Proof.context -> term -> term * (term * (term * term)) list * Proof.context*
val read_new_const : *Proof.context -> string -> term*
val relativ_tm_frm': *bool -> bool -> term -> Data.T -> Proof.context -> term -> term option * term*
val relativize_def: *bool -> bool -> bool -> bstring -> string -> Position.T -> Proof.context -> Proof.context*
val relativize_tm: *bool -> bstring -> string -> Position.T -> Proof.context -> Proof.context*
val rel_closed_goal : *string -> Position.T -> Proof.context -> Proof.state*
val iff_goal : *string -> Position.T -> Proof.context -> Proof.state*
val univalent_goal : *string -> Position.T -> Proof.context -> Proof.state*
end

structure Relativization : *Relativization = struct*

infix 6 &&&
val op &&& = Utils.&&&

*infix 6 ***

```
val op *** = Utils.***

infix 6 @@
val op @@ = Utils.@@

infix 6 ---
val op --- = Utils.---

fun insert_abs2rel ((t, u), db) = ((t, u), Database.insert Database.abs2rel (t, t) db)

fun insert_rel2is ((t, u), db) = Database.insert Database.rel2is (t, u) db

(* relativization db of relation constructors *)
val db = [ (@{const relation}, @{const Relative.is_relation})
         , (@{const function}, @{const Relative.is_function})
         , (@{const mem}, @{const mem})
         , (@{const True}, @{const True})
         , (@{const False}, @{const False})
         , (@{const Memrel}, @{const membership})
         , (@{const trancl}, @{const tran_closure})
         , (@{const IFOL.eq(i)}, @{const IFOL.eq(i)})
         , (@{const Subset}, @{const Relative.subset})
         , (@{const quasinat}, @{const Relative.is_quasinat})
         , (@{const apply}, @{const Relative.fun_apply})
         , (@{const Upair}, @{const Relative.upair})
         ]
         |> List.foldr (insert_rel2is o insert_abs2rel) Database.empty
         |> Database.insert Database.abs2is (@{const Pi}, @{const is_funspace})

fun var_i v = Free (v, @{typ i})
fun var_io v = Free (v, @{typ i => o})
val const_name = #1 o dest_Const

val lookup_tm  = AList.lookup (op aconv)
val update_tm  =  AList.update (op aconv)
val join_tm = AList.join (op aconv) (K #1)

val conj_ = Utils.binop @{const IFOL.conj}

(* generic data *)
structure Data = Generic_Data
(
  type T = Database.db
  val empty = Database.empty (* Should we initialize this outside this file? *)
  val extend = I
  val merge = Database.merge
);

fun init_db db = Context.theory_map (Data.put db)
```

```
fun get_db thy = Data.get (Context.Proof thy)

val read_const = Proof_Context.read_const {proper = true, strict = true}
val read_new_const = Proof_Context.read_term_pattern

fun add_rel_const mode c t = Database.insert mode (c, t)

fun get_consts thm =
  let val (c_rel, rhs) = Thm.concl_of thm |> Utils.dest_trueprop |>
                         Utils.dest_iff_tms |>> head_of
  in case try Utils.dest_eq_tms rhs of
       SOME tm => (c_rel, tm |> #2 |> head_of)
     | NONE => (c_rel, rhs |> Utils.dest_mem_tms |> #2 |> head_of)
  end

fun add_rule thm rs =
  let val (c_rel,c_abs) = get_consts thm
 (* in (add_rel_const Database.rel2is c_abs c_rel o add_rel_const Database.abs2rel
c_abs c_abs) rs *)
 in (add_rel_const Database.abs2rel c_abs c_abs o add_rel_const Database.abs2is
c_abs c_rel) rs
end

fun get_mode is_functional relationalising = if relationalising then Database.rel2is
else if is_functional then Database.abs2rel else Database.abs2is

fun add_constant mode abs rel thy =
  let
    val c_abs = read_new_const thy abs
    val c_rel = read_new_const thy rel
    val db_map = Data.map (Database.insert mode (c_abs, c_rel))
    fun add_to_context ctxt' = Context.proof_map db_map ctxt'
      fun add_to_theory ctxt' = Local_Theory.raw_theory (Context.theory_map
db_map) ctxt'
  in
    Local_Theory.target (add_to_theory o add_to_context) thy
 end

fun rem_constant rem_op c thy =
  let
    val c = read_new_const thy c
    val db_map = Data.map (rem_op c)
    fun add_to_context ctxt' = Context.proof_map db_map ctxt'
      fun add_to_theory ctxt' = Local_Theory.raw_theory (Context.theory_map
db_map) ctxt'
  in
    Local_Theory.target (add_to_theory o add_to_context) thy
  end
```

```
val del_rel_const = Database.remove_abs

fun del_rule thm = del_rel_const (thm |> get_consts |> #2)

val Rel_add =
  Thm.declaration_attribute (fn thm => fn context =>
    Data.map (add_rule (Thm.trim_context thm)) context);

val Rel_del =
  Thm.declaration_attribute (fn thm => fn context =>
    Data.map (del_rule (Thm.trim_context thm)) context);

(* Conjunction of a list of terms *)
fun conjs [] = @{term IFOL.True}
  | conjs (fs as _ :: _) = foldr1 (uncurry conj_) fs

(* Produces a relativized existential quantification of the term t *)
fun rex p t (Free v) = @{const rex} $ p $ lambda (Free v) t
  | rex _ t (Bound _) = t
  | rex _ t tm = raise TERM (rex shouldn't handle this.,[tm,t])

(* Constants that do not take the class predicate *)
val absolute_rels = [ @{const ZF_Base.mem}
                    , @{const IFOL.eq(i)}
                    , @{const Memrel}
                    , @{const True}
                    , @{const False}
                    ]

(* Creates the relational term corresponding to a term of type i. If the last
  argument is (SOME v) then that variable is not bound by an existential
  quantifier.
*)
fun close_rel_tm pred tm tm_var rs =
  let val news = filter (not o (fn x => is_Free x orelse is_Bound x) o #1) rs
      val (vars, tms) = split_list (map #2 news) |> (curry op @) (the_list tm)
      val vars = case tm_var of
        SOME w => filter (fn v => not (v = w)) vars
      | NONE => vars
  in fold (fn v => fn t => rex pred (incr_boundvars 1 t) v) vars (conjs tms)
  end

fun relativ_tms ___ _ _ rs ctxt [] = ([], rs, ctxt)
  | relativ_tms is_functional relationalising pred rel_db rs ctxt (u :: us) =
      let val (w_u, rs_u, ctxt_u) = relativ_tm is_functional relationalising NONE
pred rel_db (rs, ctxt) u
          val (w_us, rs_us, ctxt_us) = relativ_tms is_functional relationalising pred
rel_db rs_u ctxt_u us
```

```
    in (w_u :: w_us, join_tm (rs_u , rs_us), ctxt_us)
    end
and
  (* The result of the relativization of a term is a triple consisting of
    a. the relativized term (it can be a free or a bound variable but also a Collect)
    b. a list of (term * (term, term)), taken as a map, which is used
       to reuse relativization of different occurrences of the same term. The
       first element is the original term, the second its relativized version,
       and the last one is the predicate corresponding to it.
    c. the resulting context of created variables.
  *)
  relativ_tm is_functional relationalising mv pred rel_db (rs,ctxt) tm =
    let
    (* relativization of a fully applied constant *)
    fun mk_rel_const mv c (args, after) abs_args ctxt =
      case Database.lookup (get_mode is_functional relationalising) c rel_db of
        SOME p =>
          let
            val args' = List.filter (not o Utils.inList (Utils.frees p)) args
            val (v, ctxt1) =
              the_default
                (Variable.variant_fixes [] ctxt |>> var_i o hd)
                (Utils.map_option (I &&& K ctxt) mv)
            val args' =
                (* FIXME: This special case for functional relativization of sigma
should not be needed *)
              if c = @{const Sigma} andalso is_functional
                then
                  let
                    val t = hd args'
                    val t' = Abs (uu_, @{typ i}, (hd o tl) args' |> incr_boundvars 1)
                  in
                    [t, t']
                  end
                else
                  args'
            val arg_list = if after then abs_args @ args' else args' @ abs_args
            val r_tm =
              if is_functional
                then list_comb (p, if p = c then arg_list else pred :: arg_list)
                else list_comb (p, if (not o null) args' andalso  hd args' = pred then
arg_list @ [v] else pred :: arg_list @ [v])
          in
            if is_functional
              then (r_tm, r_tm, ctxt)
              else (v, r_tm, ctxt1)
          end
      | NONE => raise TERM (Constant  ^ const_name c ^  is not present in
the db. , nil)
```

```
(* relativization of a partially applied constant *)
fun relativ_app mv mctxt tm abs_args (Const c) (args, after) rs =
      let
        val (w_ts, rs_ts, ctxt_ts) = relativ_tms is_functional relationalising
pred rel_db rs (the_default ctxt mctxt) args
        val (w_tm, r_tm, ctxt_tm) = mk_rel_const mv (Const c) (w_ts, after)
abs_args ctxt_ts
          val rs_ts' = if is_functional then rs_ts else update_tm (tm, (w_tm,
r_tm)) rs_ts
      in
        (w_tm, rs_ts', ctxt_tm)
      end
  | relativ_app _ _ _ _ t _ _ =
       raise TERM (Tried to relativize an application with a non-constant in
head position,[t])

(* relativization of non dependent product and sum *)
fun relativ_app_no_dep mv tm c t t' rs =
    if loose_bvar1 (t', 0)
      then
        raise TERM(A dependency was found when trying to relativize, [tm])
      else
        relativ_app mv NONE tm [] c ([t, incr_boundvars ~1 t'], false) rs

fun relativ_replace mv t body after ctxt' =
  let
    val (v, b) = Term.dest_abs body |>> var_i ||> after
    val (b', (rs', ctxt'')) =
      relativ_fm is_functional relationalising pred rel_db (rs, ctxt', single v,
false) b |>> incr_boundvars 1 ||> #1 &&& #4
  in
   relativ_app mv (SOME ctxt'') tm [lambda v b'] @{const Replace} ([t], false)
rs'
  end

fun get_abs_body (Abs body) = body
  | get_abs_body t = raise TERM (Term is not Abs, [t])

fun go _ (Var _) = raise TERM (Var: Is this possible?,[])
 | go mv (@{const Replace} $ t $ Abs body) = relativ_replace mv t body I ctxt
  (* It is easier to rewrite RepFun as Replace before relativizing,
     since { f(x) . x ∈ t } = { y . x ∈ t, y = f(x) } *)
  | go mv (@{const RepFun} $ t $ Abs body) =
      let
        val (y, ctxt') = Variable.variant_fixes [] ctxt |>> var_i o hd
      in
        relativ_replace mv t body (lambda y o Utils.eq_ y o incr_boundvars 1)
ctxt'
      end
```

```
    | go mv (@{const Collect} $ t $ pc) =
      let
          val (pc', (rs', ctxt')) = relativ_fm is_functional relationalising pred
rel_db (rs,ctxt, [], false) pc ||> #1 &&& #4
      in
        relativ_app mv (SOME ctxt') tm [pc'] @{const Collect} ([t], false) rs'
      end
    | go mv (@{const Least} $ pc) =
      let
          val (pc', (rs', ctxt')) = relativ_fm is_functional relationalising pred
rel_db (rs,ctxt, [], false) pc ||> #1 &&& #4
      in
        relativ_app mv (SOME ctxt') tm [pc'] @{const Least} ([], false) rs'
      end
    | go mv (@{const transrec} $ t $ Abs body) =
      let
       val (res, ctxt') = Variable.variant_fixes [if is_functional then _aux else
] ctxt |>> var_i o hd
        val (x, b') = Term.dest_abs body |>> var_i
        val (y, b) = get_abs_body b' |> Term.dest_abs |>> var_i
        val p = Utils.eq_ res b |> lambda res
          val (p', (rs', ctxt'')) = relativ_fm is_functional relationalising pred
rel_db (rs, ctxt', [x, y], true) p |>> incr_boundvars 3 ||> #1 &&& #4
         val p' = if is_functional then p' |> #2 o Utils.dest_eq_tms o #2 o
Term.dest_abs o get_abs_body else p'
      in
        relativ_app mv (SOME ctxt'') tm [p' |> lambda x o lambda y] @{const
transrec} ([t], not is_functional) rs'
      end
    | go mv (tm as @{const Sigma} $ t $ Abs (_, _, t')) =
      relativ_app_no_dep mv tm @{const Sigma} t t' rs
    | go mv (tm as @{const Pi} $ t $ Abs (_, _, t')) =
      relativ_app_no_dep mv tm @{const Pi} t t' rs
    | go mv (tm as @{const bool_of_o} $ t) =
      let
      val (t', (rs', ctxt')) = relativ_fm is_functional relationalising pred rel_db
(rs, ctxt, [], false) t ||> #1 &&& #4
      in
        relativ_app mv (SOME ctxt') tm [t'] @{const bool_of_o} ([], false) rs'
      end
    | go mv (tm as @{const If} $ b $ t $ t') =
      let
          val (br, (rs', ctxt')) = relativ_fm is_functional relationalising pred
rel_db (rs, ctxt, [], false) b ||> #1 &&& #4
      in
        relativ_app mv (SOME ctxt') tm [br] @{const If} ([t,t'], true) rs'
      end
    | go mv (@{const The} $ pc) =
      let
```

```
            val (pc', (rs', ctxt')) = relativ_fm is_functional relationalising pred
rel_db (rs,ctxt, [], false) pc ||> #1 &&& #4
          in
            relativ_app mv (SOME ctxt') tm [pc'] @{const The} ([], false) rs'
          end
      | go mv (@{const recursor} $ t $ Abs body $ t') =
          let
           val (res, ctxt') = Variable.variant_fixes [if is_functional then _aux else
] ctxt |>> var_i o hd
            val (x, b') = Term.dest_abs body |>> var_i
            val (y, b) = get_abs_body b' |> Term.dest_abs |>> var_i
            val p = Utils.eq_ res b |> lambda res
              val (p', (rs', ctxt'')) = relativ_fm is_functional relationalising pred
rel_db (rs, ctxt', [x, y], true) p |>> incr_boundvars 3 ||> #1 &&& #4
             val p' = if is_functional then p' |> #2 o Utils.dest_eq_tms o #2 o
Term.dest_abs o get_abs_body else p'
             val (tr, rs'', ctxt''') = relativ_tm is_functional relationalising NONE
pred rel_db (rs', ctxt'') t
          in
              relativ_app mv (SOME ctxt''') tm [tr, p' |> lambda x o lambda y]
@{const recursor} ([t'], true) rs''
          end
      | go mv (@{const wfrec} $ t1 $ t2 $ Abs body) =
          let
           val (res, ctxt') = Variable.variant_fixes [if is_functional then _aux else
] ctxt |>> var_i o hd
            val (x, b') = Term.dest_abs body |>> var_i
            val (y, b) = get_abs_body b' |> Term.dest_abs |>> var_i
            val p = Utils.eq_ res b |> lambda res
              val (p', (rs', ctxt'')) = relativ_fm is_functional relationalising pred
rel_db (rs, ctxt', [x, y], true) p |>> incr_boundvars 3 ||> #1 &&& #4
             val p' = if is_functional then p' |> #2 o Utils.dest_eq_tms o #2 o
Term.dest_abs o get_abs_body else p'
          in
            relativ_app mv (SOME ctxt'') tm [p' |> lambda x o lambda y] @{const
wfrec} ([t1,t2], not is_functional) rs'
          end
      | go mv (@{const wfrec_on} $ t1 $ t2 $ t3 $ Abs body) =
          let
           val (res, ctxt') = Variable.variant_fixes [if is_functional then _aux else
] ctxt |>> var_i o hd
            val (x, b') = Term.dest_abs body |>> var_i
            val (y, b) = get_abs_body b' |> Term.dest_abs |>> var_i
            val p = Utils.eq_ res b |> lambda res
              val (p', (rs', ctxt'')) = relativ_fm is_functional relationalising pred
rel_db (rs, ctxt', [x, y], true) p |>> incr_boundvars 3 ||> #1 &&& #4
             val p' = if is_functional then p' |> #2 o Utils.dest_eq_tms o #2 o
Term.dest_abs o get_abs_body else p'
          in
```

```
            relativ_app mv (SOME ctxt'') tm [p' |> lambda x o lambda y] @{const
wfrec_on} ([t1,t2,t3], not is_functional) rs'
          end
      | go mv (@{const Lambda} $ t $ Abs body) =
          let
           val (res, ctxt') = Variable.variant_fixes [if is_functional then _aux else
] ctxt |>> var_i o hd
           val (x, b) = Term.dest_abs body |>> var_i
           val p = Utils.eq_ res b |> lambda res
             val (p', (rs', ctxt'')) = relativ_fm is_functional relationalising pred
rel_db (rs, ctxt', [x], true) p |>> incr_boundvars 2 ||> #1 &&& #4
            val p' = if is_functional then p' |> #2 o Utils.dest_eq_tms o #2 o
Term.dest_abs o get_abs_body else p'
           val (tr, rs'', ctxt''') = relativ_tm is_functional relationalising NONE
pred rel_db (rs', ctxt'') t
          in
           relativ_app mv (SOME ctxt''') tm [tr, p' |> lambda x] @{const Lambda}
([], true) rs''
          end
       (* The following are the generic cases *)
       | go mv (tm as Const _) = relativ_app mv NONE tm [] tm ([], false) rs
        | go mv (tm as _ $ _) = (strip_comb tm ||> I &&& K false |> uncurry
(relativ_app mv NONE tm [])) rs
         | go _ tm = if is_functional then (tm, rs, ctxt) else (tm, update_tm
(tm,(tm,tm)) rs, ctxt)

     (* we first check if the term has been already relativized as a variable *)
     in case lookup_tm rs tm of
          NONE => go mv tm
       | SOME (w, _) => (w, rs, ctxt)
     end
and
  relativ_fm is_functional relationalising pred rel_db (rs, ctxt, vs, is_term) fm =
  let

  (* relativization of a fully applied constant *)
  fun relativ_app (ctxt, rs) c args = case Database.lookup (get_mode is_functional
relationalising) c rel_db of
    SOME p =>
      let (* flag indicates whether the relativized constant is absolute or not. *)
        val flag = not (exists (curry op aconv c) absolute_rels orelse c = p)
       val (args, rs_ts, ctxt') = relativ_tms is_functional relationalising pred rel_db
rs ctxt args
        (* TODO: Verify if next line takes care of locales' definitions *)
        val args' = List.filter (not o Utils.inList (Utils.frees p)) args
       val args'' = if not (null args') andalso hd args' = pred then args' else pred ::
args'
        val tm = list_comb (p, if flag then args'' else args')
        (* TODO: Verify if next line is necessary *)
```

```
      val news = filter (not o (fn x => is_Free x orelse is_Bound x) o #1) rs_ts
      val (vars, tms) = split_list (map #2 news)
      (* val vars = filter (fn v => not (v = tm)) vars *) (* Verify if this line is
necessary *)
    in (tm, (rs_ts, vars, tms, ctxt'))
    end
  | NONE   => raise TERM (Constant ^const_name c ^ is not present in the
db. , nil)

 fun close_fm quantifier (f, (rs, vars, tms, ctxt)) =
   let
     fun contains_b0 t = loose_bvar1 (t, 0)

     fun contains_extra_var t = fold (fn v => fn acc => acc orelse fold_aterms
(fn t => fn acc => t = v orelse acc) t false) vs false

     fun contains_b0_extra t = contains_b0 t orelse contains_extra_var t

     (* t1 $ v ↪ t2 iff v ∈ FV(t2) *)
     fun chained_frees (_ $ v) t2 = Utils.inList (Utils.frees t2) v
       | chained_frees t _ = raise TERM (Malformed term, [t])

   val tms_to_close = filter contains_b0_extra tms |> Utils.reachable chained_frees
tms
     val tms_to_keep = map (incr_boundvars ~1) (tms --- tms_to_close)
       val vars_to_close = inter (op =) (map (List.last o #2 o strip_comb)
tms_to_close) vars
     val vars_to_keep = vars --- vars_to_close
     val new_rs =
       rs
    |> filter (fn (k, (v, rel)) => not (contains_b0_extra k orelse contains_b0_extra
v orelse contains_b0_extra rel))
       |> map (fn (k, (v, rel)) => (incr_boundvars ~1 k, (incr_boundvars ~1 v,
incr_boundvars ~1 rel)))

     val f' =
       if not is_term andalso not quantifier andalso is_functional
         then pred $ Bound 0 :: (map (curry (op $) pred) vs) @ [f]
         else [f]
   in
    (fold (fn v => fn t => rex pred (incr_boundvars 1 t) v) vars_to_close (conjs
(f' @ tms_to_close)),
      (new_rs, vars_to_keep, tms_to_keep, ctxt))
   end

 (* Handling of bounded quantifiers. *)
 fun bquant (ctxt, rs) quant conn dom pred =
   let val (v,pred') = Term.dest_abs pred |>> var_i
   in
```

```
      go (ctxt, rs, false) (quant $ (lambda v o incr_boundvars 1) (conn $ (@{const
mem} $ v $ dom) $ pred'))
    end
  and
  bind_go (ctxt, rs) const f f' =
    let
      val (r , (rs1, vars1, tms1, ctxt1)) = go (ctxt, rs, false) f
      val (r', (rs2, vars2, tms2, ctxt2)) = go (ctxt1, rs1, false) f'
    in
      (const $ r $ r', (rs2, vars1 @@ vars2, tms1 @@ tms2, ctxt2))
    end
  and
      relativ_eq_var (ctxt, rs) v t =
        let
            val (_, rs', ctxt') = relativ_tm is_functional relationalising (SOME v)
pred rel_db (rs, ctxt) t
          val f = lookup_tm rs' t |> #2 o the
          val rs'' = filter (not o (curry (op =) t) o #1) rs'
          val news = filter (not o (fn x => is_Free x orelse is_Bound x) o #1) rs''
          val (vars, tms) = split_list (map #2 news)
        in
          (f, (rs'', vars, tms, ctxt'))
        end
  and
      relativ_eq (ctxt, rs) t1 t2 =
        if is_functional orelse ((is_Free t1 orelse is_Bound t1) andalso (is_Free t2
orelse is_Bound t2)) then
          relativ_app (ctxt, rs) @{const IFOL.eq(i)} [t1, t2]
        else if is_Free t1 orelse is_Bound t1 then
          relativ_eq_var (ctxt, rs) t1 t2
        else if is_Free t2 orelse is_Bound t2 then
          relativ_eq_var (ctxt, rs) t2 t1
        else
          relativ_app (ctxt, rs) @{const IFOL.eq(i)} [t1, t2]
  and
       go (ctxt, rs, _          ) (@{const IFOL.conj} $ f $ f') = bind_go (ctxt, rs)
@{const IFOL.conj} f f'
     | go (ctxt, rs, _          ) (@{const IFOL.disj} $ f $ f') = bind_go (ctxt, rs)
@{const IFOL.disj} f f'
     | go (ctxt, rs, _          ) (@{const IFOL.Not} $ f) = go (ctxt, rs, false) f |>>
((curry op $) @{const IFOL.Not})
     | go (ctxt, rs, _          ) (@{const IFOL.iff} $ f $ f') = bind_go (ctxt, rs)
@{const IFOL.iff} f f'
     | go (ctxt, rs, _          ) (@{const IFOL.imp} $ f $ f') = bind_go (ctxt, rs)
@{const IFOL.imp} f f'
     | go (ctxt, rs, _          ) (@{const IFOL.All(i)} $ f) = go (ctxt, rs, true) f |>>
((curry op $) (@{const OrdQuant.rall} $ pred))
     | go (ctxt, rs, _          ) (@{const IFOL.Ex(i)} $ f) = go (ctxt, rs, true) f |>>
((curry op $) (@{const OrdQuant.rex} $ pred))
```

```
    | go (ctxt, rs, _         ) (@{const Bex} $ f $ Abs p) = bquant (ctxt, rs) @{const
Ex(i)} @{const IFOL.conj} f p
    | go (ctxt, rs, _         ) (@{const Ball} $ f $ Abs p) = bquant (ctxt, rs) @{const
All(i)} @{const IFOL.imp} f p
     | go (ctxt, rs, _          ) (@{const rall} $ _ $ p) = go (ctxt, rs, true) p |>>
(curry op $) (@{const rall} $ pred)
     | go (ctxt, rs, _          ) (@{const rex} $ _ $ p) = go (ctxt, rs, true) p |>>
(curry op $) (@{const rex} $ pred)
    | go (ctxt, rs, _         ) (@{const IFOL.eq(i)} $ t1 $ t2) = relativ_eq (ctxt, rs)
t1 t2
     | go (ctxt, rs, _          ) (Const c) = relativ_app (ctxt, rs) (Const c) []
   | go (ctxt, rs, _        ) (tm as _ $ _) = strip_comb tm |> uncurry (relativ_app
(ctxt, rs))
    | go (ctxt, rs, quantifier) (Abs (v, _, t)) =
      let
       val new_rs = map (fn (k, (v, rel)) => (incr_boundvars 1 k, (incr_boundvars
1 v, incr_boundvars 1 rel))) rs
      in
        go (ctxt, new_rs, false) t |> close_fm quantifier |>> lambda (var_i v)
      end
   | go _ t = raise TERM (Relativization of formulas cannot handle this case.,[t])
  in
    go (ctxt, rs, false) fm
  end


fun relativ_tm_frm' is_functional relationalising cls_pred db ctxt tm =
  let
     fun get_bounds (l as Abs _) = op @@ (strip_abs l |>> map (op #1) ||>
get_bounds)
      | get_bounds (t as _$_) = strip_comb t |> op :: |> map get_bounds |> flat
      | get_bounds _ = []

    val ty = fastype_of tm
    val initial_ctxt = fold Utils.add_to_context (get_bounds tm) ctxt
  in
    case ty of
        @{typ i} =>
          let
           val (w, rs, _) =  relativ_tm is_functional relationalising NONE cls_pred
db ([], initial_ctxt) tm
          in
            if is_functional
              then (NONE, w)
              else (SOME w, close_rel_tm cls_pred NONE (SOME w) rs)
          end
      | @{typ o} =>
          let
            fun close_fm (f, (_, vars, tms, _)) =
```

```
          fold (fn  v => fn t => rex cls_pred (incr_boundvars 1 t) v) vars (conjs
(f :: tms))
         in
        (NONE, relativ_fm is_functional relationalising cls_pred db ([], initial_ctxt,
[], false) tm |> close_fm)
         end
    | ty' => raise TYPE (We can relativize only terms of types i and o, [ty'], [tm])
  end

fun lname ctxt = Local_Theory.full_name ctxt o Binding.name

fun destroy_first_lambdas (Abs (body as (_, ty, _))) =
    Term.dest_abs body ||> destroy_first_lambdas |> (#1 o #2) &&& ((fn v =>
Free (v, ty)) *** #2) ||> op ::
  | destroy_first_lambdas t = (t, [])

fun freeType (Free (_, ty)) = ty
  | freeType t = raise TERM (freeType, [t])

fun relativize_def is_external is_functional relationalising def_name thm_ref pos
lthy =
  let
    val ctxt = lthy
    val (vars,tm,ctxt1) = Utils.thm_concl_tm ctxt (thm_ref ^ _def)
    val db' = Data.get (Context.Proof lthy)
   val (tm, lambdavars) = tm |> destroy_first_lambdas o #2 o Utils.dest_eq_tms'
o Utils.dest_trueprop
    val ctxt1 = fold Utils.add_to_context (map Utils.freeName lambdavars) ctxt1
    val (cls_pred, ctxt1, vars, lambdavars) =
      if (not o null) vars andalso (#2 o #1 o hd) vars = @{typ i ⇒ o} then
        ((Thm.term_of o #2 o hd) vars, ctxt1, tl vars, lambdavars)
       else if null vars andalso (not o null) lambdavars andalso (freeType o hd)
lambdavars = @{typ i ⇒ o} then
        (hd lambdavars, ctxt1, vars, tl lambdavars)
      else Variable.variant_fixes [N] ctxt1 |>> var_io o hd |> (fn (cls, ctxt) =>
(cls, ctxt, vars, lambdavars))
    val db' = db' |> Database.insert Database.abs2rel (cls_pred, cls_pred)
                  o Database.insert Database.rel2is (cls_pred, cls_pred)
    val (v,t) = relativ_tm_frm' is_functional relationalising cls_pred db' ctxt1 tm
    val t_vars = Term.add_free_names tm []
    val vs' = List.filter (#1 #> #1 #> #1 #> Utils.inList t_vars) vars
    val vs = cls_pred :: map (Thm.term_of o #2) vs' @ lambdavars @ the_list v
    val at = List.foldr (uncurry lambda) t vs
     val abs_const = read_const lthy (if is_external then thm_ref else lname lthy
thm_ref)
    fun new_const ctxt' = read_new_const ctxt' def_name
    fun db_map ctxt' =
      Data.map (add_rel_const (get_mode is_functional relationalising) abs_const
(new_const ctxt'))
```

```
    fun add_to_context ctxt' = Context.proof_map (db_map ctxt') ctxt'
     fun add_to_theory ctxt' = Local_Theory.raw_theory (Context.theory_map
(db_map ctxt')) ctxt'
  in
    lthy
     |> Local_Theory.define ((Binding.name def_name, NoSyn), ((Binding.name
(def_name ^ "_def"), []), at))
    |>> (#2 #> (fn (s,t) => (s,[t])))
    |> Utils.display theorem pos
    |> Local_Theory.target (add_to_theory o add_to_context)
  end

fun relativize_tm is_functional def_name term pos lthy =
  let
    val ctxt = lthy
    val (cls_pred, ctxt1) = Variable.variant_fixes ["N"] ctxt |>> var_io o hd
    val tm = Syntax.read_term ctxt1 term
    val db' = Data.get (Context.Proof lthy)
    val db' = db' |> Database.insert Database.abs2rel (cls_pred, cls_pred)
                  o Database.insert Database.rel2is (cls_pred, cls_pred)
    val vs' = Variable.add_frees ctxt1 tm []
    val ctxt2 = fold Utils.add_to_context (map #1 vs') ctxt1
    val (v,t) = relativ_tm_frm' is_functional false cls_pred db' ctxt2 tm
    val vs = cls_pred :: map Free vs' @ the_list v
    val at = List.foldr (uncurry lambda) t vs
  in
    lthy
     |> Local_Theory.define ((Binding.name def_name, NoSyn), ((Binding.name
(def_name ^ "_def"), []), at))
    |>> (#2 #> (fn (s,t) => (s,[t])))
    |> Utils.display theorem pos
  end

val op $` = curry ((op $) o swap)
infix $`

fun is_free_i (Free (_, @{typ i})) = true
  | is_free_i _ = false

fun rel_closed_goal target pos lthy =
  let
    val (_, tm, _) = Utils.thm_concl_tm lthy (target ^ "_rel_def")
    val (def, tm) = tm |> Utils.dest_eq_tms'
    fun first_lambdas (Abs (body as (_, ty, _))) =
        if ty = @{typ i}
          then (op ::) (Term.dest_abs body |>> Utils.var_i ||> first_lambdas)
          else Term.dest_abs body |> first_lambdas o #2
      | first_lambdas _ = []
    val (def, vars) = Term.strip_comb def ||> filter is_free_i
```

```
    val vs = vars @ first_lambdas tm
    val class = Free (M, @{typ i ⇒ o})
    val def = fold (op $`) (class :: vs) def
    val hyps = map (fn v => class $ v |> Utils.tp) vs
    val concl = class $ def
    val goal = Logic.list_implies (hyps, Utils.tp concl)
    val attribs = @{attributes [intro, simp]}
  in
    Proof.theorem NONE (fn thmss => Utils.display theorem pos
                                    o Local_Theory.note ((Binding.name (target ^
_rel_closed), attribs), hd thmss))
    [[(goal, [])]] lthy
  end

fun iff_goal target pos lthy =
  let
    val (_, tm, ctxt') = Utils.thm_concl_tm lthy (target ^ _rel_def)
    val (_, is_def, ctxt) = Utils.thm_concl_tm ctxt' (is_ ^ target ^ _def)
    val is_def = is_def |> Utils.dest_eq_tms' |> #1 |> Term.strip_comb |> #1
    val (def, tm) = tm |> Utils.dest_eq_tms'
    fun first_lambdas (Abs (body as (_, ty, _))) =
        if ty = @{typ i}
          then (op ::) (Term.dest_abs body |>> Utils.var_i ||> first_lambdas)
          else Term.dest_abs body |> first_lambdas o #2
      | first_lambdas _ = []
    val (def, vars) = Term.strip_comb def ||> filter is_free_i
    val vs = vars @ first_lambdas tm
    val class = Free (M, @{typ i ⇒ o})
    val def = fold (op $`) (class :: vs) def
    val ty = fastype_of def
    val res = if ty = @{typ i}
              then Variable.variant_fixes [res] ctxt |> SOME o Utils.var_i o hd o
#1
              else NONE
    val is_def = fold (op $`) (class :: vs @ the_list res) is_def
    val hyps = map (fn v => class $ v |> Utils.tp) (vs @ the_list res)
    val concl = @{const IFOL.iff} $ is_def
          $ (if ty = @{typ i} then (@{const IFOL.eq(i)} $ the res $ def) else def)
    val goal = Logic.list_implies (hyps, Utils.tp concl)
  in
    Proof.theorem NONE (fn thmss => Utils.display theorem pos
                                  o Local_Theory.note ((Binding.name (is_ ^ target ^
_iff), []), hd thmss))
    [[(goal, [])]] lthy
  end

fun univalent_goal target pos lthy =
  let
    val (_, tm, ctxt) = Utils.thm_concl_tm lthy (is_ ^ target ^ _def)
```

```
    val (def, tm) = tm |> Utils.dest_eq_tms'
    fun first_lambdas (Abs (body as (_, ty, _))) =
        if ty = @{typ i}
          then (op ::) (Term.dest_abs body |>> Utils.var_i ||> first_lambdas)
          else Term.dest_abs body |> first_lambdas o #2
      | first_lambdas _ = []
    val (def, vars) = Term.strip_comb def ||> filter is_free_i
    val vs = vars @ first_lambdas tm
    val n = length vs
    val vs = List.take (vs, n - 2)
    val class = Free (M, @{typ i ⇒ o})
    val def = fold (op $ `) (class :: vs) def
    val v = Variable.variant_fixes [A] ctxt |> Utils.var_i o hd o #1
    val hyps = map (fn v => class $ v |> Utils.tp) (v :: vs)
    val concl = @{const Relative.univalent} $ class $ v $ def
    val goal = Logic.list_implies (hyps, Utils.tp concl)
  in
    Proof.theorem NONE (fn thmss => Utils.display theorem pos
                                  o Local_Theory.note ((Binding.name (univalent_is_
^ target), []), hd thmss))
    [[(goal, [])]] lthy
  end

end
⟩

ML⟨
local
  val full_mode_parser =
     Scan.option (((Parse.$$$ functional |-- Parse.$$$ relational) >> K Database.rel2is)
                  || (((Scan.option (Parse.$$$ absolute)) |-- Parse.$$$ functional)
>> K Database.abs2rel)
                || (((Scan.option (Parse.$$$ absolute)) |-- Parse.$$$ relational) >>
K Database.abs2is))
       >> (fn mode => the_default Database.abs2is mode)

  val reldb_parser =
       Parse.position (full_mode_parser -- (Parse.string -- Parse.string));

  val singlemode_parser = (Parse.$$$ absolute >> K Database.remove_abs)
                    || (Parse.$$$ functional >> K Database.remove_rel)
                    || (Parse.$$$ relational >> K Database.remove_is)

  val reldb_rem_parser = Parse.position (singlemode_parser -- Parse.string)

  val mode_parser =
       Scan.option ((Parse.$$$ relational >> K false) || (Parse.$$$ functional >>
K true))
       >> (fn mode => if is_none mode then false else the mode)
```

```
  val relativize_parser =
    Parse.position (mode_parser -- (Parse.string -- Parse.string) -- (Scan.optional
(Parse.$$$ external >> K true) false));

  val _ =
     Outer_Syntax.local_theory command_keyword‹reldb_add› ML setup for
adding relativized/absolute pairs
      (reldb_parser >> (fn ((mode, (abs_term,rel_term)),_) =>
        Relativization.add_constant mode abs_term rel_term))

  val _ =
  Outer_Syntax.local_theory command_keyword‹reldb_rem› ML setup for adding
relativized/absolute pairs
    (reldb_rem_parser >> (uncurry Relativization.rem_constant o #1))

  val _ =
     Outer_Syntax.local_theory command_keyword‹relativize› ML setup for
relativizing definitions
      (relativize_parser >> (fn (((is_functional, (bndg,thm)), is_external),pos)
=>
       Relativization.relativize_def is_external is_functional false thm bndg pos))

  val _ =
   Outer_Syntax.local_theory command_keyword‹relativize_tm› ML setup for
relativizing definitions
      (relativize_parser >> (fn (((is_functional, (bndg,term)), _),pos) =>
        Relativization.relativize_tm is_functional term bndg pos))

  val _ =
    Outer_Syntax.local_theory command_keyword‹relationalize› ML setup for
relativizing definitions
      (relativize_parser >> (fn (((is_functional, (bndg,thm)), is_external),pos)
=>
       Relativization.relativize_def is_external is_functional true thm bndg pos))

  val _ =
     Outer_Syntax.local_theory_to_proof command_keyword‹rel_closed› ML
setup for rel_closed theorem
     (Parse.position (Parse.$$$ for |-- Parse.string) >> (fn (target,pos) =>
       Relativization.rel_closed_goal target pos))

  val _ =
     Outer_Syntax.local_theory_to_proof command_keyword‹is_iff_rel› ML
setup for rel_closed theorem
     (Parse.position (Parse.$$$ for |-- Parse.string) >> (fn (target,pos) =>
       Relativization.iff_goal target pos))

  val _ =
```

Outer_Syntax.local_theory_to_proof ***command_keyword***‹*univalent*› *ML setup*
for rel_closed theorem
(*Parse.position* (*Parse.$$$ for* |-- *Parse.string*) >> (*fn* (*target*,*pos*) =>
Relativization.univalent_goal target pos))

val _ =
Theory.setup
(*Attrib.setup* ***binding***‹*Rel*› (*Attrib.add_del Relativization.Rel_add Relativization.Rel_del*)
declaration of relativization rule) ;
in
end
›
setup‹*Relativization.init_db Relativization.db* ›

declare *relative_abs*[*Rel*]

declare *datatype_abs*[*Rel*]

ML‹
val db = Relativization.get_db @{*context*}
›

end
theory *Discipline_Base*
imports
ZF-Constructible.Rank
ZF_Miscellanea
Relativization

begin

declare [[*syntax_ambiguity_warning = false*]]

Discipline of Relativization of basic concepts.

definition
is_singleton :: [*i*⇒*o*,*i*,*i*] ⇒ *o* **where**
is_singleton(*A*,*x*,*z*) ≡ ∃ *c*[*A*]. *empty*(*A*,*c*) ∧ *is_cons*(*A*,*x*,*c*,*z*)

lemma (**in** *M_trivial*) *singleton_abs*[*simp*] :
⟦ *M*(*x*) ; *M*(*s*) ⟧ ⟹ *is_singleton*(*M*,*x*,*s*) ⟷ *s* = {*x*}
unfolding *is_singleton_def* **using** *nonempty* **by** *simp*

synthesize *singleton* **from_definition** *is_singleton*

lemma (**in** *M_trivial*) *singleton_closed* [*simp*]:
M(*x*) ⟹ *M*({*x*})
by *simp*

lemma (**in** *M_trivial*) *Upair_closed*[*simp*]: *M*(*a*) ⟹ *M*(*b*) ⟹ *M*(*Upair*(*a*,*b*))
using *Upair_eq_cons* **by** *simp*

lemma (**in** *M_trivial*) *upair_closed*[*simp*] : *M*(*x*) ⟹ *M*(*y*) ⟹ *M*({*x*,*y*})
by *simp*

The following named theorems gather instances of transitivity that arise from closure theorems

named_theorems *trans_closed*

definition
is_hcomp :: [*i*⇒*o*,*i*⇒*i*⇒*o*,*i*⇒*i*⇒*o*,*i*,*i*] ⇒ *o* **where**
is_hcomp(*M*,*is_f*,*is_g*,*a*,*w*) ≡ ∃ *z*[*M*]. *is_g*(*a*,*z*) ∧ *is_f*(*z*,*w*)

lemma (**in** *M_trivial*) *is_hcomp_abs*:
assumes
is_f_abs:⋀*a z*. *M*(*a*) ⟹ *M*(*z*) ⟹ *is_f*(*a*,*z*) ⟷ *z* = *f*(*a*) **and**
is_g_abs:⋀*a z*. *M*(*a*) ⟹ *M*(*z*) ⟹ *is_g*(*a*,*z*) ⟷ *z* = *g*(*a*) **and**
g_closed:⋀*a*. *M*(*a*) ⟹ *M*(*g*(*a*))
M(*a*) *M*(*w*)
shows
is_hcomp(*M*,*is_f*,*is_g*,*a*,*w*) ⟷ *w* = *f*(*g*(*a*))
unfolding *is_hcomp_def* **using** *assms* **by** *simp*

definition
hcomp_fm :: [*i*⇒*i*⇒*i*,*i*⇒*i*⇒*i*,*i*,*i*] ⇒ *i* **where**
hcomp_fm(*pf*,*pg*,*a*,*w*) ≡ *Exists*(*And*(*pg*(*succ*(*a*),0),*pf*(0,*succ*(*w*))))

lemma *sats_hcomp_fm*:
assumes
f_iff_sats:⋀*a b z*. *a*∈*nat* ⟹ *b*∈*nat* ⟹ *z*∈*M* ⟹
is_f(*nth*(*a*,*Cons*(*z*,*env*)),*nth*(*b*,*Cons*(*z*,*env*))) ⟷ *sats*(*M*,*pf*(*a*,*b*),*Cons*(*z*,*env*))
and
g_iff_sats:⋀*a b z*. *a*∈*nat* ⟹ *b*∈*nat* ⟹ *z*∈*M* ⟹
is_g(*nth*(*a*,*Cons*(*z*,*env*)),*nth*(*b*,*Cons*(*z*,*env*))) ⟷ *sats*(*M*,*pg*(*a*,*b*),*Cons*(*z*,*env*))
and
a∈*nat w*∈*nat env*∈*list*(*M*)
shows
sats(*M*,*hcomp_fm*(*pf*,*pg*,*a*,*w*),*env*) ⟷ *is_hcomp*(##*M*,*is_f*,*is_g*,*nth*(*a*,*env*),*nth*(*w*,*env*))
proof -
have *sats*(*M*, *pf*(0, *succ*(*w*)), *Cons*(*x*, *env*)) ⟷ *is_f*(*x*,*nth*(*w*,*env*)) **if** *x*∈*M* *w*∈*nat* **for** *x w*
using *f_iff_sats*[*of* 0 *succ*(*w*) *x*] *that* **by** *simp*

moreover
have $sats(M,\ pg(succ(a),\ 0),\ Cons(x,\ env)) \longleftrightarrow is\_g(nth(a,env),x)$ **if** $x{\in}M\ a{\in}nat$
for $x\ a$
using $g\_iff\_sats[of\ succ(a)\ 0\ x]$ *that* **by** *simp*
ultimately
show *?thesis* **unfolding** *hcomp_fm_def is_hcomp_def* **using** *assms* **by** *simp*
qed

definition
$hcomp\_r :: [i{\Rightarrow}o,[i{\Rightarrow}o,i,i]{\Rightarrow}o,[i{\Rightarrow}o,i,i]{\Rightarrow}o,i,i] \Rightarrow o$ **where**
$hcomp\_r(M,is\_f,is\_g,a,w) \equiv \exists\, z[M].\ is\_g(M,a,z) \wedge is\_f(M,z,w)$

definition
$is\_hcomp2\_2 :: [i{\Rightarrow}o,[i{\Rightarrow}o,i,i,i]{\Rightarrow}o,[i{\Rightarrow}o,i,i,i]{\Rightarrow}o,[i{\Rightarrow}o,i,i,i]{\Rightarrow}o,i,i,i] \Rightarrow o$ **where**
$is\_hcomp2\_2(M,is\_f,is\_g1,is\_g2,a,b,w) \equiv \exists\, g1ab[M].\ \exists\, g2ab[M].$
$is\_g1(M,a,b,g1ab) \wedge is\_g2(M,a,b,g2ab) \wedge is\_f(M,g1ab,g2ab,w)$

lemma (**in** *M_trivial*) *hcomp_abs*:
assumes
$is\_f\_abs{:}\bigwedge a\ z.\ M(a) \Longrightarrow M(z) \Longrightarrow is\_f(M,a,z) \longleftrightarrow z = f(a)$ **and**
$is\_g\_abs{:}\bigwedge a\ z.\ M(a) \Longrightarrow M(z) \Longrightarrow is\_g(M,a,z) \longleftrightarrow z = g(a)$ **and**
$g\_closed{:}\bigwedge a.\ M(a) \Longrightarrow M(g(a))$
$M(a)\ M(w)$
shows
$hcomp\_r(M,is\_f,is\_g,a,w) \longleftrightarrow w = f(g(a))$
unfolding *hcomp_r_def* **using** *assms* **by** *simp*

lemma *hcomp_uniqueness*:
assumes
uniq_is_f:
$\bigwedge r\ d\ d'.\ M(r) \Longrightarrow M(d) \Longrightarrow M(d') \Longrightarrow is\_f(M,\ r,\ d) \Longrightarrow is\_f(M,\ r,\ d') \Longrightarrow$
$d = d'$
and
uniq_is_g:
$\bigwedge r\ d\ d'.\ M(r) \Longrightarrow M(d) \Longrightarrow M(d') \Longrightarrow is\_g(M,\ r,\ d) \Longrightarrow is\_g(M,\ r,\ d') \Longrightarrow$
$d = d'$
and
$M(a)\ M(w)\ M(w')$
$hcomp\_r(M,is\_f,is\_g,a,w)$
$hcomp\_r(M,is\_f,is\_g,a,w')$
shows
$w{=}w'$
proof -
from *assms*
obtain $z\ z'$ **where** $is\_g(M,\ a,\ z)\ is\_g(M,\ a,\ z')$
$is\_f(M,z,w)\ is\_f(M,z',w')$
$M(z)\ M(z')$
unfolding *hcomp_r_def* **by** *blast*

moreover from *this* **and** *uniq_is_g* **and** ‹$M(a)$›
have $z{=}z'$ **by** *blast*
moreover note *uniq_is_f* **and** ‹$M(w)$› ‹$M(w')$›
ultimately
show *?thesis* **by** *blast*
qed

lemma *hcomp_witness*:
assumes
wit_is_f: $\bigwedge r.\ M(r) \Longrightarrow \exists\, d[M].\ is\_f(M,r,d)$ **and**
wit_is_g: $\bigwedge r.\ M(r) \Longrightarrow \exists\, d[M].\ is\_g(M,r,d)$ **and**
$M(a)$
shows
$\exists\, w[M].\ hcomp\_r(M,is\_f,is\_g,a,w)$
proof -
from ‹$M(a)$› **and** *wit_is_g*
obtain z **where** $is\_g(M,a,z)$ $M(z)$ **by** *blast*
moreover from *this* **and** *wit_is_f*
obtain w **where** $is\_f(M,z,w)$ $M(w)$ **by** *blast*
ultimately
show *?thesis*
using *assms* **unfolding** *hcomp_r_def* **by** *auto*
qed

lemma (**in** *M_trivial*) *hcomp2_2_abs*:
assumes
is_f_abs:$\bigwedge r1\ r2\ z.\ M(r1) \Longrightarrow M(r2) \Longrightarrow\ M(z) \Longrightarrow is\_f(M,r1,r2,z) \longleftrightarrow z = f(r1,r2)$ **and**
is_g1_abs:$\bigwedge r1\ r2\ z.\ M(r1) \Longrightarrow M(r2) \Longrightarrow\ M(z) \Longrightarrow is\_g1(M,r1,r2,z) \longleftrightarrow z = g1(r1,r2)$ **and**
is_g2_abs:$\bigwedge r1\ r2\ z.\ M(r1) \Longrightarrow M(r2) \Longrightarrow\ M(z) \Longrightarrow is\_g2(M,r1,r2,z) \longleftrightarrow z = g2(r1,r2)$ **and**
types: $M(a)\ M(b)\ M(w)\ M(g1(a,b))\ M(g2(a,b))$
shows
$is\_hcomp2\_2(M,is\_f,is\_g1,is\_g2,a,b,w) \longleftrightarrow w = f(g1(a,b),g2(a,b))$
unfolding *is_hcomp2_2_def* **using** *assms*
— We only need some particular cases of the abs assumptions

by *simp*

lemma *hcomp2_2_uniqueness*:
assumes
uniq_is_f:
$\bigwedge r1\ r2\ d\ d'.\ M(r1) \Longrightarrow M(r2) \Longrightarrow M(d) \Longrightarrow M(d') \Longrightarrow$
$is\_f(M,\ r1,\ r2\ ,\ d) \Longrightarrow is\_f(M,\ r1,\ r2,\ d') \Longrightarrow\ d = d'$
and
uniq_is_g1:
$\bigwedge r1\ r2\ d\ d'.\ M(r1) \Longrightarrow M(r2) \Longrightarrow M(d) \Longrightarrow M(d') \Longrightarrow is\_g1(M,\ r1,r2,\ d) \Longrightarrow is\_g1(M,\ r1,r2,\ d') \Longrightarrow$

$d = d'$
and
uniq_is_g2:
$\bigwedge r1\ r2\ d\ d'.\ M(r1) \Longrightarrow M(r2) \Longrightarrow M(d) \Longrightarrow M(d') \Longrightarrow is\_g2(M,\ r1,r2,\ d) \Longrightarrow is\_g2(M,\ r1,r2,\ d') \Longrightarrow$
$d = d'$
and
$M(a)\ M(b)\ M(w)\ M(w')$
is_hcomp2_2$(M,is\_f,is\_g1,is\_g2,a,b,w)$
is_hcomp2_2$(M,is\_f,is\_g1,is\_g2,a,b,w')$
shows
$w{=}w'$
proof -
from *assms*
obtain $z\ z'\ y\ y'$ **where** $is\_g1(M,\ a,b,\ z)\ is\_g1(M,\ a,b,\ z')$
$is\_g2(M,\ a,b,\ y)\ is\_g2(M,\ a,b,\ y')$
$is\_f(M,z,y,w)\ is\_f(M,z',y',w')$
$M(z)\ M(z')\ M(y)\ M(y')$
unfolding *is_hcomp2_2_def* **by** *force*
moreover from *this* **and** *uniq_is_g1 uniq_is_g2* **and** ‹$M(a)$› ‹$M(b)$›
have $z{=}z'\ y{=}y'$ **by** *blast+*
moreover note *uniq_is_f* **and** ‹$M(w)$› ‹$M(w')$›
ultimately
show *?thesis* **by** *blast*
qed

lemma *hcomp2_2_witness*:
assumes
wit_is_f: $\bigwedge r1\ r2.\ M(r1) \Longrightarrow M(r2) \Longrightarrow \exists\, d[M].\ is\_f(M,r1,r2,d)$ **and**
wit_is_g1: $\bigwedge r1\ r2.\ M(r1) \Longrightarrow M(r2) \Longrightarrow \exists\, d[M].\ is\_g1(M,r1,r2,d)$ **and**
wit_is_g2: $\bigwedge r1\ r2.\ M(r1) \Longrightarrow M(r2) \Longrightarrow \exists\, d[M].\ is\_g2(M,r1,r2,d)$ **and**
$M(a)\ M(b)$
shows
$\exists\, w[M].\ is\_hcomp2\_2(M,is\_f,is\_g1,is\_g2,a,b,w)$
proof -
from ‹$M(a)$› ‹$M(b)$› **and** *wit_is_g1*
obtain *g1a* **where** $is\_g1(M,a,b,g1a)\ M(g1a)$ **by** *blast*
moreover from ‹$M(a)$› ‹$M(b)$› **and** *wit_is_g2*
obtain *g2a* **where** $is\_g2(M,a,b,g2a)\ M(g2a)$ **by** *blast*
moreover from *calculation* **and** *wit_is_f*
obtain *w* **where** $is\_f(M,g1a,g2a,w)\ M(w)$ **by** *blast*
ultimately
show *?thesis*
using *assms* **unfolding** *is_hcomp2_2_def* **by** *auto*
qed

lemma (**in** *M_trivial*) *extensionality_trans*:
assumes
$M(d) \wedge (\forall\, x[M].\ x{\in}d \longleftrightarrow P(x))$

$M(d') \wedge (\forall\, x[M].\ x{\in}d' \longleftrightarrow P(x))$
shows
$d{=}d'$
proof -
from *assms*
have $\forall\, x.\ x{\in}d \longleftrightarrow P(x) \wedge M(x)$
using *transM*[*of* _ *d*] **by** *auto*
moreover from *assms*
have $\forall\, x.\ x{\in}d' \longleftrightarrow P(x) \wedge M(x)$
using *transM*[*of* _ *d'*] **by** *auto*
ultimately
show *?thesis* **by** *auto*
qed

definition
lt_rel :: $[i{\Rightarrow}o,i,i] \Rightarrow o$ **where**
$lt\_rel(M,a,b) \equiv a{\in}b \wedge ordinal(M,b)$

lemma (**in** *M_trans*) *lt_abs*[*absolut*]: $M(a) \Longrightarrow M(b) \Longrightarrow lt\_rel(M,a,b) \longleftrightarrow a{<}b$
unfolding *lt_rel_def lt_def* **by** *auto*

definition
le_rel :: $[i{\Rightarrow}o,i,i] \Rightarrow o$ **where**
$le\_rel(M,a,b) \equiv \exists\, sb[M].\ successor(M,b,sb) \wedge lt\_rel(M,a,sb)$

lemma (**in** *M_trivial*) *le_abs*[*absolut*]: $M(a) \Longrightarrow M(b) \Longrightarrow le\_rel(M,a,b) \longleftrightarrow a{\leq}b$
unfolding *le_rel_def* **by** (*simp add*:*absolut*)

12.1 Discipline for *Pow*

definition
is_Pow :: $[i{\Rightarrow}o,i,i] \Rightarrow o$ **where**
$is\_Pow(M,A,z) \equiv M(z) \wedge (\forall\, x[M].\ x \in z \longleftrightarrow subset(M,x,A))$

definition
Pow_rel :: $[i{\Rightarrow}o,i] \Rightarrow i$ (‹*Pow*–'(_')›) **where**
$Pow\_rel(M,r) \equiv THE\ d.\ is\_Pow(M,r,d)$

abbreviation
Pow_r_set :: $[i,i] \Rightarrow i$ (‹*Pow*–'(_')›) **where**
$Pow\_r\_set(M) \equiv Pow\_rel(\#\#M)$

context *M_basic*
begin

lemma *is_Pow_uniqueness*:
assumes

$M(r)$
$is\_Pow(M,r,d)$ $is\_Pow(M,r,d')$
shows
$d=d'$
using *assms extensionality_trans*
unfolding *is_Pow_def*
by *simp*

lemma *is_Pow_witness*: $M(r) \Longrightarrow \exists\, d[M].\ is\_Pow(M,r,d)$
using *power_ax* **unfolding** *power_ax_def powerset_def is_Pow_def*
by *simp* — We have to do this by hand, using axioms

lemma *is_Pow_closed* : $[\![\ M(r);is\_Pow(M,r,d)\ ]\!] \Longrightarrow M(d)$
unfolding *is_Pow_def* **by** *simp*

lemma *Pow_rel_closed*[*intro*,*simp*]: $M(r) \Longrightarrow M(Pow\_rel(M,r))$
unfolding *Pow_rel_def*
using *is_Pow_closed theI*[*OF ex1I*[*of* $\lambda d.\ is\_Pow(M,r,d)$], *OF _ is_Pow_uniqueness*[*of r*]]
is_Pow_witness
by *fastforce*

lemmas *trans_Pow_rel_closed*[*trans_closed*] = *transM*[*OF _ Pow_rel_closed*]

The proof of *f_rel_iff* lemma is schematic and it can reused by copy-paste replacing appropriately.

lemma *Pow_rel_iff*:
assumes $M(r)$ $M(d)$
shows $is\_Pow(M,r,d) \longleftrightarrow d = Pow\_rel(M,r)$
proof (*intro iffI*)
assume $d = Pow\_rel(M,r)$
with *assms*
show $is\_Pow(M,\ r,\ d)$
using *is_Pow_uniqueness*[*of r*] *is_Pow_witness*
theI[*OF ex1I*[*of* $\lambda d.\ is\_Pow(M,r,d)$], *OF _ is_Pow_uniqueness*[*of r*]]
unfolding *Pow_rel_def*
by *auto*
next
assume $is\_Pow(M,\ r,\ d)$
with *assms*
show $d = Pow\_rel(M,r)$
using *is_Pow_uniqueness* **unfolding** *Pow_rel_def*
by (*auto del*:*the_equality intro*:*the_equality*[*symmetric*])
qed

The next "def_" result really corresponds to $?A \in Pow(?B) \longleftrightarrow ?A \subseteq ?B$

lemma *def_Pow_rel*: $M(A) \Longrightarrow M(r) \Longrightarrow A{\in}Pow\_rel(M,r) \longleftrightarrow A \subseteq r$
using *Pow_rel_iff*[*OF _ Pow_rel_closed*, *of r r*]

unfolding *is_Pow_def* **by** *simp*

lemma *Pow_rel_char*: $M(r) \Longrightarrow Pow\_rel(M,r) = \{A \in Pow(r).\ M(A)\}$
proof -
assume $M(r)$
moreover from *this*
have $x \in Pow\_rel(M,r) \Longrightarrow x \subseteq r\ M(x) \Longrightarrow x \subseteq r \Longrightarrow x \in Pow\_rel(M,r)$ **for** x
using *def_Pow_rel* **by** (*auto intro*!:*trans_closed*)
ultimately
show *?thesis*
using *trans_closed* **by** *blast*
qed

lemma *mem_Pow_rel_abs*: $M(a) \Longrightarrow M(r) \Longrightarrow a \in Pow\_rel(M,r) \longleftrightarrow a \in Pow(r)$
using *Pow_rel_char* **by** *simp*

end

12.2 Discipline for *PiP*

definition
PiP_rel:: $[i \Rightarrow o,i,i] \Rightarrow o$ **where**
$PiP\_rel(M,A,f) \equiv \exists\, df[M].\ is\_domain(M,f,df) \wedge subset(M,A,df) \wedge$
$is\_function(M,f)$

context *M_basic*
begin

lemma *def_PiP_rel*:
assumes
$M(A)\ M(f)$
shows
$PiP\_rel(M,A,f) \longleftrightarrow A \subseteq domain(f) \wedge function(f)$
using *assms* **unfolding** *PiP_rel_def* **by** *simp*

end

definition — FIX THIS: not completely relational. Can it be?
Sigfun :: $[i,i \Rightarrow i] \Rightarrow i$ **where**
$Sigfun(x,B) \equiv \bigcup y \in B(x).\ \{\langle x,y \rangle\}$

lemma *Sigma_Sigfun*: $Sigma(A,B) = \bigcup\ \{Sigfun(x,B)\ .\ x \in A\}$
unfolding *Sigma_def Sigfun_def* **..**

definition — FIX THIS: not completely relational. Can it be?
is_Sigfun :: $[i⇒o,i,i⇒i,i]⇒o$ **where**
is_Sigfun$(M,x,B,Sd) \equiv M(Sd) \wedge (\exists\, RB[M].$ *is_Replace*$(M,B(x),\lambda y\ z.\ z=\{\langle x,y\rangle\},RB)$
$\wedge$ *big_union*$(M,RB,Sd))$

context *M_trivial*
begin

lemma *is_Sigfun_abs*:
assumes
strong_replacement$(M,\lambda y\ z.\ z=\{\langle x,y\rangle\})$
$M(x)$ $M(B(x))$ $M(Sd)$
shows
is_Sigfun$(M,x,B,Sd) \longleftrightarrow Sd =$ *Sigfun*(x,B)
proof -
have $\bigcup\{z\ .\ y \in B(x),\ z = \{\langle x,\ y\rangle\}\} = (\bigcup y{\in}B(x).\ \{\langle x,\ y\rangle\})$ **by** *auto*
then
show *?thesis*
using *assms transM*[*OF* _ ‹$M(B(x))$›] *Replace_abs*
unfolding *is_Sigfun_def Sigfun_def* **by** *auto*
qed

lemma *Sigfun_closed*:
assumes
strong_replacement$(M,\ \lambda y\ z.\ y \in B(x) \wedge z = \{\langle x,\ y\rangle\})$
$M(x)$ $M(B(x))$
shows
$M($*Sigfun*$(x,B))$
using *assms transM*[*OF* _ ‹$M(B(x))$›] *RepFun_closed2*
unfolding *Sigfun_def* **by** *simp*

lemmas *trans_Sigfun_closed*[*trans_closed*] = *transM*[*OF* _ *Sigfun_closed*]

end

definition
is_Sigma :: $[i⇒o,i,i⇒i,i]⇒o$ **where**
is_Sigma$(M,A,B,S) \equiv M(S) \wedge (\exists\, RSf[M].$
is_Replace$(M,A,\lambda x\ z.\ z=$*Sigfun*$(x,B),RSf) \wedge$ *big_union*$(M,RSf,S))$

locale *M_Pi* = *M_basic* +
assumes
Pi_separation: $M(A) \Longrightarrow$ *separation*$(M,\ PiP\_rel(M,A))$
and
Pi_replacement:
$M(x) \Longrightarrow M(y) \Longrightarrow$
strong_replacement$(M,\ \lambda ya\ z.\ ya \in y \wedge z = \{\langle x,\ ya\rangle\})$
$M(y) \Longrightarrow$

strong_replacement(M, $\lambda x\ z.\ z = (\bigcup xa{\in}y.\ \{\langle x,\ xa\rangle\})$)

locale *M_Pi_assumptions* = *M_Pi* +
  fixes A B
  assumes
    Pi_assumptions:
    $M(A)$
    $\bigwedge x.\ x{\in}A \Longrightarrow\ M(B(x))$
    $\forall\, x{\in}A.$ *strong_replacement*(M, $\lambda y\ z.\ y \in B(x) \wedge z = \{\langle x,\ y\rangle\}$)
    strong_replacement(M,$\lambda x\ z.$ z=*Sigfun*(x,B))
begin

lemma *Sigma_abs*[*simp*]:
  assumes
    $M(S)$
  shows
    is_Sigma(M,A,B,S) $\longleftrightarrow$ S = *Sigma*(A,B)
proof -
  have $\bigcup\{z\ .\ x \in A,\ z = Sigfun(x,\ B)\} = (\bigcup x{\in}A.\ Sigfun(x,\ B))$
    by *auto*
  with *assms*
  show *?thesis*
    using *Replace_abs*[*of A _* $\lambda x\ z.$ z=*Sigfun*(x,B)]
      Sigfun_closed Sigma_Sigfun[*of A B*] *transM*[*of _ A*]
      Pi_assumptions is_Sigfun_abs
    unfolding *is_Sigma_def* **by** *simp*
qed

lemma *Sigma_closed*[*intro*,*simp*]: $M(Sigma(A,B))$
proof -
  have $(\bigcup x{\in}A.\ Sigfun(x,\ B)) = \bigcup\{z\ .\ x \in A,\ z = Sigfun(x,\ B)\}$
    by *auto*
  then
  show *?thesis*
    using *Sigma_Sigfun*[*of A B*] *transM*[*of _ A*]
      Sigfun_closed Pi_assumptions
    by *simp*
qed

lemmas *trans_Sigma_closed*[*trans_closed*] = *transM*[*OF _ Sigma_closed*]

end

12.3 Discipline for *Pi*

definition
  is_Pi :: [$i{\Rightarrow}o$,i,$i{\Rightarrow}i$,i]$\Rightarrow o$ **where**
  is_Pi(M,A,B,I) $\equiv$ $M(I)$ $\wedge$ ($\exists\, S[M].\ \exists\, PS[M].$ *is_Sigma*(M,A,B,S) $\wedge$
      is_Pow(M,S,PS) $\wedge$

is_Collect(*M*,*PS*,*PiP_rel*(*M*,*A*),*I*))

definition
Pi_rel :: [*i*⇒*o*,*i*,*i*⇒*i*] ⇒ *i* (‹*Pi–'*(_,_')›) **where**
Pi_rel(*M*,*A*,*B*) ≡ *THE d*. *is_Pi*(*M*,*A*,*B*,*d*)

abbreviation
Pi_r_set :: [*i*,*i*,*i*⇒*i*] ⇒ *i* (‹*Pi–'*(_,_')›) **where**
Pi_r_set(*M*,*A*,*B*) ≡ *Pi_rel*(##*M*,*A*,*B*)

context *M_Pi_assumptions*
begin

lemma *is_Pi_uniqueness*:
assumes
is_Pi(*M*,*A*,*B*,*d*) *is_Pi*(*M*,*A*,*B*,*d'*)
shows
d=*d'*
using *assms Pi_assumptions extensionality_trans*
Pow_rel_iff
unfolding *is_Pi_def* **by** *simp*

lemma *is_Pi_witness*: ∃ *d*[*M*]. *is_Pi*(*M*,*A*,*B*,*d*)
using *Pow_rel_iff Pi_separation Pi_assumptions*
unfolding *is_Pi_def* **by** *simp*

lemma *is_Pi_closed* : *is_Pi*(*M*,*A*,*B*,*d*) ⟹ *M*(*d*)
unfolding *is_Pi_def* **by** *simp*

lemma *Pi_rel_closed*[*intro*,*simp*]: *M*(*Pi_rel*(*M*,*A*,*B*))
proof -
have *is_Pi*(*M*, *A*, *B*, *THE xa*. *is_Pi*(*M*, *A*, *B*, *xa*))
using *Pi_assumptions*
theI[*OF ex1I*[*of is_Pi*(*M*,*A*,*B*)], *OF* _ *is_Pi_uniqueness*]
is_Pi_witness is_Pi_closed
by *auto*
then show *?thesis*
using *is_Pi_closed*
unfolding *Pi_rel_def*
by *simp*
qed

— From this point on, the higher order variable *y* must be explicitly instantiated, and proof methods are slower

lemmas *trans_Pi_rel_closed*[*trans_closed*] = *transM*[*OF* _ *Pi_rel_closed*]

lemma *Pi_rel_iff*:
assumes $M(d)$
shows $is\_Pi(M,A,B,d) \longleftrightarrow d = Pi\_rel(M,A,B)$
proof (*intro iffI*)
assume $d = Pi\_rel(M,A,B)$
moreover
note *assms*
moreover from *this*
obtain e **where** $M(e)$ $is\_Pi(M,A,B,e)$
using *is_Pi_witness* **by** *blast*
ultimately
show $is\_Pi(M,\ A,\ B,\ d)$
using *is_Pi_uniqueness is_Pi_witness is_Pi_closed*
theI[*OF ex1I*[*of is_Pi*(*M*,*A*,*B*)], *OF* _ *is_Pi_uniqueness*, *of e*]
unfolding *Pi_rel_def*
by *simp*
next
assume $is\_Pi(M,\ A,\ B,\ d)$
with *assms*
show $d = Pi\_rel(M,A,B)$
using *is_Pi_uniqueness is_Pi_closed* **unfolding** *Pi_rel_def*
by (*blast del*:*the_equality intro*:*the_equality*[*symmetric*])
qed

lemma *def_Pi_rel*:
$Pi\_rel(M,A,B) = \{f \in Pow\_rel(M,Sigma(A,B)).\ A \subseteq domain(f) \wedge function(f)\}$
proof -
have $Pi\_rel(M,A,\ B) \subseteq Pow\_rel(M,Sigma(A,B))$
using *Pi_assumptions Pi_rel_iff*[*of* $Pi\_rel(M,A,B)$] *Pow_rel_iff*
unfolding *is_Pi_def* **by** *auto*
moreover
have $f \in Pi\_rel(M,A,\ B) \Longrightarrow A \subseteq domain(f) \wedge function(f)$ **for** f
using *Pi_assumptions Pi_rel_iff*[*of* $Pi\_rel(M,A,B)$]
def_PiP_rel[*of A f*] *trans_closed Pow_rel_iff*
unfolding *is_Pi_def* **by** *simp*
moreover
have $f \in Pow\_rel(M,Sigma(A,B)) \Longrightarrow A \subseteq domain(f) \wedge function(f) \Longrightarrow f \in Pi\_rel(M,A,\ B)$ **for** f
using *Pi_rel_iff*[*of* $Pi\_rel(M,A,B)$] *Pi_assumptions*
def_PiP_rel[*of A f*] *trans_closed Pow_rel_iff*
unfolding *is_Pi_def* **by** *simp*
ultimately
show *?thesis* **by** *force*
qed

lemma *Pi_rel_char*: $Pi\_rel(M,A,B) = \{f \in Pi(A,B).\ M(f)\}$
using *Pi_assumptions def_Pi_rel Pow_rel_char*[*OF Sigma_closed*] **unfolding** *Pi_def*
by *fastforce*

lemma *mem_Pi_rel_abs*:
assumes $M(f)$
shows $f \in Pi\_rel(M,A,B) \longleftrightarrow f \in Pi(A,B)$
using *assms Pi_rel_char* **by** *simp*

end

The next locale (and similar ones below) are used to show the relationship between versions of simple (i.e. Σ_1^{ZF}, Π_1^{ZF}) concepts in two different transitive models.

locale *M_N_Pi_assumptions* = *M*:*M_Pi_assumptions* + *N*:*M_Pi_assumptions* *N* **for** *N* +
assumes
M_imp_N:$M(x) \Longrightarrow N(x)$
begin

lemma *Pi_rel_transfer*: $Pi^M(A,B) \subseteq Pi^N(A,B)$
using *M.Pi_rel_char N.Pi_rel_char M_imp_N* **by** *auto*

end

locale *M_Pi_assumptions_0* = *M_Pi_assumptions* _ *0*
begin

This is used in the proof of *AC_Pi_rel*

lemma *Pi_rel_empty1*[*simp*]: $Pi^M(0,B) = \{0\}$
using *Pi_assumptions Pow_rel_char*
by (*unfold def_Pi_rel function_def*) (*auto*)

end

context *M_Pi_assumptions*
begin

12.4 Auxiliary ported results on *Pi_rel*, now unused

lemma *Pi_rel_iff'*:
assumes *types*:$M(f)$
shows
$f \in Pi\_rel(M,A,B) \longleftrightarrow function(f) \wedge f \subseteq Sigma(A,B) \wedge A \subseteq domain(f)$
using *assms Pow_rel_char*
by (*simp add*:*def_Pi_rel*, *blast*)

lemma *lam_type_M*:

assumes $M(A)$ $\bigwedge x.\ x{\in}A \Longrightarrow\ M(B(x))$
$\bigwedge x.\ x \in A \Longrightarrow b(x){\in}B(x)$ *strong_replacement*$(M,\lambda x\ y.\ y{=}\langle x,\ b(x)\rangle)$
shows $(\lambda x{\in}A.\ b(x)) \in$ *Pi_rel*(M,A,B)
proof (*auto simp add*: *lam_def def_Pi_rel function_def*)
from *assms*
have $M(\{\langle x,\ b(x)\rangle\ .\ x \in A\})$
using *Pi_assumptions transM*[*OF* _ ‹$M(A)$›]
by (*rule_tac RepFun_closed*, *auto intro*!:*transM*[*OF* _ ‹$\bigwedge x.\ x{\in}A \Longrightarrow\ M(B(x))$›])
with *assms*
show $\{\langle x,\ b(x)\rangle\ .\ x \in A\} \in Pow^M(Sigma(A,\ B))$
using *Pow_rel_char* **by** *auto*
qed

end

locale *M_Pi_assumptions2* = *M_Pi_assumptions* +
PiC: *M_Pi_assumptions* _ _ *C* **for** *C*
begin

lemma *Pi_rel_type*:
assumes $f \in Pi^M(A,C)$ $\bigwedge x.\ x \in A \Longrightarrow f`x \in B(x)$
and *types*: $M(f)$
shows $f \in Pi^M(A,B)$
using *assms Pi_assumptions*
by (*simp only*: *Pi_rel_iff'* *PiC.Pi_rel_iff'*)
(*blast dest*: *function_apply_equality*)

lemma *Pi_rel_weaken_type*:
assumes $f \in Pi^M(A,B)$ $\bigwedge x.\ x \in A \Longrightarrow B(x) \subseteq C(x)$
and *types*: $M(f)$
shows $f \in Pi^M(A,C)$
using *assms Pi_assumptions*
by (*simp only*: *Pi_rel_iff'* *PiC.Pi_rel_iff'*)
(*blast intro*: *Pi_rel_type* *dest*: *apply_type*)

end

end

13 Arities of internalized formulas

theory *Arities*
imports
Nat_Miscellanea
Internalizations
Discipline_Base
begin

declare *arity_And arity_Or arity_Implies arity_Iff arity_Exists* [*arity*]

declare *pred_Un_distrib* [*arity*]

lemma *arity_upair_fm* [*arity*] : ⟦ *t1∈nat* ; *t2∈nat* ; *up∈nat* ⟧ ⟹
arity(*upair_fm*(*t1*,*t2*,*up*)) = ⋃ {*succ*(*t1*),*succ*(*t2*),*succ*(*up*)}
unfolding *upair_fm_def*
using *union_abs1 union_abs2 pred_Un*
by *auto*

lemma *arity_pair_fm* [*arity*] : ⟦ *t1∈nat* ; *t2∈nat* ; *p∈nat* ⟧ ⟹
arity(*pair_fm*(*t1*,*t2*,*p*)) = ⋃ {*succ*(*t1*),*succ*(*t2*),*succ*(*p*)}
unfolding *pair_fm_def*
using *arity_upair_fm union_abs1 union_abs2 pred_Un*
by *auto*

lemma *arity_composition_fm* [*arity*] :
⟦ *r∈nat* ; *s∈nat* ; *t∈nat* ⟧ ⟹ *arity*(*composition_fm*(*r*,*s*,*t*)) = ⋃ {*succ*(*r*), *succ*(*s*), *succ*(*t*)}
unfolding *composition_fm_def*
using *arity_pair_fm union_abs1 union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_domain_fm* [*arity*] :
⟦ *r∈nat* ; *z∈nat* ⟧ ⟹ *arity*(*domain_fm*(*r*,*z*)) = *succ*(*r*) ∪ *succ*(*z*)
unfolding *domain_fm_def*
using *arity_pair_fm union_abs1 union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_range_fm* [*arity*] :
⟦ *r∈nat* ; *z∈nat* ⟧ ⟹ *arity*(*range_fm*(*r*,*z*)) = *succ*(*r*) ∪ *succ*(*z*)
unfolding *range_fm_def*
using *arity_pair_fm union_abs1 union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_union_fm* [*arity*] :
⟦ *x∈nat* ; *y∈nat* ; *z∈nat* ⟧ ⟹ *arity*(*union_fm*(*x*,*y*,*z*)) = ⋃ {*succ*(*x*), *succ*(*y*), *succ*(*z*)}
unfolding *union_fm_def*
using *union_abs1 union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_image_fm* [*arity*] :
⟦ *x∈nat* ; *y∈nat* ; *z∈nat* ⟧ ⟹ *arity*(*image_fm*(*x*,*y*,*z*)) = ⋃ {*succ*(*x*), *succ*(*y*), *succ*(*z*)}
unfolding *image_fm_def*
using *arity_pair_fm union_abs1 union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_pre_image_fm* [*arity*] :
 ⟦ *x*∈*nat* ; *y*∈*nat* ; *z*∈*nat* ⟧ ⟹ *arity*(*pre_image_fm*(*x*,*y*,*z*)) = ⋃ {*succ*(*x*), *succ*(*y*), *succ*(*z*)}
 unfolding *pre_image_fm_def*
 using *arity_pair_fm union_abs1 union_abs2 pred_Un_distrib*
 by *auto*

lemma *arity_big_union_fm* [*arity*] :
 ⟦ *x*∈*nat* ; *y*∈*nat* ⟧ ⟹ *arity*(*big_union_fm*(*x*,*y*)) = *succ*(*x*) ∪ *succ*(*y*)
 unfolding *big_union_fm_def*
 using *union_abs1 union_abs2 pred_Un_distrib*
 by *auto*

lemma *arity_fun_apply_fm* [*arity*] :
 ⟦ *x*∈*nat* ; *y*∈*nat* ; *f*∈*nat* ⟧ ⟹
  arity(*fun_apply_fm*(*f*,*x*,*y*)) = *succ*(*f*) ∪ *succ*(*x*) ∪ *succ*(*y*)
 unfolding *fun_apply_fm_def*
 using *arity_upair_fm arity_image_fm arity_big_union_fm union_abs2 pred_Un_distrib*
 by *auto*

lemma *arity_field_fm* [*arity*] :
  ⟦ *r*∈*nat* ; *z*∈*nat* ⟧ ⟹ *arity*(*field_fm*(*r*,*z*)) = *succ*(*r*) ∪ *succ*(*z*)
 unfolding *field_fm_def*
 using *arity_pair_fm arity_domain_fm arity_range_fm arity_union_fm*
  union_abs1 union_abs2 pred_Un_distrib
 by *auto*

lemma *arity_empty_fm* [*arity*]:
  ⟦ *r*∈*nat* ⟧ ⟹ *arity*(*empty_fm*(*r*)) = *succ*(*r*)
 unfolding *empty_fm_def*
 using *union_abs1 union_abs2 pred_Un_distrib*
 by *simp*

lemma *arity_cons_fm* [*arity*] :
 ⟦*x*∈*nat*;*y*∈*nat*;*z*∈*nat*⟧ ⟹ *arity*(*cons_fm*(*x*,*y*,*z*)) = *succ*(*x*) ∪ *succ*(*y*) ∪ *succ*(*z*)
 unfolding *cons_fm_def*
 using *arity_upair_fm arity_union_fm union_abs2 pred_Un_distrib*
 by *auto*

lemma *arity_succ_fm* [*arity*] :
 ⟦*x*∈*nat*;*y*∈*nat*⟧ ⟹ *arity*(*succ_fm*(*x*,*y*)) = *succ*(*x*) ∪ *succ*(*y*)
 unfolding *succ_fm_def*
 using *arity_cons_fm*
 by *auto*

lemma *arity_number1_fm* [*arity*] :
  ⟦ *r*∈*nat* ⟧ ⟹ *arity*(*number1_fm*(*r*)) = *succ*(*r*)

unfolding *number1_fm_def*
using *arity_empty_fm arity_succ_fm union_abs1 union_abs2 pred_Un_distrib*
by *simp*

lemma *arity_function_fm* [*arity*] :
$\llbracket\ r{\in}nat\ \rrbracket \Longrightarrow arity(function\_fm(r)) = succ(r)$
unfolding *function_fm_def*
using *arity_pair_fm union_abs1 union_abs2 pred_Un_distrib*
by *simp*

lemma *arity_relation_fm* [*arity*] :
$\llbracket\ r{\in}nat\ \rrbracket \Longrightarrow arity(relation\_fm(r)) = succ(r)$
unfolding *relation_fm_def*
using *arity_pair_fm union_abs1 union_abs2 pred_Un_distrib*
by *simp*

lemma *arity_restriction_fm* [*arity*] :
$\llbracket\ r{\in}nat\ ;\ z{\in}nat\ ;\ A{\in}nat\ \rrbracket \Longrightarrow arity(restriction\_fm(A,z,r)) = succ(A) \cup succ(r) \cup succ(z)$
unfolding *restriction_fm_def*
using *arity_pair_fm union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_typed_function_fm* [*arity*] :
$\llbracket\ x{\in}nat\ ;\ y{\in}nat\ ;\ f{\in}nat\ \rrbracket \Longrightarrow$
$arity(typed\_function\_fm(f,x,y)) = \bigcup\ \{succ(f),\ succ(x),\ succ(y)\}$
unfolding *typed_function_fm_def*
using *arity_pair_fm arity_relation_fm arity_function_fm arity_domain_fm union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_subset_fm* [*arity*] :
$[\![x{\in}nat\ ;\ y{\in}nat]\!] \Longrightarrow arity(subset\_fm(x,y)) = succ(x) \cup succ(y)$
unfolding *subset_fm_def*
using *union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_transset_fm* [*arity*] :
$[\![x{\in}nat]\!] \Longrightarrow arity(transset\_fm(x)) = succ(x)$
unfolding *transset_fm_def*
using *arity_subset_fm union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_ordinal_fm* [*arity*] :
$[\![x{\in}nat]\!] \Longrightarrow arity(ordinal\_fm(x)) = succ(x)$
unfolding *ordinal_fm_def*
using *arity_transset_fm union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_limit_ordinal_fm* [*arity*] :
  ⟦*x*∈*nat*⟧ ⟹ *arity*(*limit_ordinal_fm*(*x*)) = *succ*(*x*)
  unfolding *limit_ordinal_fm_def*
 using *arity_ordinal_fm arity_succ_fm arity_empty_fm union_abs2 pred_Un_distrib*
  by *auto*

lemma *arity_finite_ordinal_fm* [*arity*] :
  ⟦*x*∈*nat*⟧ ⟹ *arity*(*finite_ordinal_fm*(*x*)) = *succ*(*x*)
  unfolding *finite_ordinal_fm_def*
 using *arity_ordinal_fm arity_limit_ordinal_fm arity_succ_fm arity_empty_fm*

    union_abs2 pred_Un_distrib
  by *auto*

lemma *arity_omega_fm* [*arity*] :
  ⟦*x*∈*nat*⟧ ⟹ *arity*(*omega_fm*(*x*)) = *succ*(*x*)
  unfolding *omega_fm_def*
  using *arity_limit_ordinal_fm union_abs2 pred_Un_distrib*
  by *auto*

lemma *arity_cartprod_fm* [*arity*] :
  ⟦ *A*∈*nat* ; *B*∈*nat* ; *z*∈*nat* ⟧ ⟹ *arity*(*cartprod_fm*(*A*,*B*,*z*)) = *succ*(*A*) ∪ *succ*(*B*)
∪ *succ*(*z*)
  unfolding *cartprod_fm_def*
  using *arity_pair_fm union_abs2 pred_Un_distrib*
  by *auto*

lemma *arity_singleton_fm* [*arity*] :
  ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*singleton_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
  unfolding *singleton_fm_def cons_fm_def*
 using *arity_union_fm arity_upair_fm arity_empty_fm union_abs2 pred_Un_distrib*
  by *auto*

lemma *arity_Memrel_fm* [*arity*] :
  ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*Memrel_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
  unfolding *Memrel_fm_def*
  using *arity_pair_fm union_abs2 pred_Un_distrib*
  by *auto*

lemma *arity_quasinat_fm* [*arity*] :
  ⟦*x*∈*nat*⟧ ⟹ *arity*(*quasinat_fm*(*x*)) = *succ*(*x*)
  unfolding *quasinat_fm_def cons_fm_def*
  using *arity_succ_fm arity_empty_fm*
    union_abs2 pred_Un_distrib
  by *auto*

lemma *arity_is_recfun_fm* [*arity*] :
  ⟦*p*∈*formula* ; *v*∈*nat* ; *n*∈*nat*; *Z*∈*nat*;*i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹

arity(*is_recfun_fm*(*p*,*v*,*n*,*Z*)) = *succ*(*v*) ∪ *succ*(*n*) ∪ *succ*(*Z*) ∪ *pred*(*pred*(*pred*(*pred*(*i*))))
unfolding *is_recfun_fm_def*
using *arity_upair_fm arity_pair_fm arity_pre_image_fm arity_restriction_fm*
union_abs2 pred_Un_distrib
by *auto*

lemma *arity_is_wfrec_fm* [*arity*] :
⟦*p*∈*formula* ; *v*∈*nat* ; *n*∈*nat*; *Z*∈*nat* ; *i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
arity(*is_wfrec_fm*(*p*,*v*,*n*,*Z*)) = *succ*(*v*) ∪ *succ*(*n*) ∪ *succ*(*Z*) ∪ *pred*(*pred*(*pred*(*pred*(*pred*(*i*)))))
unfolding *is_wfrec_fm_def*
using *arity_succ_fm arity_is_recfun_fm*
union_abs2 pred_Un_distrib
by *auto*

lemma *arity_is_nat_case_fm* [*arity*] :
⟦*p*∈*formula* ; *v*∈*nat* ; *n*∈*nat*; *Z*∈*nat*; *i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
arity(*is_nat_case_fm*(*v*,*p*,*n*,*Z*)) = *succ*(*v*) ∪ *succ*(*n*) ∪ *succ*(*Z*) ∪ *pred*(*pred*(*i*))
unfolding *is_nat_case_fm_def*
using *arity_succ_fm arity_empty_fm arity_quasinat_fm*
union_abs2 pred_Un_distrib
by *auto*

lemma *arity_iterates_MH_fm* [*arity*] :
assumes *isF*∈*formula v*∈*nat n*∈*nat g*∈*nat z*∈*nat i*∈*nat*
arity(*isF*) = *i*
shows *arity*(*iterates_MH_fm*(*isF*,*v*,*n*,*g*,*z*)) =
succ(*v*) ∪ *succ*(*n*) ∪ *succ*(*g*) ∪ *succ*(*z*) ∪ *pred*(*pred*(*pred*(*pred*(*i*))))
proof -
let *?φ* = *Exists*(*And*(*fun_apply_fm*(*succ*(*succ*(*succ*(*g*))), *2*, *0*), *Forall*(*Implies*(*Equal*(*0*, *2*), *isF*))))
let *?ar* = *succ*(*succ*(*succ*(*g*))) ∪ *pred*(*pred*(*i*))
from *assms*
have *arity*(*?φ*) = *?ar* *?φ*∈*formula*
using *arity_fun_apply_fm*
union_abs1 union_abs2 pred_Un_distrib succ_Un_distrib Un_assoc[*symmetric*]
by *simp_all*
then
show *?thesis*
unfolding *iterates_MH_fm_def*
using *arity_is_nat_case_fm*[*OF* ‹*?φ*∈_› _ _ _ _ ‹*arity*(*?φ*) = _›] *assms*
pred_succ_eq pred_Un_distrib
by *auto*
qed

lemma *arity_is_iterates_fm* [*arity*] :
assumes *p*∈*formula v*∈*nat n*∈*nat Z*∈*nat i*∈*nat*
arity(*p*) = *i*
shows *arity*(*is_iterates_fm*(*p*,*v*,*n*,*Z*)) = *succ*(*v*) ∪ *succ*(*n*) ∪ *succ*(*Z*) ∪
pred(*pred*(*pred*(*pred*(*pred*(*pred*(*pred*(*pred*(*pred*(*pred*(*i*)))))))))))

proof -
  let *?φ = iterates_MH_fm(p, 7#+v, 2, 1, 0)*
  let *?ψ = is_wfrec_fm(?φ, 0, succ(succ(n)),succ(succ(Z)))*
  from ‹*v∈_*›
  have *arity(?φ) = (8#+v) ∪ pred(pred(pred(pred(i)))) ?φ∈formula*
    using *assms arity_iterates_MH_fm union_abs2*
    by *simp_all*
  then
  have *arity(?ψ) = succ(succ(succ(n))) ∪ succ(succ(succ(Z))) ∪ (3#+v) ∪*
      pred(pred(pred(pred(pred(pred(pred(pred(pred(i))))))))))
      using *assms arity_is_wfrec_fm[OF* ‹*?φ∈_*› *_ _ _ _* ‹*arity(?φ) = _*›*]*
union_abs1 pred_Un_distrib
    by *auto*
  then
  show *?thesis*
    unfolding *is_iterates_fm_def*
    using *arity_Memrel_fm arity_succ_fm assms union_abs1 pred_Un_distrib*
    by *auto*
qed

lemma *arity_eclose_n_fm* [*arity*] :
  assumes *A∈nat x∈nat t∈nat*
  shows *arity(eclose_n_fm(A,x,t)) = succ(A) ∪ succ(x) ∪ succ(t)*
proof -
  let *?φ = big_union_fm(1,0)*
  have *arity(?φ) = 2 ?φ∈formula*
    using *arity_big_union_fm union_abs2*
    by *simp_all*
  with *assms*
  show *?thesis*
    unfolding *eclose_n_fm_def*
    using *arity_is_iterates_fm[OF* ‹*?φ∈_*› *_ _ _*,*of _ _ _ 2*]
    by *auto*
qed

lemma *arity_mem_eclose_fm* [*arity*] :
  assumes *x∈nat t∈nat*
  shows *arity(mem_eclose_fm(x,t)) = succ(x) ∪ succ(t)*
proof -
  let *?φ=eclose_n_fm(x #+ 2, 1, 0)*
  from ‹*x∈nat*›
  have *arity(?φ) = x#+3*
    using *arity_eclose_n_fm union_abs2*
    by *simp*
  with *assms*
  show *?thesis*
    unfolding *mem_eclose_fm_def*
    using *arity_finite_ordinal_fm union_abs2 pred_Un_distrib*
    by *simp*

qed

lemma *arity_is_eclose_fm* [*arity*] :
  ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*is_eclose_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
  unfolding *is_eclose_fm_def*
  using *arity_mem_eclose_fm union_abs2 pred_Un_distrib*
  by *auto*

lemma *arity_Collect_fm* [*arity*] :
  assumes *x* ∈ *nat y* ∈ *nat p*∈*formula*
  shows *arity*(*Collect_fm*(*x*,*p*,*y*)) = *succ*(*x*) ∪ *succ*(*y*) ∪ *pred*(*arity*(*p*))
  unfolding *Collect_fm_def*
  using *assms pred_Un_distrib*
  by *auto*

schematic_goal *arity_least_fm'*:
  assumes
    i ∈ *nat q* ∈ *formula*
  shows
    arity(*least_fm*(*q*,*i*)) ≡ *?ar*
  unfolding *least_fm_def*
   using *assms pred_Un_distrib arity_And arity_Or arity_Neg arity_Implies arity_ordinal_fm*
       arity_empty_fm Un_assoc[*symmetric*] *Un_commute*
  by *auto*

lemma *arity_least_fm* [*arity*] :
  assumes
    i ∈ *nat q* ∈ *formula*
  shows
    arity(*least_fm*(*q*,*i*)) = *succ*(*i*) ∪ *pred*(*arity*(*q*))
  using *assms arity_least_fm'*
  by *auto*

lemma *arity_Replace_fm* [*arity*] :
  ⟦*p*∈*formula* ; *v*∈*nat* ; *n*∈*nat*; *i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
    arity(*Replace_fm*(*v*,*p*,*n*)) = *succ*(*n*) ∪ (*succ*(*v*) ∪ *Arith.pred*(*Arith.pred*(*i*)))
  unfolding *Replace_fm_def*
  using *union_abs2 pred_Un_distrib*
  by *simp*

lemma *arity_lambda_fm* [*arity*] :
  ⟦*p*∈*formula*; *v*∈*nat* ; *n*∈*nat*; *i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
   arity(*lambda_fm*(*p*,*v*,*n*)) = *succ*(*n*) ∪ (*succ*(*v*) ∪ *Arith.pred*(*Arith.pred*(*Arith.pred*(*i*))))
  unfolding *lambda_fm_def*
  using *arity_pair_fm pred_Un_distrib union_abs1 union_abs2*
  by *simp*

lemma *arity_transrec_fm* [*arity*] :

⟦p∈*formula* ; v∈*nat* ; n∈*nat*; i∈*nat*⟧ ⟹ *arity*(p) = i ⟹
arity(*is_transrec_fm*(p,v,n)) = *succ*(v) ∪ *succ*(n) ∪ (*pred*^8(i))
unfolding *is_transrec_fm_def*
using *arity Un_assoc*[*symmetric*]
by *simp*

end
theory *Discipline_Function*
imports
ZF_Miscellanea
ZF-Constructible.Rank
Relativization
Internalizations
Discipline_Base
Synthetic_Definition
Arities
begin

Discipline for *fst* **arity_theorem for** *empty_fm*
arity_theorem for *upair_fm*
arity_theorem for *pair_fm*
definition
is_fst :: (*i*⇒*o*)⇒*i*⇒*i*⇒*o* **where**
is_fst(M,x,t) ≡ (∃ *z*[M]. *pair*(M,t,z,x)) ∨
(¬(∃ *z*[M]. ∃ *w*[M]. *pair*(M,w,z,x)) ∧ *empty*(M,t))
synthesize *fst* **from_definition** *is_fst*
notation *fst_fm* (‹·*fst*′(_′) *is* _·›)

arity_theorem for *fst_fm*

definition *fst_rel* :: [*i*⇒*o*,*i*] ⇒ *i* **where**
fst_rel(M,p) ≡ *THE d*. *M*(d) ∧ *is_fst*(M,p,d)

reldb_add relational *fst is_fst*
reldb_add functional *fst fst_rel*

definition
is_snd :: (*i*⇒*o*)⇒*i*⇒*i*⇒*o* **where**
is_snd(M,x,t) ≡ (∃ *z*[M]. *pair*(M,z,t,x)) ∨
(¬(∃ *z*[M]. ∃ *w*[M]. *pair*(M,z,w,x)) ∧ *empty*(M,t))
synthesize *snd* **from_definition** *is_snd*
notation *snd_fm* (‹·*snd*′(_′) *is* _·›)
arity_theorem for *snd_fm*

definition *snd_rel* :: [*i*⇒*o*,*i*] ⇒ *i* **where**
snd_rel(M,p) ≡ *THE d*. *M*(d) ∧ *is_snd*(M,p,d)

reldb_add relational *snd is_snd*

reldb_add functional *snd snd_rel*

context *M_trans*
begin

lemma *fst_snd_closed*:
assumes $M(p)$
shows $M(fst(p)) \wedge M(snd(p))$
unfolding *fst_def snd_def* **using** *assms*
by (*cases* $\exists\, a.\ \exists\, b.\ p = \langle a,\ b\rangle$;*auto*)

lemma *fst_closed*[*intro*,*simp*]: $M(x) \Longrightarrow M(fst(x))$
using *fst_snd_closed* **by** *auto*

lemma *snd_closed*[*intro*,*simp*]: $M(x) \Longrightarrow M(snd(x))$
using *fst_snd_closed* **by** *auto*

lemma *fst_abs* [*absolut*]:
$[\![M(p);\ M(x)\ ]\!] \Longrightarrow is\_fst(M,p,x) \longleftrightarrow x = fst(p)$
unfolding *is_fst_def fst_def*
by (*cases* $\exists\, a.\ \exists\, b.\ p = \langle a,\ b\rangle$;*auto*)

lemma *snd_abs* [*absolut*]:
$[\![M(p);\ M(y)\ ]\!] \Longrightarrow is\_snd(M,p,y) \longleftrightarrow y = snd(p)$
unfolding *is_snd_def snd_def*
by (*cases* $\exists\, a.\ \exists\, b.\ p = \langle a,\ b\rangle$;*auto*)

lemma *empty_rel_abs* : $M(x) \Longrightarrow M(0) \Longrightarrow x = 0 \longleftrightarrow x = (THE\ d.\ M(d) \wedge empty(M,\ d))$
unfolding *the_def*
using *transM*
by *auto*

lemma *fst_rel_abs*:
$[\![M(p)\ ]\!] \Longrightarrow fst(p) = fst\_rel(M,p)$
unfolding *fst_def fst_rel_def*
using *fst_abs*
apply (*cases* $\exists\, a.\ \exists\, b.\ p = \langle a,\ b\rangle$;*auto*)
apply(*rule_tac the_equality*[*symmetric*],*simp_all*)
apply(*rule_tac the_equality*[*symmetric*],*simp_all add*:*fst_def*)
done

lemma *snd_rel_abs*:
$[\![M(p)\ ]\!] \Longrightarrow snd(p) = snd\_rel(M,p)$
unfolding *snd_def snd_rel_def*
using *snd_abs*
apply (*cases* $\exists\, a.\ \exists\, b.\ p = \langle a,\ b\rangle$;*auto*)
apply(*rule_tac the_equality*[*symmetric*],*simp_all*)
apply(*rule_tac the_equality*[*symmetric*],*simp_all add*:*snd_def*)

done

end

relativize functional *first first_rel* **external**
relativize functional *minimum minimum_rel* **external**
context *M_trans*
begin

lemma *minimum_closed*[*simp*,*intro*]:
assumes $M(A)$
shows $M(minimum(r,A))$
using *first_is_elem the_equality_if transM*[*OF _* ‹$M(A)$›]
by(*cases* $\exists\, x$. *first*(x,A,r),*auto simp*:*minimum_def*)

lemma *first_abs* :
assumes $M(B)$
shows $first(z,B,r) \longleftrightarrow first\_rel(M,z,B,r)$
unfolding *first_def first_rel_def* **using** *assms* **by** *auto*

— FIXME: find a naming convention for absoluteness results like this.
lemma *minimum_abs*:
assumes $M(B)$
shows $minimum(r,B) = minimum\_rel(M,r,B)$
proof -
from *assms*
have $first(b,\ B,\ r) \longleftrightarrow M(b) \wedge first\_rel(M,b,B,r)$ **for** b
using *first_abs*
proof (*auto*)
fix b
assume $first\_rel(M,b,B,r)$
with ‹$M(B)$›
have $b\in B$ **using** *first_abs first_is_elem* **by** *simp*
with ‹$M(B)$›
show $M(b)$ **using** *transM*[*OF* ‹$b\in B$›] **by** *simp*
qed
with *assms*
show *?thesis* **unfolding** *minimum_rel_def minimum_def*
by *simp*
qed

end

13.1 Discipline for $\lambda A\ B.\ A \to B$

definition
$is\_function\_space :: [i\Rightarrow o,i,i,i] \Rightarrow o$ **where**
$is\_function\_space(M,A,B,fs) \equiv M(fs) \wedge is\_funspace(M,A,B,fs)$

definition
function_space_rel :: $[i \Rightarrow o,i,i] \Rightarrow i$ **where**
$function\_space\_rel(M,A,B) \equiv THE\ d.\ is\_function\_space(M,A,B,d)$

reldb_rem absolute *Pi*
reldb_add relational *Pi is_function_space*
reldb_add functional *Pi function_space_rel*

abbreviation
function_space_r :: $[i,i \Rightarrow o,i] \Rightarrow i$ $(\langle\_ \to\!\!\!\!-\ \_\rangle\ [61,1,61]\ 60)$ **where**
$A \to^M B \equiv function\_space\_rel(M,A,B)$

abbreviation
function_space_r_set :: $[i,i,i] \Rightarrow i$ $(\langle\_ \to\!\!\!\!-\ \_\rangle\ [61,1,61]\ 60)$ **where**
$function\_space\_r\_set(A,M) \equiv function\_space\_rel(\#\#M,A)$

context *M_Pi*
begin

lemma *is_function_space_uniqueness*:
assumes
$M(r)$ $M(B)$
$is\_function\_space(M,r,B,d)$ $is\_function\_space(M,r,B,d')$
shows
$d{=}d'$
using *assms extensionality_trans*
unfolding *is_function_space_def is_funspace_def*
by *simp*

lemma *is_function_space_witness*:
assumes $M(A)$ $M(B)$
shows $\exists\, d[M].\ is\_function\_space(M,A,B,d)$
proof -
from *assms*
interpret *M_Pi_assumptions M A* $\lambda\_.\ B$
using *Pi_replacement Pi_separation*
by *unfold_locales* (*auto dest*:*transM simp add*:*Sigfun_def*)
have $\forall f[M].\ f \in Pi\_rel(M,A,\ \lambda\_.\ B) \longleftrightarrow f \in A \to B$
using *Pi_rel_char* **by** *simp*
with *assms*
show *?thesis* **unfolding** *is_funspace_def is_function_space_def* **by** *auto*
qed

lemma *is_function_space_closed* :
$is\_function\_space(M,A,B,d) \Longrightarrow M(d)$
unfolding *is_function_space_def* **by** *simp*

— adding closure to simpset and claset
lemma *function_space_rel_closed*[*intro*,*simp*]:

assumes $M(x)$ $M(y)$
shows $M(function\_space\_rel(M,x,y))$
proof -
have $is\_function\_space(M,\ x,\ y,\ THE\ xa.\ is\_function\_space(M,\ x,\ y,\ xa))$
using *assms*
theI[*OF ex1I*[*of is_function_space*(M,x,y)], *OF _ is_function_space_uniqueness*[*of x y*]]
is_function_space_witness
by *auto*
then show *?thesis*
using *assms is_function_space_closed*
unfolding *function_space_rel_def*
by *blast*
qed

lemmas *trans_function_space_rel_closed*[*trans_closed*] = *transM*[*OF _ function_space_rel_closed*]

lemma *function_space_rel_iff*:
assumes $M(x)$ $M(y)$ $M(d)$
shows $is\_function\_space(M,x,y,d) \longleftrightarrow d = function\_space\_rel(M,x,y)$
proof (*intro iffI*)
assume $d = function\_space\_rel(M,x,y)$
moreover
note *assms*
moreover from *this*
obtain e **where** $M(e)$ $is\_function\_space(M,x,y,e)$
using *is_function_space_witness* **by** *blast*
ultimately
show $is\_function\_space(M,\ x,\ y,\ d)$
using *is_function_space_uniqueness*[*of x y*] *is_function_space_witness*
theI[*OF ex1I*[*of is_function_space*(M,x,y)], *OF _ is_function_space_uniqueness*[*of x y*], *of e*]
unfolding *function_space_rel_def*
by *auto*
next
assume $is\_function\_space(M,\ x,\ y,\ d)$
with *assms*
show $d = function\_space\_rel(M,x,y)$
using *is_function_space_uniqueness* **unfolding** *function_space_rel_def*
by (*blast del*:*the_equality intro*:*the_equality*[*symmetric*])
qed

lemma *def_function_space_rel*:
assumes $M(A)$ $M(y)$
shows $function\_space\_rel(M,A,y) = Pi\_rel(M,A,\lambda\_.\ y)$
proof -
from *assms*
interpret *M_Pi_assumptions M A* $\lambda\_.\ y$

```
    using Pi_replacement Pi_separation
    by unfold_locales (auto dest:transM simp add:Sigfun_def)
  from assms
  have x∈function_space_rel(M,A,y) ⟷ x∈Pi_rel(M,A,λ_. y) if M(x) for x
    using that
      function_space_rel_iff[of A y, OF _ _ function_space_rel_closed, of A y]
      def_Pi_rel Pi_rel_char Pow_rel_char
    unfolding is_function_space_def is_funspace_def by (simp add:Pi_def)
  with assms
  show ?thesis — At this point, quoting "trans_rules" doesn't work
    using transM[OF _ function_space_rel_closed, OF _ ‹M(A)› ‹M(y)›]
      transM[OF _ Pi_rel_closed] by blast
qed

lemma function_space_rel_char:
  assumes M(A) M(y)
  shows function_space_rel(M,A,y) = {f ∈ A → y. M(f)}
proof -
  from assms
  interpret M_Pi_assumptions M A λ_. y
    using Pi_replacement Pi_separation
    by unfold_locales (auto dest:transM simp add:Sigfun_def)
  show ?thesis
    using assms def_function_space_rel Pi_rel_char
    by simp
qed

lemma mem_function_space_rel_abs:
  assumes M(A) M(y) M(f)
  shows f ∈ function_space_rel(M,A,y) ⟷ f ∈ A → y
  using assms function_space_rel_char by simp

end




locale M_N_Pi = M:M_Pi + N:M_Pi N for N +
  assumes
    M_imp_N:M(x) ⟹ N(x)
begin

lemma function_space_rel_transfer: M(A) ⟹ M(B) ⟹
                        function_space_rel(M,A,B) ⊆ function_space_rel(N,A,B)
  using M.function_space_rel_char N.function_space_rel_char
  by (auto dest!:M_imp_N)

end
```

abbreviation
is_apply ≡ *fun_apply*
— It is not necessary to perform the Discipline for *is_apply* since it is absolute in this context

13.2 Discipline for *Collect* terms.

We have to isolate the predicate involved and apply the Discipline to it.

definition
injP_rel:: [*i*⇒*o*,*i*,*i*]⇒*o* **where**
injP_rel(*M*,*A*,*f*) ≡ ∀ *w*[*M*]. ∀ *x*[*M*]. ∀ *fw*[*M*]. ∀ *fx*[*M*]. *w*∈*A* ∧ *x*∈*A* ∧
is_apply(*M*,*f*,*w*,*fw*) ∧ *is_apply*(*M*,*f*,*x*,*fx*) ∧ *fw*=*fx*⟶ *w*=*x*

synthesize *injP_rel* **from_definition assuming** *nonempty*

arity_theorem for *injP_rel_fm*

context *M_basic*
begin

— I'm undecided on keeping the relative quantifiers here. Same with *surjP* below. It might relieve from changing *?P*(*?x*) ⟹ ∃ *x*. *?P*(*x*)
(⋀*x*. *?P*(*x*)) ⟹ ∀ *x*. *?P*(*x*) to ⟦*?P*(*?x*); *?M*(*?x*)⟧ ⟹ ∃ *x*[*?M*]. *?P*(*x*)
(⋀*x*. *?M*(*x*) ⟹ *?P*(*x*)) ⟹ ∀ *x*[*?M*]. *?P*(*x*) in some proofs. I wonder if this escalates well. Assuming that all terms appearing in the "def_" theorem are in *M* and using ⟦*?y* ∈ *?x*; *M*(*?x*)⟧ ⟹ *M*(*?y*), it might do.
lemma *def_injP_rel*:
assumes
M(*A*) *M*(*f*)
shows
injP_rel(*M*,*A*,*f*) ⟷ (∀ *w*[*M*]. ∀ *x*[*M*]. *w*∈*A* ∧ *x*∈*A* ∧ *f*‘*w*=*f*‘*x* ⟶ *w*=*x*)
using *assms* **unfolding** *injP_rel_def* **by** *simp*

end

13.3 Discipline for *inj*

term *function_space_rel*

ML‹
@{*term is_function_space*}
›

definition
is_inj :: [*i*⇒*o*,*i*,*i*,*i*]⇒*o* **where**
is_inj(*M*,*A*,*B*,*I*) ≡ *M*(*I*) ∧ (∃ *F*[*M*]. *is_function_space*(*M*,*A*,*B*,*F*) ∧

is_Collect(*M*,*F*,*injP_rel*(*M*,*A*),*I*))

declare *typed_function_iff_sats Collect_iff_sats* [*iff_sats*]

synthesize *is_funspace* **from_definition assuming** *nonempty*
arity_theorem for *is_funspace_fm*

synthesize *is_function_space* **from_definition assuming** *nonempty*
notation *is_function_space_fm* (‹·_ → _ *is* _·›)

arity_theorem for *is_function_space_fm*

synthesize *is_inj* **from_definition assuming** *nonempty*
notation *is_inj_fm* (‹·*inj*′(_,_′) *is* _·›)

arity_theorem intermediate for *is_inj_fm*

lemma *arity_is_inj_fm*[*arity*]:
A ∈ *nat* ⟹
B ∈ *nat* ⟹ *I* ∈ *nat* ⟹ *arity*(*is_inj_fm*(*A*, *B*, *I*)) = *succ*(*A*) ∪ *succ*(*B*) ∪ *succ*(*I*)
using *arity_is_inj_fm*′ **by** *auto*

definition
inj_rel :: [*i*⇒*o*,*i*,*i*] ⇒ *i* (‹*inj*−′(_,_′)›) **where**
inj_rel(*M*,*A*,*B*) ≡ *THE d*. *is_inj*(*M*,*A*,*B*,*d*)

abbreviation
inj_r_set :: [*i*,*i*,*i*] ⇒ *i* (‹*inj*−′(_,_′)›) **where**
inj_r_set(*M*) ≡ *inj_rel*(*##M*)

locale *M_inj* = *M_Pi* +
assumes
injP_separation: *M*(*r*) ⟹ *separation*(*M*,*injP_rel*(*M*, *r*))
begin

lemma *is_inj_uniqueness*:
assumes
M(*r*) *M*(*B*)
is_inj(*M*,*r*,*B*,*d*) *is_inj*(*M*,*r*,*B*,*d*′)
shows
d=*d*′
using *assms function_space_rel_iff extensionality_trans*
unfolding *is_inj_def* **by** *simp*

lemma *is_inj_witness*: *M*(*r*) ⟹ *M*(*B*)⟹ ∃ *d*[*M*]. *is_inj*(*M*,*r*,*B*,*d*)
using *injP_separation function_space_rel_iff*
unfolding *is_inj_def* **by** *simp*

lemma *is_inj_closed* :
is_inj(*M*,*x*,*y*,*d*) $\Longrightarrow$ *M*(*d*)
unfolding *is_inj_def* **by** *simp*

lemma *inj_rel_closed*[*intro*,*simp*]:
assumes *M*(*x*) *M*(*y*)
shows *M*(*inj_rel*(*M*,*x*,*y*))
proof -
have *is_inj*(*M*, *x*, *y*, *THE xa*. *is_inj*(*M*, *x*, *y*, *xa*))
using *assms*
theI[*OF ex1I*[*of is_inj*(*M*,*x*,*y*)], *OF _ is_inj_uniqueness*[*of x y*]]
is_inj_witness
by *auto*
then show *?thesis*
using *assms is_inj_closed*
unfolding *inj_rel_def*
by *blast*
qed

lemmas *trans_inj_rel_closed*[*trans_closed*] = *transM*[*OF _ inj_rel_closed*]

lemma *inj_rel_iff*:
assumes *M*(*x*) *M*(*y*) *M*(*d*)
shows *is_inj*(*M*,*x*,*y*,*d*) $\longleftrightarrow$ *d* = *inj_rel*(*M*,*x*,*y*)
proof (*intro iffI*)
assume *d* = *inj_rel*(*M*,*x*,*y*)
moreover
note *assms*
moreover from *this*
obtain *e* **where** *M*(*e*) *is_inj*(*M*,*x*,*y*,*e*)
using *is_inj_witness* **by** *blast*
ultimately
show *is_inj*(*M*, *x*, *y*, *d*)
using *is_inj_uniqueness*[*of x y*] *is_inj_witness*
theI[*OF ex1I*[*of is_inj*(*M*,*x*,*y*)], *OF _ is_inj_uniqueness*[*of x y*], *of e*]
unfolding *inj_rel_def*
by *auto*
next
assume *is_inj*(*M*, *x*, *y*, *d*)
with *assms*
show *d* = *inj_rel*(*M*,*x*,*y*)
using *is_inj_uniqueness* **unfolding** *inj_rel_def*
by (*blast del*:*the_equality intro*:*the_equality*[*symmetric*])
qed

lemma *def_inj_rel*:
assumes *M*(*A*) *M*(*B*)
shows *inj_rel*(*M*,*A*,*B*) =

$\{f \in function\_space\_rel(M,A,B).\ \forall w[M].\ \forall x[M].\ w{\in}A \wedge x{\in}A \wedge f`w = f`x \longrightarrow w{=}x\}$
(**is** _ = *Collect*(_,*?P*))
proof -
from *assms*
have *inj_rel*(*M*,*A*,*B*) ⊆ *function_space_rel*(*M*,*A*,*B*)
using *inj_rel_iff*[*of A B inj_rel*(*M*,*A*,*B*)] *function_space_rel_iff*
unfolding *is_inj_def* **by** *auto*
moreover from *assms*
have *f* ∈ *inj_rel*(*M*,*A*,*B*) ⟹ *?P*(*f*) **for** *f*
using *inj_rel_iff*[*of A B inj_rel*(*M*,*A*,*B*)] *function_space_rel_iff*
def_injP_rel transM[*OF _ function_space_rel_closed*, *OF _* ‹*M*(*A*)› ‹*M*(*B*)›]
unfolding *is_inj_def* **by** *auto*
moreover from *assms*
have *f* ∈ *function_space_rel*(*M*,*A*,*B*) ⟹ *?P*(*f*) ⟹ *f* ∈ *inj_rel*(*M*,*A*,*B*) **for** *f*
using *inj_rel_iff*[*of A B inj_rel*(*M*,*A*,*B*)] *function_space_rel_iff*
def_injP_rel transM[*OF _ function_space_rel_closed*, *OF _* ‹*M*(*A*)› ‹*M*(*B*)›]
unfolding *is_inj_def* **by** *auto*
ultimately
show *?thesis* **by** *force*
qed

lemma *inj_rel_char*:
assumes *M*(*A*) *M*(*B*)
shows *inj_rel*(*M*,*A*,*B*) = {*f* ∈ *inj*(*A*,*B*). *M*(*f*)}
proof -
from *assms*
interpret *M_Pi_assumptions M A* λ_. *B*
using *Pi_replacement Pi_separation*
by *unfold_locales* (*auto dest*:*transM simp add*:*Sigfun_def*)
from *assms*
show *?thesis*
using *def_inj_rel*[*OF assms*] *def_function_space_rel*[*OF assms*]
transM[*OF _* ‹*M*(*A*)›] *Pi_rel_char*
unfolding *inj_def*
by *auto*
qed

end

locale *M_N_inj* = *M*:*M_inj* + *N*:*M_inj N* **for** *N* +
assumes
M_imp_N:*M*(*x*) ⟹ *N*(*x*)
begin

lemma *inj_rel_transfer*: *M*(*A*) ⟹ *M*(*B*) ⟹ *inj_rel*(*M*,*A*,*B*) ⊆ *inj_rel*(*N*,*A*,*B*)
using *M.inj_rel_char N.inj_rel_char*
by (*auto dest*!:*M_imp_N*)

end

definition
surjP_rel:: $[i \Rightarrow o,i,i,i] \Rightarrow o$ **where**
surjP_rel$(M,A,B,f) \equiv$
$\forall\, y[M].\ \exists\, x[M].\ \exists\, fx[M].\ y{\in}B \longrightarrow x{\in}A \wedge is\_apply(M,f,x,fx) \wedge fx{=}y$

synthesize *surjP_rel* **from_definition assuming** *nonempty*

context *M_basic*
begin

lemma *def_surjP_rel*:
assumes
$M(A)\ M(B)\ M(f)$
shows
$surjP\_rel(M,A,B,f) \longleftrightarrow (\forall\, y[M].\ \exists\, x[M].\ y{\in}B \longrightarrow x{\in}A \wedge f`x{=}y)$
using *assms* **unfolding** *surjP_rel_def* **by** *auto*

end

13.4 Discipline for *surj*

definition
is_surj :: $[i \Rightarrow o,i,i,i] \Rightarrow o$ **where**
$is\_surj(M,A,B,I) \equiv M(I) \wedge (\exists\, F[M].\ is\_function\_space(M,A,B,F) \wedge$
$is\_Collect(M,F,surjP\_rel(M,A,B),I))$

synthesize *is_surj* **from_definition assuming** *nonempty*
notation *is_surj_fm* $(\langle\cdot surj'(\_,\_')\ is\ \_\cdot\rangle)$

definition
surj_rel :: $[i \Rightarrow o,i,i] \Rightarrow i$ $(\langle surj\text{-}'(\_,\_')\rangle)$ **where**
$surj\_rel(M,A,B) \equiv THE\ d.\ is\_surj(M,A,B,d)$

abbreviation
surj_r_set :: $[i,i,i] \Rightarrow i$ $(\langle surj\text{-}'(\_,\_')\rangle)$ **where**
$surj\_r\_set(M) \equiv surj\_rel(\#\#M)$

locale *M_surj* = *M_Pi* +
assumes
surjP_separation: $M(A) \Longrightarrow M(B) \Longrightarrow separation(M,\lambda x.\ surjP\_rel(M,A,B,x))$
begin

lemma *is_surj_uniqueness*:
  assumes
    $M(r)$ $M(B)$
    is_surj(M,r,B,d) *is_surj*(M,r,B,d')
  shows
    $d{=}d'$
  using *assms function_space_rel_iff extensionality_trans*
  unfolding *is_surj_def* **by** *simp*

lemma *is_surj_witness*: $M(r) \Longrightarrow M(B) \Longrightarrow \exists\, d[M].$ *is_surj*(M,r,B,d)
  using *surjP_separation function_space_rel_iff*
  unfolding *is_surj_def* **by** *simp*

lemma *is_surj_closed* :
 is_surj(M,x,y,d) $\Longrightarrow$ $M(d)$
  unfolding *is_surj_def* **by** *simp*

lemma *surj_rel_closed*[*intro*,*simp*]:
  assumes $M(x)$ $M(y)$
  shows M(*surj_rel*(M,x,y))
proof -
  have *is_surj*(M, x, y, *THE xa. is_surj*(M, x, y, xa))
    using *assms*
        theI[*OF ex1I*[*of is_surj*(M,x,y)], *OF _ is_surj_uniqueness*[*of x y*]]
        is_surj_witness
    by *auto*
  then show *?thesis*
    using *assms is_surj_closed*
    unfolding *surj_rel_def*
    by *blast*
qed

lemmas *trans_surj_rel_closed*[*trans_closed*] = *transM*[*OF _ surj_rel_closed*]

lemma *surj_rel_iff*:
  assumes $M(x)$ $M(y)$ $M(d)$
  shows *is_surj*(M,x,y,d) $\longleftrightarrow$ d = *surj_rel*(M,x,y)
proof (*intro iffI*)
  assume d = *surj_rel*(M,x,y)
  moreover
  note *assms*
  moreover from *this*
  obtain e **where** $M(e)$ *is_surj*(M,x,y,e)
    using *is_surj_witness* **by** *blast*
  ultimately
  show *is_surj*(M, x, y, d)
    using *is_surj_uniqueness*[*of x y*] *is_surj_witness*
      theI[*OF ex1I*[*of is_surj*(M,x,y)], *OF _ is_surj_uniqueness*[*of x y*], *of e*]

unfolding *surj_rel_def*
by *auto*
next
assume *is_surj*(M, x, y, d)
with *assms*
show d = *surj_rel*(M,x,y)
using *is_surj_uniqueness* **unfolding** *surj_rel_def*
by (*blast del*:*the_equality intro*:*the_equality*[*symmetric*])
qed

lemma *def_surj_rel*:
assumes $M(A)$ $M(B)$
shows *surj_rel*(M,A,B) =
$\{f \in$ *function_space_rel*(M,A,B). $\forall\, y[M].\ \exists\, x[M].\ y{\in}B \longrightarrow x{\in}A \wedge f`x{=}y\ \}$
(**is** _ = *Collect*(_,*?P*))
proof -
from *assms*
have *surj_rel*(M,A,B) $\subseteq$ *function_space_rel*(M,A,B)
using *surj_rel_iff*[*of A B surj_rel*(M,A,B)] *function_space_rel_iff*
unfolding *is_surj_def* **by** *auto*
moreover from *assms*
have $f \in$ *surj_rel*(M,A,B) $\Longrightarrow$ *?P*(f) **for** f
using *surj_rel_iff*[*of A B surj_rel*(M,A,B)] *function_space_rel_iff*
def_surjP_rel transM[*OF* _ *function_space_rel_closed*, *OF* _ ‹$M(A)$› ‹$M(B)$›]
unfolding *is_surj_def* **by** *auto*
moreover from *assms*
have $f \in$ *function_space_rel*(M,A,B) $\Longrightarrow$ *?P*(f) $\Longrightarrow$ $f \in$ *surj_rel*(M,A,B) **for** f
using *surj_rel_iff*[*of A B surj_rel*(M,A,B)] *function_space_rel_iff*
def_surjP_rel transM[*OF* _ *function_space_rel_closed*, *OF* _ ‹$M(A)$› ‹$M(B)$›]
unfolding *is_surj_def* **by** *auto*
ultimately
show *?thesis* **by** *force*
qed

lemma *surj_rel_char*:
assumes $M(A)$ $M(B)$
shows *surj_rel*(M,A,B) = $\{f \in$ *surj*(A,B). $M(f)\}$
proof -
from *assms*
interpret *M_Pi_assumptions M A* λ_. *B*
using *Pi_replacement Pi_separation*
by *unfold_locales* (*auto dest*:*transM simp add*:*Sigfun_def*)
from *assms*
show *?thesis*
using *def_surj_rel*[*OF assms*] *def_function_space_rel*[*OF assms*]
transM[*OF* _ ‹$M(A)$›] *transM*[*OF* _ ‹$M(B)$›] *Pi_rel_char*
unfolding *surj_def*
by *auto*
qed

end

locale *M_N_surj* = *M*:*M_surj* + *N*:*M_surj N* **for** *N* +
  assumes
    M_imp_N:$M(x) \Longrightarrow N(x)$
begin

lemma *surj_rel_transfer*: $M(A) \Longrightarrow M(B) \Longrightarrow$ *surj_rel*(*M*,*A*,*B*) $\subseteq$ *surj_rel*(*N*,*A*,*B*)
  using *M.surj_rel_char N.surj_rel_char*
  by (*auto dest*!:*M_imp_N*)

end

definition
  is_Int :: [*i*⇒*o*,*i*,*i*,*i*]⇒*o* **where**
  is_Int(*M*,*A*,*B*,*I*) ≡ $M(I) \wedge (\forall x[M].\ x \in I \longleftrightarrow x \in A \wedge x \in B)$

reldb_rem relational *inter*
reldb_add absolute relational *ZF_Base.Int is_Int*

synthesize *is_Int* **from_definition assuming** *nonempty*
notation *is_Int_fm* (‹_ ∩ _ *is* _›)

context *M_basic*
begin

lemma *is_Int_closed* :
 is_Int(*M*,*A*,*B*,*I*) $\Longrightarrow M(I)$
  unfolding *is_Int_def* **by** *simp*

lemma *is_Int_abs*:
  assumes
    $M(A)$ $M(B)$ $M(I)$
  shows
    is_Int(*M*,*A*,*B*,*I*) $\longleftrightarrow I = A \cap B$
  using *assms transM*[*OF* _ ‹$M(B)$›] *transM*[*OF* _ ‹$M(I)$›]
  unfolding *is_Int_def* **by** *blast*

lemma *is_Int_uniqueness*:
  assumes
    $M(r)$ $M(B)$
    is_Int(*M*,*r*,*B*,*d*) *is_Int*(*M*,*r*,*B*,*d*′)
  shows
    d=*d*′

proof -
  have $M(d)$ **and** $M(d')$
    using *assms is_Int_closed* **by** *simp+*
  then show *?thesis*
    using *assms is_Int_abs* **by** *simp*
qed

Note: $[\![M(?A); M(?B)]\!] \Longrightarrow M(?A \cap ?B)$ already in *ZF-Constructible.Relative.*

end

13.5 Discipline for *bij*

reldb_add functional *inj inj_rel*
reldb_add functional relational *inj_rel is_inj*
reldb_add functional *surj surj_rel*
reldb_add functional relational *surj_rel is_surj*
relativize functional *bij bij_rel* **external**
relationalize *bij_rel is_bij*

synthesize *is_bij* **from_definition assuming** *nonempty*
notation *is_bij_fm* ($\langle$"*bij*'(_,_') *is* _")

abbreviation
  bij_r_class :: $[i \Rightarrow o, i, i] \Rightarrow i$ ($\langle$*bij*–'(_,_')$\rangle$) **where**
  bij_r_class $\equiv$ *bij_rel*

abbreviation
  bij_r_set :: $[i,i,i] \Rightarrow i$ ($\langle$*bij*–'(_,_')$\rangle$) **where**
  bij_r_set(M) $\equiv$ *bij_rel*($\#\#M$)

locale *M_Perm* = *M_Pi* + *M_inj* + *M_surj*
begin

lemma *is_bij_closed* : *is_bij*(M,f,y,d) $\Longrightarrow$ $M(d)$
  unfolding *is_bij_def* **using** *is_Int_closed is_inj_witness is_surj_witness* **by** *auto*

lemma *bij_rel_closed*[*intro*,*simp*]:
  assumes $M(x)$ $M(y)$
  shows M(*bij_rel*(M,x,y))
  unfolding *bij_rel_def*
  using *assms Int_closed surj_rel_closed inj_rel_closed*
  by *auto*

lemmas *trans_bij_rel_closed*[*trans_closed*] = *transM*[*OF* _ *bij_rel_closed*]

lemma *bij_rel_iff*:
assumes *M*(*x*) *M*(*y*) *M*(*d*)
shows *is_bij*(*M*,*x*,*y*,*d*) ⟷ *d* = *bij_rel*(*M*,*x*,*y*)
unfolding *is_bij_def bij_rel_def*
using *assms surj_rel_iff inj_rel_iff is_Int_abs*
by *auto*

lemma *def_bij_rel*:
assumes *M*(*A*) *M*(*B*)
shows *bij_rel*(*M*,*A*,*B*) = *inj_rel*(*M*,*A*,*B*) ∩ *surj_rel*(*M*,*A*,*B*)
using *assms bij_rel_iff inj_rel_iff surj_rel_iff*
is_Int_abs— For absolute terms, "_abs" replaces "_iff". Also, in this case "_closed" is in the simpset.
unfolding *is_bij_def* **by** *simp*

lemma *bij_rel_char*:
assumes *M*(*A*) *M*(*B*)
shows *bij_rel*(*M*,*A*,*B*) = {*f* ∈ *bij*(*A*,*B*). *M*(*f*)}
using *assms def_bij_rel inj_rel_char surj_rel_char*
unfolding *bij_def*— Unfolding this might be a pattern already
by *auto*

end

locale *M_N_Perm* = *M_N_Pi* + *M_N_inj* + *M_N_surj* + *M*:*M_Perm* + *N*:*M_Perm N*

begin

lemma *bij_rel_transfer*: *M*(*A*) ⟹ *M*(*B*) ⟹ *bij_rel*(*M*,*A*,*B*) ⊆ *bij_rel*(*N*,*A*,*B*)
using *M*.*bij_rel_char N*.*bij_rel_char*
by (*auto dest*!:*M_imp_N*)

end

13.6 Discipline for (≈)

relativize functional *eqpoll eqpoll_rel* **external**
relationalize *eqpoll_rel is_eqpoll*

synthesize *is_eqpoll* **from_definition assuming** *nonempty*
arity_theorem for *is_eqpoll_fm*
notation *is_eqpoll_fm* (‹_ ≈ _›)

context *M_Perm* **begin**

is_iff_rel for *eqpoll*
  using *bij_rel_iff* **unfolding** *is_eqpoll_def eqpoll_rel_def* **by** *simp*

end

abbreviation
  eqpoll_r :: [*i*,*i*⇒*o*,*i*] => *o* (‹_ ≈– _› [*51*,*1*,*51*] *50*) **where**
  A ≈^{M} *B* ≡ *eqpoll_rel*(*M*,*A*,*B*)

abbreviation
  eqpoll_r_set :: [*i*,*i*,*i*] ⇒ *o* (‹_ ≈– _› [*51*,*1*,*51*] *50*) **where**
  eqpoll_r_set(*A*,*M*) ≡ *eqpoll_rel*(*##M*,*A*)

context *M_Perm*
begin

lemma *def_eqpoll_rel*:
  assumes
    M(*A*) *M*(*B*)
  shows
    eqpoll_rel(*M*,*A*,*B*) ⟷ (∃*f*[*M*]. *f* ∈ *bij_rel*(*M*,*A*,*B*))
  using *assms bij_rel_iff*
  unfolding *eqpoll_rel_def* **by** *simp*

end

context *M_N_Perm*
begin

lemma *eqpoll_rel_transfer*: **assumes** *A* ≈^{M} *B* *M*(*A*) *M*(*B*)
  shows *A* ≈^{N} *B*
proof -
  note *assms*
  moreover from *this*
  obtain *f* **where** *f* ∈ *bij*^{M}(*A*,*B*) *N*(*f*)
    using *M.def_eqpoll_rel* **by** (*auto dest*!:*M_imp_N*)
  moreover from *calculation*
  have *f* ∈ *bij*^{N}(*A*,*B*)
    using *bij_rel_transfer* **by** (*auto*)
  ultimately
  show *?thesis*
    using *N.def_eqpoll_rel* **by** (*blast dest*!:*M_imp_N*)
qed

end

13.7 Discipline for $(\lesssim)$

relativize functional *lepoll lepoll_rel* **external**
relationalize *lepoll_rel is_lepoll*

synthesize *is_lepoll* **from_definition assuming** *nonempty*
notation *is_lepoll_fm* ($\langle\cdot\_ \lesssim \_\cdot\rangle$)
arity_theorem for *is_lepoll_fm*

context *M_inj* **begin**

is_iff_rel for *lepoll*
  using *inj_rel_iff* **unfolding** *is_lepoll_def lepoll_rel_def* **by** *simp*

end

abbreviation
  lepoll_r :: $[i,i\Rightarrow o,i]$ => *o* ($\langle\_ \lesssim^{-} \_\rangle$ [*51*,*1*,*51*] *50*) **where**
  $A \lesssim^{M} B \equiv$ *lepoll_rel*(M,A,B)

abbreviation
  lepoll_r_set :: $[i,i,i] \Rightarrow o$ ($\langle\_ \lesssim^{-} \_\rangle$ [*51*,*1*,*51*] *50*) **where**
  lepoll_r_set(A,M) $\equiv$ *lepoll_rel*($\#\#M$,A)

context *M_Perm*
begin

lemma *def_lepoll_rel*:
  assumes
    $M(A)$ $M(B)$
  shows
    lepoll_rel(M,A,B) $\longleftrightarrow$ ($\exists f[M].\ f \in$ *inj_rel*(M,A,B))
  using *assms inj_rel_iff*
  unfolding *lepoll_rel_def* **by** *simp*

end

context *M_N_Perm*
begin

lemma *lepoll_rel_transfer*: **assumes** $A \lesssim^{M} B$ $M(A)$ $M(B)$
  shows $A \lesssim^{N} B$
proof -
  note *assms*
  moreover from *this*
  obtain f **where** $f \in inj^{M}(A,B)$ $N(f)$
    using *M.def_lepoll_rel* **by** (*auto dest*!:*M_imp_N*)
  moreover from *calculation*
  have $f \in inj^{N}(A,B)$

using *inj_rel_transfer* **by** (*auto*)
ultimately
show *?thesis*
using *N.def_lepoll_rel* **by** (*blast dest*!:*M_imp_N*)
qed

end

13.8 Discipline for (≺)

relativize functional *lesspoll lesspoll_rel* **external**
relationalize *lesspoll_rel is_lesspoll*

synthesize *is_lesspoll* **from_definition assuming** *nonempty*
notation *is_lesspoll_fm* (‹·_ ≺ _·›)
arity_theorem for *is_lesspoll_fm*

context *M_Perm* **begin**

is_iff_rel for *lesspoll*
using *is_lepoll_iff is_eqpoll_iff*
unfolding *is_lesspoll_def lesspoll_rel_def* **by** *simp*

end

abbreviation
lesspoll_r :: [*i*,*i*⇒*o*,*i*] => *o* (‹_ ≺⁻ _› [*51*,*1*,*51*] *50*) **where**
$A \prec^M B$ ≡ *lesspoll_rel*(*M*,*A*,*B*)

abbreviation
lesspoll_r_set :: [*i*,*i*,*i*] ⇒ *o* (‹_ ≺⁻ _› [*51*,*1*,*51*] *50*) **where**
lesspoll_r_set(*A*,*M*) ≡ *lesspoll_rel*(##*M*,*A*)

Since *lesspoll_rel* is defined as a propositional combination of older terms, there is no need for a separate "def" theorem for it.

Note that *lesspoll_rel* is neither Σ_1^{ZF} nor Π_1^{ZF}, so there is no "transfer" theorem for it.

end

14 Relativization of the cumulative hierarchy

theory *Relative_Univ*
imports
ZF-Constructible.Rank
Internalizations
Recursion_Thms

begin

declare (**in** *M_trivial*) *powerset_abs*[*simp*]

lemma *Collect_inter_Transset*:
assumes
Transset(*M*) $b \in M$
shows
$\{x{\in}b\ .\ P(x)\} = \{x{\in}b\ .\ P(x)\} \cap M$
using *assms* **unfolding** *Transset_def*
by (*auto*)

lemma (**in** *M_trivial*) *family_union_closed*: $[\![$*strong_replacement*(M, $\lambda x\ y.\ y = f(x)$); $M(A)$; $\forall x{\in}A.\ M(f(x))]\!]$
$\Longrightarrow M(\bigcup x{\in}A.\ f(x))$
using *RepFun_closed* **..**

lemma (**in** *M_trivial*) *family_union_closed'*: $[\![$*strong_replacement*(M, $\lambda x\ y.\ x{\in}A \wedge y = f(x)$); $M(A)$; $\forall x{\in}A.\ M(f(x))]\!]$
$\Longrightarrow M(\bigcup x{\in}A.\ f(x))$
using *RepFun_closed2*
by *simp*

definition
HVfrom :: $[i{\Rightarrow}o,i,i,i] \Rightarrow i$ **where**
HVfrom(M,A,x,f) $\equiv A \cup (\bigcup y{\in}x.\ \{a{\in}Pow(f`y).\ M(a)\})$

definition
is_powapply :: $[i{\Rightarrow}o,i,i,i] \Rightarrow o$ **where**
is_powapply(M,f,y,z) $\equiv M(z) \wedge (\exists fy[M].$ *fun_apply*(M,f,y,fy) $\wedge$ *powerset*(M,fy,z))

lemma *is_powapply_closed*: *is_powapply*(M,f,y,z) $\Longrightarrow M(z)$
unfolding *is_powapply_def* **by** *simp*

definition
is_HVfrom :: $[i{\Rightarrow}o,i,i,i,i] \Rightarrow o$ **where**
is_HVfrom(M,A,x,f,h) $\equiv \exists U[M].\ \exists R[M].$ *union*(M,A,U,h)
$\wedge$ *big_union*(M,R,U) $\wedge$ *is_Replace*(M,x,*is_powapply*(M,f),R)

definition
is_Vfrom :: $[i{\Rightarrow}o,i,i,i] \Rightarrow o$ **where**
is_Vfrom(M,A,i,V) $\equiv$ *is_transrec*(M,*is_HVfrom*(M,A),i,V)

definition

is_Vset :: $[i \Rightarrow o,i,i] \Rightarrow o$ **where**
$is\_Vset(M,i,V) \equiv \exists\, z[M].\ empty(M,z) \wedge is\_Vfrom(M,z,i,V)$

14.1 Formula synthesis

schematic_goal *sats_is_powapply_fm_auto*:
assumes
$f \in nat\ y \in nat\ z \in nat\ env \in list(A)\ 0 \in A$
shows
$is\_powapply(\#\#A,nth(f,\ env),nth(y,\ env),nth(z,\ env))$
$\longleftrightarrow sats(A,?ipa\_fm(f,y,z),env)$
unfolding *is_powapply_def powerset_def subset_def*
using *nth_closed assms*
by (*simp*) (*rule sep_rules* | *simp*)+

schematic_goal *is_powapply_iff_sats*:
assumes
$nth(f,env) = ff\ nth(y,env) = yy\ nth(z,env) = zz\ 0 \in A$
$f \in nat\ \ y \in nat\ z \in nat\ env \in list(A)$
shows
$is\_powapply(\#\#A,ff,yy,zz) \longleftrightarrow sats(A,\ ?is\_one\_fm(a,r),\ env)$
unfolding $\langle nth(f,env) = ff\rangle[symmetric]\ \langle nth(y,env) = yy\rangle[symmetric]$
$\langle nth(z,env) = zz\rangle[symmetric]$
by (*rule sats_is_powapply_fm_auto*(*1*); *simp add*:*assms*)

definition
Hrank :: $[i,i] \Rightarrow i$ **where**
$Hrank(x,f) = (\bigcup y{\in}x.\ succ(f`y))$

definition
PHrank :: $[i \Rightarrow o,i,i,i] \Rightarrow o$ **where**
$PHrank(M,f,y,z) \equiv M(z) \wedge (\exists\, fy[M].\ fun\_apply(M,f,y,fy) \wedge successor(M,fy,z))$

definition
is_Hrank :: $[i \Rightarrow o,i,i,i] \Rightarrow o$ **where**
$is\_Hrank(M,x,f,hc) \equiv (\exists\, R[M].\ big\_union(M,R,hc) \wedge is\_Replace(M,x,PHrank(M,f),R))$

definition
rrank :: $i \Rightarrow i$ **where**
$rrank(a) \equiv Memrel(eclose(\{a\}))^{+}$

lemma (**in** *M_eclose*) *wf_rrank* : $M(x) \Longrightarrow wf(rrank(x))$
unfolding *rrank_def* **using** *wf_trancl*[*OF wf_Memrel*] **.**

lemma (**in** *M_eclose*) *trans_rrank* : $M(x) \Longrightarrow trans(rrank(x))$
unfolding *rrank_def* **using** *trans_trancl* **.**

lemma (**in** *M_eclose*) *relation_rrank* : $M(x) \Longrightarrow relation(rrank(x))$
unfolding *rrank_def* **using** *relation_trancl* **.**

lemma (**in** *M_eclose*) *rrank_in_M* : $M(x) \Longrightarrow M(rrank(x))$
unfolding *rrank_def* **by** *simp*

14.2 Absoluteness results

locale *M_eclose_pow* = *M_eclose* +
assumes
power_ax : *power_ax*(M) **and**
powapply_replacement : $M(f) \Longrightarrow$ *strong_replacement*(M,*is_powapply*(M,f))
and
HVfrom_replacement : $[\![\ M(i)\ ;\ M(A)\ ]\!] \Longrightarrow$
transrec_replacement(M,*is_HVfrom*(M,A),i) **and**
PHrank_replacement : $M(f) \Longrightarrow$ *strong_replacement*(M,*PHrank*(M,f)) **and**
is_Hrank_replacement : $M(x) \Longrightarrow$ *wfrec_replacement*(M,*is_Hrank*(M),*rrank*(x))

begin

lemma *is_powapply_abs*: $[\![M(f);\ M(y)]\!] \Longrightarrow$ *is_powapply*(M,f,y,z) $\longleftrightarrow$ $M(z) \wedge z = \{x \in Pow(f`y).\ M(x)\}$
unfolding *is_powapply_def* **by** *simp*

lemma $[\![M(A);\ M(x);\ M(f);\ M(h)\ ]\!] \Longrightarrow$
is_HVfrom(M,A,x,f,h) $\longleftrightarrow$
$(\exists\, R[M].\ h = A \cup \bigcup R \wedge$ *is_Replace*(M, x,$\lambda x\ y.\ y = \{x \in Pow(f\ `\ x)\ .\ M(x)\}$, R))
using *is_powapply_abs* **unfolding** *is_HVfrom_def* **by** *auto*

lemma *Replace_is_powapply*:
assumes
$M(R)$ $M(A)$ $M(f)$
shows
is_Replace(M, A, *is_powapply*(M, f), R) $\longleftrightarrow$ $R =$ *Replace*(A,*is_powapply*(M,f))
proof -
have *univalent*(M,A,*is_powapply*(M,f))
using ‹$M(A)$› ‹$M(f)$› **unfolding** *univalent_def is_powapply_def* **by** *simp*
moreover
have $\bigwedge x\ y.\ [\![\ x{\in}A;\ $ *is_powapply*(M,f,x,y) $]\!] \Longrightarrow M(y)$
using ‹$M(A)$› ‹$M(f)$› **unfolding** *is_powapply_def* **by** *simp*
ultimately
show *?thesis* **using** ‹$M(A)$› ‹$M(R)$› *Replace_abs* **by** *simp*
qed

lemma *powapply_closed*:
$[\![\ M(y)\ ;\ M(f)\ ]\!] \Longrightarrow M(\{x \in Pow(f\ `\ y)\ .\ M(x)\})$
using *apply_closed power_ax* **unfolding** *power_ax_def* **by** *simp*

lemma *RepFun_is_powapply*:
  assumes
    $M(R)$ $M(A)$ $M(f)$
  shows
  $Replace(A,is\_powapply(M,f)) = RepFun(A,\lambda y.\{x\in Pow(f`y).\ M(x)\})$
proof -
  have $\{y\ .\ x\in A,\ M(y)\wedge y=\{x\in Pow(f\ `\ x)\ .\ M(x)\}\} = \{y\ .\ x\in A,\ y=\{x$
$\in Pow(f\ `\ x)\ .\ M(x)\}\}$
    using *assms powapply_closed transM*[*of* _ *A*] **by** *blast*
  also
  have $\ ... = \{\{x\in Pow(f\ `\ y)\ .\ M(x)\}\ .\ y\in A\}$ **by** *auto*
  finally
  show *?thesis* **using** *assms is_powapply_abs transM*[*of* _ *A*] **by** *simp*
qed

lemma *RepFun_powapply_closed*:
  assumes
    $M(f)$ $M(A)$
  shows
    $M(Replace(A,is\_powapply(M,f)))$
proof -
  have $univalent(M,A,is\_powapply(M,f))$
    using ‹$M(A)$› ‹$M(f)$› **unfolding** *univalent_def is_powapply_def* **by** *simp*
  moreover
  have $[\![\ x\in A\ ;\ is\_powapply(M,f,x,y)\ ]\!] \Longrightarrow M(y)$ **for** *x y*
    using *assms* **unfolding** *is_powapply_def* **by** *simp*
  ultimately
  show *?thesis* **using** *assms powapply_replacement* **by** *simp*
qed

lemma *Union_powapply_closed*:
  assumes
    $M(x)$ $M(f)$
  shows
    $M(\bigcup y\in x.\ \{a\in Pow(f`y).\ M(a)\})$
proof -
  have $M(\{a\in Pow(f`y).\ M(a)\})$ **if** $y\in x$ **for** *y*
    using *that assms transM*[*of* _ *x*] *powapply_closed* **by** *simp*
  then
  have $M(\{\{a\in Pow(f`y).\ M(a)\}.\ y\in x\})$
    using *assms transM*[*of* _ *x*] *RepFun_powapply_closed RepFun_is_powapply*
by *simp*
  then show *?thesis* **using** *assms* **by** *simp*
qed

lemma *relation2_HVfrom*: $M(A) \Longrightarrow relation2(M,is\_HVfrom(M,A),HVfrom(M,A))$
    unfolding *is_HVfrom_def HVfrom_def relation2_def*
    using *Replace_is_powapply RepFun_is_powapply*
        Union_powapply_closed RepFun_powapply_closed **by** *auto*

lemma *HVfrom_closed* :
  $M(A) \Longrightarrow \forall\, x[M].\ \forall\, g[M].\ function(g) \longrightarrow M(HVfrom(M,A,x,g))$
  unfolding *HVfrom_def* **using** *Union_powapply_closed* **by** *simp*

lemma *transrec_HVfrom*:
  assumes $M(A)$
  shows $Ord(i) \Longrightarrow \{x{\in}Vfrom(A,i).\ M(x)\} = transrec(i,HVfrom(M,A))$
proof (*induct rule*:*trans_induct*)
  case (*step i*)
  have $Vfrom(A,i) = A \cup (\bigcup y{\in}i.\ Pow((\lambda x{\in}i.\ Vfrom(A,\ x))\ `\ y))$
    using *def_transrec*[*OF Vfrom_def*, *of A i*] **by** *simp*
  then
  have $Vfrom(A,i) = A \cup (\bigcup y{\in}i.\ Pow(Vfrom(A,\ y)))$
    by *simp*
  then
  have $\{x{\in}Vfrom(A,i).\ M(x)\} = \{x{\in}A.\ M(x)\} \cup (\bigcup y{\in}i.\ \{x{\in}Pow(Vfrom(A,\ y)).$
$M(x)\})$
    by *auto*
  with ‹$M(A)$›
  have $\{x{\in}Vfrom(A,i).\ M(x)\} = A \cup (\bigcup y{\in}i.\ \{x{\in}Pow(Vfrom(A,\ y)).\ M(x)\})$
    by (*auto intro*:*transM*)
  also
  have $... = A \cup (\bigcup y{\in}i.\ \{x{\in}Pow(\{z{\in}Vfrom(A,y).\ M(z)\}).\ M(x)\})$
  proof -
   have $\{x{\in}Pow(Vfrom(A,\ y)).\ M(x)\} = \{x{\in}Pow(\{z{\in}Vfrom(A,y).\ M(z)\}).\ M(x)\}$
     if $y{\in}i$ **for** y **by** (*auto intro*:*transM*)
    then
    show *?thesis* **by** *simp*
  qed
  also from *step*
  have $... = A \cup (\bigcup y{\in}i.\ \{x{\in}Pow(transrec(y,\ HVfrom(M,\ A))).\ M(x)\})$ **by** *auto*
  also
  have $... = transrec(i,\ HVfrom(M,\ A))$
   using *def_transrec*[*of* $\lambda y.\ transrec(y,\ HVfrom(M,\ A))\ HVfrom(M,\ A)$ *i*,*symmetric*]

    unfolding *HVfrom_def* **by** *simp*
  finally
  show *?case* **.**
qed

lemma *Vfrom_abs*: $[\![\ M(A);\ M(i);\ M(V);\ Ord(i)\ ]\!] \Longrightarrow is\_Vfrom(M,A,i,V) \longleftrightarrow$
$V = \{x{\in}Vfrom(A,i).\ M(x)\}$
  unfolding *is_Vfrom_def*
  using *relation2_HVfrom HVfrom_closed HVfrom_replacement*
    transrec_abs[*of is_HVfrom*(M,A) *i HVfrom*(M,A)] *transrec_HVfrom* **by** *simp*

lemma *Vfrom_closed*: $[\![\ M(A);\ M(i);\ Ord(i)\ ]\!] \Longrightarrow M(\{x{\in}Vfrom(A,i).\ M(x)\})$
  unfolding *is_Vfrom_def*

using *relation2_HVfrom HVfrom_closed HVfrom_replacement*
transrec_closed[*of is_HVfrom*(*M*,*A*) *i HVfrom*(*M*,*A*)] *transrec_HVfrom* **by** *simp*

lemma *Vset_abs*: ⟦ *M*(*i*); *M*(*V*); *Ord*(*i*) ⟧ ⟹ *is_Vset*(*M*,*i*,*V*) ⟷ *V* = {*x*∈*Vset*(*i*). *M*(*x*)}
using *Vfrom_abs* **unfolding** *is_Vset_def* **by** *simp*

lemma *Vset_closed*: ⟦ *M*(*i*); *Ord*(*i*) ⟧ ⟹ *M*({*x*∈*Vset*(*i*). *M*(*x*)})
using *Vfrom_closed* **unfolding** *is_Vset_def* **by** *simp*

lemma *Hrank_trancl*:*Hrank*(*y*, *restrict*(*f*,*Memrel*(*eclose*({*x*}))-‘‘{*y*}))
= *Hrank*(*y*, *restrict*(*f*,(*Memrel*(*eclose*({*x*}))^+)-‘‘{*y*}))
unfolding *Hrank_def*
using *restrict_trans_eq* **by** *simp*

lemma *rank_trancl*: *rank*(*x*) = *wfrec*(*rrank*(*x*), *x*, *Hrank*)
proof -
have *rank*(*x*) = *wfrec*(*Memrel*(*eclose*({*x*})), *x*, *Hrank*)
(**is** _ = *wfrec*(*?r*,_,_))
unfolding *rank_def transrec_def Hrank_def* **by** *simp*
also
have ... = *wftrec*(*?r*^+, *x*, λ*y f*. *Hrank*(*y*, *restrict*(*f*,*?r*-‘‘{*y*})))
unfolding *wfrec_def* **..**
also
have ... = *wftrec*(*?r*^+, *x*, λ*y f*. *Hrank*(*y*, *restrict*(*f*,(*?r*^+)-‘‘{*y*})))
using *Hrank_trancl* **by** *simp*
also
have ... = *wfrec*(*?r*^+, *x*, *Hrank*)
unfolding *wfrec_def* **using** *trancl_eq_r*[*OF relation_trancl trans_trancl*] **by** *simp*
finally
show *?thesis* **unfolding** *rrank_def* **.**
qed

lemma *univ_PHrank* : ⟦ *M*(*z*) ; *M*(*f*) ⟧ ⟹ *univalent*(*M*,*z*,*PHrank*(*M*,*f*))
unfolding *univalent_def PHrank_def* **by** *simp*

lemma *PHrank_abs* :
⟦ *M*(*f*) ; *M*(*y*) ⟧ ⟹ *PHrank*(*M*,*f*,*y*,*z*) ⟷ *M*(*z*) ∧ *z* = *succ*(*f*‘*y*)
unfolding *PHrank_def* **by** *simp*

lemma *PHrank_closed* : *PHrank*(*M*,*f*,*y*,*z*) ⟹ *M*(*z*)
unfolding *PHrank_def* **by** *simp*

lemma *Replace_PHrank_abs*:
assumes
M(*z*) *M*(*f*) *M*(*hr*)

shows
is_Replace(M,z,PHrank(M,f),hr) $\longleftrightarrow$ *hr* = *Replace(z,PHrank(M,f))*
proof -
have $\bigwedge x\ y.\ [\![x{\in}z;\ PHrank(M,f,x,y)\ ]\!] \Longrightarrow M(y)$
using ‹*M*(*z*)› ‹*M*(*f*)› **unfolding** *PHrank_def* **by** *simp*
then
show *?thesis* **using** ‹*M*(*z*)› ‹*M*(*hr*)› ‹*M*(*f*)› *univ_PHrank Replace_abs* **by** *simp*
qed

lemma *RepFun_PHrank*:
assumes
M(*R*) *M*(*A*) *M*(*f*)
shows
Replace(*A*,*PHrank*(*M*,*f*)) = *RepFun*(*A*,$\lambda y.$ *succ*(*f*‘*y*))
proof -
have {*z* . *y* ∈ *A*, *M*(*z*) ∧ *z* = *succ*(*f*‘*y*)} = {*z* . *y* ∈ *A*, *z* = *succ*(*f*‘*y*)}
using *assms PHrank_closed transM*[*of* _ *A*] **by** *blast*
also
have ... = {*succ*(*f*‘*y*) . *y* ∈ *A*} **by** *auto*
finally
show *?thesis* **using** *assms PHrank_abs transM*[*of* _ *A*] **by** *simp*
qed

lemma *RepFun_PHrank_closed* :
assumes
M(*f*) *M*(*A*)
shows
M(*Replace*(*A*,*PHrank*(*M*,*f*)))
proof -
have $[\![\ x{\in}A\ ;\ PHrank(M,f,x,y)\ ]\!] \Longrightarrow M(y)$ **for** *x y*
using *assms* **unfolding** *PHrank_def* **by** *simp*
with *univ_PHrank*
show *?thesis* **using** *assms PHrank_replacement* **by** *simp*
qed

lemma *relation2_Hrank* :
relation2(*M*,*is_Hrank*(*M*),*Hrank*)
unfolding *is_Hrank_def Hrank_def relation2_def*
using *Replace_PHrank_abs RepFun_PHrank RepFun_PHrank_closed* **by** *auto*

lemma *Union_PHrank_closed*:
assumes
M(*x*) *M*(*f*)
shows
M($\bigcup y{\in}x.$ *succ*(*f*‘*y*))
proof -
have *M*(*succ*(*f*‘*y*)) **if** *y*∈*x* **for** *y*
using *that assms transM*[*of* _ *x*] **by** *simp*

then
have $M(\{succ(f`y).\ y{\in}x\})$
using *assms transM*[*of* _ *x*] *RepFun_PHrank_closed RepFun_PHrank* **by** *simp*
then show *?thesis* **using** *assms* **by** *simp*
qed

lemma *is_Hrank_closed* :
$M(A) \Longrightarrow \forall\, x[M].\ \forall\, g[M].\ function(g) \longrightarrow M(Hrank(x,g))$
unfolding *Hrank_def* **using** *RepFun_PHrank_closed Union_PHrank_closed* **by** *simp*

lemma *rank_closed*: $M(a) \Longrightarrow M(rank(a))$
unfolding *rank_trancl*
using *relation2_Hrank is_Hrank_closed is_Hrank_replacement*
wf_rrank relation_rrank trans_rrank rrank_in_M
trans_wfrec_closed[*of rrank*(*a*) *a is_Hrank*(*M*)] **by** *simp*

lemma *M_into_Vset*:
assumes $M(a)$
shows $\exists\, i[M].\ \exists\, V[M].\ ordinal(M,i) \wedge is\_Vfrom(M,0,i,V) \wedge a{\in}V$
proof -
let *?i*=$succ(rank(a))$
from *assms*
have $a{\in}\{x{\in}Vfrom(0,?i).\ M(x)\}$ (**is** $a{\in}?V$)
using *Vset_Ord_rank_iff* **by** *simp*
moreover from *assms*
have $M(?i)$
using *rank_closed* **by** *simp*
moreover
note ‹$M(a)$›
moreover from *calculation*
have $M(?V)$
using *Vfrom_closed* **by** *simp*
moreover from *calculation*
have $ordinal(M,?i) \wedge is\_Vfrom(M,\ 0,\ ?i,\ ?V) \wedge a \in ?V$
using *Ord_rank Vfrom_abs* **by** *simp*
ultimately
show *?thesis* **by** *blast*
qed

end
end

15 Renaming of variables in internalized formulas

theory *Renaming*
imports

Nat_Miscellanea
ZF_Miscellanea
ZF-Constructible.Formula
begin

15.1 Renaming of free variables

definition
union_fun :: [*i*,*i*,*i*,*i*] ⇒ *i* **where**
union_fun(*f*,*g*,*m*,*p*) ≡ λ*j* ∈ *m* ∪ *p* . *if j*∈*m then f*‘*j else g*‘*j*

lemma *union_fun_type*:
assumes *f* ∈ *m* → *n*
g ∈ *p* → *q*
shows *union_fun*(*f*,*g*,*m*,*p*) ∈ *m* ∪ *p* → *n* ∪ *q*
proof -
let *?h*=*union_fun*(*f*,*g*,*m*,*p*)
have
D: *?h*‘*x* ∈ *n* ∪ *q* **if** *x* ∈ *m* ∪ *p* **for** *x*
proof (*cases x* ∈ *m*)
case *True*
then have
x ∈ *m* ∪ *p* **by** *simp*
with ‹*x*∈*m*›
have *?h*‘*x* = *f*‘*x*
unfolding *union_fun_def beta* **by** *simp*
with ‹*f* ∈ *m* → *n*› ‹*x*∈*m*›
have *?h*‘*x* ∈ *n* **by** *simp*
then show *?thesis* **..**
next
case *False*
with ‹*x* ∈ *m* ∪ *p*›
have *x* ∈ *p*
by *auto*
with ‹*x*∉*m*›
have *?h*‘*x* = *g*‘*x*
unfolding *union_fun_def* **using** *beta* **by** *simp*
with ‹*g* ∈ *p* → *q*› ‹*x*∈*p*›
have *?h*‘*x* ∈ *q* **by** *simp*
then show *?thesis* **..**
qed
have *A*:*function*(*?h*) **unfolding** *union_fun_def* **using** *function_lam* **by** *simp*
have *x*∈ (*m* ∪ *p*) × (*n* ∪ *q*) **if** *x*∈ *?h* **for** *x*
using *that lamE*[*of x m* ∪ *p* _ *x* ∈ (*m* ∪ *p*) × (*n* ∪ *q*)] *D* **unfolding** *union_fun_def*
by *auto*
then have *B*:*?h* ⊆ (*m* ∪ *p*) × (*n* ∪ *q*) **..**
have *m* ∪ *p* ⊆ *domain*(*?h*)
unfolding *union_fun_def* **using** *domain_lam* **by** *simp*

```
  with A B
  show ?thesis using  Pi_iff [THEN iffD2] by simp
qed

lemma union_fun_action :
  assumes
    env ∈ list(M)
    env′ ∈ list(M)
    length(env) = m ∪ p
    ∀ i . i ∈ m ⟶  nth(f‘i,env′) = nth(i,env)
    ∀ j . j ∈ p ⟶ nth(g‘j,env′) = nth(j,env)
  shows ∀ i . i ∈ m ∪ p ⟶
        nth(i,env) = nth(union_fun(f,g,m,p)‘i,env′)
proof -
  let ?h = union_fun(f,g,m,p)
  have nth(x, env) = nth(?h‘x,env′) if x ∈ m ∪ p for x
    using that
  proof (cases x∈m)
    case True
    with ‹x∈m›
    have ?h‘x = f‘x
      unfolding union_fun_def  beta by simp
    with assms ‹x∈m›
    have nth(x,env) = nth(?h‘x,env′) by simp
    then show ?thesis .
  next
    case False
    with ‹x ∈ m ∪ p›
    have
      x ∈ p x∉m  by auto
    then
    have ?h‘x = g‘x
      unfolding union_fun_def beta by simp
    with assms ‹x∈p›
    have nth(x,env) = nth(?h‘x,env′) by simp
    then show ?thesis .
  qed
  then show ?thesis by simp
qed


lemma id_fn_type :
  assumes n ∈ nat
  shows id(n) ∈ n → n
  unfolding id_def using ‹n∈nat› by simp

lemma id_fn_action:
  assumes n ∈ nat env∈list(M)
  shows ⋀ j . j < n ⟹ nth(j,env) = nth(id(n)‘j,env)
```

proof -
  show *nth(j,env) = nth(id(n)‘j,env)* **if** *j < n* **for** *j* **using** *that ‹n∈nat› ltD* **by** *simp*
qed

definition
  rsum :: [i,i,i,i,i] ⇒ i **where**
  rsum(f,g,m,n,p) ≡ λj ∈ m#+p . if j<m then f‘j else (g‘(j#-m))#+n

lemma *sum_inl*:
  assumes *m ∈ nat n∈nat*
    f ∈ m→n x ∈ m
  shows *rsum(f,g,m,n,p)‘x = f‘x*
proof -
  from *‹m∈nat›*
  have *m≤m#+p*
    using *add_le_self[of m]* **by** *simp*
  with *assms*
  have *x∈m#+p*
    using *ltI[of x m] lt_trans2[of x m m#+p] ltD* **by** *simp*
  from *assms*
  have *x<m*
    using *ltI* **by** *simp*
  with *‹x∈m#+p›*
  show *?thesis* **unfolding** *rsum_def* **by** *simp*
qed

lemma *sum_inr*:
  assumes *m ∈ nat n∈nat p∈nat*
    g∈p→q m ≤ x x < m#+p
  shows *rsum(f,g,m,n,p)‘x = g‘(x#-m)#+n*
proof -
  from *assms*
  have *x∈nat*
    using *in_n_in_nat[of m#+p] ltD*
    by *simp*
  with *assms*
  have *¬ x<m*
    using *not_lt_iff_le[THEN iffD2]* **by** *simp*
  from *assms*
  have *x∈m#+p*
    using *ltD* **by** *simp*
  with *‹¬ x<m›*
  show *?thesis* **unfolding** *rsum_def* **by** *simp*
qed

lemma *sum_action* :

```
  assumes m ∈ nat n∈nat p∈nat q∈nat
    f ∈ m→n g∈p→q
    env ∈ list(M)
    env′ ∈ list(M)
    env1 ∈ list(M)
    env2 ∈ list(M)
    length(env) = m
    length(env1) = p
    length(env′) = n
    ⋀ i . i < m ⟹ nth(i,env) = nth(f‘i,env′)
    ⋀ j. j < p ⟹ nth(j,env1) = nth(g‘j,env2)
  shows ∀ i . i < m#+p ⟶
          nth(i,env@env1) = nth(rsum(f,g,m,n,p)‘i,env′@env2)
proof -
  let ?h = rsum(f,g,m,n,p)
  from ‹m∈nat› ‹n∈nat› ‹q∈nat›
  have m≤m#+p n≤n#+q q≤n#+q
    using add_le_self[of m]  add_le_self2[of n q] by simp_all
  from ‹p∈nat›
  have p = (m#+p)#-m using diff_add_inverse2 by simp
  have nth(x, env @ env1) = nth(?h‘x,env′@env2) if x<m#+p for x
  proof (cases x<m)
    case True
    then
    have 2: ?h‘x= f‘x x∈m f‘x ∈ n x∈nat
      using assms sum_inl ltD apply_type[of f m _ x] in_n_in_nat by simp_all
    with ‹x<m› assms
    have f‘x < n f‘x<length(env′)  f‘x∈nat
      using ltI in_n_in_nat by simp_all
    with 2 ‹x<m› assms
    have nth(x,env@env1) = nth(x,env)
      using nth_append[OF ‹env∈list(M)›] ‹x∈nat› by simp
    also
    have
      ... = nth(f‘x,env′)
      using 2 ‹x<m› assms by simp
    also
    have ... = nth(f‘x,env′@env2)
      using nth_append[OF ‹env′∈list(M)›] ‹f‘x<length(env′)› ‹f‘x ∈nat› by simp
    also
    have ... = nth(?h‘x,env′@env2)
      using 2 by simp
    finally
    have nth(x, env @ env1) = nth(?h‘x,env′@env2) .
    then show ?thesis .
  next
    case False
    have x∈nat
      using that in_n_in_nat[of m#+p x] ltD ‹p∈nat› ‹m∈nat› by simp
```

```
    with ‹length(env) = m›
    have m≤x length(env) ≤ x
      using not_lt_iff_le ‹m∈nat› ‹¬x<m› by simp_all
    with ‹¬x<m› ‹length(env) = m›
    have 2 : ?h`x= g`(x#-m)#+n  ¬ x <length(env)
      unfolding rsum_def
      using  sum_inr that beta ltD by simp_all
    from assms ‹x∈nat› ‹p=m#+p#-m›
    have x#-m < p
      using diff_mono[OF _ _ _ ‹x<m#+p› ‹m≤x›] by simp
    then have x#-m∈p using ltD by simp
    with ‹g∈p→q›
    have g`(x#-m) ∈ q  by simp
    with ‹q∈nat› ‹length(env') = n›
    have g`(x#-m) < q g`(x#-m)∈nat using ltI in_n_in_nat by simp_all
    with ‹q∈nat› ‹n∈nat›
   have (g`(x#-m))#+n <n#+q n ≤ g`(x#-m)#+n ¬ g`(x#-m)#+n < length(env')
      using add_lt_mono1[of g`(x#-m) _ n,OF _ ‹q∈nat›]
        add_le_self2[of n] ‹length(env') = n›
      by simp_all
    from assms ‹¬ x < length(env)› ‹length(env) = m›
    have nth(x,env @ env1) = nth(x#-m,env1)
      using nth_append[OF ‹env∈list(M)› ‹x∈nat›] by simp
    also
    have ... = nth(g`(x#-m),env2)
      using assms ‹x#-m < p› by simp
    also
    have ... = nth((g`(x#-m)#+n)#-length(env'),env2)
      using  ‹length(env') = n›
        diff_add_inverse2 ‹g`(x#-m)∈nat›
      by simp
    also
    have ... = nth((g`(x#-m)#+n),env'@env2)
    using nth_append[OF ‹env'∈list(M)›] ‹n∈nat› ‹¬ g`(x#-m)#+n < length(env')›
      by simp
    also
    have ... = nth(?h`x,env'@env2)
      using 2 by simp
    finally
    have nth(x, env @ env1) = nth(?h`x,env'@env2) .
    then show ?thesis .
  qed
  then show ?thesis by simp
qed

lemma sum_type :
  assumes m ∈ nat n∈nat p∈nat q∈nat
    f ∈ m→n g∈p→q
  shows rsum(f,g,m,n,p) ∈ (m#+p) → (n#+q)
```

```
proof -
  let ?h = rsum(f,g,m,n,p)
  from ‹m∈nat› ‹n∈nat› ‹q∈nat›
  have m≤m#+p n≤n#+q q≤n#+q
    using add_le_self[of m]  add_le_self2[of n q] by simp_all
  from ‹p∈nat›
  have p = (m#+p)#-m using diff_add_inverse2 by simp
  {fix x
    assume 1: x∈m#+p x<m
    with 1 have ?h‘x= f‘x x∈m
      using assms sum_inl ltD by simp_all
    with ‹f∈m→n›
    have ?h‘x ∈ n by simp
    with ‹n∈nat› have ?h‘x < n using ltI by simp
    with ‹n≤n#+q›
    have ?h‘x < n#+q using lt_trans2 by simp
    then
    have ?h‘x ∈ n#+q  using ltD by simp
  }
  then have 1:?h‘x ∈ n#+q if x∈m#+p x<m for x using that .
  {fix x
    assume 1: x∈m#+p m≤x
   then have x<m#+p x∈nat using ltI in_n_in_nat[of m#+p] ltD by simp_all
    with 1
    have 2 : ?h‘x= g‘(x#-m)#+n
      using assms sum_inr ltD by simp_all
    from assms ‹x∈nat› ‹p=m#+p#-m›
    have x#-m < p using diff_mono[OF _ _ _ ‹x<m#+p› ‹m≤x›] by simp
    then have x#-m∈p using ltD by simp
    with ‹g∈p→q›
    have g‘(x#-m) ∈ q  by simp
    with ‹q∈nat› have g‘(x#-m) < q using ltI by simp
    with ‹q∈nat›
     have (g‘(x#-m))#+n <n#+q using add_lt_mono1[of g‘(x#-m) _ n,OF _
‹q∈nat›] by simp
    with 2
    have ?h‘x ∈ n#+q  using ltD by simp
  }
  then have 2:?h‘x ∈ n#+q if x∈m#+p m≤x for x using that .
  have
    D: ?h‘x ∈ n#+q if x∈m#+p for x
    using that
  proof (cases x<m)
    case True
    then show ?thesis using 1 that by simp
  next
    case False
     with ‹m∈nat› have m≤x using not_lt_iff_le that in_n_in_nat[of m#+p]
by simp
```

```
    then show ?thesis using 2 that by simp
  qed
  have A:function(?h) unfolding rsum_def using function_lam by simp
  have  x∈ (m #+ p) × (n #+ q) if x∈ ?h for x
     using that lamE[of x m#+p _ x ∈ (m #+ p) × (n #+ q)] D unfolding
rsum_def
    by auto
  then have B:?h ⊆ (m #+ p) × (n #+ q) ..
  have m #+ p ⊆ domain(?h)
    unfolding rsum_def using domain_lam by simp
  with A B
  show ?thesis using  Pi_iff [THEN iffD2] by simp
qed

lemma sum_type_id :
  assumes
    f ∈ length(env)→length(env')
    env ∈ list(M)
    env' ∈ list(M)
    env1 ∈ list(M)
  shows
    rsum(f,id(length(env1)),length(env),length(env'),length(env1)) ∈
        (length(env)#+length(env1)) → (length(env')#+length(env1))
  using assms length_type id_fn_type sum_type
  by simp

lemma sum_type_id_aux2 :
  assumes
    f ∈ m→n
    m ∈ nat n ∈ nat
    env1 ∈ list(M)
  shows
    rsum(f,id(length(env1)),m,n,length(env1)) ∈
        (m#+length(env1)) → (n#+length(env1))
  using assms id_fn_type sum_type
  by auto

lemma sum_action_id :
  assumes
    env ∈ list(M)
    env' ∈ list(M)
    f ∈ length(env)→length(env')
    env1 ∈ list(M)
    ⋀ i . i < length(env) ⟹ nth(i,env) = nth(f‘i,env')
  shows ⋀ i . i < length(env)#+length(env1) ⟹
      nth(i,env@env1) = nth(rsum(f,id(length(env1)),length(env),length(env'),length(env1))‘i,env'@env1)
proof -
  from assms
  have length(env)∈nat (is ?m ∈ _) by simp
```

```
  from assms have length(env')∈nat (is ?n ∈ _) by simp
  from assms have length(env1)∈nat (is ?p ∈ _) by simp
  note lenv = id_fn_action[OF ‹?p∈nat› ‹env1∈list(M)›]
  note lenv_ty = id_fn_type[OF ‹?p∈nat›]
  {
    fix i
    assume i < length(env)#+length(env1)
    have nth(i,env@env1) = nth(rsum(f,id(length(env1)),?m,?n,?p)‘i,env'@env1)
      using sum_action[OF ‹?m∈nat› ‹?n∈nat› ‹?p∈nat› ‹?p∈nat› ‹f∈?m→?n›
          lenv_ty ‹env∈list(M)› ‹env'∈list(M)›
          ‹env1∈list(M)› ‹env1∈list(M)› _
          _ _  assms(5) lenv
          ] ‹i<?m#+length(env1)› by simp
  }
  then show ⋀ i . i < ?m#+length(env1) ⟹
          nth(i,env@env1) = nth(rsum(f,id(?p),?m,?n,?p)‘i,env'@env1) by simp
qed

lemma sum_action_id_aux :
  assumes
    f ∈ m→n
    env ∈ list(M)
    env' ∈ list(M)
    env1 ∈ list(M)
    length(env) = m
    length(env') = n
    length(env1) = p
    ⋀ i . i < m ⟹ nth(i,env) = nth(f‘i,env')
  shows ⋀ i . i < m#+length(env1) ⟹
      nth(i,env@env1) = nth(rsum(f,id(length(env1)),m,n,length(env1))‘i,env'@env1)
  using assms length_type id_fn_type sum_action_id
  by auto


definition
  sum_id :: [i,i] ⇒ i where
  sum_id(m,f) ≡ rsum(λx∈1.x,f,1,1,m)

lemma sum_id0 : m∈nat⟹sum_id(m,f)‘0 = 0
  by(unfold sum_id_def,subst sum_inl,auto)

lemma sum_idS : p∈nat ⟹ q∈nat ⟹ f∈p→q ⟹ x ∈ p ⟹ sum_id(p,f)‘(succ(x))
= succ(f‘x)
  by(subgoal_tac x∈nat,unfold sum_id_def,subst sum_inr,
      simp_all add:ltI,simp_all add: app_nm in_n_in_nat)

lemma sum_id_tc_aux :
  p ∈ nat ⟹  q ∈ nat ⟹ f ∈ p → q ⟹ sum_id(p,f) ∈ 1#+p → 1#+q
  by (unfold sum_id_def,rule sum_type,simp_all)
```

lemma *sum_id_tc* :
n ∈ *nat* ⟹ *m* ∈ *nat* ⟹ *f* ∈ *n* → *m* ⟹ *sum_id*(*n*,*f*) ∈ *succ*(*n*) → *succ*(*m*)
by(*rule ssubst*[*of succ*(*n*) → *succ*(*m*) *1*#+*n* → *1*#+*m*],
simp,*rule sum_id_tc_aux*,*simp_all*)

15.2 Renaming of formulas

consts *ren* :: *i*⇒*i*
primrec
ren(*Member*(*x*,*y*)) =
(λ *n* ∈ *nat* . λ *m* ∈ *nat*. λ*f* ∈ *n* → *m*. *Member* (*f*‘*x*, *f*‘*y*))

ren(*Equal*(*x*,*y*)) =
(λ *n* ∈ *nat* . λ *m* ∈ *nat*. λ*f* ∈ *n* → *m*. *Equal* (*f*‘*x*, *f*‘*y*))

ren(*Nand*(*p*,*q*)) =
(λ *n* ∈ *nat* . λ *m* ∈ *nat*. λ*f* ∈ *n* → *m*. *Nand* (*ren*(*p*)‘*n*‘*m*‘*f*, *ren*(*q*)‘*n*‘*m*‘*f*))

ren(*Forall*(*p*)) =
(λ *n* ∈ *nat* . λ *m* ∈ *nat*. λ*f* ∈ *n* → *m*. *Forall* (*ren*(*p*)‘*succ*(*n*)‘*succ*(*m*)‘*sum_id*(*n*,*f*)))

lemma *arity_meml* : *l* ∈ *nat* ⟹ *Member*(*x*,*y*) ∈ *formula* ⟹ *arity*(*Member*(*x*,*y*)) ≤ *l* ⟹ *x* ∈ *l*
by(*simp*,*rule subsetD*,*rule le_imp_subset*,*assumption*,*simp*)
lemma *arity_memr* : *l* ∈ *nat* ⟹ *Member*(*x*,*y*) ∈ *formula* ⟹ *arity*(*Member*(*x*,*y*)) ≤ *l* ⟹ *y* ∈ *l*
by(*simp*,*rule subsetD*,*rule le_imp_subset*,*assumption*,*simp*)
lemma *arity_eql* : *l* ∈ *nat* ⟹ *Equal*(*x*,*y*) ∈ *formula* ⟹ *arity*(*Equal*(*x*,*y*)) ≤ *l* ⟹ *x* ∈ *l*
by(*simp*,*rule subsetD*,*rule le_imp_subset*,*assumption*,*simp*)
lemma *arity_eqr* : *l* ∈ *nat* ⟹ *Equal*(*x*,*y*) ∈ *formula* ⟹ *arity*(*Equal*(*x*,*y*)) ≤ *l* ⟹ *y* ∈ *l*
by(*simp*,*rule subsetD*,*rule le_imp_subset*,*assumption*,*simp*)
lemma *nand_ar1* : *p* ∈ *formula* ⟹ *q*∈*formula* ⟹*arity*(*p*) ≤ *arity*(*Nand*(*p*,*q*))
by (*simp*,*rule Un_upper1_le*,*simp*+)
lemma *nand_ar2* : *p* ∈ *formula* ⟹ *q*∈*formula* ⟹*arity*(*q*) ≤ *arity*(*Nand*(*p*,*q*))
by (*simp*,*rule Un_upper2_le*,*simp*+)

lemma *nand_ar1D* : *p* ∈ *formula* ⟹ *q*∈*formula* ⟹ *arity*(*Nand*(*p*,*q*)) ≤ *n* ⟹ *arity*(*p*) ≤ *n*
by(*auto simp add*: *le_trans*[*OF Un_upper1_le*[*of arity*(*p*) *arity*(*q*)]])
lemma *nand_ar2D* : *p* ∈ *formula* ⟹ *q*∈*formula* ⟹ *arity*(*Nand*(*p*,*q*)) ≤ *n* ⟹ *arity*(*q*) ≤ *n*
by(*auto simp add*: *le_trans*[*OF Un_upper2_le*[*of arity*(*p*) *arity*(*q*)]])

lemma *ren_tc* : *p* ∈ *formula* ⟹
(⋀ *n m f* . *n* ∈ *nat* ⟹ *m* ∈ *nat* ⟹ *f* ∈ *n*→*m* ⟹ *ren*(*p*)‘*n*‘*m*‘*f* ∈ *formula*)

by (*induct set*:*formula*,*auto simp add*: *app_nm sum_id_tc*)

lemma *arity_ren* :
fixes *p*
assumes $p \in formula$
shows $\bigwedge n\ m\ f\ .\ n \in nat \Longrightarrow m \in nat \Longrightarrow f \in n \to m \Longrightarrow arity(p) \leq n \Longrightarrow arity(ren(p)`n`m`f) \leq m$
using *assms*
proof (*induct set*:*formula*)
case (*Member x y*)
then have $f`x \in m\ f`y \in m$
using *Member assms* **by** (*simp add*: *arity_meml apply_funtype*,*simp add*:*arity_memr apply_funtype*)
then show *?case* **using** *Member* **by** (*simp add*: *Un_least_lt ltI*)
next
case (*Equal x y*)
then have $f`x \in m\ f`y \in m$
using *Equal assms* **by** (*simp add*: *arity_eql apply_funtype*,*simp add*:*arity_eqr apply_funtype*)
then show *?case* **using** *Equal* **by** (*simp add*: *Un_least_lt ltI*)
next
case (*Nand p q*)
then have $arity(p) \leq arity(Nand(p,q))$
$arity(q) \leq arity(Nand(p,q))$
by (*subst nand_ar1*,*simp*,*simp*,*simp*,*subst nand_ar2*,*simp*+)
then have $arity(p) \leq n$
and $arity(q) \leq n$ **using** *Nand*
by (*rule_tac j*=$arity(Nand(p,q))$ **in** *le_trans*,*simp*,*simp*)+
then have $arity(ren(p)`n`m`f) \leq m$ **and** $arity(ren(q)`n`m`f) \leq m$
using *Nand* **by** *auto*
then show *?case* **using** *Nand* **by** (*simp add*:*Un_least_lt*)
next
case (*Forall p*)
from *Forall* **have** $succ(n) \in nat$ $succ(m) \in nat$ **by** *auto*
from *Forall* **have** *2*: $sum\_id(n,f) \in succ(n) \to succ(m)$ **by** (*simp add*:*sum_id_tc*)
from *Forall* **have** *3*:$arity(p) \leq succ(n)$ **by** (*rule_tac n*=$arity(p)$ **in** *natE*,*simp*+)
then have $arity(ren(p)`succ(n)`succ(m)`sum\_id(n,f)) \leq succ(m)$ **using**
Forall ‹$succ(n) \in nat$› ‹$succ(m) \in nat$› *2* **by** *force*
then show *?case* **using** *Forall 2 3 ren_tc arity_type pred_le* **by** *auto*
qed

lemma *arity_forallE* : $p \in formula \Longrightarrow m \in nat \Longrightarrow arity(Forall(p)) \leq m \Longrightarrow arity(p) \leq succ(m)$
by(*rule_tac n*=$arity(p)$ **in** *natE*,*erule arity_type*,*simp*+)

lemma *env_coincidence_sum_id* :
assumes $m \in nat\ n \in nat$
$\varrho \in list(A)\ \varrho' \in list(A)$

```
    f ∈ n → m
    ⋀ i . i < n ⟹ nth(i,ϱ) = nth(f‘i,ϱ′)
    a ∈ A j ∈ succ(n)
  shows nth(j,Cons(a,ϱ)) = nth(sum_id(n,f)‘j,Cons(a,ϱ′))
proof -
  let ?g=sum_id(n,f)
  have succ(n) ∈ nat using ‹n∈nat› by simp
  then have j ∈ nat using ‹j∈succ(n)› in_n_in_nat by blast
  then have nth(j,Cons(a,ϱ)) = nth(?g‘j,Cons(a,ϱ′))
  proof (cases rule:natE[OF ‹j∈nat›])
    case 1
    then show ?thesis using assms sum_id0 by simp
  next
    case (2 i)
    with ‹j∈succ(n)› have succ(i)∈succ(n) by simp
    with ‹n∈nat› have i ∈ n using nat_succD assms by simp
    have f‘i∈m using ‹f∈n→m› apply_type ‹i∈n› by simp
    then have f‘i ∈ nat using in_n_in_nat ‹m∈nat› by simp
    have nth(succ(i),Cons(a,ϱ)) = nth(i,ϱ) using ‹i∈nat› by simp
    also have ... = nth(f‘i,ϱ′) using assms ‹i∈n› ltI by simp
    also have ... = nth(succ(f‘i),Cons(a,ϱ′)) using ‹f‘i∈nat› by simp
    also have ... = nth(?g‘succ(i),Cons(a,ϱ′))
      using assms sum_idS[OF ‹n∈nat› ‹m∈nat› ‹f∈n→m› ‹i ∈ n›] cases by simp
    finally have nth(succ(i),Cons(a,ϱ)) = nth(?g‘succ(i),Cons(a,ϱ′)) .
    then show ?thesis using ‹j=succ(i)› by simp
  qed
  then show ?thesis .
qed

lemma sats_iff_sats_ren :
  assumes φ ∈ formula
  shows ⟦ n ∈ nat ; m ∈ nat ; ϱ ∈ list(M) ; ϱ′ ∈ list(M) ; f ∈ n → m ;
          arity(φ) ≤ n ;
          ⋀ i . i < n ⟹ nth(i,ϱ) = nth(f‘i,ϱ′) ⟧ ⟹
        sats(M,φ,ϱ) ⟷ sats(M,ren(φ)‘n‘m‘f,ϱ′)
  using ‹φ ∈ formula›
proof(induct φ arbitrary:n m ϱ ϱ′ f)
  case (Member x y)
 have ren(Member(x,y))‘n‘m‘f = Member(f‘x,f‘y) using Member assms arity_type
by force
  moreover
  have x ∈ n using Member arity_meml by simp
  moreover
  have y ∈ n using Member arity_memr by simp
  ultimately
  show ?case using Member ltI by simp
next
  case (Equal x y)
  have ren(Equal(x,y))‘n‘m‘f = Equal(f‘x,f‘y) using Equal assms arity_type by
```

```
force
  moreover
  have x ∈ n using Equal arity_eql by simp
  moreover
  have y ∈ n using Equal arity_eqr by simp
  ultimately show ?case using Equal ltI by simp
next
  case (Nand p q)
  have ren(Nand(p,q))`n`m`f = Nand(ren(p)`n`m`f,ren(q)`n`m`f) using Nand by
simp
  moreover
  have arity(p) ≤ n using Nand nand_ar1D by simp
  moreover from this
  have i ∈ arity(p) ⟹ i ∈ n for i using subsetD[OF le_imp_subset[OF ‹arity(p)
≤ n›]] by simp
  moreover from this
  have i ∈ arity(p) ⟹ nth(i,ϱ) = nth(f`i,ϱ′) for i using Nand ltI by simp
  moreover from this
  have sats(M,p,ϱ) ⟷ sats(M,ren(p)`n`m`f,ϱ′) using ‹arity(p)≤n› Nand by simp
  have arity(q) ≤ n using Nand nand_ar2D by simp
  moreover from this
  have i ∈ arity(q) ⟹ i ∈ n for i using subsetD[OF le_imp_subset[OF ‹arity(q)
≤ n›]] by simp
  moreover from this
  have i ∈ arity(q) ⟹ nth(i,ϱ) = nth(f`i,ϱ′) for i using Nand ltI by simp
  moreover from this
  have sats(M,q,ϱ) ⟷ sats(M,ren(q)`n`m`f,ϱ′) using assms ‹arity(q)≤n› Nand
by simp
  ultimately
  show ?case using Nand by simp
next
  case (Forall p)
  have 0:ren(Forall(p))`n`m`f = Forall(ren(p)`succ(n)`succ(m)`sum_id(n,f))
    using Forall by simp
  have 1:sum_id(n,f) ∈ succ(n) → succ(m) (is ?g ∈ _) using sum_id_tc Forall
by simp
  then have 2: arity(p) ≤ succ(n)
    using Forall le_trans[of _ succ(pred(arity(p)))] succpred_leI by simp
  have succ(n)∈nat succ(m)∈nat using Forall by auto
  then have A:⋀ j .j < succ(n) ⟹ nth(j, Cons(a, ϱ)) = nth(?g`j, Cons(a, ϱ′))
if a∈M for a
    using that env_coincidence_sum_id Forall ltD by force
  have
   sats(M,p,Cons(a,ϱ)) ⟷ sats(M,ren(p)`succ(n)`succ(m)`?g,Cons(a,ϱ′)) if a∈M
for a
  proof -
    have C:Cons(a,ϱ) ∈ list(M) Cons(a,ϱ′)∈list(M) using Forall that by auto
    have sats(M,p,Cons(a,ϱ)) ⟷ sats(M,ren(p)`succ(n)`succ(m)`?g,Cons(a,ϱ′))
     using Forall(2)[OF ‹succ(n)∈nat› ‹succ(m)∈nat› C(1) C(2) 1 2 A[OF ‹a∈M›]]
```

```
by simp
   then show ?thesis .
  qed
  then show ?case using Forall 0 1 2 by simp
qed

end
theory Renaming_Auto
  imports
    Renaming
    ZF.Finite
    ZF.List
    Utils
keywords
  rename :: thy_decl % ML
and
  simple_rename :: thy_decl % ML
and
  src
and
  tgt
abbrevs
  simple_rename =

begin

lemmas app_fun = apply_iff[THEN iffD1]
lemmas nat_succI = nat_succ_iff[THEN iffD2]
ML_file‹Renaming_ML.ml›
ML‹
  open Renaming_ML

  fun renaming_def mk_ren name from to ctxt =
    let val to = to |> Syntax.read_term ctxt
        val from = from |> Syntax.read_term ctxt
        val (tc_lemma,action_lemma,fvs,r) = mk_ren from to ctxt
          val (tc_lemma,action_lemma) = (fix_vars tc_lemma fvs ctxt , fix_vars
action_lemma fvs ctxt)
        val ren_fun_name = Binding.name (name ^ "_fn")
        val ren_fun_def =  Binding.name (name ^ "_fn_def")
        val ren_thm = Binding.name (name ^ "_thm")
    in
      Local_Theory.note   ((ren_thm, []), [tc_lemma,action_lemma]) ctxt |> snd
|>
     Local_Theory.define ((ren_fun_name, NoSyn), ((ren_fun_def, []), r)) |> snd

  end;
›
```

ML‹
local

val ren_parser = Parse.position (Parse.string --
(Parse.$$$ src |-- Parse.string --| Parse.$$$ tgt -- Parse.string));

val _ =
Outer_Syntax.local_theory ***command_keyword***‹*rename*› *ML setup for synthetic definitions*
(ren_parser >> (fn ((name,(from,to)),_) => renaming_def sum_rename name from to))

val _ =
Outer_Syntax.local_theory ***command_keyword***‹*simple_rename*› *ML setup for synthetic definitions*
(ren_parser >> (fn ((name,(from,to)),_) => renaming_def ren_thm name from to))

in
end
›
end

16 Interface between set models and Constructibility

This theory provides an interface between Paulson's relativization results and set models of ZFC. In particular, it is used to prove that the locale *forcing_data* is a sublocale of all relevant locales in ZF-Constructibility (*M_trivial*, *M_basic*, *M_eclose*, etc).

theory *Interface*
imports
Nat_Miscellanea
Relative_Univ
Synthetic_Definition
Arities
Renaming_Auto
Discipline_Function
begin

abbreviation
dec10 :: *i* (*10*) **where** *10* ≡ *succ*(*9*)

abbreviation
dec11 :: *i* (*11*) **where** *11* ≡ *succ*(*10*)

abbreviation
dec12 :: *i* (*12*) **where** *12* ≡ *succ*(*11*)

abbreviation
dec13 :: *i* (*13*) **where** *13* ≡ *succ*(*12*)

abbreviation
dec14 :: *i* (*14*) **where** *14* ≡ *succ*(*13*)

definition
infinity_ax :: (*i* ⇒ *o*) ⇒ *o* **where**
infinity_ax(*M*) ≡
(∃ *I*[*M*]. (∃ *z*[*M*]. *empty*(*M*,*z*) ∧ *z*∈*I*) ∧ (∀ *y*[*M*]. *y*∈*I* ⟶ (∃ *sy*[*M*]. *successor*(*M*,*y*,*sy*) ∧ *sy*∈*I*)))

definition
choice_ax :: (*i*⇒*o*) ⇒ *o* **where**
choice_ax(*M*) ≡ ∀ *x*[*M*]. ∃ *a*[*M*]. ∃ *f*[*M*]. *ordinal*(*M*,*a*) ∧ *surjection*(*M*,*a*,*x*,*f*)

context *M_basic* **begin**

lemma *choice_ax_abs* :
choice_ax(*M*) ⟷ (∀ *x*[*M*]. ∃ *a*[*M*]. ∃ *f*[*M*]. *Ord*(*a*) ∧ *f* ∈ *surj*(*a*,*x*))
unfolding *choice_ax_def*
by (*simp*)

end

definition
wellfounded_trancl :: [*i*=>*o*,*i*,*i*,*i*] => *o* **where**
wellfounded_trancl(*M*,*Z*,*r*,*p*) ≡
∃ *w*[*M*]. ∃ *wx*[*M*]. ∃ *rp*[*M*].
w ∈ *Z* & *pair*(*M*,*w*,*p*,*wx*) & *tran_closure*(*M*,*r*,*rp*) & *wx* ∈ *rp*

lemma *empty_intf* :
infinity_ax(*M*) ⟹
(∃ *z*[*M*]. *empty*(*M*,*z*))
by (*auto simp add*: *empty_def infinity_ax_def*)

lemma *Transset_intf* :
Transset(*M*) ⟹ *y*∈*x* ⟹ *x* ∈ *M* ⟹ *y* ∈ *M*
by (*simp add*: *Transset_def*,*auto*)

locale *M_ZF* =
fixes *M*
assumes
upair_ax: *upair_ax*(##*M*) **and**
Union_ax: *Union_ax*(##*M*) **and**
power_ax: *power_ax*(##*M*) **and**
extensionality:*extensionality*(##*M*) **and**
foundation_ax: *foundation_ax*(##*M*) **and**

infinity_ax: *infinity_ax*(##*M*) **and**
separation_ax: $\varphi \in$ *formula* $\Longrightarrow$ *env* $\in$ *list*(*M*) $\Longrightarrow$
arity(φ) $\leq$ *1* #+ *length*(*env*) $\Longrightarrow$
separation(##*M*,λx. *sats*(*M*,φ,[*x*] @ *env*)) **and**
replacement_ax:$\varphi \in$ *formula* $\Longrightarrow$ *env* $\in$ *list*(*M*) $\Longrightarrow$
arity(φ) $\leq$ *2* #+ *length*(*env*) $\Longrightarrow$
strong_replacement(##*M*,λx *y*. *sats*(*M*,φ,[*x*,*y*] @ *env*))

locale *M_ZF_trans* = *M_ZF* +
assumes
trans_M: *Transset*(*M*)
begin

lemmas *transitivity* = *Transset_intf*[*OF trans_M*]

16.1 Interface with *M_trivial*

lemma *zero_in_M*: *0* $\in$ *M*
proof -
from *infinity_ax*
have ($\exists$ *z*[##*M*]. *empty*(##*M*,*z*))
by (*rule empty_intf*)
then obtain *z* **where**
zm: *empty*(##*M*,*z*) *z*$\in$*M*
by *auto*
then
have *z=0*
using *transitivity empty_def* **by** *auto*
with *zm* **show** *?thesis*
by *simp*
qed

end

locale *M_ZFC* = *M_ZF* +
assumes
choice_ax:*choice_ax*(##*M*)

locale *M_ZFC_trans* = *M_ZF_trans* + *M_ZFC*

sublocale *M_ZF_trans* $\subseteq$ *M_trans* ##*M*
using *transitivity zero_in_M exI*[*of* λx. *x*$\in$*M*]
by *unfold_locales simp_all*

sublocale *M_ZF_trans* $\subseteq$ *M_trivial* ##*M*
using *trans_M upair_ax Union_ax* **by** *unfold_locales*

16.2 Interface with *M_basic*

definition *Intersection* **where**

```
 Intersection(N,B,x) ≡ (∀ y[N]. y∈B ⟶ x∈y)

manual_schematic Inter_fm for Intersection
  unfolding Intersection_def
  by (rule sep_rules | simp)+
synthesize Intersection from_schematic Inter_fm
arity_theorem for Intersection_fm

context M_ZF_trans
begin

lemma inter_sep_intf :
  assumes
    A∈M
  shows
    separation(##M,λx . ∀ y∈M . y∈A ⟶ x∈y)
proof -
  have arity(Intersection_fm(1,0)) = 2 0∈nat 1∈nat
    using arity_Intersection_fm pred_Un_distrib by auto
  then
    have ∀ a∈M. separation(##M, λx. sats(M,Intersection_fm(1,0) , [x, a]))
      using separation_ax Intersection_fm_type
      by simp
  moreover
  have (∀ y∈M . y∈a ⟶ x∈y) ⟷ sats(M,Intersection_fm(1,0),[x,a])
    if a∈M x∈M for a x
    using that Intersection_iff_sats[of 1 [x,a] a 0 x M]
    unfolding Intersection_def by simp
  ultimately
  have ∀ a∈M. separation(##M, λx . ∀ y∈M . y∈a ⟶ x∈y)
    unfolding separation_def by simp
  with ‹A∈M› show ?thesis by simp
qed



schematic_goal diff_fm_auto:
  assumes
    nth(i,env) = x nth(j,env) = B
    i ∈ nat j ∈ nat env ∈ list(A)
  shows
    x∉B ⟷ sats(A,?dfm(i,j),env)
  by (insert assms ; (rule sep_rules | simp)+)

lemma diff_sep_intf :
  assumes
    B∈M
  shows
    separation(##M,λx . x∉B)
```

proof -
obtain *dfm* **where**
fmsats:$\bigwedge$*env*. *env*∈*list*(*M*) ⟹ *nth*(*0*,*env*)∉*nth*(*1*,*env*)
⟷ *sats*(*M*,*dfm*(*0*,*1*),*env*)
and
dfm(*0*,*1*) ∈ *formula*
and
arity(*dfm*(*0*,*1*)) = *2*
using ‹*B*∈*M*› *diff_fm_auto*
by (*simp del*:*FOL_sats_iff add*: *ord_simp_union*)
then
have ∀ *b*∈*M*. *separation*(*##M*, λ*x*. *sats*(*M*,*dfm*(*0*,*1*) , [*x*, *b*]))
using *separation_ax* **by** *simp*
moreover
have *x*∉*b* ⟷ *sats*(*M*,*dfm*(*0*,*1*),[*x*,*b*])
if *b*∈*M x*∈*M* **for** *b x*
using *that fmsats*[*of* [*x*,*b*]] **by** *simp*
ultimately
have ∀ *b*∈*M*. *separation*(*##M*, λ*x* . *x*∉*b*)
unfolding *separation_def* **by** *simp*
with ‹*B*∈*M*› **show** *?thesis* **by** *simp*
qed

schematic_goal *cprod_fm_auto*:
assumes
nth(*i*,*env*) = *z nth*(*j*,*env*) = *B nth*(*h*,*env*) = *C*
i ∈ *nat j* ∈ *nat h* ∈ *nat env* ∈ *list*(*A*)
shows
(∃ *x*∈*A*. *x*∈*B* ∧ (∃ *y*∈*A*. *y*∈*C* ∧ *pair*(*##A*,*x*,*y*,*z*))) ⟷ *sats*(*A*,*?cpfm*(*i*,*j*,*h*),*env*)
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *cartprod_sep_intf* :
assumes
A∈*M*
and
B∈*M*
shows
separation(*##M*,λ*z*. ∃ *x*∈*M*. *x*∈*A* ∧ (∃ *y*∈*M*. *y*∈*B* ∧ *pair*(*##M*,*x*,*y*,*z*)))
proof -
obtain *cpfm* **where**
fmsats:$\bigwedge$*env*. *env*∈*list*(*M*) ⟹
(∃ *x*∈*M*. *x*∈*nth*(*1*,*env*) ∧ (∃ *y*∈*M*. *y*∈*nth*(*2*,*env*) ∧ *pair*(*##M*,*x*,*y*,*nth*(*0*,*env*))))
⟷ *sats*(*M*,*cpfm*(*0*,*1*,*2*),*env*)
and
cpfm(*0*,*1*,*2*) ∈ *formula*
and
arity(*cpfm*(*0*,*1*,*2*)) = *3*
using *cprod_fm_auto* **by** (*simp del*:*FOL_sats_iff add*: *fm_definitions ord_simp_union*)

then
have $\forall\, a{\in}M.\ \forall\, b{\in}M.\ separation(\#\#M,\ \lambda z.\ sats(M,cpfm(0,1,2)\ ,\ [z,\ a,\ b]))$
using *separation_ax* **by** *simp*
moreover
have $(\exists\, x{\in}M.\ x{\in}a \wedge (\exists\, y{\in}M.\ y{\in}b \wedge pair(\#\#M,x,y,z))) \longleftrightarrow sats(M,cpfm(0,1,2),[z,a,b])$
if $a{\in}M\ b{\in}M\ z{\in}M$ **for** *a b z*
using *that fmsats*[*of* [*z*,*a*,*b*]] **by** *simp*
ultimately
have $\forall\, a{\in}M.\ \forall\, b{\in}M.\ separation(\#\#M,\ \lambda z\ .\ (\exists\, x{\in}M.\ x{\in}a\ \wedge\ (\exists\, y{\in}M.\ y{\in}b\ \wedge\ pair(\#\#M,x,y,z))))$
unfolding *separation_def* **by** *simp*
with ‹$A{\in}M$› ‹$B{\in}M$› **show** *?thesis* **by** *simp*
qed

schematic_goal *im_fm_auto*:
assumes
$nth(i,env) = y\ nth(j,env) = r\ nth(h,env) = B$
$i \in nat\ j \in nat\ h \in nat\ env \in list(A)$
shows
$(\exists\, p{\in}A.\ p{\in}r\ \&\ (\exists\, x{\in}A.\ x{\in}B\ \&\ pair(\#\#A,x,y,p))) \longleftrightarrow sats(A,?imfm(i,j,h),env)$
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *image_sep_intf* :
assumes
$A{\in}M$
and
$r{\in}M$
shows
$separation(\#\#M,\ \lambda y.\ \exists\, p{\in}M.\ p{\in}r\ \&\ (\exists\, x{\in}M.\ x{\in}A\ \&\ pair(\#\#M,x,y,p)))$
proof -
obtain *imfm* **where**
$fmsats{:}\bigwedge env.\ env{\in}list(M) \Longrightarrow$
$(\exists\, p{\in}M.\ p{\in}nth(1,env)\ \&\ (\exists\, x{\in}M.\ x{\in}nth(2,env)\ \&\ pair(\#\#M,x,nth(0,env),p)))$
$\longleftrightarrow sats(M,imfm(0,1,2),env)$
and
$imfm(0,1,2) \in formula$
and
$arity(imfm(0,1,2)) = 3$
using *im_fm_auto* **by** (*simp del*:*FOL_sats_iff pair_abs add*: *fm_definitions ord_simp_union*)
then
have $\forall\, r{\in}M.\ \forall\, a{\in}M.\ separation(\#\#M,\ \lambda y.\ sats(M,imfm(0,1,2)\ ,\ [y,r,a]))$
using *separation_ax* **by** *simp*
moreover
have $(\exists\, p{\in}M.\ p{\in}k\ \&\ (\exists\, x{\in}M.\ x{\in}a\ \&\ pair(\#\#M,x,y,p))) \longleftrightarrow sats(M,imfm(0,1,2),[y,k,a])$
if $k{\in}M\ a{\in}M\ y{\in}M$ **for** *k a y*
using *that fmsats*[*of* [*y*,*k*,*a*]] **by** *simp*
ultimately
have $\forall\, k{\in}M.\ \forall\, a{\in}M.\ separation(\#\#M,\ \lambda y\ .\ \exists\, p{\in}M.\ p{\in}k\ \&\ (\exists\, x{\in}M.\ x{\in}a\ \&$

pair(##*M*,*x*,*y*,*p*)))
unfolding *separation_def* **by** *simp*
with ‹*r*∈*M*› ‹*A*∈*M*› **show** *?thesis* **by** *simp*
qed

schematic_goal *con_fm_auto*:
assumes
nth(*i*,*env*) = *z* *nth*(*j*,*env*) = *R*
i ∈ *nat* *j* ∈ *nat* *env* ∈ *list*(*A*)
shows
(∃ *p*∈*A*. *p*∈*R* & (∃ *x*∈*A*.∃ *y*∈*A*. *pair*(##*A*,*x*,*y*,*p*) & *pair*(##*A*,*y*,*x*,*z*)))
⟷ *sats*(*A*,*?cfm*(*i*,*j*),*env*)
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *converse_sep_intf* :
assumes
R∈*M*
shows
separation(##*M*,λ*z*. ∃ *p*∈*M*. *p*∈*R* & (∃ *x*∈*M*.∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*p*) &
pair(##*M*,*y*,*x*,*z*)))
proof -
obtain *cfm* **where**
fmsats:⋀*env*. *env*∈*list*(*M*) ⟹
(∃ *p*∈*M*. *p*∈*nth*(1,*env*) & (∃ *x*∈*M*.∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*p*) & *pair*(##*M*,*y*,*x*,*nth*(0,*env*))))
⟷ *sats*(*M*,*cfm*(0,1),*env*)
and
cfm(0,1) ∈ *formula*
and
arity(*cfm*(0,1)) = 2
using *con_fm_auto* **by** (*simp del*:*FOL_sats_iff pair_abs add*: *fm_definitions*
ord_simp_union)
then
have ∀ *r*∈*M*. *separation*(##*M*, λ*z*. *sats*(*M*,*cfm*(0,1) , [*z*,*r*]))
using *separation_ax* **by** *simp*
moreover
have (∃ *p*∈*M*. *p*∈*r* & (∃ *x*∈*M*.∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*p*) & *pair*(##*M*,*y*,*x*,*z*)))
⟷
sats(*M*,*cfm*(0,1),[*z*,*r*])
if *z*∈*M* *r*∈*M* **for** *z* *r*
using *that fmsats*[*of* [*z*,*r*]] **by** *simp*
ultimately
have ∀ *r*∈*M*. *separation*(##*M*, λ*z* . ∃ *p*∈*M*. *p*∈*r* & (∃ *x*∈*M*.∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*p*)
& *pair*(##*M*,*y*,*x*,*z*)))
unfolding *separation_def* **by** *simp*
with ‹*R*∈*M*› **show** *?thesis* **by** *simp*
qed

schematic_goal *rest_fm_auto*:
assumes
nth(*i*,*env*) = *z* *nth*(*j*,*env*) = *C*
i ∈ *nat* *j* ∈ *nat* *env* ∈ *list*(*A*)
shows
(∃ *x*∈*A*. *x*∈*C* & (∃ *y*∈*A*. *pair*(##*A*,*x*,*y*,*z*)))
⟷ *sats*(*A*,*?rfm*(*i*,*j*),*env*)
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *restrict_sep_intf* :
assumes
A∈*M*
shows
separation(##*M*,λ*z*. ∃ *x*∈*M*. *x*∈*A* & (∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*z*)))
proof -
obtain *rfm* **where**
fmsats:⋀*env*. *env*∈*list*(*M*) ⟹
(∃ *x*∈*M*. *x*∈*nth*(*1*,*env*) & (∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*nth*(*0*,*env*))))
⟷ *sats*(*M*,*rfm*(*0*,*1*),*env*)
and
rfm(*0*,*1*) ∈ *formula*
and
arity(*rfm*(*0*,*1*)) = *2*
using *rest_fm_auto* **by** (*simp del*:*FOL_sats_iff pair_abs add*: *fm_definitions ord_simp_union*)
then
have ∀ *a*∈*M*. *separation*(##*M*, λ*z*. *sats*(*M*,*rfm*(*0*,*1*) , [*z*,*a*]))
using *separation_ax* **by** *simp*
moreover
have (∃ *x*∈*M*. *x*∈*a* & (∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*z*))) ⟷
sats(*M*,*rfm*(*0*,*1*),[*z*,*a*])
if *z*∈*M* *a*∈*M* **for** *z* *a*
using *that fmsats*[*of* [*z*,*a*]] **by** *simp*
ultimately
have ∀ *a*∈*M*. *separation*(##*M*, λ*z* . ∃ *x*∈*M*. *x*∈*a* & (∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*z*)))
unfolding *separation_def* **by** *simp*
with ‹*A*∈*M*› **show** *?thesis* **by** *simp*
qed

schematic_goal *comp_fm_auto*:
assumes
nth(*i*,*env*) = *xz* *nth*(*j*,*env*) = *S* *nth*(*h*,*env*) = *R*
i ∈ *nat* *j* ∈ *nat* *h* ∈ *nat* *env* ∈ *list*(*A*)
shows
(∃ *x*∈*A*. ∃ *y*∈*A*. ∃ *z*∈*A*. ∃ *xy*∈*A*. ∃ *yz*∈*A*.
pair(##*A*,*x*,*z*,*xz*) & *pair*(##*A*,*x*,*y*,*xy*) & *pair*(##*A*,*y*,*z*,*yz*) & *xy*∈*S* & *yz*∈*R*)
⟷ *sats*(*A*,*?cfm*(*i*,*j*,*h*),*env*)

by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *comp_sep_intf* :
assumes
R∈M
and
S∈M
shows
separation(*##M*,*λxz*. ∃ *x*∈*M*. ∃ *y*∈*M*. ∃ *z*∈*M*. ∃ *xy*∈*M*. ∃ *yz*∈*M*.
pair(*##M*,*x*,*z*,*xz*) & *pair*(*##M*,*x*,*y*,*xy*) & *pair*(*##M*,*y*,*z*,*yz*) & *xy*∈*S* & *yz*∈*R*)
proof -
obtain *cfm* **where**
fmsats:⋀*env*. *env*∈*list*(*M*) ⟹
(∃ *x*∈*M*. ∃ *y*∈*M*. ∃ *z*∈*M*. ∃ *xy*∈*M*. ∃ *yz*∈*M*. *pair*(*##M*,*x*,*z*,*nth*(*0*,*env*)) &
pair(*##M*,*x*,*y*,*xy*) & *pair*(*##M*,*y*,*z*,*yz*) & *xy*∈*nth*(*1*,*env*) & *yz*∈*nth*(*2*,*env*))
⟷ *sats*(*M*,*cfm*(*0*,*1*,*2*),*env*)
and
cfm(*0*,*1*,*2*) ∈ *formula*
and
arity(*cfm*(*0*,*1*,*2*)) = *3*
using *comp_fm_auto* **by** (*simp del*:*FOL_sats_iff pair_abs add*: *fm_definitions ord_simp_union*)
then
have ∀ *r*∈*M*. ∀ *s*∈*M*. *separation*(*##M*, *λy*. *sats*(*M*,*cfm*(*0*,*1*,*2*) , [*y*,*s*,*r*]))
using *separation_ax* **by** *simp*
moreover
have (∃ *x*∈*M*. ∃ *y*∈*M*. ∃ *z*∈*M*. ∃ *xy*∈*M*. ∃ *yz*∈*M*.
pair(*##M*,*x*,*z*,*xz*) & *pair*(*##M*,*x*,*y*,*xy*) & *pair*(*##M*,*y*,*z*,*yz*) & *xy*∈*s* & *yz*∈*r*)
⟷ *sats*(*M*,*cfm*(*0*,*1*,*2*) , [*xz*,*s*,*r*])
if *xz*∈*M s*∈*M r*∈*M* **for** *xz s r*
using *that fmsats*[*of* [*xz*,*s*,*r*]] **by** *simp*
ultimately
have ∀ *s*∈*M*. ∀ *r*∈*M*. *separation*(*##M*, *λxz* . ∃ *x*∈*M*. ∃ *y*∈*M*. ∃ *z*∈*M*. ∃ *xy*∈*M*. ∃ *yz*∈*M*.
pair(*##M*,*x*,*z*,*xz*) & *pair*(*##M*,*x*,*y*,*xy*) & *pair*(*##M*,*y*,*z*,*yz*) & *xy*∈*s* & *yz*∈*r*)
unfolding *separation_def* **by** *simp*
with ‹*S*∈*M*› ‹*R*∈*M*› **show** *?thesis* **by** *simp*
qed

schematic_goal *pred_fm_auto*:
assumes
nth(*i*,*env*) = *y nth*(*j*,*env*) = *R nth*(*h*,*env*) = *X*
i ∈ *nat j* ∈ *nat h* ∈ *nat env* ∈ *list*(*A*)
shows

(∃ p∈A. p∈R & pair(##A,y,X,p)) ⟷ sats(A,?pfm(i,j,h),env)
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *pred_sep_intf*:
assumes
R∈M
and
X∈M
shows
separation(##M, λy. ∃ p∈M. p∈R & pair(##M,y,X,p))
proof -
obtain *pfm* **where**
fmsats:⋀*env*. *env∈list*(*M*) ⟹
(∃ p∈M. p∈nth(1,env) & pair(##M,nth(0,env),nth(2,env),p)) ⟷ sats(M,pfm(0,1,2),env)
and
pfm(0,1,2) ∈ *formula*
and
arity(*pfm*(0,1,2)) = 3
using *pred_fm_auto* **by** (*simp del*:*FOL_sats_iff pair_abs add*: *fm_definitions ord_simp_union*)
then
have ∀ x∈M. ∀ r∈M. *separation*(##M, λy. *sats*(M,*pfm*(0,1,2) , [y,r,x]))
using *separation_ax* **by** *simp*
moreover
have (∃ p∈M. p∈r & *pair*(##M,y,x,p))
⟷ *sats*(M,*pfm*(0,1,2) , [y,r,x])
if y∈M r∈M x∈M **for** *y x r*
using *that fmsats*[*of* [y,r,x]] **by** *simp*
ultimately
have ∀ x∈M. ∀ r∈M. *separation*(##M, λ *y* . ∃ p∈M. p∈r & *pair*(##M,y,x,p))
unfolding *separation_def* **by** *simp*
with ‹X∈M› ‹R∈M› **show** *?thesis* **by** *simp*
qed

schematic_goal *mem_fm_auto*:
assumes
nth(*i*,*env*) = *z i* ∈ *nat env* ∈ *list*(*A*)
shows
(∃ x∈A. ∃ y∈A. *pair*(##A,x,y,z) & *x* ∈ *y*) ⟷ *sats*(A,*?mfm*(*i*),*env*)
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *memrel_sep_intf*:
separation(##M, λz. ∃ x∈M. ∃ y∈M. *pair*(##M,x,y,z) & *x* ∈ *y*)
proof -
obtain *mfm* **where**
fmsats:⋀*env*. *env∈list*(*M*) ⟹
(∃ x∈M. ∃ y∈M. *pair*(##M,x,y,nth(0,env)) & *x* ∈ *y*) ⟷ *sats*(M,*mfm*(0),*env*)

and
mfm(0) ∈ *formula*
and
arity(mfm(0)) = *1*
using *mem_fm_auto* **by** (*simp del*:*FOL_sats_iff pair_abs add*: *fm_definitions ord_simp_union*)
then
have *separation*(*##M*, λ*z*. *sats*(*M*,*mfm(0)* , [*z*]))
using *separation_ax* **by** *simp*
moreover
have (∃ *x*∈*M*. ∃ *y*∈*M*. *pair*(*##M*,*x*,*y*,*z*) & *x* ∈ *y*) ⟷ *sats*(*M*,*mfm(0)*,[*z*])
if *z*∈*M* **for** *z*
using *that fmsats*[*of* [*z*]] **by** *simp*
ultimately
have *separation*(*##M*, λ*z* . ∃ *x*∈*M*. ∃ *y*∈*M*. *pair*(*##M*,*x*,*y*,*z*) & *x* ∈ *y*)
unfolding *separation_def* **by** *simp*
then show *?thesis* **by** *simp*
qed

schematic_goal *recfun_fm_auto*:
assumes
nth(i1,env) = *x nth(i2,env)* = *r nth(i3,env)* = *f nth(i4,env)* = *g nth(i5,env)* = *a*
nth(i6,env) = *b i1*∈*nat i2*∈*nat i3*∈*nat i4*∈*nat i5*∈*nat i6*∈*nat env* ∈ *list*(*A*)
shows
(∃ *xa*∈*A*. ∃ *xb*∈*A*. *pair*(*##A*,*x*,*a*,*xa*) & *xa* ∈ *r* & *pair*(*##A*,*x*,*b*,*xb*) & *xb* ∈ *r* &
(∃ *fx*∈*A*. ∃ *gx*∈*A*. *fun_apply*(*##A*,*f*,*x*,*fx*) & *fun_apply*(*##A*,*g*,*x*,*gx*) & *fx* ≠ *gx*))
⟷ *sats*(*A*,*?rffm*(*i1*,*i2*,*i3*,*i4*,*i5*,*i6*),*env*)
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *is_recfun_sep_intf* :
assumes
r∈*M f*∈*M g*∈*M a*∈*M b*∈*M*
shows
separation(*##M*,λ*x*. ∃ *xa*∈*M*. ∃ *xb*∈*M*.
pair(*##M*,*x*,*a*,*xa*) & *xa* ∈ *r* & *pair*(*##M*,*x*,*b*,*xb*) & *xb* ∈ *r* &
(∃ *fx*∈*M*. ∃ *gx*∈*M*. *fun_apply*(*##M*,*f*,*x*,*fx*) & *fun_apply*(*##M*,*g*,*x*,*gx*) &
fx ≠ *gx*))
proof -
obtain *rffm* **where**
fmsats:⋀*env*. *env*∈*list*(*M*) ⟹
(∃ *xa*∈*M*. ∃ *xb*∈*M*. *pair*(*##M*,*nth*(*0*,*env*),*nth*(*4*,*env*),*xa*) & *xa* ∈ *nth*(*1*,*env*) &
pair(*##M*,*nth*(*0*,*env*),*nth*(*5*,*env*),*xb*) & *xb* ∈ *nth*(*1*,*env*) & (∃ *fx*∈*M*. ∃ *gx*∈*M*.
fun_apply(*##M*,*nth*(*2*,*env*),*nth*(*0*,*env*),*fx*) & *fun_apply*(*##M*,*nth*(*3*,*env*),*nth*(*0*,*env*),*gx*) & *fx* ≠ *gx*))
⟷ *sats*(*M*,*rffm*(*0*,*1*,*2*,*3*,*4*,*5*),*env*)

and
rffm(*0*,*1*,*2*,*3*,*4*,*5*) ∈ *formula*
and
arity(*rffm*(*0*,*1*,*2*,*3*,*4*,*5*)) = *6*
using *recfun_fm_auto* **by** (*simp del*:*FOL_sats_iff pair_abs add*: *fm_definitions ord_simp_union*)
then
have ∀ *a1*∈*M*. ∀ *a2*∈*M*. ∀ *a3*∈*M*. ∀ *a4*∈*M*. ∀ *a5*∈*M*.
separation(*##M*, λ*x*. *sats*(*M*,*rffm*(*0*,*1*,*2*,*3*,*4*,*5*) , [*x*,*a1*,*a2*,*a3*,*a4*,*a5*]))
using *separation_ax* **by** *simp*
moreover
have (∃ *xa*∈*M*. ∃ *xb*∈*M*. *pair*(*##M*,*x*,*a4*,*xa*) & *xa* ∈ *a1* & *pair*(*##M*,*x*,*a5*,*xb*) & *xb* ∈ *a1* &
(∃ *fx*∈*M*. ∃ *gx*∈*M*. *fun_apply*(*##M*,*a2*,*x*,*fx*) & *fun_apply*(*##M*,*a3*,*x*,*gx*) & *fx* ≠ *gx*))
⟷ *sats*(*M*,*rffm*(*0*,*1*,*2*,*3*,*4*,*5*) , [*x*,*a1*,*a2*,*a3*,*a4*,*a5*])
if *x*∈*M a1*∈*M a2*∈*M a3*∈*M a4*∈*M a5*∈*M* **for** *x a1 a2 a3 a4 a5*
using *that fmsats*[*of* [*x*,*a1*,*a2*,*a3*,*a4*,*a5*]] **by** *simp*
ultimately
have ∀ *a1*∈*M*. ∀ *a2*∈*M*. ∀ *a3*∈*M*. ∀ *a4*∈*M*. ∀ *a5*∈*M*. *separation*(*##M*, λ *x* .
∃ *xa*∈*M*. ∃ *xb*∈*M*. *pair*(*##M*,*x*,*a4*,*xa*) & *xa* ∈ *a1* & *pair*(*##M*,*x*,*a5*,*xb*) & *xb* ∈ *a1* &
(∃ *fx*∈*M*. ∃ *gx*∈*M*. *fun_apply*(*##M*,*a2*,*x*,*fx*) & *fun_apply*(*##M*,*a3*,*x*,*gx*) & *fx* ≠ *gx*))
unfolding *separation_def* **by** *simp*
with ‹*r*∈*M*› ‹*f*∈*M*› ‹*g*∈*M*› ‹*a*∈*M*› ‹*b*∈*M*› **show** *?thesis* **by** *simp*
qed

schematic_goal *funsp_fm_auto*:
assumes
nth(*i*,*env*) = *p nth*(*j*,*env*) = *z nth*(*h*,*env*) = *n*
i ∈ *nat j* ∈ *nat h* ∈ *nat env* ∈ *list*(*A*)
shows
(∃ *f*∈*A*. ∃ *b*∈*A*. ∃ *nb*∈*A*. ∃ *cnbf*∈*A*. *pair*(*##A*,*f*,*b*,*p*) & *pair*(*##A*,*n*,*b*,*nb*) & *is_cons*(*##A*,*nb*,*f*,*cnbf*) &
upair(*##A*,*cnbf*,*cnbf*,*z*)) ⟷ *sats*(*A*,*?fsfm*(*i*,*j*,*h*),*env*)
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *funspace_succ_rep_intf* :
assumes
n∈*M*
shows
strong_replacement(*##M*,
λ*p z*. ∃ *f*∈*M*. ∃ *b*∈*M*. ∃ *nb*∈*M*. ∃ *cnbf*∈*M*.
pair(*##M*,*f*,*b*,*p*) & *pair*(*##M*,*n*,*b*,*nb*) & *is_cons*(*##M*,*nb*,*f*,*cnbf*) &

upair(##*M*,*cnbf*,*cnbf*,*z*))
proof -
obtain *fsfm* **where**
fmsats:*env*∈*list*(*M*) ⟹
(∃ *f*∈*M*. ∃ *b*∈*M*. ∃ *nb*∈*M*. ∃ *cnbf*∈*M*. *pair*(##*M*,*f*,*b*,*nth*(*0*,*env*)) & *pair*(##*M*,*nth*(*2*,*env*),*b*,*nb*)
& *is_cons*(##*M*,*nb*,*f*,*cnbf*) & *upair*(##*M*,*cnbf*,*cnbf*,*nth*(*1*,*env*)))
⟷ *sats*(*M*,*fsfm*(*0*,*1*,*2*),*env*)
and *fsfm*(*0*,*1*,*2*) ∈ *formula* **and** *arity*(*fsfm*(*0*,*1*,*2*)) = *3* **for** *env*
using *funsp_fm_auto*[*of concl*:*M*] **by** (*simp del*:*FOL_sats_iff pair_abs add*:
fm_definitions ord_simp_union)

then
have ∀ *n0*∈*M*. *strong_replacement*(##*M*, λ*p z*. *sats*(*M*,*fsfm*(*0*,*1*,*2*) , [*p*,*z*,*n0*]))
using *replacement_ax*[*of fsfm*(*0*,*1*,*2*)] **by** *simp*
moreover
have (∃ *f*∈*M*. ∃ *b*∈*M*. ∃ *nb*∈*M*. ∃ *cnbf*∈*M*. *pair*(##*M*,*f*,*b*,*p*) & *pair*(##*M*,*n0*,*b*,*nb*)
&
is_cons(##*M*,*nb*,*f*,*cnbf*) & *upair*(##*M*,*cnbf*,*cnbf*,*z*))
⟷ *sats*(*M*,*fsfm*(*0*,*1*,*2*) , [*p*,*z*,*n0*])
if *p*∈*M z*∈*M n0*∈*M* **for** *p z n0*
using *that fmsats*[*of* [*p*,*z*,*n0*]] **by** *simp*
ultimately
have ∀ *n0*∈*M*. *strong_replacement*(##*M*, λ *p z*.
∃ *f*∈*M*. ∃ *b*∈*M*. ∃ *nb*∈*M*. ∃ *cnbf*∈*M*. *pair*(##*M*,*f*,*b*,*p*) & *pair*(##*M*,*n0*,*b*,*nb*)
&
is_cons(##*M*,*nb*,*f*,*cnbf*) & *upair*(##*M*,*cnbf*,*cnbf*,*z*))
unfolding *strong_replacement_def univalent_def* **by** *simp*
with ‹*n*∈*M*› **show** *?thesis* **by** *simp*
qed

lemmas *M_basic_sep_instances* =
inter_sep_intf diff_sep_intf cartprod_sep_intf
image_sep_intf converse_sep_intf restrict_sep_intf
pred_sep_intf memrel_sep_intf comp_sep_intf is_recfun_sep_intf

end

sublocale *M_ZF_trans* ⊆ *M_basic* ##*M*
using *trans_M zero_in_M power_ax M_basic_sep_instances funspace_succ_rep_intf*
by *unfold_locales auto*

16.3 Interface with *M_trancl*

schematic_goal *rtran_closure_mem_auto*:
assumes
nth(*i*,*env*) = *p nth*(*j*,*env*) = *r* *nth*(*k*,*env*) = *B*

```
    i ∈ nat j ∈ nat k ∈ nat env ∈ list(A)
  shows
    rtran_closure_mem(##A,B,r,p) ⟷ sats(A,?rcfm(i,j,k),env)
  unfolding rtran_closure_mem_def
  by (insert assms ; (rule sep_rules | simp)+)


lemma (in M_ZF_trans) rtrancl_separation_intf:
  assumes
    r∈M
    and
    A∈M
  shows
    separation (##M, rtran_closure_mem(##M,A,r))
proof -
  obtain rcfm where
    fmsats:⋀env. env∈list(M) ⟹
   (rtran_closure_mem(##M,nth(2,env),nth(1,env),nth(0,env))) ⟷ sats(M,rcfm(0,1,2),env)
    and
    rcfm(0,1,2) ∈ formula
    and
    arity(rcfm(0,1,2)) = 3
     using rtran_closure_mem_auto by (simp del:FOL_sats_iff pair_abs add:
fm_definitions ord_simp_union)
  then
  have ∀ x∈M. ∀ a∈M. separation(##M, λy. sats(M,rcfm(0,1,2) , [y,x,a]))
    using separation_ax by simp
  moreover
  have (rtran_closure_mem(##M,a,x,y))
        ⟷ sats(M,rcfm(0,1,2) , [y,x,a])
    if y∈M x∈M a∈M for y x a
    using that fmsats[of [y,x,a]] by simp
  ultimately
  have ∀ x∈M. ∀ a∈M. separation(##M, rtran_closure_mem(##M,a,x))
    unfolding separation_def by simp
  with ‹r∈M› ‹A∈M› show ?thesis by simp
qed

schematic_goal rtran_closure_fm_auto:
  assumes
    nth(i,env) = r nth(j,env) = rp
    i ∈ nat j ∈ nat env ∈ list(A)
  shows
    rtran_closure(##A,r,rp) ⟷ sats(A,?rtc(i,j),env)
  unfolding rtran_closure_def
  by (insert assms ; (rule sep_rules rtran_closure_mem_auto| simp)+)

schematic_goal trans_closure_fm_auto:
  assumes
```

```
    i ∈ nat j ∈ nat env ∈ list(A)
  shows
    tran_closure(##A,nth(i,env),nth(j,env)) ⟷ sats(A,?tc(i,j),env)
  unfolding tran_closure_def
  by (insert assms ; (rule sep_rules rtran_closure_fm_auto | simp))+

synthesize trans_closure from_schematic trans_closure_fm_auto

lemma arity_tran_closure_fm :
  ⟦x∈nat;f∈nat⟧ ⟹ arity(trans_closure_fm(x,f)) = succ(x) ∪ succ(f)
  unfolding trans_closure_fm_def
  using arity_omega_fm arity_field_fm arity_typed_function_fm arity_pair_fm
arity_empty_fm arity_fun_apply_fm
    arity_composition_fm arity_succ_fm union_abs2 pred_Un_distrib
  by auto

schematic_goal wellfounded_trancl_fm_auto:
  assumes
    nth(i,env) = p nth(j,env) = r  nth(k,env) = B
    i ∈ nat j ∈ nat k ∈ nat env ∈ list(A)
  shows
    wellfounded_trancl(##A,B,r,p) ⟷ sats(A,?wtf(i,j,k),env)
  unfolding  wellfounded_trancl_def
  by (insert assms ; (rule sep_rules trans_closure_iff_sats | simp)+)

context M_ZF_trans
begin

lemma wftrancl_separation_intf:
  assumes
    r∈M and Z∈M
  shows
    separation (##M, wellfounded_trancl(##M,Z,r))
proof -
  obtain rcfm where
    fmsats:⋀env. env∈list(M) ⟹
   (wellfounded_trancl(##M,nth(2,env),nth(1,env),nth(0,env))) ⟷ sats(M,rcfm(0,1,2),env)
    and
    rcfm(0,1,2) ∈ formula
    and
    arity(rcfm(0,1,2)) = 3
   using wellfounded_trancl_fm_auto[of concl:M nth(2,_)] unfolding fm_definitions
    by (simp del:FOL_sats_iff pair_abs add: ord_simp_union)
  then
  have ∀ x∈M. ∀ z∈M. separation(##M, λy. sats(M,rcfm(0,1,2) , [y,x,z]))
    using separation_ax by simp
  moreover
  have (wellfounded_trancl(##M,z,x,y))
        ⟷ sats(M,rcfm(0,1,2) , [y,x,z])
```

if $y{\in}M$ $x{\in}M$ $z{\in}M$ **for** y x z
using *that fmsats*[*of* [y,x,z]] **by** *simp*
ultimately
have $\forall\, x{\in}M.\ \forall\, z{\in}M.$ *separation*(*##M*, *wellfounded_trancl*(*##M*,z,x))
unfolding *separation_def* **by** *simp*
with ‹$r{\in}M$› ‹$Z{\in}M$› **show** *?thesis* **by** *simp*
qed

Proof that $nat \in M$

lemma *finite_sep_intf*: *separation*(*##M*, $\lambda x.\ x{\in}nat$)
proof -
have *arity*(*finite_ordinal_fm*(*0*)) = *1*
unfolding *finite_ordinal_fm_def limit_ordinal_fm_def empty_fm_def succ_fm_def cons_fm_def*
union_fm_def upair_fm_def
by (*simp add*: *union_abs1 Un_commute*)
with *separation_ax*
have ($\forall\, v{\in}M.$ *separation*(*##M*,$\lambda x.$ *sats*(*M*,*finite_ordinal_fm*(*0*),[x,v])))
by *simp*
then have ($\forall\, v{\in}M.$ *separation*(*##M*,*finite_ordinal*(*##M*)))
unfolding *separation_def* **by** *simp*
then have *separation*(*##M*,*finite_ordinal*(*##M*))
using *zero_in_M* **by** *auto*
then show *?thesis* **unfolding** *separation_def* **by** *simp*
qed

lemma *nat_subset_I'*:
$[\![\ I{\in}M\ ;\ 0{\in}I\ ;\ \bigwedge x.\ x{\in}I \Longrightarrow succ(x){\in}I\ ]\!] \Longrightarrow nat \subseteq I$
by (*rule subsetI*,*induct_tac x*,*simp*+)

lemma *nat_subset_I*: $\exists\, I{\in}M.\ nat \subseteq I$
proof -
have $\exists\, I{\in}M.\ 0{\in}I \wedge (\forall\, x{\in}M.\ x{\in}I \longrightarrow succ(x){\in}I)$
using *infinity_ax* **unfolding** *infinity_ax_def* **by** *auto*
then obtain I **where**
$I{\in}M$ $0{\in}I$ $(\forall\, x{\in}M.\ x{\in}I \longrightarrow succ(x){\in}I)$
by *auto*
then have $\bigwedge x.\ x{\in}I \Longrightarrow succ(x){\in}I$
using *transitivity* **by** *simp*
then have *nat*$\subseteq I$
using ‹$I{\in}M$› ‹$0{\in}I$› *nat_subset_I'* **by** *simp*
then show *?thesis* **using** ‹$I{\in}M$› **by** *auto*
qed

lemma *nat_in_M*: $nat \in M$
proof -
have *1*:$\{x{\in}B\ .\ x{\in}A\}{=}A$ **if** $A{\subseteq}B$ **for** A B
using *that* **by** *auto*
obtain I **where**

```
    I∈M nat⊆I
    using nat_subset_I by auto
  then have {x∈I . x∈nat} ∈ M
    using finite_sep_intf separation_closed[of λx . x∈nat] by simp
  then show ?thesis
    using ‹nat⊆I› 1 by simp
qed

end

sublocale M_ZF_trans ⊆ M_trancl ##M
  using rtrancl_separation_intf wftrancl_separation_intf nat_in_M
    wellfounded_trancl_def by unfold_locales auto
```

16.4 Interface with *M_eclose*

```
lemma repl_sats:
  assumes
    sat:⋀x z. x∈M ⟹ z∈M ⟹ sats(M,φ,Cons(x,Cons(z,env))) ⟷ P(x,z)
  shows
    strong_replacement(##M,λx z. sats(M,φ,Cons(x,Cons(z,env)))) ⟷
   strong_replacement(##M,P)
  by (rule strong_replacement_cong,simp add:sat)

lemma (in M_ZF_trans) list_repl1_intf:
  assumes
    A∈M
  shows
    iterates_replacement(##M, is_list_functor(##M,A), 0)
proof -
  {
    fix n
    assume n∈nat
    have succ(n)∈M
      using ‹n∈nat› nat_into_M by simp
    then have 1:Memrel(succ(n))∈M
      using ‹n∈nat› Memrel_closed by simp
    have 0∈M
      using  nat_0I nat_into_M by simp
    then have is_list_functor(##M, A, a, b)
    ⟷ sats(M, list_functor_fm(13,1,0), [b,a,c,d,a0,a1,a2,a3,a4,y,x,z,Memrel(succ(n)),A,0])
      if a∈M b∈M c∈M d∈M a0∈M a1∈M a2∈M a3∈M a4∈M y∈M x∈M z∈M
      for a b c d a0 a1 a2 a3 a4 y x z
      using that 1 ‹A∈M› list_functor_iff_sats by simp
   then have sats(M, iterates_MH_fm(list_functor_fm(13,1,0),10,2,1,0), [a0,a1,a2,a3,a4,y,x,z,Memrel(succ(
        ⟷ iterates_MH(##M,is_list_functor(##M,A),0,a2, a1, a0)
      if a0∈M a1∈M a2∈M a3∈M a4∈M y∈M x∈M z∈M
      for a0 a1 a2 a3 a4 y x z
     using that sats_iterates_MH_fm[of M is_list_functor(##M,A) _] 1 ‹0∈M›
```

```
‹A∈M›  by simp
  then have 2:sats(M, is_wfrec_fm(iterates_MH_fm(list_functor_fm(13,1,0),10,2,1,0),3,1,0),
                          [y,x,z,Memrel(succ(n)),A,0])
        ⟷
     is_wfrec(##M, iterates_MH(##M,is_list_functor(##M,A),0) , Memrel(succ(n)),
x, y)
      if y∈M x∈M z∈M for y x z
      using  that sats_is_wfrec_fm 1 ‹0∈M› ‹A∈M› by simp
    let
      ?f=Exists(And(pair_fm(1,0,2),
          is_wfrec_fm(iterates_MH_fm(list_functor_fm(13,1,0),10,2,1,0),3,1,0)))
    have satsf:sats(M, ?f, [x,z,Memrel(succ(n)),A,0])
        ⟷
        (∃ y∈M. pair(##M,x,y,z) &
     is_wfrec(##M, iterates_MH(##M,is_list_functor(##M,A),0) , Memrel(succ(n)),
x, y))
      if x∈M z∈M for x z
      using that 2 1 ‹0∈M› ‹A∈M› by (simp del:pair_abs)
    have arity(?f) = 5
      unfolding fm_definitions
      by (simp add:ord_simp_union)
    then
    have strong_replacement(##M,λx z. sats(M,?f,[x,z,Memrel(succ(n)),A,0]))
      using replacement_ax[of ?f] 1 ‹A∈M› ‹0∈M› by simp
    then
    have strong_replacement(##M,λx z.
      ∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, iterates_MH(##M,is_list_functor(##M,A),0)
,
                Memrel(succ(n)), x, y))
     using repl_sats[of M ?f [Memrel(succ(n)),A,0]]  satsf by (simp del:pair_abs)
  }
  then
   show ?thesis unfolding iterates_replacement_def wfrec_replacement_def by
simp
qed




lemma (in M_ZF_trans) iterates_repl_intf :
  assumes
    v∈M and
    isfm:is_F_fm ∈ formula and
    arty:arity(is_F_fm)=2 and
    satsf: ⋀a b env'. ⟦ a∈M ; b∈M ; env'∈list(M) ⟧
             ⟹ is_F(a,b) ⟷ sats(M, is_F_fm, [b,a]@env')
  shows
    iterates_replacement(##M,is_F,v)
proof -
```

```
  {
    fix n
    assume n∈nat
    have succ(n)∈M
      using ‹n∈nat› nat_into_M by simp
    then have 1:Memrel(succ(n))∈M
      using ‹n∈nat› Memrel_closed by simp
    {
      fix a0 a1 a2 a3 a4 y x z
      assume as:a0∈M a1∈M a2∈M a3∈M a4∈M y∈M x∈M z∈M
    have sats(M, is_F_fm, Cons(b,Cons(a,Cons(c,Cons(d,[a0,a1,a2,a3,a4,y,x,z,Memrel(succ(n)),v])))))
          ⟷ is_F(a,b)
        if a∈M b∈M c∈M d∈M for a b c d
         using as that 1 satsf[of a b [c,d,a0,a1,a2,a3,a4,y,x,z,Memrel(succ(n)),v]]
‹v∈M› by simp
      then
    have sats(M, iterates_MH_fm(is_F_fm,9,2,1,0), [a0,a1,a2,a3,a4,y,x,z,Memrel(succ(n)),v])
          ⟷ iterates_MH(##M,is_F,v,a2, a1, a0)
        using as
          sats_iterates_MH_fm[of M is_F is_F_fm] 1 ‹v∈M› by simp
    }
    then have 2:sats(M, is_wfrec_fm(iterates_MH_fm(is_F_fm,9,2,1,0),3,1,0),
                           [y,x,z,Memrel(succ(n)),v])
        ⟷
        is_wfrec(##M, iterates_MH(##M,is_F,v),Memrel(succ(n)), x, y)
      if y∈M x∈M z∈M for y x z
      using  that sats_is_wfrec_fm 1 ‹v∈M› by simp
    let
      ?f=Exists(And(pair_fm(1,0,2),
              is_wfrec_fm(iterates_MH_fm(is_F_fm,9,2,1,0),3,1,0)))
    have satsf:sats(M, ?f, [x,z,Memrel(succ(n)),v])
        ⟷
        (∃ y∈M. pair(##M,x,y,z) &
        is_wfrec(##M, iterates_MH(##M,is_F,v) , Memrel(succ(n)), x, y))
      if x∈M z∈M for x z
      using that 2 1 ‹v∈M› by (simp del:pair_abs)
    have arity(?f) = 4
      unfolding fm_definitions
      using arty by (simp add:ord_simp_union)
    then
    have strong_replacement(##M,λx z. sats(M,?f,[x,z,Memrel(succ(n)),v]))
      using replacement_ax[of ?f] 1 ‹v∈M› ‹is_F_fm∈formula› by simp
    then
    have strong_replacement(##M,λx z.
          ∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, iterates_MH(##M,is_F,v)
,
                 Memrel(succ(n)), x, y))
      using repl_sats[of M ?f [Memrel(succ(n)),v]]  satsf by (simp del:pair_abs)
  }
```

then
show *?thesis* **unfolding** *iterates_replacement_def wfrec_replacement_def* **by** *simp*
qed

lemma (**in** *M_ZF_trans*) *formula_repl1_intf* :
iterates_replacement(*##M*, *is_formula_functor*(*##M*), *0*)
proof -
have *0∈M*
using *nat_0I nat_into_M* **by** *simp*
have *1*:*arity*(*formula_functor_fm*(*1*,*0*)) = *2*
unfolding *fm_definitions*
by (*simp add*:*ord_simp_union*)
have *2*:*formula_functor_fm*(*1*,*0*)∈*formula* **by** *simp*
have *is_formula_functor*(*##M*,*a*,*b*) ⟷
sats(*M*, *formula_functor_fm*(*1*,*0*), [*b*,*a*])
if *a∈M b∈M* **for** *a b*
using *that* **by** *simp*
then show *?thesis* **using** ‹*0∈M*› *1 2*
iterates_repl_intf[**where** *is_F_fm*=*formula_functor_fm*(*1*,*0*)] **by** *simp*
qed

lemma (**in** *M_ZF_trans*) *nth_repl_intf*:
assumes
l ∈ M
shows
iterates_replacement(*##M*,λ*l′ t*. *is_tl*(*##M*,*l′*,*t*),*l*)
proof -
have *1*:*arity*(*tl_fm*(*1*,*0*)) = *2*
unfolding *fm_definitions* **by** (*simp add*:*ord_simp_union*)
have *2*:*tl_fm*(*1*,*0*)∈*formula* **by** *simp*
have *is_tl*(*##M*,*a*,*b*) ⟷ *sats*(*M*, *tl_fm*(*1*,*0*), [*b*,*a*])
if *a∈M b∈M* **for** *a b*
using *that* **by** *simp*
then show *?thesis* **using** ‹*l∈M*› *1 2*
iterates_repl_intf[**where** *is_F_fm*=*tl_fm*(*1*,*0*)] **by** *simp*
qed

lemma (**in** *M_ZF_trans*) *eclose_repl1_intf*:
assumes
A∈M
shows
iterates_replacement(*##M*, *big_union*(*##M*), *A*)
proof -
have *1*:*arity*(*big_union_fm*(*1*,*0*)) = *2*
unfolding *fm_definitions* **by** (*simp add*:*ord_simp_union*)
have *2*:*big_union_fm*(*1*,*0*)∈*formula* **by** *simp*
have *big_union*(*##M*,*a*,*b*) ⟷ *sats*(*M*, *big_union_fm*(*1*,*0*), [*b*,*a*])

```
    if a∈M b∈M for a b
    using that by simp
  then show ?thesis using ‹A∈M› 1 2
      iterates_repl_intf[where is_F_fm=big_union_fm(1,0)] by simp
qed


lemma (in M_ZF_trans) list_repl2_intf:
  assumes
    A∈M
  shows
   strong_replacement(##M,λn y. n∈nat & is_iterates(##M, is_list_functor(##M,A),
0, n, y))
proof -
  have 0∈M
    using  nat_0I nat_into_M by simp
  have is_list_functor(##M,A,a,b) ⟷
       sats(M,list_functor_fm(13,1,0),[b,a,c,d,e,f,g,h,i,j,k,n,y,A,0,nat])
    if a∈M b∈M c∈M d∈M e∈M f∈Mg∈Mh∈Mi∈Mj∈M k∈M n∈M y∈M
    for a b c d e f g h i j k n y
    using that ‹0∈M› nat_in_M ‹A∈M› by simp
  then
   have 1:sats(M,  is_iterates_fm(list_functor_fm(13,1,0),3,0,1),[n,y,A,0,nat]  )
⟷
          is_iterates(##M, is_list_functor(##M,A), 0, n , y)
    if n∈M y∈M for n y
    using that ‹0∈M› ‹A∈M› nat_in_M
      sats_is_iterates_fm[of M is_list_functor(##M,A)] by simp
  let ?f = And(Member(0,4),is_iterates_fm(list_functor_fm(13,1,0),3,0,1))
  have satsf:sats(M, ?f,[n,y,A,0,nat] ) ⟷
       n∈nat & is_iterates(##M, is_list_functor(##M,A), 0, n, y)
    if n∈M y∈M for n y
    using that ‹0∈M› ‹A∈M› nat_in_M 1 by simp
  have arity(?f) = 5
    unfolding fm_definitions
    by (simp add:ord_simp_union)
  then
  have strong_replacement(##M,λn y. sats(M,?f,[n,y,A,0,nat]))
    using replacement_ax[of ?f] 1 nat_in_M ‹A∈M› ‹0∈M› by simp
  then
  show ?thesis using repl_sats[of M ?f [A,0,nat]]  satsf  by simp
qed

lemma (in M_ZF_trans) formula_repl2_intf:
 strong_replacement(##M,λn y. n∈nat & is_iterates(##M, is_formula_functor(##M),
0, n, y))
proof -
  have 0∈M
    using  nat_0I nat_into_M by simp
```

```
  have is_formula_functor(##M,a,b) ⟷
        sats(M,formula_functor_fm(1,0),[b,a,c,d,e,f,g,h,i,j,k,n,y,0,nat])
    if a∈M b∈M c∈M d∈M e∈M f∈Mg∈Mh∈Mi∈Mj∈M k∈M n∈M y∈M
    for a b c d e f g h i j k n y
    using that ‹0∈M› nat_in_M by simp
  then
 have 1:sats(M, is_iterates_fm(formula_functor_fm(1,0),2,0,1),[n,y,0,nat] ) ⟷
          is_iterates(##M, is_formula_functor(##M), 0, n , y)
    if n∈M y∈M for n y
    using that ‹0∈M› nat_in_M
      sats_is_iterates_fm[of M is_formula_functor(##M)] by simp
  let ?f = And(Member(0,3),is_iterates_fm(formula_functor_fm(1,0),2,0,1))
  have satsf:sats(M, ?f,[n,y,0,nat] ) ⟷
        n∈nat & is_iterates(##M, is_formula_functor(##M), 0, n, y)
    if n∈M y∈M for n y
    using that ‹0∈M› nat_in_M 1 by simp
  have artyf:arity(?f) = 4
    unfolding fm_definitions
    by (simp add:ord_simp_union)
  then
  have strong_replacement(##M,λn y. sats(M,?f,[n,y,0,nat]))
    using replacement_ax[of ?f] 1 artyf ‹0∈M› nat_in_M by simp
  then
  show ?thesis using repl_sats[of M ?f [0,nat]]  satsf  by simp
qed
```

```
lemma (in M_ZF_trans) eclose_repl2_intf:
  assumes
    A∈M
  shows
   strong_replacement(##M,λn y. n∈nat & is_iterates(##M, big_union(##M),
A, n, y))
proof -
  have big_union(##M,a,b) ⟷
        sats(M,big_union_fm(1,0),[b,a,c,d,e,f,g,h,i,j,k,n,y,A,nat])
    if a∈M b∈M c∈M d∈M e∈M f∈Mg∈Mh∈Mi∈Mj∈M k∈M n∈M y∈M
    for a b c d e f g h i j k n y
    using that ‹A∈M› nat_in_M by simp
  then
  have 1:sats(M, is_iterates_fm(big_union_fm(1,0),2,0,1),[n,y,A,nat] ) ⟷
          is_iterates(##M, big_union(##M), A, n , y)
    if n∈M y∈M for n y
    using that ‹A∈M› nat_in_M
      sats_is_iterates_fm[of M big_union(##M)] by simp
  let ?f = And(Member(0,3),is_iterates_fm(big_union_fm(1,0),2,0,1))
  have satsf:sats(M, ?f,[n,y,A,nat] ) ⟷
```

```
      n∈nat & is_iterates(##M, big_union(##M), A, n, y)
    if n∈M y∈M for n y
    using that ‹A∈M› nat_in_M 1 by simp
  have artyf:arity(?f) = 4
    unfolding fm_definitions
    by (simp add:ord_simp_union)
  then
  have strong_replacement(##M,λn y. sats(M,?f,[n,y,A,nat]))
    using replacement_ax[of ?f] 1 artyf ‹A∈M› nat_in_M by simp
  then
  show ?thesis using repl_sats[of M ?f [A,nat]]  satsf  by simp
qed

sublocale M_ZF_trans ⊆ M_datatypes ##M
  using list_repl1_intf list_repl2_intf formula_repl1_intf
    formula_repl2_intf nth_repl_intf
  by unfold_locales auto

sublocale M_ZF_trans ⊆ M_eclose ##M
  using eclose_repl1_intf eclose_repl2_intf
  by unfold_locales auto




definition
  powerset_fm :: [i,i] ⇒ i where
  powerset_fm(A,z) ≡ Forall(Iff(Member(0,succ(z)),subset_fm(0,succ(A))))

lemma powerset_type [TC]:
  ⟦ x ∈ nat; y ∈ nat ⟧ ⟹ powerset_fm(x,y) ∈ formula
  by (simp add:powerset_fm_def)

definition
  is_powapply_fm :: [i,i,i] ⇒ i where
  is_powapply_fm(f,y,z) ≡
      Exists(And(fun_apply_fm(succ(f), succ(y), 0),
            Forall(Iff(Member(0, succ(succ(z))),
            Forall(Implies(Member(0, 1), Member(0, 2)))))))

lemma is_powapply_type [TC] :
  ⟦f∈nat ; y∈nat; z∈nat⟧ ⟹ is_powapply_fm(f,y,z)∈formula
  unfolding is_powapply_fm_def by simp

declare is_powapply_fm_def[fm_definitions add]

lemma sats_is_powapply_fm :
  assumes
    f∈nat y∈nat z∈nat env∈list(A) 0∈A
```

shows
is_powapply(##A,nth(f, env),nth(y, env),nth(z, env))
⟷ sats(A,is_powapply_fm(f,y,z),env)
unfolding *is_powapply_def is_powapply_fm_def powerset_def subset_def*
using *nth_closed assms* **by** *simp*

lemma (**in** *M_ZF_trans*) *powapply_repl* :
assumes
f∈M
shows
strong_replacement(##M,is_powapply(##M,f))
proof -
have *arity(is_powapply_fm(2,0,1)) = 3*
unfolding *is_powapply_fm_def*
by (*simp add*: *fm_definitions ord_simp_union*)
then
have *∀ f0∈M. strong_replacement(##M, λp z. sats(M,is_powapply_fm(2,0,1)*
, [p,z,f0]))
using *replacement_ax*[*of is_powapply_fm(2,0,1)*] **by** *simp*
moreover
have *is_powapply(##M,f0,p,z) ⟷ sats(M,is_powapply_fm(2,0,1) , [p,z,f0])*
if *p∈M z∈M f0∈M* **for** *p z f0*
using *that zero_in_M sats_is_powapply_fm*[*of 2 0 1* [*p,z,f0*] *M*] **by** *simp*
ultimately
have *∀ f0∈M. strong_replacement(##M, is_powapply(##M,f0))*
unfolding *strong_replacement_def univalent_def* **by** *simp*
with ‹*f∈M*› **show** *?thesis* **by** *simp*
qed

definition
PHrank_fm :: [*i*,*i*,*i*] ⇒ *i* **where**
PHrank_fm(f,y,z) ≡ Exists(And(fun_apply_fm(succ(f),succ(y),0)
,succ_fm(0,succ(z))))

lemma *PHrank_type* [*TC*]:
⟦ *x ∈ nat*; *y ∈ nat*; *z ∈ nat* ⟧ ⟹ *PHrank_fm(x,y,z) ∈ formula*
by (*simp add*:*PHrank_fm_def*)

lemma (**in** *M_ZF_trans*) *sats_PHrank_fm*:
⟦ *x ∈ nat*; *y ∈ nat*; *z ∈ nat*; *env ∈ list(M)* ⟧
⟹ *sats(M,PHrank_fm(x,y,z),env) ⟷*
PHrank(##M,nth(x,env),nth(y,env),nth(z,env))
using *zero_in_M Internalizations.nth_closed* **by** (*simp add*: *PHrank_def PHrank_fm_def*)

lemma (**in** *M_ZF_trans*) *phrank_repl* :
  assumes
    $f{\in}M$
  shows
    strong_replacement(##*M*,*PHrank*(##*M*,*f*))
proof -
  have *arity*(*PHrank_fm*(2,0,1)) = 3
    unfolding *PHrank_fm_def*
    by (*simp add*: *fm_definitions ord_simp_union*)
  then
   have $\forall f0{\in}M.$ *strong_replacement*(##*M*, $\lambda p\ z.$ *sats*(*M*,*PHrank_fm*(2,0,1) , [*p*,*z*,*f0*]))
    using *replacement_ax*[*of PHrank_fm*(2,0,1)] **by** *simp*
  then
  have $\forall f0{\in}M.$ *strong_replacement*(##*M*, *PHrank*(##*M*,*f0*))
   unfolding *strong_replacement_def univalent_def* **by** (*simp add*:*sats_PHrank_fm*)
  with ‹$f{\in}M$› **show** *?thesis* **by** *simp*
qed

definition
  is_Hrank_fm :: [*i*,*i*,*i*] ⇒ *i* **where**
  is_Hrank_fm(*x*,*f*,*hc*) ≡ *Exists*(*And*(*big_union_fm*(0,*succ*(*hc*)),
                        Replace_fm(*succ*(*x*),*PHrank_fm*(*succ*(*succ*(*succ*(*f*))),0,1),0)))

lemma *is_Hrank_type* [*TC*]:
  ⟦ *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* ⟧ ⟹ *is_Hrank_fm*(*x*,*y*,*z*) ∈ *formula*
  by (*simp add*:*is_Hrank_fm_def*)

lemma (**in** *M_ZF_trans*) *sats_is_Hrank_fm*:
  ⟦ *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*M*)⟧
    ⟹ *sats*(*M*,*is_Hrank_fm*(*x*,*y*,*z*),*env*) ⟷
        is_Hrank(##*M*,*nth*(*x*,*env*),*nth*(*y*,*env*),*nth*(*z*,*env*))
  using *zero_in_M is_Hrank_def is_Hrank_fm_def sats_Replace_fm*
  by (*simp add*:*sats_PHrank_fm*)

declare *is_Hrank_fm_def*[*fm_definitions add*]
declare *PHrank_fm_def*[*fm_definitions add*]

lemma (**in** *M_ZF_trans*) *wfrec_rank* :
  assumes
    $X{\in}M$
  shows
    wfrec_replacement(##*M*,*is_Hrank*(##*M*),*rrank*(*X*))
proof -
  have
    is_Hrank(##*M*,*a2*, *a1*, *a0*) ⟷
          sats(*M*, *is_Hrank_fm*(2,1,0), [*a0*,*a1*,*a2*,*a3*,*a4*,*y*,*x*,*z*,*rrank*(*X*)])

if *a4∈M a3∈M a2∈M a1∈M a0∈M y∈M x∈M z∈M* **for** *a4 a3 a2 a1 a0 y x z*
using *that rrank_in_M ‹X∈M›* **by** (*simp add:sats_is_Hrank_fm*)
then
have
1:sats(M, is_wfrec_fm(is_Hrank_fm(2,1,0),3,1,0),[y,x,z,rrank(X)])
⟷ is_wfrec(##M, is_Hrank(##M) ,rrank(X), x, y)
if *y∈M x∈M z∈M* **for** *y x z*
using *that ‹X∈M› rrank_in_M sats_is_wfrec_fm* **by** (*simp add:sats_is_Hrank_fm*)
let
?f=Exists(And(pair_fm(1,0,2),is_wfrec_fm(is_Hrank_fm(2,1,0),3,1,0)))
have *satsf:sats(M, ?f, [x,z,rrank(X)])*
⟷ (∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, is_Hrank(##M) , rrank(X), x, y))
if *x∈M z∈M* **for** *x z*
using *that 1 ‹X∈M› rrank_in_M* **by** (*simp del:pair_abs*)
have *arity(?f) = 3*
unfolding *fm_definitions*
by (*simp add:ord_simp_union*)
then
have *strong_replacement(##M,λx z. sats(M,?f,[x,z,rrank(X)]))*
using *replacement_ax[of ?f] 1 ‹X∈M› rrank_in_M* **by** *simp*
then
have *strong_replacement(##M,λx z.*
∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, is_Hrank(##M) , rrank(X), x, y))
using *repl_sats[of M ?f [rrank(X)]] satsf* **by** (*simp del:pair_abs*)
then
show *?thesis* **unfolding** *wfrec_replacement_def* **by** *simp*
qed

definition
is_HVfrom_fm :: [i,i,i,i] ⇒ i **where**
is_HVfrom_fm(A,x,f,h) ≡ Exists(Exists(And(union_fm(A #+ 2,1,h #+ 2),
And(big_union_fm(0,1),
Replace_fm(x #+ 2,is_powapply_fm(f #+ 4,0,1),0)))))
declare *is_HVfrom_fm_def[fm_definitions add]*

lemma *is_HVfrom_type [TC]:*
⟦ A∈nat; x ∈ nat; f ∈ nat; h ∈ nat ⟧ ⟹ is_HVfrom_fm(A,x,f,h) ∈ formula
by (*simp add:is_HVfrom_fm_def*)

lemma *sats_is_HVfrom_fm :*
⟦ a∈nat; x ∈ nat; f ∈ nat; h ∈ nat; env ∈ list(A); 0∈A⟧
⟹ sats(A,is_HVfrom_fm(a,x,f,h),env) ⟷
is_HVfrom(##A,nth(a,env),nth(x,env),nth(f,env),nth(h,env))
using *is_HVfrom_def is_HVfrom_fm_def sats_Replace_fm[OF sats_is_powapply_fm]*
by *simp*

lemma *is_HVfrom_iff_sats*:
assumes
nth(a,env) = aa nth(x,env) = xx nth(f,env) = ff nth(h,env) = hh
a∈nat x∈nat f∈nat h∈nat env∈list(A) 0∈A
shows
is_HVfrom(##A,aa,xx,ff,hh) ⟷ sats(A, is_HVfrom_fm(a,x,f,h), env)
using *assms sats_is_HVfrom_fm* **by** *simp*

schematic_goal *sats_is_Vset_fm_auto*:
assumes
i∈nat v∈nat env∈list(A) 0∈A
i < length(env) v < length(env)
shows
is_Vset(##A,nth(i, env),nth(v, env))
⟷ sats(A,?ivs_fm(i,v),env)
unfolding *is_Vset_def is_Vfrom_def*
by (*insert assms*; (*rule sep_rules is_HVfrom_iff_sats is_transrec_iff_sats* | *simp*)+)

schematic_goal *is_Vset_iff_sats*:
assumes
nth(i,env) = ii nth(v,env) = vv
i∈nat v∈nat env∈list(A) 0∈A
i < length(env) v < length(env)
shows
is_Vset(##A,ii,vv) ⟷ sats(A, ?ivs_fm(i,v), env)
unfolding ‹*nth(i,env) = ii*›[*symmetric*] ‹*nth(v,env) = vv*›[*symmetric*]
by (*rule sats_is_Vset_fm_auto*(*1*); *simp add*:*assms*)

lemma (**in** *M_ZF_trans*) *memrel_eclose_sing* :
a∈M ⟹ ∃ sa∈M. ∃ esa∈M. ∃ mesa∈M.
upair(##M,a,a,sa) & is_eclose(##M,sa,esa) & membership(##M,esa,mesa)
using *upair_ax eclose_closed Memrel_closed* **unfolding** *upair_ax_def*
by (*simp del*:*upair_abs*)

lemma (**in** *M_ZF_trans*) *trans_repl_HVFrom* :
assumes
A∈M i∈M
shows
transrec_replacement(##M,is_HVfrom(##M,A),i)
proof -
{ fix *mesa*
assume *mesa∈M*
have
0:*is_HVfrom(##M,A,a2, a1, a0) ⟷*
sats(M, is_HVfrom_fm(8,2,1,0), [a0,a1,a2,a3,a4,y,x,z,A,mesa])
if *a4∈M a3∈M a2∈M a1∈M a0∈M y∈M x∈M z∈M* **for** *a4 a3 a2 a1 a0 y x z*

```
    using that zero_in_M sats_is_HVfrom_fm ‹mesa∈M› ‹A∈M› by simp
  have
    1:sats(M, is_wfrec_fm(is_HVfrom_fm(8,2,1,0),4,1,0),[y,x,z,A,mesa])
      ⟷ is_wfrec(##M, is_HVfrom(##M,A),mesa, x, y)
    if y∈M x∈M z∈M for y x z
    using that ‹A∈M› ‹mesa∈M› sats_is_wfrec_fm[OF 0] by simp
  let
   ?f=Exists(And(pair_fm(1,0,2),is_wfrec_fm(is_HVfrom_fm(8,2,1,0),4,1,0)))
  have satsf:sats(M, ?f, [x,z,A,mesa])
        ⟷ (∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, is_HVfrom(##M,A)
, mesa, x, y))
    if x∈M z∈M for x z
    using that 1 ‹A∈M› ‹mesa∈M› by (simp del:pair_abs)
  have arity(?f) = 4
    unfolding fm_definitions
    by (simp add:ord_simp_union)
  then
  have strong_replacement(##M,λx z. sats(M,?f,[x,z,A,mesa]))
    using replacement_ax[of ?f] 1 ‹A∈M› ‹mesa∈M› by simp
  then
  have strong_replacement(##M,λx z.
      ∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, is_HVfrom(##M,A) , mesa,
x, y))
    using repl_sats[of M ?f [A,mesa]]  satsf by (simp del:pair_abs)
  then
  have wfrec_replacement(##M,is_HVfrom(##M,A),mesa)
    unfolding wfrec_replacement_def  by simp
 }
 then show ?thesis unfolding transrec_replacement_def
  using ‹i∈M› memrel_eclose_sing by simp
qed

sublocale M_ZF_trans ⊆ M_eclose_pow ##M
 using power_ax powapply_repl phrank_repl trans_repl_HVFrom
  wfrec_rank by unfold_locales auto
```

16.5 Interface for proving Collects and Replace in M.

```
context M_ZF_trans
begin

lemma Collect_in_M :
 assumes
  φ ∈ formula env∈list(M)
  arity(φ) ≤ 1 #+ length(env) A∈M and
  satsQ: ⋀x. x∈M ⟹ sats(M,φ,[x]@env) ⟷ Q(x)
 shows
  {y∈A . Q(y)}∈M
proof -
```

have *separation*(*##M*,*λx. sats*(*M*,*φ*,[*x*] @ *env*))
using *assms separation_ax* **by** *simp*
then show *?thesis* **using**
‹*A∈M*› *satsQ trans_M*
separation_cong[*of ##M λy. sats*(*M*,*φ*,[*y*]@*env*) *Q*]
separation_closed **by** *simp*
qed

— This version has a weaker assumption.
lemma *separation_in_M* :
assumes
φ ∈ formula env∈list(*M*)
arity(*φ*) ≤ *1 #+ length*(*env*) *A∈M* **and**
satsQ: ⋀*x. x∈A ⟹ sats*(*M*,*φ*,[*x*]@*env*) ⟷ *Q*(*x*)
shows
{*y∈A . Q*(*y*)} ∈ *M*
proof -
let *?φ′* = *And*(*φ*,*Member*(*0*,*length*(*env*)*#+1*))
have *arity*(*?φ′*) ≤ *1 #+ length*(*env*@[*A*])
using *assms Un_le*
le_trans[*of arity*(*φ*) *1#+length*(*env*) *2#+length*(*env*)]
by *force*
moreover from *assms*
have *?φ′∈formula*
nth(*length*(*env*), *env* @ [*A*]) = *A* **using** *assms nth_append* **by** *auto*
moreover from *calculation*
have ⋀ *x . x ∈ M ⟹ sats*(*M*,*?φ′*,[*x*]@*env*@[*A*]) ⟷ *Q*(*x*) ∧ *x∈A*
using *arity_sats_iff*[*of _ [A] _ [_]@env*] *assms*
by *auto*
ultimately
show *?thesis* **using** *assms*
Collect_in_M[*of ?φ′ env*@[*A*] *_ λx . Q*(*x*) ∧ *x∈A*, *OF _ _ _ ‹A∈M›*]
by *auto*
qed

lemma *Replace_in_M* :
assumes
f_fm: *φ ∈ formula* **and**
f_ar: *arity*(*φ*)≤ *2 #+ length*(*env*) **and**
fsats: ⋀*x y. x∈A ⟹ y∈M ⟹* (*M*,[*x*,*y*]@*env* ⊨ *φ*) ⟷ *y* = *f*(*x*) **and**
fclosed: ⋀*x. x∈A ⟹ f*(*x*) ∈ *M* **and**
A∈M env∈list(*M*)
shows {*f*(*x*) . *x∈A*}∈*M*
proof -
let *?φ′* = *And*(*φ*,*Member*(*0*,*length*(*env*)*#+2*))
have *arity*(*?φ′*) ≤ *2 #+ length*(*env*@[*A*])
using *assms Un_le*
le_trans[*of arity*(*φ*) *2#+*(*length*(*env*)) *3#+length*(*env*)]
by *force*

moreover from *assms*
have *?φ′∈formula nth(length(env), env @ [A]) = A*
using *assms nth_append* **by** *auto*
moreover from *calculation*
have $\bigwedge x\ y.\ x \in M \Longrightarrow y \in M \Longrightarrow (M,[x,y]@env@[A] \models ?\varphi') \longleftrightarrow y = f(x) \wedge x \in A$
using *arity_sats_iff*[*of* _ [*A*] _ [_,_]@*env*] *assms*
by *auto*
moreover from *calculation*
have *strong_replacement*(##*M*, $\lambda x\ y.\ M,[x,y]@env@[A] \models ?\varphi'$)
using *replacement_ax*[*of ?φ′*] ‹*env∈list*(*M*)› *assms* **by** *simp*
ultimately
have *4:strong_replacement*(##*M*, $\lambda x\ y.\ y = f(x) \wedge x \in A$)
using
strong_replacement_cong[*of* ##*M* $\lambda x\ y.\ M,[x,y]@env@[A] \models ?\varphi'\ \lambda x\ y.\ y = f(x) \wedge x \in A$]
by *simp*
then
have $\{y\ .\ x \in A\ ,\ y = f(x)\} \in M$
using ‹*A∈M*› *strong_replacement_closed*[*OF 4*,*of A*] *fclosed* **by** *simp*
moreover
have $\{f(x).\ x \in A\} = \{\ y\ .\ x \in A\ ,\ y = f(x)\}$
by *auto*
ultimately show *?thesis* **by** *simp*
qed

lemma *Replace_relativized_in_M* :
assumes
f_fm: $\varphi \in formula$ **and**
f_ar: $arity(\varphi) \le 2\ \#+\ length(env)$ **and**
fsats: $\bigwedge x\ y.\ x \in A \Longrightarrow y \in M \Longrightarrow (M,[x,y]@env \models \varphi) \longleftrightarrow is\_f(x,y)$ **and**
fabs: $\bigwedge x\ y.\ x \in A \Longrightarrow y \in M \Longrightarrow is\_f(x,y) \longleftrightarrow y = f(x)$ **and**
fclosed: $\bigwedge x.\ x \in A \Longrightarrow f(x) \in M$ **and**
$A \in M\ env \in list(M)$
shows $\{f(x)\ .\ x \in A\} \in M$
using *assms Replace_in_M*[*of* φ] **by** *auto*

definition $\varrho\_repl :: i \Rightarrow i$ **where**
$\varrho\_repl(l) \equiv rsum(\{\langle 0,\ 1\rangle, \langle 1,\ 0\rangle\}, id(l), 2, 3, l)$

lemma *f_type* : $\{\langle 0,\ 1\rangle, \langle 1,\ 0\rangle\} \in 2 \to 3$
using *Pi_iff* **unfolding** *function_def* **by** *auto*

lemma *ren_type* :
assumes $l \in nat$
shows $\varrho\_repl(l) : 2\#+l \to 3\#+l$
using *sum_type*[*of 2 3 l l* $\{\langle 0,\ 1\rangle, \langle 1,\ 0\rangle\}$ *id*(*l*)] *f_type assms id_type*
unfolding ϱ_repl_def **by** *auto*

lemma *ren_action* :

```
  assumes
    env∈list(M) x∈M y∈M z∈M
  shows ∀ i . i < 2#+length(env) ⟶
        nth(i,[x,z]@env) = nth(ϱ_repl(length(env))‘i,[z,x,y]@env)
proof -
  let ?f={⟨0, 1⟩, ⟨1, 0⟩}
  have 1:(⋀j. j < length(env) ⟹ nth(j, env) = nth(id(length(env)) ‘ j, env))
    using assms ltD by simp
  have 2:nth(j, [x,z]) = nth(?f ‘ j, [z,x,y]) if j<2 for j
  proof -
    consider j=0 | j=1 using  ltD[OF ‹j<2›] by auto
    then show ?thesis
    proof(cases)
      case 1
      then show ?thesis using apply_equality f_type by simp
    next
      case 2
      then show ?thesis using apply_equality f_type by simp
    qed
  qed
  show ?thesis
   using sum_action[OF _ _ _ _ f_type id_type _ _ _ _ _ _ _ _ 2 1,simplified]
assms
    unfolding ϱ_repl_def by simp
qed

lemma Lambda_in_M :
  assumes
    f_fm:  φ ∈ formula and
    f_ar:  arity(φ)≤ 2 #+ length(env) and
    fsats: ⋀x y. x∈A ⟹ y∈M ⟹ (M,[x,y]@env ⊨ φ) ⟷ is_f(x,y) and
    fabs:  ⋀x y. x∈A ⟹ y∈M ⟹ is_f(x,y) ⟷ y = f(x) and
    fclosed: ⋀x. x∈A ⟹ f(x) ∈ M and
    A∈M env∈list(M)
  shows (λx∈A . f(x)) ∈M
  unfolding lam_def
proof -
  let ?ren=ϱ_repl(length(env))
  let ?j=2#+length(env)
  let ?k=3#+length(env)
  let ?ψ=ren(φ)‘?j‘?k‘?ren
  let ?φ'=Exists(And(pair_fm(1,0,2),?ψ))
  let ?p=λx y. ∃z∈M. pair(##M,x,z,y) ∧ is_f(x,z)
  have ?φ'∈formula ?ψ∈formula
   using ‹env∈_› length_type f_fm ren_type ren_tc[of φ 2#+length(env) 3#+length(env)
?ren]
    by simp_all
  moreover from this
  have arity(?ψ)≤3#+(length(env)) arity(?ψ)∈nat
```

```
    using assms arity_ren[OF f_fm _ _ ren_type,of length(env)] by simp_all
  then
  have arity(?φ') ≤ 2#+(length(env))
    using arity_pair_fm Un_le pred_Un_distrib assms pred_le
    by simp
  moreover from this calculation
  have x∈A ⟹ y∈M ⟹ (M,[x,y]@env ⊨ ?φ') ⟷ ?p(x,y) for x y
    using ‹env∈_› length_type[OF ‹env∈_›] assms transitivity[OF _ ‹A∈M›]
     sats_iff_sats_ren[OF f_fm _ _ _ _ ren_type f_ar ren_action[rule_format,of
_ x y],of _ M ]
    by auto
  moreover
  have x∈A ⟹ y∈M ⟹ ?p(x,y) ⟷ y = <x,f(x)> for x y
    using assms transitivity[OF _ ‹A∈_›] fclosed
    by simp
  moreover
  have ⋀ x . x∈A ⟹ <x,f(x)> ∈ M
    using transitivity[OF _ ‹A∈M›] pair_in_M_iff fclosed by simp
  ultimately
  show {⟨x,f(x)⟩ . x∈A } ∈ M
    using Replace_in_M[of ?φ'] ‹A∈M› ‹env∈_›
    by simp
qed

definition ϱ_pair_repl :: i⇒i where
  ϱ_pair_repl(l) ≡ rsum({⟨0, 0⟩, ⟨1, 1⟩, ⟨2, 3⟩}, id(l), 3, 4, l)

lemma f_type' : {⟨0,0 ⟩, ⟨1, 1⟩, ⟨2, 3⟩} ∈ 3 → 4
  using Pi_iff unfolding function_def by auto

lemma ren_type' :
  assumes l∈nat
  shows ϱ_pair_repl(l) : 3#+l → 4#+l
  using sum_type[of 3 4 l l {⟨0, 0⟩, ⟨1, 1⟩, ⟨2, 3⟩} id(l)] f_type' assms id_type
  unfolding ϱ_pair_repl_def by auto

lemma ren_action' :
  assumes
    env∈list(M) x∈M y∈M z∈M u∈M
  shows ∀ i . i < 3#+length(env) ⟶
        nth(i,[x,z,u]@env) = nth(ϱ_pair_repl(length(env))`i,[x,z,y,u]@env)
proof -
  let ?f={⟨0, 0⟩, ⟨1, 1⟩, ⟨2,3⟩}
  have 1:(⋀j. j < length(env) ⟹ nth(j, env) = nth(id(length(env)) ` j, env))
    using assms ltD by simp
  have 2:nth(j, [x,z,u]) = nth(?f ` j, [x,z,y,u]) if j<3 for j
  proof -
    consider j=0 | j=1 | j=2 using  ltD[OF ‹j<3›] by auto
    then show ?thesis
```

```
    proof(cases)
      case 1
      then show ?thesis using apply_equality f_type' by simp
    next
      case 2
      then show ?thesis using apply_equality f_type' by simp
    next
      case 3
      then show ?thesis using apply_equality f_type' by simp
    qed
  qed
  show ?thesis
   using sum_action[OF _ _ _ _ f_type' id_type _ _ _ _ _ _ _ _ 2 1,simplified]
assms
    unfolding ϱ_pair_repl_def by simp
qed

lemma LambdaPair_in_M :
  assumes
    f_fm: φ ∈ formula and
    f_ar: arity(φ)≤ 3 #+ length(env) and
   fsats: ⋀x z r. x∈M ⟹ z∈M ⟹ r∈M ⟹ (M,[x,z,r]@env ⊨ φ) ⟷ is_f(x,z,r)
and
    fabs: ⋀x z r. x∈M ⟹ z∈M ⟹ r∈M ⟹ is_f(x,z,r) ⟷ r = f(x,z) and
    fclosed: ⋀x z. x∈M ⟹ z∈M ⟹ f(x,z) ∈ M and
    A∈M env∈list(M)
  shows (λx∈A . f(fst(x),snd(x))) ∈M
proof -
  let ?ren=ϱ_pair_repl(length(env))
  let ?j=3#+length(env)
  let ?k=4#+length(env)
  let ?ψ=ren(φ)‘?j‘?k‘?ren
  let ?φ'=Exists(Exists(And(fst_fm(2,0),(And(snd_fm(2,1),?ψ)))))
  let ?p=λx y. is_f(fst(x),snd(x),y)
  have ?φ'∈formula ?ψ∈formula
    using ‹env∈_› length_type f_fm ren_type' ren_tc[of φ ?j ?k ?ren]
    by simp_all
  moreover from this
  have arity(?ψ)≤4#+(length(env)) arity(?ψ)∈nat
    using assms arity_ren[OF f_fm _ _ ren_type',of length(env)] by simp_all
  moreover from calculation
  have 1:arity(?φ') ≤ 2#+(length(env)) ?φ'∈formula
    using arity_fst_fm arity_snd_fm Un_le pred_Un_distrib assms pred_le
    by simp_all
  moreover from this calculation
  have 2:x∈A ⟹ y∈M ⟹ (M,[x,y]@env ⊨ ?φ') ⟷ ?p(x,y) for x y
    using
      sats_iff_sats_ren[OF f_fm _ _ _ _ ren_type' f_ar
        ren_action'[rule_format,of _ fst(x) x snd(x) y],simplified]
```

```
      ‹env∈_› length_type[OF ‹env∈_›] transitivity[OF _ ‹A∈M›]
      fst_snd_closed pair_in_M_iff fsats[of fst(x) snd(x) y,symmetric]
      fst_abs snd_abs
    by auto
  moreover from assms
  have 3:x∈A ⟹ y∈M ⟹ ?p(x,y) ⟷ y = f(fst(x),snd(x)) for x y
    using fclosed fst_snd_closed pair_in_M_iff fabs transitivity
    by auto
  moreover
  have 4:⋀ x . x∈A ⟹ <x,f(fst(x),snd(x))> ∈ M ⋀ x . x∈A ⟹ f(fst(x),snd(x))
∈ M
    using transitivity[OF _ ‹A∈M›] pair_in_M_iff fclosed fst_snd_closed
    by simp_all
  ultimately
  show ?thesis
    using Lambda_in_M[of ?φ'] ‹env∈_› ‹A∈_› by simp
qed

end

end
```

17 Transitive set models of ZF

This theory defines the locale *M_ZF_trans* for transitive models of ZF, and the associated *forcing_data* that adds a forcing notion

```
theory Forcing_Data
  imports
    Forcing_Notions
    Interface
begin

locale M_ctm = M_ZF_trans +
  fixes enum
  assumes M_countable:      enum∈bij(nat,M)
begin


end

locale M_ctm_AC = M_ctm + M_ZFC_trans
```

17.1 A forcing locale and generic filters

```
locale forcing_data = forcing_notion + M_ctm +
  assumes P_in_M:           P ∈ M
    and leq_in_M:         leq ∈ M
```

begin

lemma *P_sub_M* : $P \subseteq M$
using *transitivity*[*OF* _ *P_in_M*] **by** *auto*

definition
$M\_generic :: i \Rightarrow o$ **where**
$M\_generic(G) \equiv filter(G) \wedge (\forall\, D{\in}M.\ D{\subseteq}P \wedge dense(D) \longrightarrow D \cap G \neq 0)$

lemma *M_genericD* [*dest*]: $M\_generic(G) \Longrightarrow x{\in}G \Longrightarrow x{\in}P$
unfolding *M_generic_def* **by** (*blast dest*:*filterD*)

lemma *M_generic_leqD* [*dest*]: $M\_generic(G) \Longrightarrow p{\in}G \Longrightarrow q{\in}P \Longrightarrow p \preceq q \Longrightarrow q{\in}G$
unfolding *M_generic_def* **by** (*blast dest*:*filter_leqD*)

lemma *M_generic_compatD* [*dest*]: $M\_generic(G) \Longrightarrow p{\in}G \Longrightarrow r{\in}G \Longrightarrow \exists\, q{\in}G.\ q \preceq p \wedge q \preceq r$
unfolding *M_generic_def* **by** (*blast dest*:*low_bound_filter*)

lemma *M_generic_denseD* [*dest*]: $M\_generic(G) \Longrightarrow dense(D) \Longrightarrow D{\subseteq}P \Longrightarrow D{\in}M \Longrightarrow \exists\, q{\in}G.\ q{\in}D$
unfolding *M_generic_def* **by** *blast*

lemma *G_nonempty*: $M\_generic(G) \Longrightarrow G{\neq}0$
proof -
have $P{\subseteq}P$ **..**
assume
$M\_generic(G)$
with *P_in_M P_dense* ‹$P{\subseteq}P$› **show**
$G \neq 0$
unfolding *M_generic_def* **by** *auto*
qed

lemma *one_in_G* :
assumes $M\_generic(G)$
shows $one \in G$
proof -
from *assms* **have** $G{\subseteq}P$
unfolding *M_generic_def* **and** *filter_def* **by** *simp*
from ‹$M\_generic(G)$› **have** $increasing(G)$
unfolding *M_generic_def* **and** *filter_def* **by** *simp*
with ‹$G{\subseteq}P$› **and** ‹$M\_generic(G)$›
show *?thesis*
using *G_nonempty* **and** *one_in_P* **and** *one_max*
unfolding *increasing_def* **by** *blast*
qed

lemma *G_subset_M*: *M_generic*(*G*) $\Longrightarrow$ *G* $\subseteq$ *M*
using *transitivity*[*OF* _ *P_in_M*] **by** *auto*

declare *iff_trans* [*trans*]

lemma *generic_filter_existence*:
p$\in$*P* $\Longrightarrow$ $\exists$ *G*. *p*$\in$*G* $\wedge$ *M_generic*(*G*)
proof -
assume *p*$\in$*P*
let *?D*=λ*n*$\in$*nat*. (*if* (*enum*‘*n*$\subseteq$*P* $\wedge$ *dense*(*enum*‘*n*)) *then enum*‘*n else P*)
have $\forall$ *n*$\in$*nat*. *?D*‘*n* $\in$ *Pow*(*P*)
by *auto*
then
have *?D*:*nat*$\rightarrow$*Pow*(*P*)
using *lam_type* **by** *auto*
have *Eq4*: $\forall$ *n*$\in$*nat*. *dense*(*?D*‘*n*)
proof(*intro ballI*)
fix *n*
assume *n*$\in$*nat*
then
have *dense*(*?D*‘*n*) $\longleftrightarrow$ *dense*(*if enum*‘*n* $\subseteq$ *P* $\wedge$ *dense*(*enum*‘*n*) *then enum*‘*n else P*)
by *simp*
also
have ... $\longleftrightarrow$ (¬(*enum*‘*n* $\subseteq$ *P* $\wedge$ *dense*(*enum*‘*n*)) $\longrightarrow$ *dense*(*P*))
using *split_if* **by** *simp*
finally
show *dense*(*?D*‘*n*)
using *P_dense* ‹*n*$\in$*nat*› **by** *auto*
qed
from ‹*?D*$\in$_› **and** *Eq4*
interpret *cg*: *countable_generic P leq one ?D*
by (*unfold_locales*, *auto*)
from ‹*p*$\in$*P*›
obtain *G* **where** *Eq6*: *p*$\in$*G* $\wedge$ *filter*(*G*) $\wedge$ ($\forall$ *n*$\in$*nat*.(*?D*‘*n*)$\cap$*G*$\neq$*0*)
using *cg.countable_rasiowa_sikorski*[**where** *M*=λ_. *M*] *P_sub_M*
M_countable[*THEN bij_is_fun*] *M_countable*[*THEN bij_is_surj*, *THEN surj_range*]
unfolding *cg.D_generic_def* **by** *blast*
then
have *Eq7*: ($\forall$ *D*$\in$*M*. *D*$\subseteq$*P* $\wedge$ *dense*(*D*)$\longrightarrow$*D*$\cap$*G*$\neq$*0*)
proof (*intro ballI impI*)
fix *D*
assume *D*$\in$*M* **and** *Eq9*: *D* $\subseteq$ *P* $\wedge$ *dense*(*D*)
have $\forall$ *y*$\in$*M*. $\exists$ *x*$\in$*nat*. *enum*‘*x*= *y*
using *M_countable* **and** *bij_is_surj* **unfolding** *surj_def* **by** (*simp*)
with ‹*D*$\in$*M*› **obtain** *n* **where** *Eq10*: *n*$\in$*nat* $\wedge$ *enum*‘*n* = *D*
by *auto*
with *Eq9* **and** *if_P*

have *?D`n = D* **by** (*simp*)
with *Eq6* **and** *Eq10*
show *D∩G≠0* **by** *auto*
qed
with *Eq6*
show *?thesis* **unfolding** *M_generic_def* **by** *auto*
qed

lemma *one_in_M*: *one ∈ M*
by (*insert one_in_P P_in_M*, *simp add*: *transitivity*)

end

lemma (**in** *M_trivial*) *compat_in_abs* :
assumes
M(A) M(r) M(p) M(q)
shows
is_compat_in(M,A,r,p,q) ⟷ compat_in(A,r,p,q)
using *assms* **unfolding** *is_compat_in_def compat_in_def* **by** *simp*

context *forcing_data* **begin**

definition
compat_in_fm :: [*i*,*i*,*i*,*i*] ⇒ *i* **where**
compat_in_fm(A,r,p,q) ≡
Exists(And(Member(0,succ(A)),Exists(And(pair_fm(1,p#+2,0),
And(Member(0,r#+2),
Exists(And(pair_fm(2,q#+3,0),Member(0,r#+3))))))))

lemma *compat_in_fm_type*[*TC*] :
⟦ *A∈nat*;*r∈nat*;*p∈nat*;*q∈nat*⟧ ⟹ *compat_in_fm(A,r,p,q)∈formula*
unfolding *compat_in_fm_def* **by** *simp*

lemma *sats_compat_in_fm*:
assumes
A∈nat r∈nat p∈nat q∈nat env∈list(M)
shows
sats(M,compat_in_fm(A,r,p,q),env) ⟷
is_compat_in(##M,nth(A, env),nth(r, env),nth(p, env),nth(q, env))
unfolding *compat_in_fm_def is_compat_in_def* **using** *assms* **by** *simp*

end

end

18 Names and generic extensions

theory *Names*

imports
Forcing_Data
Interface
Recursion_Thms
Relativization
Discipline_Base
Synthetic_Definition
ZF_Miscellanea
begin

18.1 The well-founded relation *ed*

lemma *eclose_sing* : $x \in eclose(a) \Longrightarrow x \in eclose(\{a\})$
by(*rule subsetD*[*OF mem_eclose_subset*],*simp+*)

lemma *ecloseE* :
assumes $x \in eclose(A)$
shows $x \in A \vee (\exists\ B \in A\ .\ x \in eclose(B))$
using *assms*
proof (*induct rule*:*eclose_induct_down*)
case (*1 y*)
then
show *?case*
using *arg_into_eclose* **by** *auto*
next
case (*2 y z*)
from ‹$y \in A \vee (\exists\, B{\in}A.\ y \in eclose(B))$›
consider (*inA*) $y \in A$ | (*exB*) $(\exists\, B{\in}A.\ y \in eclose(B))$
by *auto*
then show *?case*
proof (*cases*)
case *inA*
then
show *?thesis* **using** *2 arg_into_eclose* **by** *auto*
next
case *exB*
then obtain *B* **where** $y \in eclose(B)$ $B{\in}A$
by *auto*
then
show *?thesis* **using** *2 ecloseD*[*of y B z*] **by** *auto*
qed
qed

lemma *eclose_singE* : $x \in eclose(\{a\}) \Longrightarrow x = a \vee x \in eclose(a)$
by(*blast dest*: *ecloseE*)

lemma *in_eclose_sing* :
assumes $x \in eclose(\{a\})$ $a \in eclose(z)$
shows $x \in eclose(\{z\})$

proof -
  from ‹$x \in eclose(\{a\})$›
  consider (eq) $x=a$ | (lt) $x \in eclose(a)$
    using *eclose_singE* **by** *auto*
  then
  show *?thesis*
    using *eclose_sing mem_eclose_trans assms*
    by (*cases*, *auto*)
qed

lemma *in_dom_in_eclose* :
  assumes $x \in domain(z)$
  shows $x \in eclose(z)$
proof -
  from *assms*
  obtain y **where** $\langle x,y\rangle \in z$
    unfolding *domain_def* **by** *auto*
  then
  show *?thesis*
    unfolding *Pair_def*
    using *ecloseD*[*of* $\{x,x\}$] *ecloseD*[*of* $\{\{x,x\},\{x,y\}\}$] *arg_into_eclose*
    by *auto*
qed

term*ed* is the well-founded relation on which *val* is defined.

definition
  $ed :: [i,i] \Rightarrow o$ **where**
  $ed(x,y) \equiv x \in domain(y)$

definition
  $edrel :: i \Rightarrow i$ **where**
  $edrel(A) \equiv Rrel(ed,A)$

lemma *edI*[*intro*!]: $t \in domain(x) \Longrightarrow ed(t,x)$
  unfolding *ed_def* **.**

lemma *edD*[*dest*!]: $ed(t,x) \Longrightarrow t \in domain(x)$
  unfolding *ed_def* **.**

lemma *rank_ed*:
  assumes $ed(y,x)$
  shows $succ(rank(y)) \leq rank(x)$
proof
  from *assms*
  obtain p **where** $\langle y,p\rangle \in x$ **by** *auto*
  moreover
  obtain s **where** $y \in s$ $s \in \langle y,p\rangle$ **unfolding** *Pair_def* **by** *auto*

ultimately
have *rank*(*y*) < *rank*(*s*) *rank*(*s*) < *rank*(⟨*y*,*p*⟩) *rank*(⟨*y*,*p*⟩) < *rank*(*x*)
using *rank_lt* **by** *blast*+
then
show *rank*(*y*) < *rank*(*x*)
using *lt_trans* **by** *blast*
qed

lemma *edrel_dest* [*dest*]: *x* ∈ *edrel*(*A*) ⟹ ∃ *a*∈ *A*. ∃ *b* ∈ *A*. *x* =⟨*a*,*b*⟩
by(*auto simp add*:*ed_def edrel_def Rrel_def*)

lemma *edrelD* : *x* ∈ *edrel*(*A*) ⟹ ∃ *a*∈ *A*. ∃ *b* ∈ *A*. *x* =⟨*a*,*b*⟩ ∧ *a* ∈ *domain*(*b*)
by(*auto simp add*:*ed_def edrel_def Rrel_def*)

lemma *edrelI* [*intro*!]: *x*∈*A* ⟹ *y*∈*A* ⟹ *x* ∈ *domain*(*y*) ⟹ ⟨*x*,*y*⟩∈*edrel*(*A*)
by (*simp add*:*ed_def edrel_def Rrel_def*)

lemma *edrel_trans*: *Transset*(*A*) ⟹ *y*∈*A* ⟹ *x* ∈ *domain*(*y*) ⟹ ⟨*x*,*y*⟩∈*edrel*(*A*)
by (*rule edrelI*, *auto simp add*:*Transset_def domain_def Pair_def*)

lemma *domain_trans*: *Transset*(*A*) ⟹ *y*∈*A* ⟹ *x* ∈ *domain*(*y*) ⟹ *x*∈*A*
by (*auto simp add*: *Transset_def domain_def Pair_def*)

lemma *relation_edrel* : *relation*(*edrel*(*A*))
by(*auto simp add*: *relation_def*)

lemma *field_edrel* : *field*(*edrel*(*A*))⊆*A*
by *blast*

lemma *edrel_sub_memrel*: *edrel*(*A*) ⊆ *trancl*(*Memrel*(*eclose*(*A*)))
proof
fix *z*
assume
z∈*edrel*(*A*)
then obtain *x y* **where**
Eq1: *x*∈*A* *y*∈*A* *z*=⟨*x*,*y*⟩ *x*∈*domain*(*y*)
using *edrelD*
by *blast*
then obtain *u v* **where**
Eq2: *x*∈*u* *u*∈*v* *v*∈*y*
unfolding *domain_def Pair_def* **by** *auto*
with *Eq1* **have**
Eq3: *x*∈*eclose*(*A*) *y*∈*eclose*(*A*) *u*∈*eclose*(*A*) *v*∈*eclose*(*A*)
by (*auto*, *rule_tac* [*3-4*] *ecloseD*, *rule_tac* [*3*] *ecloseD*, *simp_all add*:*arg_into_eclose*)
let
?r=*trancl*(*Memrel*(*eclose*(*A*)))
from *Eq2* **and** *Eq3* **have**
⟨*x*,*u*⟩∈*?r* ⟨*u*,*v*⟩∈*?r* ⟨*v*,*y*⟩∈*?r*
by (*auto simp add*: *r_into_trancl*)

```
  then  have
    ⟨x,y⟩∈?r
    by (rule_tac trancl_trans, rule_tac [2] trancl_trans, simp)
  with Eq1 show z∈?r by simp
qed

lemma wf_edrel : wf(edrel(A))
  using wf_subset [of trancl(Memrel(eclose(A)))] edrel_sub_memrel
    wf_trancl wf_Memrel
  by auto

lemma ed_induction:
  assumes ⋀x. ⟦⋀y.  ed(y,x) ⟹ Q(y) ⟧ ⟹ Q(x)
  shows Q(a)
proof(induct rule: wf_induct2[OF wf_edrel[of eclose({a})] ,of a eclose({a})])
  case 1
  then show ?case using arg_into_eclose by simp
next
  case 2
  then show ?case using field_edrel .
next
  case (3 x)
  then
  show ?case
    using assms[of x] edrelI domain_trans[OF Transset_eclose 3(1)] by blast
qed

lemma dom_under_edrel_eclose: edrel(eclose({x})) -`` {x} = domain(x)
proof
  show edrel(eclose({x})) -`` {x} ⊆ domain(x)
    unfolding edrel_def Rrel_def ed_def
    by auto
next
  show domain(x) ⊆ edrel(eclose({x})) -`` {x}
    unfolding edrel_def Rrel_def
    using in_dom_in_eclose eclose_sing arg_into_eclose
    by blast
qed

lemma ed_eclose : ⟨y,z⟩ ∈ edrel(A) ⟹ y ∈ eclose(z)
  by(drule edrelD,auto simp add:domain_def in_dom_in_eclose)

lemma tr_edrel_eclose : ⟨y,z⟩ ∈ edrel(eclose({x}))^+ ⟹ y ∈ eclose(z)
  by(rule trancl_induct,(simp add: ed_eclose mem_eclose_trans)+)


lemma restrict_edrel_eq :
  assumes z ∈ domain(x)
  shows edrel(eclose({x})) ∩ eclose({z})×eclose({z}) = edrel(eclose({z}))
```

proof(*intro equalityI subsetI*)
let *?ec*=λ *y* . *edrel*(*eclose*({*y*}))
let *?ez*=*eclose*({*z*})
let *?rr*=*?ec*(*x*) $\cap$ *?ez* $\times$ *?ez*
fix *y*
assume *yr*:*y* $\in$ *?rr*
with *yr* **obtain** *a b* **where** *1*:$\langle a,b\rangle$ $\in$ *?rr a* $\in$ *?ez b* $\in$ *?ez* $\langle a,b\rangle$ $\in$ *?ec*(*x*) *y*=$\langle a,b\rangle$
by *blast*
moreover
from *this*
have *a* $\in$ *domain*(*b*) **using** *edrelD* **by** *blast*
ultimately
show *y* $\in$ *edrel*(*eclose*({*z*})) **by** *blast*
next
let *?ec*=λ *y* . *edrel*(*eclose*({*y*}))
let *?ez*=*eclose*({*z*})
let *?rr*=*?ec*(*x*) $\cap$ *?ez* $\times$ *?ez*
fix *y*
assume *yr*:*y* $\in$ *edrel*(*?ez*)
then obtain *a b* **where** *a* $\in$ *?ez b* $\in$ *?ez y*=$\langle a,b\rangle$ *a* $\in$ *domain*(*b*)
using *edrelD* **by** *blast*
moreover
from *this assms*
have *z* $\in$ *eclose*(*x*) **using** *in_dom_in_eclose* **by** *simp*
moreover
from *assms calculation*
have *a* $\in$ *eclose*({*x*}) *b* $\in$ *eclose*({*x*}) **using** *in_eclose_sing* **by** *simp_all*
moreover
from *this* ‹*a*$\in$*domain*(*b*)›
have $\langle a,b\rangle$ $\in$ *edrel*(*eclose*({*x*})) **by** *blast*
ultimately
show *y* $\in$ *?rr* **by** *simp*
qed

lemma *tr_edrel_subset* :
assumes *z* $\in$ *domain*(*x*)
shows *tr_down*(*edrel*(*eclose*({*x*})),*z*) $\subseteq$ *eclose*({*z*})
proof(*intro subsetI*)
let *?r*=λ *x* . *edrel*(*eclose*({*x*}))
fix *y*
assume *y* $\in$ *tr_down*(*?r*(*x*),*z*)
then
have $\langle y,z\rangle$ $\in$ *?r*(*x*)$^\frown$+ **using** *tr_downD* **by** *simp*
with *assms*
show *y* $\in$ *eclose*({*z*}) **using** *tr_edrel_eclose eclose_sing* **by** *simp*
qed

definition
Hv :: [*i*,*i*,*i*,*i*]$\Rightarrow$*i* **where**

$Hv(P,G,x,f) \equiv \{\ f`y\ ..\ y{\in}\ domain(x),\ \exists\, p{\in}P.\ \langle y,p\rangle \in x \wedge p \in G\ \}$

The funcion *val* interprets a name in M according to a (generic) filter G. Note the definition in terms of the well-founded recursor.

definition
$val :: [i,i,i]{\Rightarrow}i$ **where**
$val(P,G,\tau) \equiv wfrec(edrel(eclose(\{\tau\})),\ \tau\ ,Hv(P,G))$

definition
$GenExt :: [i,i,i]{\Rightarrow}i \quad (\_-[\_]\ [71,1])$
where $M^P[G] \equiv \{val(P,G,\tau).\ \tau \in M\}$

abbreviation (**in** *forcing_notion*)
$GenExt\_at\_P :: i{\Rightarrow}i{\Rightarrow}i \quad (\_[\_]\ [71,1])$
where $M[G] \equiv M^P[G]$

18.2 Values and check-names

context *forcing_data*
begin

definition
$Hcheck :: [i,i] \Rightarrow i$ **where**
$Hcheck(z,f)\ \equiv \{\ \langle f`y,one\rangle\ .\ y \in z\}$

definition
$check :: i \Rightarrow i$ **where**
$check(x) \equiv transrec(x\ ,\ Hcheck)$

lemma *checkD*:
$check(x) =\ \ wfrec(Memrel(eclose(\{x\})),\ x,\ Hcheck)$
unfolding *check_def transrec_def* **..**

definition
$rcheck :: i \Rightarrow i$ **where**
$rcheck(x) \equiv Memrel(eclose(\{x\}))^{+}$

lemma *Hcheck_trancl*:$Hcheck(y,\ restrict(f,Memrel(eclose(\{x\}))\text{-``}\{y\}))$
$= Hcheck(y,\ restrict(f,(Memrel(eclose(\{x\}))^{+})\text{-``}\{y\}))$
unfolding *Hcheck_def*
using *restrict_trans_eq* **by** *simp*

lemma *check_trancl*: $check(x) = wfrec(rcheck(x),\ x,\ Hcheck)$
using *checkD wf_eq_trancl Hcheck_trancl* **unfolding** *rcheck_def* **by** *simp*

lemma *rcheck_in_M* :
$x \in M \Longrightarrow rcheck(x) \in M$
unfolding *rcheck_def* **by** (*simp flip*: *setclass_iff*)

lemma *aux_def_check*: $x \in y \Longrightarrow$
wfrec(*Memrel*(*eclose*({*y*})), *x*, *Hcheck*) =
wfrec(*Memrel*(*eclose*({*x*})), *x*, *Hcheck*)
by (*rule wfrec_eclose_eq*,*auto simp add*: *arg_into_eclose eclose_sing*)

lemma *def_check* : *check*(*y*) = { ⟨*check*(*w*),*one*⟩ . *w* ∈ *y*}
proof -
let
?r=λ*y*. *Memrel*(*eclose*({*y*}))
have *wfr*: ∀ *w* . *wf*(*?r*(*w*))
using *wf_Memrel* **..**
then
have *check*(*y*)= *Hcheck*(*y*, λ*x*∈*?r*(*y*) -‘‘ {*y*}. *wfrec*(*?r*(*y*), *x*, *Hcheck*))
using *wfrec*[*of ?r*(*y*) *y Hcheck*] *checkD* **by** *simp*
also
have ... = *Hcheck*(*y*, λ*x*∈*y*. *wfrec*(*?r*(*y*), *x*, *Hcheck*))
using *under_Memrel_eclose arg_into_eclose* **by** *simp*
also
have ... = *Hcheck*(*y*, λ*x*∈*y*. *check*(*x*))
using *aux_def_check checkD* **by** *simp*
finally show *?thesis* **using** *Hcheck_def* **by** *simp*
qed

lemma *def_checkS* :
fixes *n*
assumes *n* ∈ *nat*
shows *check*(*succ*(*n*)) = *check*(*n*) ∪ {⟨*check*(*n*),*one*⟩}
proof -
have *check*(*succ*(*n*)) = {⟨*check*(*i*),*one*⟩ . *i* ∈ *succ*(*n*)}
using *def_check* **by** *blast*
also have ... = {⟨*check*(*i*),*one*⟩ . *i* ∈ *n*} ∪ {⟨*check*(*n*),*one*⟩}
by *blast*
also have ... = *check*(*n*) ∪ {⟨*check*(*n*),*one*⟩}
using *def_check*[*of n*,*symmetric*] **by** *simp*
finally show *?thesis* **.**
qed

lemma *field_Memrel2* :
assumes *x* ∈ *M*
shows *field*(*Memrel*(*eclose*({*x*}))) ⊆ *M*
proof -
have *field*(*Memrel*(*eclose*({*x*}))) ⊆ *eclose*({*x*}) *eclose*({*x*}) ⊆ *M*
using *Ordinal.Memrel_type field_rel_subset assms eclose_least*[*OF trans_M*]
by *auto*
then
show *?thesis* **using** *subset_trans* **by** *simp*
qed

lemma *aux_def_val*:
assumes $z \in domain(x)$
shows $wfrec(edrel(eclose(\{x\})),z,Hv(P,G)) = wfrec(edrel(eclose(\{z\})),z,Hv(P,G))$
proof -
let $?r=\lambda x\ .\ edrel(eclose(\{x\}))$
have $z\in eclose(\{z\})$ **using** *arg_in_eclose_sing* **.**
moreover
have $relation(?r(x))$ **using** *relation_edrel* **.**
moreover
have $wf(?r(x))$ **using** *wf_edrel* **.**
moreover from *assms*
have $tr\_down(?r(x),z) \subseteq eclose(\{z\})$ **using** *tr_edrel_subset* **by** *simp*
ultimately
have $wfrec(?r(x),z,Hv(P,G)) = wfrec[eclose(\{z\})](?r(x),z,Hv(P,G))$
using *wfrec_restr* **by** *simp*
also from ‹$z\in domain(x)$›
have $... = wfrec(?r(z),z,Hv(P,G))$
using *restrict_edrel_eq wfrec_restr_eq* **by** *simp*
finally show *?thesis* **.**
qed

The next lemma provides the usual recursive expresion for the definition of term*val*.

lemma *def_val*: $val(P,G,x) = \{val(P,G,t)\ ..\ t\in domain(x)\ ,\ \exists\, p\in P\ .\ \langle t,p\rangle\in x \wedge p \in G\ \}$
proof -
let
$?r=\lambda\tau\ .\ edrel(eclose(\{\tau\}))$
let
$?f=\lambda z\in ?r(x)\text{-``}\{x\}.\ wfrec(?r(x),z,Hv(P,G))$
have $\forall\,\tau.\ wf(?r(\tau))$ **using** *wf_edrel* **by** *simp*
with *wfrec* [*of* _ *x*]
have $val(P,G,x) = Hv(P,G,x,?f)$ **using** *val_def* **by** *simp*
also
have $... = Hv(P,G,x,\lambda z\in domain(x).\ wfrec(?r(x),z,Hv(P,G)))$
using *dom_under_edrel_eclose* **by** *simp*
also
have $... = Hv(P,G,x,\lambda z\in domain(x).\ val(P,G,z))$
using *aux_def_val val_def* **by** *simp*
finally
show *?thesis* **using** *Hv_def SepReplace_def* **by** *simp*
qed

lemma *val_mono* : $x\subseteq y \Longrightarrow val(P,G,x) \subseteq val(P,G,y)$
by (*subst* (*1 2*) *def_val*, *force*)

Check-names are the canonical names for elements of the ground model. Here we show that this is the case.

lemma *valcheck* : $one \in G \Longrightarrow\ one \in P \Longrightarrow val(P,G,check(y))\ = y$
proof (*induct rule:eps_induct*)
case (*1 y*)
then show *?case*
proof -
have $check(y) = \{\ \langle check(w),\ one\rangle\ .\ w \in y\}$ (**is** _ = *?C*)
using *def_check* **.**
then
have $val(P,G,check(y)) = val(P,G,\ \{\langle check(w),\ one\rangle\ .\ w \in y\})$
by *simp*
also
have $...\ = \{val(P,G,t)\ ..\ t{\in}domain(?C)\ ,\ \exists\, p{\in}P\ .\ \langle t,\ p\rangle{\in}?C \wedge p \in G\ \}$
using *def_val* **by** *blast*
also
have $...\ = \ \{val(P,G,t)\ ..\ t{\in}domain(?C)\ ,\ \exists\, w{\in}y.\ t{=}check(w)\ \}$
using *1* **by** *simp*
also
have $...\ = \{val(P,G,check(w))\ .\ w{\in}y\ \}$
by *force*
finally
show $val(P,G,check(y)) = y$
using *1* **by** *simp*
qed
qed

lemma *val_of_name* :
$val(P,G,\{x{\in}A{\times}P.\ Q(x)\}) = \{val(P,G,t)\ ..\ t{\in}A\ ,\ \exists\, p{\in}P\ .\ \ Q(\langle t,p\rangle) \wedge p \in G\ \}$
proof -
let
$?n{=}\{x{\in}A{\times}P.\ Q(x)\}$ **and**
$?r{=}\lambda\tau\ .\ edrel(eclose(\{\tau\}))$
let
$?f{=}\lambda z{\in}?r(?n)\text{-}``\{?n\}.\ val(P,G,z)$
have
wfR : $wf(?r(\tau))$ **for** τ
by (*simp add*: *wf_edrel*)
have $domain(?n) \subseteq A$ **by** *auto*
{ fix *t*
assume *H*:$t \in domain(\{x \in A \times P\ .\ Q(x)\})$
then have $?f\ `\ t = (if\ t \in\ ?r(?n)\text{-}``\{?n\}\ then\ val(P,G,t)\ else\ 0)$
by *simp*
moreover have ... = $val(P,G,t)$
using *dom_under_edrel_eclose H if_P* **by** *auto*
}
then
have *Eq1*: $t \in domain(\{x \in A \times P\ .\ Q(x)\}) \Longrightarrow val(P,G,t) = ?f`\ t$ **for** *t*
by *simp*
have $val(P,G,?n) = \{val(P,G,t)\ ..\ t{\in}domain(?n),\ \exists\, p \in P\ .\ \langle t,p\rangle \in ?n \wedge p \in G\}$

```
    by (subst def_val,simp)
  also
  have ... = {?f`t .. t∈domain(?n), ∃p∈P . ⟨t,p⟩∈?n ∧ p∈G}
    unfolding Hv_def
    by (subst SepReplace_dom_implies,auto simp add:Eq1)
  also
  have  ... = { (if t∈?r(?n)-``{?n} then val(P,G,t) else 0) .. t∈domain(?n), ∃p∈P
. ⟨t,p⟩∈?n ∧ p∈G}
    by (simp)
  also
  have Eq2:  ... = { val(P,G,t) .. t∈domain(?n), ∃p∈P . ⟨t,p⟩∈?n ∧ p∈G}
  proof -
    have domain(?n) ⊆ ?r(?n)-``{?n}
      using dom_under_edrel_eclose by simp
    then
    have ∀t∈domain(?n). (if t∈?r(?n)-``{?n} then val(P,G,t) else 0) = val(P,G,t)
      by auto
    then
    show { (if t∈?r(?n)-``{?n} then val(P,G,t) else 0) .. t∈domain(?n), ∃p∈P .
⟨t,p⟩∈?n ∧ p∈G} =
          { val(P,G,t) .. t∈domain(?n), ∃p∈P . ⟨t,p⟩∈?n ∧ p∈G}
      by auto
  qed
  also
  have  ... = { val(P,G,t) .. t∈A, ∃p∈P . ⟨t,p⟩∈?n ∧ p∈G}
    by force
  finally
  show  val(P,G,?n)  = { val(P,G,t) .. t∈A, ∃p∈P . Q(⟨t,p⟩) ∧ p∈G}
    by auto
qed

lemma val_of_name_alt :
  val(P,G,{x∈A×P. Q(x)}) = {val(P,G,t) .. t∈A , ∃p∈P∩G .  Q(⟨t,p⟩) }
  using val_of_name by force

lemma val_only_names: val(P,F,τ) = val(P,F,{x∈τ. ∃t∈domain(τ). ∃p∈P. x=⟨t,p⟩})
  (is _ = val(P,F,?name))
proof -
  have val(P,F,?name) = {val(P,F, t).. t∈domain(?name), ∃p∈P. ⟨t, p⟩ ∈ ?name
∧ p ∈ F}
    using def_val by blast
  also
  have  ... = {val(P,F, t). t∈{y∈domain(?name). ∃p∈P. ⟨y, p⟩ ∈ ?name ∧ p ∈
F}}
    using Sep_and_Replace by simp
  also
  have  ... = {val(P,F, t). t∈{y∈domain(τ). ∃p∈P. ⟨y, p⟩ ∈ τ ∧ p ∈ F}}
    by blast
  also
```

have $... = \{val(P,F,\ t)..\ t{\in}domain(\tau),\ \exists\, p{\in}P.\ \langle t,\ p\rangle \in \tau \wedge p \in F\}$
using *Sep_and_Replace* **by** *simp*
also
have $... = val(P,F,\ \tau)$
using *def_val*[*symmetric*] **by** *blast*
finally
show *?thesis* **..**
qed

lemma *val_only_pairs*: $val(P,F,\tau) = val(P,F,\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\})$
proof
have $val(P,F,\tau) = val(P,F,\{x{\in}\tau.\ \exists\, t{\in}domain(\tau).\ \exists\, p{\in}P.\ x{=}\langle t,p\rangle\})$
(**is** $\_ = val(P,F,?name)$)
using *val_only_names* **.**
also
have $... \subseteq val(P,F,\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\})$
using *val_mono*[*of ?name* $\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\}$] **by** *auto*
finally
show $val(P,F,\tau) \subseteq val(P,F,\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\})$ **by** *simp*
next
show $val(P,F,\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\}) \subseteq val(P,F,\tau)$
using *val_mono*[*of* $\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\}$] **by** *auto*
qed

lemma *val_subset_domain_times_range*: $val(P,F,\tau) \subseteq val(P,F,domain(\tau){\times}range(\tau))$
using *val_only_pairs*[*THEN equalityD1*]
val_mono[*of* $\{x \in \tau\ .\ \exists\, t\ p.\ x = \langle t,\ p\rangle\}$ $domain(\tau){\times}range(\tau)$] **by** *blast*

lemma *val_subset_domain_times_P*: $val(P,F,\tau) \subseteq val(P,F,domain(\tau){\times}P)$
using *val_only_names*[*of F* τ] *val_mono*[*of* $\{x{\in}\tau.\ \exists\, t{\in}domain(\tau).\ \exists\, p{\in}P.\ x{=}\langle t,p\rangle\}$
$domain(\tau){\times}P$ *F*]
by *auto*

lemma *val_of_elem*: $\langle\vartheta,p\rangle \in \pi \Longrightarrow p{\in}G \Longrightarrow p{\in}P \Longrightarrow val(P,G,\vartheta) \in val(P,G,\pi)$
proof -
assume
$\langle\vartheta,p\rangle \in \pi$
then
have $\vartheta{\in}domain(\pi)$ **by** *auto*
assume $p{\in}G$ $p{\in}P$
with ‹$\vartheta{\in}domain(\pi)$› ‹$\langle\vartheta,p\rangle \in \pi$›
have $val(P,G,\vartheta) \in \{val(P,G,t)\ ..\ t{\in}domain(\pi)\ ,\ \exists\, p{\in}P\ .\ \ \langle t,\ p\rangle{\in}\pi \wedge p \in G\ \}$
by *auto*
then
show *?thesis* **by** (*subst def_val*)
qed

lemma *elem_of_val*: $x{\in}val(P,G,\pi) \Longrightarrow \exists\, \vartheta{\in}domain(\pi).\ val(P,G,\vartheta) = x$
by (*subst* (*asm*) *def_val*,*auto*)

lemma *elem_of_val_pair*: $x{\in}val(P,G,\pi) \Longrightarrow \exists\vartheta.\ \exists p{\in}G.\ \ \langle\vartheta,p\rangle{\in}\pi \wedge val(P,G,\vartheta) = x$
by (*subst* (*asm*) *def_val*,*auto*)

lemma *elem_of_val_pair'*:
assumes $\pi{\in}M$ $x{\in}val(P,G,\pi)$
shows $\exists\vartheta{\in}M.\ \exists p{\in}G.\ \ \langle\vartheta,p\rangle{\in}\pi \wedge val(P,G,\vartheta) = x$
proof -
from *assms*
obtain ϑ p **where** $p{\in}G$ $\langle\vartheta,p\rangle{\in}\pi$ $val(P,G,\vartheta) = x$
using *elem_of_val_pair* **by** *blast*
moreover from *this* ‹$\pi{\in}M$›
have $\vartheta{\in}M$
using *pair_in_M_iff*[*THEN iffD1*, *THEN conjunct1*, *simplified*]
transitivity **by** *blast*
ultimately
show *?thesis* **by** *blast*
qed

lemma *GenExtD*:
$x \in M[G] \Longrightarrow \exists\tau{\in}M.\ x = val(P,G,\tau)$
by (*simp add*:*GenExt_def*)

lemma *GenExtI*:
$x \in M \Longrightarrow val(P,G,x) \in M[G]$
by (*auto simp add*: *GenExt_def*)

lemma *Transset_MG* : *Transset*($M[G]$)
proof -
{ fix *vc y*
assume $vc \in M[G]$ **and** $y \in vc$
then obtain c **where** $c{\in}M$ $val(P,G,c){\in}M[G]$ $y \in val(P,G,c)$
using *GenExtD* **by** *auto*
from ‹$y \in val(P,G,c)$›
obtain ϑ **where** $\vartheta{\in}domain(c)$ $val(P,G,\vartheta) = y$
using *elem_of_val* **by** *blast*
with *trans_M* ‹$c{\in}M$›
have $y \in M[G]$
using *domain_trans GenExtI* **by** *blast*
}
then
show *?thesis* **using** *Transset_def* **by** *auto*
qed

lemmas *transitivity_MG* = *Transset_intf*[*OF Transset_MG*]

lemma *check_n_M* :

fixes n
assumes $n \in nat$
shows $check(n) \in M$
using ‹$n \in nat$›
proof (*induct* n)
case *0*
then show *?case* **using** *zero_in_M* **by** (*subst def_check*,*simp*)
next
case (*succ* x)
have $one \in M$ **using** *one_in_P P_sub_M subsetD* **by** *simp*
with ‹$check(x) \in M$›
have $\langle check(x), one\rangle \in M$
using *pair_in_M_iff* **by** *simp*
then
have $\{\langle check(x), one\rangle\} \in M$
using *singleton_closed* **by** *simp*
with ‹$check(x) \in M$›
have $check(x) \cup \{\langle check(x), one\rangle\} \in M$
using *Un_closed* **by** *simp*
then show *?case* **using** ‹$x \in nat$› *def_checkS* **by** *simp*
qed

definition
$PHcheck :: [i,i,i,i] \Rightarrow o$ **where**
$PHcheck(o,f,y,p) \equiv p \in M \wedge (\exists fy[\#\#M].\ fun\_apply(\#\#M,f,y,fy) \wedge pair(\#\#M,fy,o,p))$

definition
$is\_Hcheck :: [i,i,i,i] \Rightarrow o$ **where**
$is\_Hcheck(o,z,f,hc) \equiv is\_Replace(\#\#M,z,PHcheck(o,f),hc)$

lemma *def_PHcheck*:
assumes
$z \in M\ f \in M$
shows
$Hcheck(z,f) = Replace(z,PHcheck(one,f))$
proof -
from *assms*
have $\langle f`x, one\rangle \in M\ f`x \in M$ **if** $x \in z$ **for** x
using *pair_in_M_iff one_in_M transitivity that apply_closed* **by** *simp_all*
then
have $\{y\ .\ x \in z,\ y = \langle f\ `\ x,\ one\rangle\} = \{y\ .\ x \in z,\ y = \langle f\ `\ x,\ one\rangle \wedge y \in M \wedge f`x \in M\}$
by *simp*
then
show *?thesis*
using ‹$z \in M$› ‹$f \in M$› *transitivity*
unfolding *Hcheck_def PHcheck_def RepFun_def*
by *auto*

qed

definition
PHcheck_fm :: $[i,i,i,i] \Rightarrow i$ **where**
PHcheck_fm(o,f,y,p) $\equiv$ *Exists*(*And*(*fun_apply_fm*(*succ*(f),*succ*(y),0)
,*pair_fm*(0,*succ*(o),*succ*(p))))

declare *PHcheck_fm_def*[*fm_definitions*]

lemma *PHcheck_type* [*TC*]:
$[\![\ x \in nat;\ y \in nat;\ z \in nat;\ u \in nat\ ]\!] \Longrightarrow$ *PHcheck_fm*(x,y,z,u) $\in$ *formula*
by (*simp add*:*PHcheck_fm_def*)

lemma *sats_PHcheck_fm* [*simp*]:
$[\![\ x \in nat;\ y \in nat;\ z \in nat;\ u \in nat\ ;\ env \in list(M)]\!]$
$\Longrightarrow$ *sats*(M,*PHcheck_fm*(x,y,z,u),*env*) $\longleftrightarrow$
PHcheck(*nth*(x,*env*),*nth*(y,*env*),*nth*(z,*env*),*nth*(u,*env*))
using *zero_in_M Internalizations.nth_closed* **by** (*simp add*: *PHcheck_def PHcheck_fm_def*)

definition
is_Hcheck_fm :: $[i,i,i,i] \Rightarrow i$ **where**
is_Hcheck_fm(o,z,f,hc) $\equiv$ *Replace_fm*(z,*PHcheck_fm*(*succ*(*succ*(o)),*succ*(*succ*(f)),0,1),hc)

declare *is_Hcheck_fm_def* [*fm_definitions*]

lemma *is_Hcheck_type* [*TC*]:
$[\![\ x \in nat;\ y \in nat;\ z \in nat;\ u \in nat\ ]\!] \Longrightarrow$ *is_Hcheck_fm*(x,y,z,u) $\in$ *formula*
by (*simp add*:*is_Hcheck_fm_def*)

lemma *sats_is_Hcheck_fm* [*simp*]:
$[\![\ x \in nat;\ y \in nat;\ z \in nat;\ u \in nat\ ;\ env \in list(M)]\!]$
$\Longrightarrow$ *sats*(M,*is_Hcheck_fm*(x,y,z,u),*env*) $\longleftrightarrow$
is_Hcheck(*nth*(x,*env*),*nth*(y,*env*),*nth*(z,*env*),*nth*(u,*env*))
using *sats_Replace_fm* **unfolding** *is_Hcheck_def is_Hcheck_fm_def*
by *simp*

lemma *wfrec_Hcheck* :
assumes
$X \in M$
shows
wfrec_replacement(##M,*is_Hcheck*(*one*),*rcheck*(X))
proof -
have *is_Hcheck*(*one*,a,b,c) $\longleftrightarrow$
sats(M,*is_Hcheck_fm*(8,2,1,0),[c,b,a,d,e,y,x,z,*one*,*rcheck*(x)])
if $a \in M$ $b \in M$ $c \in M$ $d \in M$ $e \in M$ $y \in M$ $x \in M$ $z \in M$

```
    for a b c d e y x z
    using that one_in_M ‹X∈M› rcheck_in_M by simp
  then have 1:sats(M,is_wfrec_fm(is_Hcheck_fm(8,2,1,0),4,1,0),
                  [y,x,z,one,rcheck(X)]) ⟷
          is_wfrec(##M, is_Hcheck(one),rcheck(X), x, y)
    if x∈M y∈M z∈M for x y z
    using  that sats_is_wfrec_fm ‹X∈M› rcheck_in_M one_in_M by simp
  let
    ?f=Exists(And(pair_fm(1,0,2),
              is_wfrec_fm(is_Hcheck_fm(8,2,1,0),4,1,0)))
  have satsf:sats(M, ?f, [x,z,one,rcheck(X)]) ⟷
         (∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, is_Hcheck(one),rcheck(X),
x, y))
    if x∈M z∈M for x z
    using that 1 ‹X∈M› rcheck_in_M one_in_M by (simp del:pair_abs)
  have artyf:arity(?f) = 4
    unfolding fm_definitions
    by (simp add:ord_simp_union)
  then
  have strong_replacement(##M,λx z. sats(M,?f,[x,z,one,rcheck(X)]))
   using replacement_ax[of ?f] 1 artyf ‹X∈M› rcheck_in_M one_in_M by simp
  then
  have strong_replacement(##M,λx z.
         ∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, is_Hcheck(one),rcheck(X),
x, y))
    using repl_sats[of M ?f [one,rcheck(X)]] satsf by (simp del:pair_abs)
  then
  show ?thesis unfolding wfrec_replacement_def by simp
qed

lemma repl_PHcheck :
  assumes
    f∈M
  shows
    strong_replacement(##M,PHcheck(one,f))
proof -
  have arity(PHcheck_fm(2,3,0,1)) = 4
   unfolding PHcheck_fm_def fun_apply_fm_def big_union_fm_def pair_fm_def
image_fm_def
      upair_fm_def
    by (simp add:ord_simp_union)
  with ‹f∈M›
  have strong_replacement(##M,λx y. sats(M,PHcheck_fm(2,3,0,1),[x,y,one,f]))
    using replacement_ax[of PHcheck_fm(2,3,0,1)] one_in_M by simp
  with ‹f∈M›
 show ?thesis using one_in_M unfolding strong_replacement_def univalent_def
by simp
qed
```

lemma *univ_PHcheck* : ⟦ *z*∈*M* ; *f*∈*M* ⟧ ⟹ *univalent*(*##M*,*z*,*PHcheck*(*one*,*f*))
unfolding *univalent_def PHcheck_def* **by** *simp*

lemma *relation2_Hcheck* :
relation2(*##M*,*is_Hcheck*(*one*),*Hcheck*)
proof -
have *1*:⟦*x*∈*z*; *PHcheck*(*one*,*f*,*x*,*y*) ⟧ ⟹ (*##M*)(*y*)
if *z*∈*M f*∈*M* **for** *z f x y*
using *that* **unfolding** *PHcheck_def* **by** *simp*
have *is_Replace*(*##M*,*z*,*PHcheck*(*one*,*f*),*hc*) ⟷ *hc* = *Replace*(*z*,*PHcheck*(*one*,*f*))
if *z*∈*M f*∈*M hc*∈*M* **for** *z f hc*
using *that Replace_abs*[*OF _ _ univ_PHcheck 1*] **by** *simp*
with *def_PHcheck*
show *?thesis*
unfolding *relation2_def is_Hcheck_def Hcheck_def* **by** *simp*
qed

lemma *PHcheck_closed* :
⟦*z*∈*M* ; *f*∈*M* ; *x*∈*z*; *PHcheck*(*one*,*f*,*x*,*y*) ⟧ ⟹ (*##M*)(*y*)
unfolding *PHcheck_def* **by** *simp*

lemma *Hcheck_closed* :
∀ *y*∈*M*. ∀ *g*∈*M*. *function*(*g*) ⟶ *Hcheck*(*y*,*g*)∈*M*
proof -
have *Replace*(*y*,*PHcheck*(*one*,*f*))∈*M* **if** *f*∈*M y*∈*M* **for** *f y*
using *that repl_PHcheck PHcheck_closed*[*of y f*] *univ_PHcheck*
strong_replacement_closed
by (*simp flip*: *setclass_iff*)
then show *?thesis* **using** *def_PHcheck* **by** *auto*
qed

lemma *wf_rcheck* : *x*∈*M* ⟹ *wf*(*rcheck*(*x*))
unfolding *rcheck_def* **using** *wf_trancl*[*OF wf_Memrel*] **.**

lemma *trans_rcheck* : *x*∈*M* ⟹ *trans*(*rcheck*(*x*))
unfolding *rcheck_def* **using** *trans_trancl* **.**

lemma *relation_rcheck* : *x*∈*M* ⟹ *relation*(*rcheck*(*x*))
unfolding *rcheck_def* **using** *relation_trancl* **.**

lemma *check_in_M* : *x*∈*M* ⟹ *check*(*x*) ∈ *M*
unfolding *transrec_def*
using *wfrec_Hcheck*[*of x*] *check_trancl wf_rcheck trans_rcheck relation_rcheck*
rcheck_in_M
Hcheck_closed relation2_Hcheck trans_wfrec_closed[*of rcheck*(*x*) *x is_Hcheck*(*one*)
Hcheck]
by (*simp flip*: *setclass_iff*)

end

context *forcing_data* **begin**

definition
is_rcheck :: [*i*,*i*] ⇒ *o* **where**
is_rcheck(*x*,*z*) ≡ ∃ *r*∈*M*. *tran_closure*(##*M*,*r*,*z*) ∧ (∃ *ec*∈*M*. *membership*(##*M*,*ec*,*r*) ∧
(∃ *s*∈*M*. *is_singleton*(##*M*,*x*,*s*) ∧ *is_eclose*(##*M*,*s*,*ec*)))

lemma *rcheck_abs*[*Rel*] :
⟦ *x*∈*M* ; *r*∈*M* ⟧ ⟹ *is_rcheck*(*x*,*r*) ⟷ *r* = *rcheck*(*x*)
unfolding *rcheck_def is_rcheck_def*
using *singleton_closed trancl_closed Memrel_closed eclose_closed* **by** *simp*

schematic_goal *rcheck_fm_auto*:
assumes
i ∈ *nat j* ∈ *nat env* ∈ *list*(*M*)
shows
is_rcheck(*nth*(*i*,*env*),*nth*(*j*,*env*)) ⟷ *sats*(*M*,*?rch*(*i*,*j*),*env*)
unfolding *is_rcheck_def*
by (*insert assms* ; (*rule sep_rules singleton_iff_sats is_eclose_iff_sats*
trans_closure_iff_sats | *simp*)+)

synthesize *rcheck* **from_schematic** *rcheck_fm_auto*
arity_theorem for *rcheck_fm*

definition
is_check :: [*i*,*i*] ⇒ *o* **where**
is_check(*x*,*z*) ≡ ∃ *rch*∈*M*. *is_rcheck*(*x*,*rch*) ∧ *is_wfrec*(##*M*,*is_Hcheck*(*one*),*rch*,*x*,*z*)

lemma *check_abs*[*Rel*] :
assumes
x∈*M z*∈*M*
shows
is_check(*x*,*z*) ⟷ *z* = *check*(*x*)
proof -
have
is_check(*x*,*z*) ⟷ *is_wfrec*(##*M*,*is_Hcheck*(*one*),*rcheck*(*x*),*x*,*z*)
unfolding *is_check_def* **using** *assms rcheck_abs rcheck_in_M*
unfolding *check_trancl is_check_def* **by** *simp*
then show *?thesis*
unfolding *check_trancl*
using *assms wfrec_Hcheck*[*of x*] *wf_rcheck trans_rcheck relation_rcheck rcheck_in_M*
Hcheck_closed relation2_Hcheck trans_wfrec_abs[*of rcheck*(*x*) *x z is_Hcheck*(*one*)
Hcheck]
by (*simp flip*: *setclass_iff*)
qed

definition
check_fm :: $[i,i,i] \Rightarrow i$ **where**
[*fm_definitions*] :
check_fm$(x,o,z) \equiv$ *Exists*(*And*(*rcheck_fm*(1#+x,0),
is_wfrec_fm(*is_Hcheck_fm*(6#+o,2,1,0),0,1#+x,1#+z)))

notation *check_fm* ($\langle\cdot$_ ^{v} _ *is* _$\cdot\rangle$)

lemma *check_fm_type*[*TC*] :
$[\![x\in nat; o\in nat; z\in nat]\!] \Longrightarrow$ *check_fm*$(x,o,z)\in$*formula*
unfolding *check_fm_def* **by** *simp*

arity_theorem for *PHcheck_fm*

lemma *arity_is_Hcheck_fm* :
assumes $m\in nat$ $n\in nat$ $p\in nat$ $o\in nat$
shows *arity*(*is_Hcheck_fm*(m,n,p,o)) = *succ*(o) $\cup$ *succ*(n) $\cup$ *succ*(p) $\cup$ *succ*(m)
unfolding *is_Hcheck_fm_def*
using *assms arity_Replace_fm*[*rule_format*,*OF PHcheck_type _ _ _ arity_PHcheck_fm*]
pred_Un_distrib Un_assoc Un_nat_type
by *simp*

lemma *arity_check_fm* :
assumes $m\in nat$ $n\in nat$ $o\in nat$
shows *arity*(*check_fm*(m,n,o)) = *succ*(o) $\cup$ *succ*(n) $\cup$ *succ*(m)
unfolding *check_fm_def*
using *assms arity_is_wfrec_fm*[*rule_format*,*OF _ _ _ _ _ arity_is_Hcheck_fm*]
pred_Un_distrib Un_assoc arity_tran_closure_fm
by (*auto simp add*:*arity*)

lemma *sats_check_fm* :
assumes
nth(o,*env*) = *one* $x\in nat$ $z\in nat$ $o\in nat$ *env*$\in$*list*(M) $x <$ *length*(*env*) $z <$ *length*(*env*)
shows
sats(M, *check_fm*(x,o,z), *env*) $\longleftrightarrow$ *is_check*(*nth*(x,*env*),*nth*(z,*env*))
proof -
have *sats_is_Hcheck_fm*:
$\bigwedge a0\ a1\ a2\ a3\ a4.\ [\![\ a0\in M;\ a1\in M;\ a2\in M;\ a3\in M;\ a4\in M\ ]\!] \Longrightarrow$
is_Hcheck(*one*,*a2*, *a1*, *a0*) $\longleftrightarrow$
sats(M, *is_Hcheck_fm*(6#+o,2,1,0), [*a0*,*a1*,*a2*,*a3*,*a4*,r]@*env*) **if** $r\in M$ **for** r
using *that one_in_M assms* **by** *simp*
then
have *sats*(M, *is_wfrec_fm*(*is_Hcheck_fm*(6#+o,2,1,0),0,1#+x,1#+z),*Cons*(r,*env*))
$\longleftrightarrow$ *is_wfrec*(##M,*is_Hcheck*(*one*),r,*nth*(x,*env*),*nth*(z,*env*)) **if** $r\in M$ **for** r
using *that assms one_in_M sats_is_wfrec_fm* **by** *simp*

then
show *?thesis* **unfolding** *is_check_def check_fm_def*
using *assms rcheck_in_M one_in_M rcheck_iff_sats*[*symmetric*] **by** *simp*
qed

lemma *check_replacement*:
$\{check(x).\ x{\in}P\} \in M$
proof -
have $arity(check\_fm(0,2,1)) = 3$
unfolding *eclose_n_fm_def is_eclose_fm_def mem_eclose_fm_def fm_definitions*
by (*simp add*:*ord_simp_union*)
moreover
have $check(x){\in}M$ **if** $x{\in}P$ **for** x
using *that transitivity check_in_M P_in_M* **by** *simp*
ultimately
show *?thesis* **using** *sats_check_fm check_abs P_in_M check_in_M one_in_M transitivity*
Replace_relativized_in_M[*of check_fm*(*0*,*2*,*1*) [*one*] _ *is_check check*] **by** *simp*
qed

lemma *pair_check* : $[\![\ p{\in}M\ ;\ y{\in}M\ ]\!] \Longrightarrow (\exists\, c{\in}M.\ is\_check(p,c) \wedge pair(\#\#M,c,p,y)) \longleftrightarrow y = \langle check(p),p\rangle$
using *check_abs check_in_M pair_in_M_iff* **by** *simp*

lemma *M_subset_MG* : $one \in G \Longrightarrow M \subseteq M[G]$
using *check_in_M one_in_P GenExtI*
by (*intro subsetI*, *subst valcheck* [*of G*,*symmetric*], *auto*)

The name for the generic filter

definition
$G\_dot :: i$ **where**
$G\_dot \equiv \{\langle check(p),p\rangle\ .\ p{\in}P\}$

lemma *G_dot_in_M* :
$G\_dot \in M$
proof -
let *?is_pcheck* $= \lambda x\ y.\ \exists\, ch{\in}M.\ is\_check(x,ch) \wedge pair(\#\#M,ch,x,y)$
let *?pcheck_fm* $= Exists(And(check\_fm(1,3,0),pair\_fm(0,1,2)))$
have $sats(M,?pcheck\_fm,[x,y,one]) \longleftrightarrow$ *?is_pcheck*(x,y) **if** $x{\in}M\ y{\in}M$ **for** $x\ y$
using *sats_check_fm that one_in_M* **by** *simp*
moreover
have *?is_pcheck*$(x,y) \longleftrightarrow y = \langle check(x),x\rangle$ **if** $x{\in}M\ y{\in}M$ **for** $x\ y$
using *that check_abs check_in_M* **by** *simp*
moreover
have *?pcheck_fm*$\in$*formula* **by** *simp*
moreover

```
  have arity(?pcheck_fm)=3
   unfolding is_eclose_fm_def mem_eclose_fm_def eclose_n_fm_def fm_definitions
    by (simp add:ord_simp_union)
  moreover
  from P_in_M check_in_M pair_in_M_iff P_sub_M
  have ⟨check(p),p⟩ ∈ M if p∈P for p
    using that by auto
  ultimately
  show ?thesis
    unfolding G_dot_def
   using one_in_M P_in_M transitivity Replace_relativized_in_M[of ?pcheck_fm
[one]]
    by simp
qed

lemma val_G_dot :
  assumes G ⊆ P
    one ∈ G
  shows val(P,G,G_dot) = G
proof (intro equalityI subsetI)
  fix x
  assume x∈val(P,G,G_dot)
  then obtain ϑ p where p∈G ⟨ϑ,p⟩ ∈ G_dot val(P,G,ϑ) = x ϑ = check(p)
    unfolding G_dot_def using elem_of_val_pair G_dot_in_M
    by force
  with ‹one∈G› ‹G⊆P› show
    x ∈ G
    using valcheck P_sub_M by auto
next
  fix p
  assume p∈G
  have ⟨check(q),q⟩ ∈ G_dot if q∈P for q
    unfolding G_dot_def using that by simp
  with ‹p∈G› ‹G⊆P›
  have val(P,G,check(p)) ∈ val(P,G,G_dot)
    using val_of_elem G_dot_in_M by blast
  with ‹p∈G› ‹G⊆P› ‹one∈G›
  show p ∈ val(P,G,G_dot)
    using P_sub_M valcheck by auto
qed


lemma G_in_Gen_Ext :
  assumes G ⊆ P and one ∈ G
  shows   G ∈ M[G]
  using assms val_G_dot GenExtI[of _ G] G_dot_in_M
  by force

end
```

locale *G_generic* = *forcing_data* +
  fixes *G* :: *i*
  assumes *generic* : *M_generic*(*G*)
begin

lemma *zero_in_MG* :
  0 ∈ *M*[*G*]
proof -
  have *0* = *val*(*P*,*G*,*0*)
    using *zero_in_M elem_of_val* **by** *auto*
  also
  have ... ∈ *M*[*G*]
    using *GenExtI zero_in_M* **by** *simp*
  finally show *?thesis* **.**
qed

lemma *G_nonempty*: *G*≠*0*
proof -
  have *P*⊆*P* **..**
  with *P_in_M P_dense* ‹*P*⊆*P*›
  show *G* ≠ *0*
    using *generic* **unfolding** *M_generic_def* **by** *auto*
qed

end

locale *G_generic_AC* = *G_generic* + *M_ctm_AC*

end

19 Well-founded relation on names

theory *FrecR*
  imports
    Names
    Synthetic_Definition
    Internalizations
    Discipline_Function
begin

frecR is the well-founded relation on names that allows us to define forcing for atomic formulas.

definition
  ftype :: *i*⇒*i* **where**
  ftype ≡ *fst*

definition
  name1 :: *i*⇒*i* **where**

name1(x) ≡ *fst*(*snd*(x))

definition
name2 :: $i \Rightarrow i$ **where**
name2(x) ≡ *fst*(*snd*(*snd*(x)))

definition
cond_of :: $i \Rightarrow i$ **where**
cond_of(x) ≡ *snd*(*snd*(*snd*((x))))

lemma *components_simp*:
ftype($\langle f,n1,n2,c\rangle$) = f
name1($\langle f,n1,n2,c\rangle$) = $n1$
name2($\langle f,n1,n2,c\rangle$) = $n2$
cond_of($\langle f,n1,n2,c\rangle$) = c
unfolding *ftype_def name1_def name2_def cond_of_def*
by *simp_all*

definition *eclose_n* :: $[i \Rightarrow i, i] \Rightarrow i$ **where**
eclose_n(*name*,x) = *eclose*($\{$*name*(x)$\}$)

definition
ecloseN :: $i \Rightarrow i$ **where**
ecloseN(x) = *eclose_n*(*name1*,x) ∪ *eclose_n*(*name2*,x)

lemma *components_in_eclose* :
$n1 \in$ *ecloseN*($\langle f,n1,n2,c\rangle$)
$n2 \in$ *ecloseN*($\langle f,n1,n2,c\rangle$)
unfolding *ecloseN_def eclose_n_def*
using *components_simp arg_into_eclose* **by** *auto*

lemmas *names_simp* = *components_simp*(*2*) *components_simp*(*3*)

lemma *ecloseNI1* :
assumes $x \in$ *eclose*(*n1*) ∨ $x \in$ *eclose*(*n2*)
shows $x \in$ *ecloseN*($\langle f,n1,n2,c\rangle$)
unfolding *ecloseN_def eclose_n_def*
using *assms eclose_sing names_simp*
by *auto*

lemmas *ecloseNI* = *ecloseNI1*

lemma *ecloseN_mono* :
assumes $u \in$ *ecloseN*(x) *name1*(x) ∈ *ecloseN*(y) *name2*(x) ∈ *ecloseN*(y)
shows $u \in$ *ecloseN*(y)
proof -
from ‹$u \in$ _›
consider (*a*) $u \in$ *eclose*($\{$*name1*(x)$\}$) | (*b*) $u \in$ *eclose*($\{$*name2*(x)$\}$)
unfolding *ecloseN_def eclose_n_def* **by** *auto*

```
  then
  show ?thesis
  proof cases
    case a
    with ‹name1(x) ∈ _›
    show ?thesis
      unfolding ecloseN_def  eclose_n_def
      using eclose_singE[OF a] mem_eclose_trans[of u name1(x) ] by auto
  next
    case b
    with ‹name2(x) ∈ _›
    show ?thesis
      unfolding ecloseN_def eclose_n_def
      using eclose_singE[OF b] mem_eclose_trans[of u name2(x)] by auto
  qed
qed

definition
  is_ftype :: (i⇒o)⇒i⇒i⇒o where
  is_ftype ≡ is_fst

definition
  ftype_fm :: [i,i] ⇒ i where
  ftype_fm ≡ fst_fm

lemma is_ftype_iff_sats [iff_sats]:
  assumes
    nth(a,env) = aa nth(b,env) = bb a∈nat b∈nat env ∈ list(A)
  shows
    is_ftype(##A,aa,bb)  ⟷ sats(A,ftype_fm(a,b), env)
  unfolding ftype_fm_def is_ftype_def
  using assms sats_fst_fm
  by simp

definition
  is_name1 :: (i⇒o)⇒i⇒i⇒o where
  is_name1(M,x,t2) ≡ is_hcomp(M,is_fst(M),is_snd(M),x,t2)

definition
  name1_fm :: [i,i] ⇒ i where
  name1_fm(x,t) ≡ hcomp_fm(fst_fm,snd_fm,x,t)

lemma sats_name1_fm [simp]:
  ⟦ x ∈ nat; y ∈ nat;env ∈ list(A) ⟧
    ⟹ sats(A, name1_fm(x,y), env) ⟷
        is_name1(##A, nth(x,env), nth(y,env))
  unfolding name1_fm_def is_name1_def using sats_fst_fm sats_snd_fm
    sats_hcomp_fm[of A is_fst(##A) _ fst_fm is_snd(##A)] by simp
```

lemma *is_name1_iff_sats* [*iff_sats*]:
assumes
nth(*a*,*env*) = *aa* *nth*(*b*,*env*) = *bb* *a*∈*nat* *b*∈*nat* *env* ∈ *list*(*A*)
shows
is_name1(##*A*,*aa*,*bb*) ⟷ *sats*(*A*,*name1_fm*(*a*,*b*), *env*)
using *assms* *sats_name1_fm*
by *simp*

definition
is_snd_snd :: (*i*⇒*o*)⇒*i*⇒*i*⇒*o* **where**
is_snd_snd(*M*,*x*,*t*) ≡ *is_hcomp*(*M*,*is_snd*(*M*),*is_snd*(*M*),*x*,*t*)

definition
snd_snd_fm :: [*i*,*i*]⇒*i* **where**
snd_snd_fm(*x*,*t*) ≡ *hcomp_fm*(*snd_fm*,*snd_fm*,*x*,*t*)

lemma *sats_snd2_fm* [*simp*]:
⟦ *x* ∈ *nat*; *y* ∈ *nat*;*env* ∈ *list*(*A*) ⟧
⟹ *sats*(*A*,*snd_snd_fm*(*x*,*y*), *env*) ⟷
is_snd_snd(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
unfolding *snd_snd_fm_def* *is_snd_snd_def* **using** *sats_snd_fm*
sats_hcomp_fm[*of* *A* *is_snd*(##*A*) _ *snd_fm* *is_snd*(##*A*)] **by** *simp*

definition
is_name2 :: (*i*⇒*o*)⇒*i*⇒*i*⇒*o* **where**
is_name2(*M*,*x*,*t3*) ≡ *is_hcomp*(*M*,*is_fst*(*M*),*is_snd_snd*(*M*),*x*,*t3*)

definition
name2_fm :: [*i*,*i*] ⇒ *i* **where**
name2_fm(*x*,*t3*) ≡ *hcomp_fm*(*fst_fm*,*snd_snd_fm*,*x*,*t3*)

lemma *sats_name2_fm* :
⟦ *x* ∈ *nat*; *y* ∈ *nat*;*env* ∈ *list*(*A*) ⟧
⟹ *sats*(*A*,*name2_fm*(*x*,*y*), *env*) ⟷
is_name2(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
unfolding *name2_fm_def* *is_name2_def* **using** *sats_fst_fm* *sats_snd2_fm*
sats_hcomp_fm[*of* *A* *is_fst*(##*A*) _ *fst_fm* *is_snd_snd*(##*A*)] **by** *simp*

lemma *is_name2_iff_sats*:
assumes
nth(*a*,*env*) = *aa* *nth*(*b*,*env*) = *bb* *a*∈*nat* *b*∈*nat* *env* ∈ *list*(*A*)
shows
is_name2(##*A*,*aa*,*bb*) ⟷ *sats*(*A*,*name2_fm*(*a*,*b*), *env*)
using *assms*
by (*simp* *add*:*sats_name2_fm*)

definition
is_cond_of :: (*i*⇒*o*)⇒*i*⇒*i*⇒*o* **where**
is_cond_of(*M*,*x*,*t4*) ≡ *is_hcomp*(*M*,*is_snd*(*M*),*is_snd_snd*(*M*),*x*,*t4*)

definition
cond_of_fm :: [*i*,*i*] ⇒ *i* **where**
cond_of_fm(*x*,*t4*) ≡ *hcomp_fm*(*snd_fm*,*snd_snd_fm*,*x*,*t4*)

lemma *sats_cond_of_fm* :
⟦ *x* ∈ *nat*; *y* ∈ *nat*;*env* ∈ *list*(*A*) ⟧
⟹ *sats*(*A*,*cond_of_fm*(*x*,*y*), *env*) ⟷
is_cond_of(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
unfolding *cond_of_fm_def is_cond_of_def* **using** *sats_snd_fm sats_snd2_fm*
sats_hcomp_fm[*of A is_snd*(##*A*) _ *snd_fm is_snd_snd*(##*A*)] **by** *simp*

lemma *is_cond_of_iff_sats*:
assumes
nth(*a*,*env*) = *aa nth*(*b*,*env*) = *bb a*∈*nat b*∈*nat env* ∈ *list*(*A*)
shows
is_cond_of(##*A*,*aa*,*bb*) ⟷ *sats*(*A*,*cond_of_fm*(*a*,*b*), *env*)
using *assms*
by (*simp add*:*sats_cond_of_fm*)

lemma *components_type*[*TC*] :
assumes *a*∈*nat b*∈*nat*
shows
ftype_fm(*a*,*b*)∈*formula*
name1_fm(*a*,*b*)∈*formula*
name2_fm(*a*,*b*)∈*formula*
cond_of_fm(*a*,*b*)∈*formula*
using *assms*
unfolding *ftype_fm_def fst_fm_def snd_fm_def snd_snd_fm_def name1_fm_def name2_fm_def*
cond_of_fm_def hcomp_fm_def
by *simp_all*

lemmas *components_iff_sats* = *is_ftype_iff_sats is_name1_iff_sats is_name2_iff_sats*
is_cond_of_iff_sats

lemmas *components_defs* = *ftype_fm_def snd_snd_fm_def hcomp_fm_def*
name1_fm_def name2_fm_def cond_of_fm_def

definition
is_eclose_n :: [*i*⇒*o*,[*i*⇒*o*,*i*,*i*]⇒*o*,*i*,*i*] ⇒ *o* **where**
is_eclose_n(*N*,*is_name*,*en*,*t*) ≡
∃ *n1*[*N*].∃ *s1*[*N*]. *is_name*(*N*,*t*,*n1*) ∧ *is_singleton*(*N*,*n1*,*s1*) ∧ *is_eclose*(*N*,*s1*,*en*)

definition
eclose_n1_fm :: [*i*,*i*] ⇒ *i* **where**
eclose_n1_fm(*m*,*t*) ≡ *Exists*(*Exists*(*And*(*And*(*name1_fm*(*t*#+2,0),*singleton_fm*(0,1)),
is_eclose_fm(1,*m*#+2))))

definition
eclose_n2_fm :: [*i*,*i*] ⇒ *i* **where**
eclose_n2_fm(*m*,*t*) ≡ *Exists*(*Exists*(*And*(*And*(*name2_fm*(*t*#+2,0),*singleton_fm*(0,1)),
is_eclose_fm(1,*m*#+2))))

definition
is_ecloseN :: [*i*⇒*o*,*i*,*i*] ⇒ *o* **where**
is_ecloseN(*N*,*t*,*en*) ≡ ∃ *en1*[*N*].∃ *en2*[*N*].
is_eclose_n(*N*,*is_name1*,*en1*,*t*) ∧ *is_eclose_n*(*N*,*is_name2*,*en2*,*t*)∧
union(*N*,*en1*,*en2*,*en*)

definition
ecloseN_fm :: [*i*,*i*] ⇒ *i* **where**
ecloseN_fm(*en*,*t*) ≡ *Exists*(*Exists*(*And*(*eclose_n1_fm*(1,*t*#+2),
And(*eclose_n2_fm*(0,*t*#+2),*union_fm*(1,0,*en*#+2)))))
lemma *ecloseN_fm_type* [*TC*] :
⟦ *en* ∈ *nat* ; *t* ∈ *nat* ⟧ ⟹ *ecloseN_fm*(*en*,*t*) ∈ *formula*
unfolding *ecloseN_fm_def eclose_n1_fm_def eclose_n2_fm_def* **by** *simp*

lemma *sats_ecloseN_fm* [*simp*]:
⟦ *en* ∈ *nat*; *t* ∈ *nat* ; *env* ∈ *list*(*A*) ⟧
⟹ *sats*(*A*, *ecloseN_fm*(*en*,*t*), *env*) ⟷ *is_ecloseN*(##*A*,*nth*(*t*,*env*),*nth*(*en*,*env*))
unfolding *ecloseN_fm_def is_ecloseN_def eclose_n1_fm_def eclose_n2_fm_def is_eclose_n_def*
using *nth_0 nth_ConsI sats_name1_fm sats_name2_fm*
singleton_iff_sats[*symmetric*]
by *auto*

lemma *is_ecloseN_iff_sats* [*iff_sats*]:
⟦ *nth*(*en*, *env*) = *ena*; *nth*(*t*, *env*) = *ta*; *en* ∈ *nat*; *t* ∈ *nat* ; *env* ∈ *list*(*A*) ⟧
⟹ *is_ecloseN*(##*A*,*ta*,*ena*) ⟷ *sats*(*A*, *ecloseN_fm*(*en*,*t*), *env*)
by *simp*

definition
frecR :: *i* ⇒ *i* ⇒ *o* **where**
frecR(*x*,*y*) ≡
(*ftype*(*x*) = 1 ∧ *ftype*(*y*) = 0
∧ (*name1*(*x*) ∈ *domain*(*name1*(*y*)) ∪ *domain*(*name2*(*y*)) ∧ (*name2*(*x*) = *name1*(*y*) ∨ *name2*(*x*) = *name2*(*y*))))
∨ (*ftype*(*x*) = 0 ∧ *ftype*(*y*) = 1 ∧ *name1*(*x*) = *name1*(*y*) ∧ *name2*(*x*) ∈ *domain*(*name2*(*y*)))

lemma *frecR_ftypeD* :
assumes *frecR*(*x*,*y*)
shows (*ftype*(*x*) = 0 ∧ *ftype*(*y*) = 1) ∨ (*ftype*(*x*) = 1 ∧ *ftype*(*y*) = 0)
using *assms* **unfolding** *frecR_def* **by** *auto*

lemma *frecRI1*: $s \in domain(n1) \vee s \in domain(n2) \Longrightarrow frecR(\langle 1, s, n1, q\rangle, \langle 0, n1, n2, q'\rangle)$
unfolding *frecR_def* **by** (*simp add*:*components_simp*)

lemma *frecRI1'*: $s \in domain(n1) \cup domain(n2) \Longrightarrow frecR(\langle 1, s, n1, q\rangle, \langle 0, n1, n2, q'\rangle)$
unfolding *frecR_def* **by** (*simp add*:*components_simp*)

lemma *frecRI2*: $s \in domain(n1) \vee s \in domain(n2) \Longrightarrow frecR(\langle 1, s, n2, q\rangle, \langle 0, n1, n2, q'\rangle)$
unfolding *frecR_def* **by** (*simp add*:*components_simp*)

lemma *frecRI2'*: $s \in domain(n1) \cup domain(n2) \Longrightarrow frecR(\langle 1, s, n2, q\rangle, \langle 0, n1, n2, q'\rangle)$
unfolding *frecR_def* **by** (*simp add*:*components_simp*)

lemma *frecRI3*: $\langle s, r\rangle \in n2 \Longrightarrow frecR(\langle 0, n1, s, q\rangle, \langle 1, n1, n2, q'\rangle)$
unfolding *frecR_def* **by** (*auto simp add*:*components_simp*)

lemma *frecRI3'*: $s \in domain(n2) \Longrightarrow frecR(\langle 0, n1, s, q\rangle, \langle 1, n1, n2, q'\rangle)$
unfolding *frecR_def* **by** (*auto simp add*:*components_simp*)

lemma *frecR_iff* :
$frecR(x,y) \longleftrightarrow$
$(ftype(x) = 1 \wedge ftype(y) = 0$
$\wedge (name1(x) \in domain(name1(y)) \cup domain(name2(y)) \wedge (name2(x) = name1(y) \vee name2(x) = name2(y))))$
$\vee (ftype(x) = 0 \wedge ftype(y) = 1 \wedge name1(x) = name1(y) \wedge name2(x) \in domain(name2(y)))$
unfolding *frecR_def* **..**

lemma *frecR_D1* :
$frecR(x,y) \Longrightarrow ftype(y) = 0 \Longrightarrow ftype(x) = 1 \wedge$
$(name1(x) \in domain(name1(y)) \cup domain(name2(y)) \wedge (name2(x) = name1(y) \vee name2(x) = name2(y)))$
using *frecR_iff*
by *auto*

lemma *frecR_D2* :
$frecR(x,y) \Longrightarrow ftype(y) = 1 \Longrightarrow ftype(x) = 0 \wedge$
$ftype(x) = 0 \wedge ftype(y) = 1 \wedge name1(x) = name1(y) \wedge name2(x) \in domain(name2(y))$
using *frecR_iff*
by *auto*

lemma *frecR_DI* :
assumes $frecR(\langle a,b,c,d\rangle,\langle ftype(y),name1(y),name2(y),cond\_of(y)\rangle)$
shows $frecR(\langle a,b,c,d\rangle,y)$

using *assms* **unfolding** *frecR_def* **by** (*force simp add*:*components_simp*)

reldb_add *ftype is_ftype*
reldb_add *name1 is_name1*
reldb_add *name2 is_name2*

relativize *frecR is_frecR*

schematic_goal *sats_frecR_fm_auto*:
assumes
i∈*nat j*∈*nat env*∈*list*(*A*)
shows
is_frecR(##*A*,*nth*(*i*,*env*),*nth*(*j*,*env*)) ⟷ *sats*(*A*,*?fr_fm*(*i*,*j*),*env*)
unfolding *is_frecR_def*
by (*insert assms* ; (*rule sep_rules*′ *cartprod_iff_sats components_iff_sats*
| *simp del*:*sats_cartprod_fm*)+)

synthesize *frecR* **from_schematic** *sats_frecR_fm_auto*

lemma *eq_ftypep_not_frecrR*:
assumes *ftype*(*x*) = *ftype*(*y*)
shows ¬ *frecR*(*x*,*y*)
using *assms frecR_ftypeD* **by** *force*

definition
rank_names :: *i* ⇒ *i* **where**
rank_names(*x*) ≡ *max*(*rank*(*name1*(*x*)),*rank*(*name2*(*x*)))

lemma *rank_names_types* [*TC*]:
shows *Ord*(*rank_names*(*x*))
unfolding *rank_names_def max_def* **using** *Ord_rank Ord_Un* **by** *auto*

definition
mtype_form :: *i* ⇒ *i* **where**
mtype_form(*x*) ≡ *if rank*(*name1*(*x*)) < *rank*(*name2*(*x*)) *then 0 else 2*

definition
type_form :: *i* ⇒ *i* **where**
type_form(*x*) ≡ *if ftype*(*x*) = *0 then 1 else mtype_form*(*x*)

lemma *type_form_tc* [*TC*]:
shows *type_form*(*x*) ∈ *3*
unfolding *type_form_def mtype_form_def* **by** *auto*

lemma *frecR_le_rnk_names* :
assumes *frecR*(*x*,*y*)
shows *rank_names*(*x*)≤*rank_names*(*y*)

```
proof -
  obtain a b c d  where
    H: a = name1(x) b = name2(x)
    c = name1(y) d = name2(y)
    (a ∈ domain(c)∪domain(d) ∧ (b=c ∨ b = d)) ∨ (a = c ∧ b ∈ domain(d))
    using assms unfolding frecR_def by force
  then
  consider
    (m) a ∈ domain(c) ∧ (b = c ∨ b = d)
    | (n) a ∈ domain(d) ∧ (b = c ∨ b = d)
    | (o) b ∈ domain(d) ∧ a = c
    by auto
  then show ?thesis proof(cases)
    case m
    then
    have rank(a) < rank(c)
      using eclose_rank_lt  in_dom_in_eclose  by simp
    with ‹rank(a) < rank(c)› H m
   show ?thesis unfolding rank_names_def using Ord_rank max_cong max_cong2
leI by auto
  next
    case n
    then
    have rank(a) < rank(d)
      using eclose_rank_lt in_dom_in_eclose  by simp
    with ‹rank(a) < rank(d)› H n
    show ?thesis unfolding rank_names_def
      using Ord_rank max_cong2 max_cong max_commutes[of rank(c) rank(d)]
leI by auto
  next
    case o
    then
    have rank(b) < rank(d) (is ?b < ?d) rank(a) = rank(c) (is ?a = _)
      using eclose_rank_lt in_dom_in_eclose  by simp_all
    with H
    show ?thesis unfolding rank_names_def
     using Ord_rank max_commutes max_cong2[OF leI[OF ‹?b < ?d›], of ?a] by
simp
  qed
qed


definition
  Γ :: i ⇒ i where
  Γ(x) = 3 ** rank_names(x) ++ type_form(x)

lemma Γ_type [TC]:
  shows Ord(Γ(x))
  unfolding Γ_def by simp
```

```
lemma Γ_mono :
  assumes frecR(x,y)
  shows Γ(x) < Γ(y)
proof -
  have F: type_form(x) < 3 type_form(y) < 3
    using ltI by simp_all
  from assms
  have A: rank_names(x) ≤ rank_names(y) (is ?x ≤ ?y)
    using frecR_le_rnk_names by simp
  then
  have Ord(?y) unfolding rank_names_def using Ord_rank max_def by simp
  note leE[OF ‹?x≤?y›]
  then
  show ?thesis
  proof(cases)
    case 1
    then
    show ?thesis unfolding Γ_def using oadd_lt_mono2 ‹?x < ?y› F by auto
  next
    case 2
    consider (a) ftype(x) = 0 ∧ ftype(y) = 1 | (b) ftype(x) = 1 ∧ ftype(y) = 0
      using frecR_ftypeD[OF ‹frecR(x,y)›] by auto
    then show ?thesis proof(cases)
      case b
      then
      have type_form(y) = 1
        using type_form_def by simp
      from b
      have H: name2(x) = name1(y) ∨ name2(x) = name2(y)  (is ?τ = ?σ' ∨ ?τ
= ?τ')
        name1(x) ∈ domain(name1(y)) ∪ domain(name2(y))
        (is ?σ ∈ domain(?σ') ∪ domain(?τ'))
        using assms unfolding type_form_def frecR_def by auto
      then
      have E: rank(?τ) = rank(?σ') ∨ rank(?τ) = rank(?τ') by auto
      from H
      consider (a) rank(?σ) < rank(?σ') | (b) rank(?σ) < rank(?τ')
        using eclose_rank_lt in_dom_in_eclose by force
      then
      have rank(?σ) < rank(?τ) proof (cases)
        case a
        with ‹rank_names(x) = rank_names(y) ›
       show ?thesis unfolding rank_names_def mtype_form_def type_form_def
using max_D2[OF E a]
           E assms Ord_rank by simp
      next
        case b
```

```
      with ‹rank_names(x) = rank_names(y) ›
      show ?thesis unfolding rank_names_def mtype_form_def type_form_def
       using max_D2[OF _ b] max_commutes E assms Ord_rank disj_commute
by auto
     qed
     with b
    have type_form(x) = 0 unfolding type_form_def mtype_form_def by simp
     with ‹rank_names(x) = rank_names(y) › ‹type_form(y) = 1› ‹type_form(x)
= 0›
     show ?thesis
      unfolding Γ_def by auto
   next
     case a
     then
     have name1(x) = name1(y) (is ?σ = ?σ')
      name2(x) ∈ domain(name2(y)) (is ?τ ∈ domain(?τ'))
      type_form(x) = 1
      using assms unfolding type_form_def frecR_def by auto
     then
     have rank(?σ) = rank(?σ') rank(?τ) < rank(?τ')
      using  eclose_rank_lt in_dom_in_eclose by simp_all
     with ‹rank_names(x) = rank_names(y) ›
     have rank(?τ') ≤ rank(?σ')
      unfolding rank_names_def using Ord_rank max_D1 by simp
     with a
     have type_form(y) = 2
      unfolding type_form_def mtype_form_def using not_lt_iff_le assms by
simp
    with ‹rank_names(x) = rank_names(y) › ‹type_form(y) = 2› ‹type_form(x)
= 1›
     show ?thesis
      unfolding Γ_def by auto
   qed
  qed
qed

definition
  frecrel :: i ⇒ i where
  frecrel(A) ≡ Rrel(frecR,A)

lemma frecrelI :
  assumes x ∈ A y∈A frecR(x,y)
  shows ⟨x,y⟩∈frecrel(A)
  using assms unfolding frecrel_def Rrel_def by auto

lemma frecrelD :
  assumes ⟨x,y⟩ ∈ frecrel(A1×A2×A3×A4)
  shows ftype(x) ∈ A1 ftype(x) ∈ A1
    name1(x) ∈ A2 name1(y) ∈ A2 name2(x) ∈ A3 name2(x) ∈ A3
```

```
    cond_of(x) ∈ A4 cond_of(y) ∈ A4
    frecR(x,y)
  using assms unfolding frecrel_def Rrel_def ftype_def by (auto simp add:components_simp)

lemma wf_frecrel :
  shows wf(frecrel(A))
proof -
  have frecrel(A) ⊆ measure(A,Γ)
    unfolding frecrel_def Rrel_def measure_def
    using Γ_mono by force
  then show ?thesis using wf_subset wf_measure by auto
qed

lemma core_induction_aux:
  fixes A1 A2 :: i
  assumes
    Transset(A1)
    ⋀τ ϑ p.  p ∈ A2 ⟹ ⟦⋀q σ. ⟦ q∈A2 ; σ∈domain(ϑ)⟧ ⟹ Q(0,τ,σ,q)⟧ ⟹
Q(1,τ,ϑ,p)
   ⋀τ ϑ p.  p ∈ A2 ⟹ ⟦⋀q σ. ⟦ q∈A2 ; σ∈domain(τ) ∪ domain(ϑ)⟧ ⟹ Q(1,σ,τ,q)
∧ Q(1,σ,ϑ,q)⟧ ⟹ Q(0,τ,ϑ,p)
  shows a∈2×A1×A1×A2 ⟹ Q(ftype(a),name1(a),name2(a),cond_of(a))
proof (induct a rule:wf_induct[OF wf_frecrel[of 2×A1×A1×A2]])
  case (1 x)
  let ?τ = name1(x)
  let ?ϑ = name2(x)
  let ?D = 2×A1×A1×A2
  assume x ∈ ?D
  then
  have cond_of(x)∈A2
    by (auto simp add:components_simp)
  from ‹x∈?D›
  consider (eq) ftype(x)=0 | (mem) ftype(x)=1
    by (auto simp add:components_simp)
  then
  show ?case
  proof cases
    case eq
    then
     have Q(1, σ, ?τ, q) ∧ Q(1, σ, ?ϑ, q) if σ ∈ domain(?τ) ∪ domain(?ϑ) and
q∈A2 for q σ
    proof -
      from 1
      have A: ?τ∈A1 ?ϑ∈A1 ?τ∈eclose(A1) ?ϑ∈eclose(A1)
        using arg_into_eclose by (auto simp add:components_simp)
      with ‹Transset(A1)› that(1)
      have σ∈eclose(?τ) ∪ eclose(?ϑ)
        using in_dom_in_eclose by auto
      then
```

```
    have σ∈A1
      using mem_eclose_subset[OF ‹?τ∈A1›] mem_eclose_subset[OF ‹?ϑ∈A1›]
        Transset_eclose_eq_arg[OF ‹Transset(A1)›]
      by auto
    with ‹q∈A2› ‹?ϑ ∈ A1› ‹cond_of(x)∈A2› ‹?τ∈A1›
    have frecR(⟨1, σ, ?τ, q⟩, x) (is frecR(?T,_))
      frecR(⟨1, σ, ?ϑ, q⟩, x) (is frecR(?U,_))
      using frecRI1'[OF that(1)] frecR_DI ‹ftype(x) = 0›
        frecRI2'[OF that(1)]
      by (auto simp add:components_simp)
    with ‹x∈?D› ‹σ∈A1› ‹q∈A2›
    have ⟨?T,x⟩∈ frecrel(?D) ⟨?U,x⟩∈ frecrel(?D)
    using frecrelI[of ?T ?D x] frecrelI[of ?U ?D x] by (auto simp add:components_simp)
    with ‹q∈A2› ‹σ∈A1› ‹?τ∈A1› ‹?ϑ∈A1›
    have Q(1, σ, ?τ, q) using 1 by (force simp add:components_simp)
    moreover from ‹q∈A2› ‹σ∈A1› ‹?τ∈A1› ‹?ϑ∈A1› ‹⟨?U,x⟩∈ frecrel(?D)›
    have Q(1, σ, ?ϑ, q) using 1 by (force simp add:components_simp)
    ultimately
    show ?thesis using A by simp
  qed
  then show ?thesis using assms(3) ‹ftype(x) = 0› ‹cond_of(x)∈A2› by auto
next
  case mem
  have Q(0, ?τ, σ, q) if σ ∈ domain(?ϑ) and q∈A2 for q σ
  proof -
    from 1 assms
    have ?τ∈A1 ?ϑ∈A1 cond_of(x)∈A2 ?τ∈eclose(A1) ?ϑ∈eclose(A1)
      using arg_into_eclose by (auto simp add:components_simp)
    with ‹Transset(A1)› that(1)
    have σ∈ eclose(?ϑ)
      using in_dom_in_eclose by auto
    then
    have σ∈A1
    using mem_eclose_subset[OF ‹?ϑ∈A1›] Transset_eclose_eq_arg[OF ‹Transset(A1)›]
      by auto
    with ‹q∈A2› ‹?ϑ ∈ A1› ‹cond_of(x)∈A2› ‹?τ∈A1›
    have frecR(⟨0, ?τ, σ, q⟩, x) (is frecR(?T,_))
      using frecRI3'[OF that(1)] frecR_DI ‹ftype(x) = 1›
      by (auto simp add:components_simp)
    with ‹x∈?D› ‹σ∈A1› ‹q∈A2› ‹?τ∈A1›
    have ⟨?T,x⟩∈ frecrel(?D) ?T∈?D
      using frecrelI[of ?T ?D x] by (auto simp add:components_simp)
    with ‹q∈A2› ‹σ∈A1› ‹?τ∈A1› ‹?ϑ∈A1› 1
    show ?thesis by (force simp add:components_simp)
  qed
  then show ?thesis using assms(2) ‹ftype(x) = 1› ‹cond_of(x)∈A2› by auto
qed
qed
```

lemma *def_frecrel* : *frecrel*(*A*) = {*z*∈*A*×*A*. ∃ *x y*. *z* = ⟨*x*, *y*⟩ ∧ *frecR*(*x*,*y*)}
unfolding *frecrel_def Rrel_def* **..**

lemma *frecrel_fst_snd*:
frecrel(*A*) = {*z* ∈ *A*×*A* .
ftype(*fst*(*z*)) = *1* ∧
ftype(*snd*(*z*)) = *0* ∧ *name1*(*fst*(*z*)) ∈ *domain*(*name1*(*snd*(*z*))) ∪ *domain*(*name2*(*snd*(*z*)))
∧
(*name2*(*fst*(*z*)) = *name1*(*snd*(*z*)) ∨ *name2*(*fst*(*z*)) = *name2*(*snd*(*z*)))
∨ (*ftype*(*fst*(*z*)) = *0* ∧
ftype(*snd*(*z*)) = *1* ∧ *name1*(*fst*(*z*)) = *name1*(*snd*(*z*)) ∧ *name2*(*fst*(*z*)) ∈
domain(*name2*(*snd*(*z*))))}
unfolding *def_frecrel frecR_def*
by (*intro equalityI subsetI CollectI*; *elim CollectE*; *auto*)

end
theory *FrecR_Arities*
imports *Arities FrecR*
begin
lemma *arity_fst_fm* [*arity*] :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*fst_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
unfolding *fst_fm_def*
using *arity_pair_fm arity_empty_fm union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_snd_fm* [*arity*] :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*snd_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
unfolding *snd_fm_def*
using *arity_pair_fm arity_empty_fm union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_snd_snd_fm* [*arity*] :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*snd_snd_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
unfolding *snd_snd_fm_def hcomp_fm_def*
using *arity_snd_fm arity_empty_fm union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_ftype_fm* [*arity*] :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*ftype_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
unfolding *ftype_fm_def*
using *arity_fst_fm*
by *auto*

lemma *arity_name1_fm* [*arity*] :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*name1_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
unfolding *name1_fm_def hcomp_fm_def*
using *arity_fst_fm arity_snd_fm union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_name2_fm* [*arity*] :
  ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*name2_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
  unfolding *name2_fm_def hcomp_fm_def*
  using *arity_fst_fm arity_snd_snd_fm union_abs2 pred_Un_distrib*
  by *auto*

lemma *arity_cond_of_fm* [*arity*] :
  ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*cond_of_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
  unfolding *cond_of_fm_def hcomp_fm_def*
  using *arity_snd_fm arity_snd_snd_fm union_abs2 pred_Un_distrib*
  by *auto*

lemma *arity_eclose_n1_fm* [*arity*] :
  ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*eclose_n1_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
  unfolding *eclose_n1_fm_def*
 using *arity_is_eclose_fm arity_singleton_fm arity_name1_fm union_abs2 pred_Un_distrib*
  by *auto*

lemma *arity_eclose_n2_fm* [*arity*] :
  ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*eclose_n2_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
  unfolding *eclose_n2_fm_def*
 using *arity_is_eclose_fm arity_singleton_fm arity_name2_fm union_abs2 pred_Un_distrib*
  by *auto*

lemma *arity_ecloseN_fm* [*arity*] :
  ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*ecloseN_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
  unfolding *ecloseN_fm_def*
   using *arity_eclose_n1_fm arity_eclose_n2_fm arity_union_fm union_abs2*
pred_Un_distrib
  by *auto*

lemma *arity_frecR_fm* [*arity*]:
  ⟦*a*∈*nat*;*b*∈*nat*⟧ ⟹ *arity*(*frecR_fm*(*a*,*b*)) = *succ*(*a*) ∪ *succ*(*b*)
  unfolding *frecR_fm_def*
  using *arity_ftype_fm arity_name1_fm arity_name2_fm arity_domain_fm*
     arity_empty_fm arity_union_fm pred_Un_distrib arity_succ_fm
  by *auto*
end
theory *Discipline_Cardinal*
  imports
    Discipline_Base
    Discipline_Function
    Least
    FrecR
    Arities
    FrecR_Arities
begin

declare [[*syntax_ambiguity_warning* = *false*]]

relativize functional *cardinal cardinal_rel* **external**
relationalize *cardinal_rel is_cardinal*
synthesize *is_cardinal* **from_definition assuming** *nonempty*

notation *is_cardinal_fm* (‹*cardinal'*(_') *is* _›)

abbreviation
cardinal_r :: [*i*,*i*⇒*o*] ⇒ *i* (‹|_|⟶›) **where**
$|x|^M \equiv$ *cardinal_rel*(*M*,*x*)

abbreviation
cardinal_r_set :: [*i*,*i*]⇒*i* (‹|_|⟶›) **where**
$|x|^M \equiv$ *cardinal_rel*(##*M*,*x*)

context *M_trivial* **begin**
rel_closed for *cardinal*
using *Least_closed'*[*of* λ*i*. *M*(*i*) ∧ *i* ≈^{M} *A*]
unfolding *cardinal_rel_def*
by *simp*
end

manual_arity intermediate for *is_Int_fm*
unfolding *is_Int_fm_def*
using *arity*
by *simp*

arity_theorem for *is_Int_fm*

arity_theorem for *is_funspace_fm*

arity_theorem for *is_function_space_fm*

arity_theorem for *surjP_rel_fm*

arity_theorem intermediate for *is_surj_fm*

lemma *arity_is_surj_fm* [*arity*] :
A ∈ *nat* ⟹ *B* ∈ *nat* ⟹ *I* ∈ *nat* ⟹ *arity*(*is_surj_fm*(*A*, *B*, *I*)) = *succ*(*A*) ∪ *succ*(*B*) ∪ *succ*(*I*)
using *arity_is_surj_fm'*
by *auto*

arity_theorem for *injP_rel_fm*

arity_theorem intermediate for *is_inj_fm*

lemma *arity_is_inj_fm* [*arity*]:
A ∈ *nat* ⟹ *B* ∈ *nat* ⟹ *I* ∈ *nat* ⟹ *arity*(*is_inj_fm*(*A*, *B*, *I*)) = *succ*(*A*) ∪

$succ(B) \cup succ(I)$
using *arity_is_inj_fm'*
by *auto*

arity_theorem for *is_bij_fm*

arity_theorem for *is_eqpoll_fm*

arity_theorem for *is_cardinal_fm*

context *M_Perm* **begin**

is_iff_rel for *cardinal*
using *least_abs'*[*of* $\lambda i.\ M(i) \wedge i \approx^M A$]
is_eqpoll_iff
unfolding *is_cardinal_def cardinal_rel_def*
by *simp*
end

reldb_add relational *Ord ordinal*
reldb_add functional *lt lt*
reldb_add relational *lt lt_rel*
synthesize *lt_rel* **from_definition**
notation *lt_rel_fm* ($\langle \cdot \_ < \_ \cdot \rangle$)
arity_theorem intermediate for *lt_rel_fm*

lemma *arity_lt_rel_fm*[*arity*]: $a \in nat \Longrightarrow b \in nat \Longrightarrow arity(lt\_rel\_fm(a,\ b)) = succ(a) \cup succ(b)$
using *arity_lt_rel_fm'*
by *auto*

relativize functional *Card Card_rel* **external**
relationalize *Card_rel is_Card*
synthesize *is_Card* **from_definition assuming** *nonempty*
notation *is_Card_fm* ($\langle \cdot Card'(\_') \cdot \rangle$)
arity_theorem for *is_Card_fm*

notation *Card_rel* ($\langle Card{-}'(\_') \rangle$)

lemma (**in** *M_Perm*) *is_Card_iff*: $M(A) \Longrightarrow is\_Card(M,\ A) \longleftrightarrow Card^M(A)$
using *is_cardinal_iff*
unfolding *is_Card_def Card_rel_def* **by** *simp*

abbreviation
$Card\_r\_set\ ::\ [i,i] \Rightarrow o\ \ (\langle Card{-}'(\_') \rangle)$ **where**
$Card^M(i) \equiv Card\_rel(\#\#M,i)$

relativize functional *InfCard InfCard_rel* **external**
relationalize *InfCard_rel is_InfCard*

synthesize *is_InfCard* **from_definition assuming** *nonempty*
notation *is_InfCard_fm* (‹·*InfCard*′(_′)·›)
arity_theorem for *is_InfCard_fm*

notation *InfCard_rel* (‹*InfCard*–′(_′)›)

abbreviation
InfCard_r_set :: [*i*,*i*]⇒*o* (‹*InfCard*–′(_′)›) **where**
InfCard^M(*i*) ≡ *InfCard_rel*(##*M*,*i*)

relativize functional *cadd cadd_rel* **external**

abbreviation
cadd_r :: [*i*,*i*⇒*o*,*i*] ⇒ *i* (‹_ ⊕– _› [66,1,66] 65) **where**
A ⊕^{M} *B* ≡ *cadd_rel*(*M*,*A*,*B*)

context *M_basic* **begin**
rel_closed for *cadd*
using *cardinal_rel_closed*
unfolding *cadd_rel_def*
by *simp*
end

relationalize *cadd_rel is_cadd*

manual_schematic for *is_cadd* **assuming** *nonempty*
unfolding *is_cadd_def*
by (*rule iff_sats sum_iff_sats* | *simp*)+
synthesize *is_cadd* **from_schematic**

arity_theorem for *sum_fm*

arity_theorem for *is_cadd_fm*

context *M_Perm* **begin**
is_iff_rel for *cadd*
using *is_cardinal_iff*
unfolding *is_cadd_def cadd_rel_def*
by *simp*
end

relativize functional *cmult cmult_rel* **external**

abbreviation
cmult_r :: [*i*,*i*⇒*o*,*i*] ⇒ *i* (‹_ ⊗– _› [66,1,66] 65) **where**
A ⊗^{M} *B* ≡ *cmult_rel*(*M*,*A*,*B*)

relationalize *cmult_rel is_cmult*

declare *cartprod_iff_sats* [*iff_sats*]

synthesize *is_cmult* **from_definition assuming** *nonempty*

arity_theorem for *is_cmult_fm*

context *M_Perm* **begin**

rel_closed for *cmult*
  using *cardinal_rel_closed*
  unfolding *cmult_rel_def*
  by *simp*

is_iff_rel for *cmult*
  using *is_cardinal_iff*
  unfolding *is_cmult_def cmult_rel_def*
  by *simp*

end

definition
  Powapply :: [*i*,*i*] ⇒ *i* **where**
  Powapply(*f*,*y*) ≡ *Pow*(*f*‘*y*)

reldb_add functional *Pow Pow_rel*
reldb_add relational *Pow is_Pow*

lemma *subset_iff_sats*[*iff_sats*]:
  nth(*i*, *env*) = *x* ⟹ *nth*(*j*, *env*) = *y* ⟹ *i*∈*nat* ⟹ *j*∈*nat* ⟹
   env ∈ *list*(*A*) ⟹ *subset*(##*A*, *x*, *y*) ⟷ *A*, *env* ⊨ *subset_fm*(*i*, *j*)
  using *sats_subset_fm′* **by** *simp*

declare *Replace_iff_sats*[*iff_sats*]

synthesize *is_Pow* **from_definition assuming** *nonempty*
arity_theorem for *is_Pow_fm*

relativize functional *Powapply Powapply_rel*
relationalize *Powapply_rel is_Powapply*
synthesize *is_Powapply* **from_definition assuming** *nonempty*
arity_theorem for *is_Powapply_fm*

notation *Powapply_rel* (‹*Powapply*–′(_,_′)›)

context *M_basic*
begin

rel_closed for *Powapply*
unfolding *Powapply_rel_def*
by *simp*

is_iff_rel for *Powapply*
using *Pow_rel_iff*
unfolding *is_Powapply_def Powapply_rel_def*
by *simp*

univalent for *Powapply*
using *is_Powapply_iff*
unfolding *univalent_def*
by *simp*

end

definition
$HVfrom :: [i,i,i] \Rightarrow i$ **where**
$HVfrom(A,x,f) \equiv A \cup (\bigcup y{\in}x.\ Powapply(f,y))$

relativize functional *HVfrom HVfrom_rel*
relationalize *HVfrom_rel is_HVfrom*
synthesize *is_HVfrom* **from_definition assuming** *nonempty*
arity_theorem for *is_HVfrom_fm*

notation *HVfrom_rel* ($\langle HVfrom\text{–}'(\_,\_,\_')\rangle$)

locale *M_HVfrom* = *M_eclose* +
assumes
Powapply_replacement:
$M(K) \Longrightarrow strong\_replacement(M,\lambda y\ z.\ z = Powapply^M(f,y))$
begin

is_iff_rel for *HVfrom*
proof -
assume *assms*:$M(A)\ M(x)\ M(f)\ M(res)$
then
have $is\_Replace(M,x,\lambda y\ z.\ z = Powapply^M(f,y),r) \longleftrightarrow r = \{z\ .\ y{\in}x,\ z = Powapply^M(f,y)\}$
if $M(r)$ **for** r
using *that Powapply_rel_closed*
$Replace\_abs[of\ x\ r\ \lambda y\ z.\ z = Powapply^M(f,y)]\ transM[of\ \_\ x]$
by *simp*
moreover
have $is\_Replace(M,x,is\_Powapply(M,f),r) \longleftrightarrow$
$is\_Replace(M,x,\lambda y\ z.\ z = Powapply^M(f,y),r)$ **if** $M(r)$ **for** r

using *assms that is_Powapply_iff is_Replace_cong*
by *simp*
ultimately
have *is_Replace*(M,x,*is_Powapply*(M,f),r) $\longleftrightarrow$ $r = \{z\ .\ y{\in}x,\ z = Powapply^M(f,y)\}$
if $M(r)$ **for** r
using *that assms*
by *simp*
moreover
have $M(\{z\ .\ y \in x,\ z = Powapply^M(f,y)\})$
using *assms strong_replacement_closed*[*OF Powapply_replacement*]
Powapply_rel_closed transM[*of _ x*]
by *simp*
moreover from *assms*
— intermediate step for body of Replace
have $\{z\ .\ y \in x,\ z = Powapply^M(f,y)\} = \{y\ .\ w \in x,\ M(y) \wedge M(w) \wedge y = Powapply^M(f,w)\}$
by (*auto dest*:*transM*)
ultimately
show *?thesis*
using *assms*
unfolding *is_HVfrom_def HVfrom_rel_def*
by (*auto dest*:*transM*)
qed

univalent for *HVfrom*
using *is_HVfrom_iff*
unfolding *univalent_def*
by *simp*

rel_closed for *HVfrom*
proof -
assume *assms*:$M(A)$ $M(x)$ $M(f)$
have $M(\{z\ .\ y \in x,\ z = Powapply^M(f,y)\})$
using *assms strong_replacement_closed*[*OF Powapply_replacement*]
Powapply_rel_closed transM[*of _ x*]
by *simp*
then
have $M(A \cup \bigcup(\{z\ .\ y{\in}x,\ z = Powapply^M(f,y)\}))$
using *assms*
by *simp*
moreover from *assms*
— intermediate step for body of Replace
have $\{z\ .\ y \in x,\ z = Powapply^M(f,y)\} = \{y\ .\ w \in x,\ M(y) \wedge M(w) \wedge y = Powapply^M(f,w)\}$
by (*auto dest*:*transM*)
ultimately
show *?thesis*
using *assms*
unfolding *HVfrom_rel_def*

by *simp*
qed

end

definition
Vfrom_rel :: [*i*⇒*o*,*i*,*i*] ⇒ *i* (‹*Vfrom*–'(_,_')›) **where**
$Vfrom^M(A,i) = transrec(i, HVfrom\_rel(M,A))$

definition
is_Vfrom :: [*i*⇒*o*,*i*,*i*,*i*] ⇒ *o* **where**
is_Vfrom(*M*,*A*,*i*,*z*) ≡ *is_transrec*(*M*,*is_HVfrom*(*M*,*A*),*i*,*z*)

locale *M_Vfrom* = *M_HVfrom* +
assumes
trepl_HVfrom : ⟦ *M*(*A*);*M*(*i*) ⟧ ⟹ *transrec_replacement*(*M*,*is_HVfrom*(*M*,*A*),*i*)

begin

lemma *Vfrom_rel_iff* :
assumes *M*(*A*) *M*(*i*) *M*(*z*) *Ord*(*i*)
shows *is_Vfrom*(*M*,*A*,*i*,*z*) ⟷ *z* = $Vfrom^M(A,i)$
proof -
have *relation2*(*M*, *is_HVfrom*(*M*, *A*), *HVfrom_rel*(*M*, *A*))
using *assms is_HVfrom_iff*
unfolding *relation2_def*
by *simp*
then
show *?thesis*
using *assms HVfrom_rel_closed trepl_HVfrom*
transrec_abs[*of is_HVfrom*(*M*,*A*) *i HVfrom_rel*(*M*,*A*) *z*]
unfolding *is_Vfrom_def Vfrom_rel_def*
by *simp*
qed

end

end

20 Replacements using Lambdas

theory *Lambda_Replacement*
imports
ZF-Constructible.Relative
ZF_Miscellanea— for *SepReplace*
Discipline_Function

begin

In this theory we prove several instances of separation and replacement in *M_basic*. Moreover by assuming a seven instances of separation and ten instances of "lambda" replacements we prove a bunch of other instances.

definition
lam_replacement :: $[i⇒o,i⇒i] ⇒ o$ **where**
lam_replacement$(M,b) ≡$ *strong_replacement*$(M, \lambda x\ y.\ y = \langle x,\ b(x)\rangle)$

lemma *separation_univ* :
shows *separation*(M,M)
unfolding *separation_def* **by** *auto*

context *M_basic*
begin

lemma *separation_in* :
assumes $M(a)$
shows *separation*$(M,\lambda x\ .\ x∈a)$
proof -
have $\{x∈A\ .\ x∈a\} = A ∩ a$ **for** A **by** *auto*
with ‹$M(a)$›
show *?thesis* **using** *separation_iff Collect_abs*
by *simp*
qed

lemma *separation_equal* :
shows *separation*$(M,\lambda x\ .\ x=a)$
proof -
have $\{x∈A\ .\ x=a\} =$ (*if* $a∈A$ *then* $\{a\}$ *else* 0) **for** A
by *auto*
then
have $M(\{x∈A\ .\ x=a\})$ **if** $M(A)$ **for** A
using *transM*[*OF* _ ‹$M(A)$›] **by** *simp*
then
show *?thesis* **using** *separation_iff Collect_abs*
by *simp*
qed

lemma (**in** *M_basic*) *separation_in_rev*:
assumes $(M)(a)$
shows *separation*$(M,\lambda x\ .\ a∈x)$
proof -
have *eq*: $\{x∈A.\ a∈x\} =$ *Memrel*$(A∪\{a\})$ `` $\{a\}$ **for** A
unfolding *ZF_Base.image_def*
by(*intro equalityI*,*auto simp*:*mem_not_refl*)
moreover from *assms*
have $M($*Memrel*$(A∪\{a\})$ `` $\{a\})$ **if** $M(A)$ **for** A
using *that* **by** *simp*

ultimately
show *?thesis*
using *separation_iff Collect_abs*
by *simp*
qed

lemma *lam_replacement_iff_lam_closed*:
assumes $\forall\, x[M].\ M(b(x))$
shows *lam_replacement*$(M,\ b) \longleftrightarrow\ (\forall\, A[M].\ M(\lambda x{\in}A.\ b(x)))$
using *assms lam_closed lam_funtype*[*of* _ *b*, *THEN Pi_memberD*]
unfolding *lam_replacement_def strong_replacement_def*
by (*auto intro*:*lamI dest*:*transM*)
(*rule lam_closed*, *auto simp add*:*strong_replacement_def dest*:*transM*)

lemma *lam_replacement_cong*:
assumes *lam_replacement*(M,f) $\forall\, x[M].\ f(x) = g(x)$ $\forall\, x[M].\ M(f(x))$
shows *lam_replacement*(M,g)
proof -
note *assms*
moreover from *this*
have $\forall\, A[M].\ M(\lambda x{\in}A.\ f(x))$
using *lam_replacement_iff_lam_closed*
by *simp*
moreover from *calculation*
have $(\lambda x{\in}A\ .\ f(x)) = (\lambda x{\in}A\ .\ g(x))$ **if** $M(A)$ **for** A
using *lam_cong*[*OF refl*,*of A f g*] *transM*[*OF* _ *that*]
by *simp*
ultimately
show *?thesis*
using *lam_replacement_iff_lam_closed*
by *simp*
qed

lemma *converse_subset* : *converse*$(r) \subseteq \{<snd(x),fst(x)>\ .\ x{\in}r\}$
unfolding *converse_def*
proof(*intro subsetI*, *auto*)
fix *u v*
assume $<u,v>{\in}r$ (**is** *?z*$\in r$)
moreover
have *v*=*snd*(*?z*) *u*=*fst*(*?z*) **by** *simp_all*
ultimately
show $\exists\, z{\in}r.\ v{=}snd(z) \wedge\ u = fst(z)$
using *rexI*[**where** $x{=}<u,v>$] **by** *force*
qed

lemma *converse_eq_aux* :
assumes $<0,0>{\in}r$
shows *converse*$(r) = \{<snd(x),fst(x)>\ .\ x{\in}r\}$
using *converse_subset*

proof(*intro equalityI subsetI*,*auto*)
fix *z*
assume *z*∈*r*
then show <*fst*(*z*),*snd*(*z*)> ∈ *r*
proof(*cases* ∃ *a b* . *z* =<*a*,*b*>)
case *True*
with ‹*z*∈*r*›
show *?thesis* **by** *auto*
next
case *False*
then
have *fst*(*z*) = *0 snd*(*z*)=*0*
unfolding *fst_def snd_def* **by** *auto*
with ‹*z*∈*r*› *assms*
show *?thesis* **by** *auto*
qed
qed

lemma *converse_eq_aux′* :
assumes <*0*,*0*>∉*r*
shows *converse*(*r*) = {<*snd*(*x*),*fst*(*x*)> . *x*∈*r*} - {<*0*,*0*>}
using *converse_subset assms*
proof(*intro equalityI subsetI*,*auto*)
fix *z*
assume *z*∈*r snd*(*z*)≠*0*
then
obtain *a b* **where** *z* = <*a*,*b*> **unfolding** *snd_def* **by** *force*
with ‹*z*∈*r*›
show <*fst*(*z*),*snd*(*z*)> ∈ *r*
by *auto*
next
fix *z*
assume *z*∈*r fst*(*z*)≠*0*
then
obtain *a b* **where** *z* = <*a*,*b*> **unfolding** *fst_def* **by** *force*
with ‹*z*∈*r*›
show <*fst*(*z*),*snd*(*z*)> ∈ *r*
by *auto*
qed

lemma *diff_un* : *b*⊆*a* ⟹ (*a*-*b*) ∪ *b* = *a*
by *auto*

lemma *converse_eq*: *converse*(*r*) = ({<*snd*(*x*),*fst*(*x*)> . *x*∈*r*} - {<*0*,*0*>}) ∪ (*r*∩{<*0*,*0*>})

proof(*cases* <*0*,*0*>∈*r*)
case *True*
then
have *converse*(*r*) = {<*snd*(*x*),*fst*(*x*)> . *x*∈*r*}

```
    using converse_eq_aux by auto
  moreover
  from True
  have r∩{<0,0>} = {<0,0>} {<0,0>}⊆{<snd(x),fst(x)> . x∈r}
    using converse_subset by auto
  moreover from this True
   have {<snd(x),fst(x)> . x∈r} = ({<snd(x),fst(x)> . x∈r} - {<0,0>}) ∪
({<0,0>})
    using diff_un[of {<0,0>},symmetric] converse_eq_aux by auto
  ultimately
  show ?thesis
    by simp
next
  case False
  then
  have r∩{<0,0>} = 0 by auto
  then
 have ({<snd(x),fst(x)> . x∈r} - {<0,0>}) ∪ (r∩{<0,0>}) = ({<snd(x),fst(x)>
. x∈r} - {<0,0>})
    by simp
  with False
  show ?thesis
    using converse_eq_aux' by auto
qed

lemma range_subset : range(r) ⊆ {snd(x). x∈r}
  unfolding range_def domain_def converse_def
proof(intro subsetI, auto)
  fix u v
  assume <u,v>∈r (is ?z∈r)
  moreover
  have v=snd(?z) u=fst(?z) by simp_all
  ultimately
  show ∃ z∈r. v=snd(z)
    using rexI[where x=v] by force
qed

lemma lam_replacement_imp_strong_replacement_aux:
  assumes lam_replacement(M, b) ∀ x[M]. M(b(x))
  shows strong_replacement(M, λx y. y = b(x))
proof -
  {
    fix A
    note assms
    moreover
    assume M(A)
    moreover from calculation
    have M(λx∈A. b(x)) using lam_replacement_iff_lam_closed by auto
    ultimately
```

```
    have M((λx∈A. b(x))‘‘A) ∀z[M]. z ∈ (λx∈A. b(x))‘‘A ⟷ (∃x∈A. z = b(x))
      by (auto simp:lam_def)
  }
  then
  show ?thesis unfolding strong_replacement_def
    by clarsimp (rule_tac x=(λx∈A. b(x))‘‘A in rexI, auto)
qed

lemma lam_replacement_imp_RepFun_Lam:
  assumes lam_replacement(M, f) M(A)
  shows M({y . x∈A , M(y) ∧ y=⟨x,f(x)⟩})
proof -
  from assms
  obtain Y where 1:M(Y) ∀b[M]. b ∈ Y ⟷ (∃x[M]. x ∈ A ∧ b = ⟨x,f(x)⟩)
    unfolding lam_replacement_def strong_replacement_def
    by auto
  moreover from calculation
  have Y = {y . x∈A , M(y) ∧ y = ⟨x,f(x)⟩} (is Y=?R)
  proof(intro equalityI subsetI)
    fix y
    assume y∈Y
    moreover from this 1
    obtain x where x∈A y=⟨x,f(x)⟩ M(y)
      using transM[OF _ ‹M(Y)›] by auto
    ultimately
    show y∈?R
      by auto
  next
    fix z
    assume z∈?R
    moreover from this
    obtain a where a∈A z=⟨a,f(a)⟩ M(a) M(f(a))
      using transM[OF _ ‹M(A)›]
      by auto
    ultimately
    show z∈Y using 1 by simp
  qed
  ultimately
  show ?thesis by auto
qed

lemma lam_closed_imp_closed:
  assumes ∀A[M]. M(λx∈A. f(x))
  shows ∀x[M]. M(f(x))
proof
  fix x
  assume M(x)
  moreover from this and assms
  have M(λx∈{x}. f(x)) by simp
```

ultimately
show $M(f(x))$
using *image_lam*[*of* $\{x\}$ $\{x\}$ f]
image_closed[*of* $\{x\}$ $(\lambda x{\in}\{x\}.\ f(x))$] **by** (*auto dest*:*transM*)
qed

lemma *lam_replacement_if*:
assumes *lam_replacement*(M,f) *lam_replacement*(M,g) *separation*(M,b)
$\forall\, x[M].\ M(f(x))$ $\forall\, x[M].\ M(g(x))$
shows *lam_replacement*(M, $\lambda x.$ *if* $b(x)$ *then* $f(x)$ *else* $g(x)$)
proof -
let *?G*=$\lambda x.$ *if* $b(x)$ *then* $f(x)$ *else* $g(x)$
let *?b*=λA . $\{x{\in}A.\ b(x)\}$ **and** *?b'*=λA . $\{x{\in}A.\ \neg b(x)\}$
have *eq*:$(\lambda x{\in}A\ .\ ?G(x)) = (\lambda x{\in}?b(A)\ .\ f(x)) \cup (\lambda x{\in}?b'(A).g(x))$ **for** A
unfolding *lam_def* **by** *auto*
have $?b'(A) = A$ - $?b(A)$ **for** A **by** *auto*
moreover
have $M(?b(A))$ **if** $M(A)$ **for** A **using** *assms that* **by** *simp*
moreover from *calculation*
have $M(?b'(A))$ **if** $M(A)$ **for** A **using** *that* **by** *simp*
moreover from *calculation assms*
have $M(\lambda x{\in}?b(A).\ f(x))$ $M(\lambda x{\in}?b'(A)\ .\ g(x))$ **if** $M(A)$ **for** A
using *lam_replacement_iff_lam_closed that*
by *simp_all*
moreover from *this*
have $M((\lambda x{\in}?b(A)\ .\ f(x)) \cup (\lambda x{\in}?b'(A).g(x)))$ **if** $M(A)$ **for** A
using *that* **by** *simp*
ultimately
have $M(\lambda x{\in}A.$ *if* $b(x)$ *then* $f(x)$ *else* $g(x))$ **if** $M(A)$ **for** A
using *that eq* **by** *simp*
with *assms*
show *?thesis* **using** *lam_replacement_iff_lam_closed* **by** *simp*
qed

lemma *lam_replacement_constant*: $M(b) \Longrightarrow$ *lam_replacement*(M,$\lambda\_.\ b$)
unfolding *lam_replacement_def strong_replacement_def*
by *safe* (*rule_tac* x=$\_{\times}\{b\}$ **in** *rexI*; *blast*)

20.1 Replacement instances obtained through Powerset

The next few lemmas provide bounds for certain constructions.

lemma *not_functional_Replace_0*:
assumes $\neg(\forall\, y\ y'.\ P(y) \wedge P(y') \longrightarrow y{=}y')$
shows $\{y\ .\ x \in A,\ P(y)\} = 0$
using *assms* **by** (*blast elim*!: *ReplaceE*)

lemma *Replace_in_Pow_rel*:
assumes $\bigwedge x\ b.\ x \in A \Longrightarrow P(x,b) \Longrightarrow b \in U\ \forall\, x{\in}A.\ \forall\, y\ y'.\ P(x,y) \wedge P(x,y') \longrightarrow y{=}y'$

separation(M, $\lambda y.\ \exists\, x[M].\ x \in A \wedge P(x,\ y)$)
M(U) *M*(A)
shows $\{y\ .\ x \in A,\ P(x,\ y)\} \in Pow^M(U)$
proof -
from *assms*
have $\{y\ .\ x \in A,\ P(x,\ y)\} \subseteq U$
$z \in \{y\ .\ x \in A,\ P(x,\ y)\} \Longrightarrow M(z)$ **for** z
by (*auto dest*:*transM*)
with *assms*
have $\{y\ .\ x \in A,\ P(x,\ y)\} = \{y{\in}U\ .\ \exists\, x[M].\ x{\in}A \wedge P(x,y)\}$
by (*intro equalityI*) (*auto*, *blast*)
with *assms*
have $M(\{y\ .\ x \in A,\ P(x,\ y)\})$
by *simp*
with *assms*
show *?thesis*
using *mem_Pow_rel_abs* **by** *auto*
qed

lemma *Replace_sing_0_in_Pow_rel*:
assumes $\bigwedge b.\ P(b) \Longrightarrow b \in U$
separation(M, $\lambda y.\ P(y)$) *M*(U)
shows $\{y\ .\ x \in \{0\},\ P(y)\} \in Pow^M(U)$
proof (*cases* $\forall\ y\ y'.\ P(y) \wedge P(y') \longrightarrow y{=}y'$)
case *True*
with *assms*
show *?thesis* **by** (*rule_tac Replace_in_Pow_rel*) *auto*
next
case *False*
with *assms*
show *?thesis*
using *nonempty not_functional_Replace_0*[*of P* $\{0\}$] *Pow_rel_char* **by** *auto*
qed

lemma *The_in_Pow_rel_Union*:
assumes $\bigwedge b.\ P(b) \Longrightarrow b \in U$ *separation*(M, $\lambda y.\ P(y)$) *M*(U)
shows (*THE i. P*(i)) $\in Pow^M(\bigcup U)$
proof -
note *assms*
moreover from *this*
have (*THE i. P*(i)) $\in Pow(\bigcup U)$
unfolding *the_def* **by** *auto*
moreover from *assms*
have *M*(*THE i. P*(i))
using *Replace_sing_0_in_Pow_rel*[*of P U*] **unfolding** *the_def*
by (*auto dest*:*transM*)
ultimately
show *?thesis*
using *Pow_rel_char* **by** *auto*

qed

lemma *separation_least*: *separation*(M, $\lambda y.\ Ord(y) \wedge P(y) \wedge (\forall j.\ j < y \longrightarrow \neg P(j))$)
unfolding *separation_def*
proof
fix z
assume $M(z)$
have $M(\{x \in z\ .\ x \in z \wedge Ord(x) \wedge P(x) \wedge (\forall j.\ j < x \longrightarrow \neg P(j))\})$
(**is** $M(?y)$)
proof (*cases* $\exists\, x{\in}z.\ Ord(x) \wedge P(x) \wedge (\forall j.\ j < x \longrightarrow \neg P(j))$)
case *True*
with ‹$M(z)$›
have $\exists\, x[M].\ ?y = \{x\}$
by (*safe*, *rename_tac* x, *rule_tac* $x{=}x$ **in** *rexI*)
(*auto dest*:*transM*, *intro equalityI*, *auto elim*:*Ord_linear_lt*)
then
show *?thesis*
by *auto*
next
case *False*
then
have $\{x \in z\ .\ x \in z \wedge Ord(x) \wedge P(x) \wedge (\forall j.\ j < x \longrightarrow \neg P(j))\} = 0$
by *auto*
then
show *?thesis* **by** *auto*
qed
moreover from *this*
have $\forall\, x[M].\ x \in ?y \longleftrightarrow x \in z \wedge Ord(x) \wedge P(x) \wedge (\forall j.\ j < x \longrightarrow \neg P(j))$ **by** *simp*
ultimately
show $\exists\, y[M].\ \forall\, x[M].\ x \in y \longleftrightarrow x \in z \wedge Ord(x) \wedge P(x) \wedge (\forall j.\ j < x \longrightarrow \neg P(j))$
by *blast*
qed

lemma *Least_in_Pow_rel_Union*:
assumes $\bigwedge b.\ P(b) \Longrightarrow b \in U$
$M(U)$
shows $(\mu\ i.\ P(i)) \in Pow^M(\bigcup U)$
using *assms separation_least* **unfolding** *Least_def*
by (*rule_tac The_in_Pow_rel_Union*) *simp*

lemma *bounded_lam_replacement*:
fixes U
assumes $\forall\, X[M].\ \forall\, x{\in}X.\ f(x) \in U(X)$
and *separation_f*:$\forall\, A[M].\ separation(M,\lambda y.\ \exists\, x[M].\ x{\in}A \wedge y = \langle x, f(x)\rangle)$
and *U_closed* [*intro*,*simp*]: $\bigwedge X.\ M(X) \Longrightarrow M(U(X))$
shows *lam_replacement*(M, f)

proof -
  have $M(\lambda x{\in}A.\ f(x))$ **if** $M(A)$ **for** A
  proof -
    have $(\lambda x{\in}A.\ f(x)) = \{y{\in}\ Pow^M(Pow^M(A \cup U(A)))).\ \exists\, x[M].\ x{\in}A \wedge y = \langle x,$
$f(x)\rangle\}$
      using ‹$M(A)$› **unfolding** *lam_def*
    proof (*intro equalityI*, *auto*)
      fix x
      assume $x{\in}A$
      moreover
      note ‹$M(A)$›
      moreover from *calculation assms*
      have $f(x) \in U(A)$ **by** *simp*
      moreover from *calculation*
      have $\{x,\ f(x)\} \in Pow^M(A \cup U(A))$ $\{x,x\} \in Pow^M(A \cup U(A))$
        using *Pow_rel_char*[*of* $A \cup U(A)$] **by** (*auto dest*:*transM*)
      ultimately
      show $\langle x,\ f(x)\rangle \in Pow^M(Pow^M(A \cup U(A)))$
        using *Pow_rel_char*[*of* $Pow^M(A \cup U(A))$] **unfolding** *Pair_def*
        by (*auto dest*:*transM*)
    qed
    moreover from ‹$M(A)$›
    have $M(\{y{\in}\ Pow^M(Pow^M(A \cup U(A))).\ \exists\, x[M].\ x{\in}A \wedge y = \langle x,\ f(x)\rangle\})$
      using *separation_f*
      by (*rule_tac separation_closed*) *simp_all*
    ultimately
    show *?thesis*
      by *simp*
  qed
  moreover from *this*
  have $\forall\, x[M].\ M(f(x))$
    using *lam_closed_imp_closed* **by** *simp*
  ultimately
  show *?thesis*
    using *assms*
    by (*rule_tac lam_replacement_iff_lam_closed*[*THEN iffD2*]) *simp_all*
qed

lemma *lam_replacement_domain'*:
  assumes $\forall\, A[M].\ separation(M,\ \lambda y.\ \exists\, x{\in}A.\ y = \langle x,\ domain(x)\rangle)$
  shows *lam_replacement*(M,*domain*)
proof -
  have $\forall\, x{\in}X.\ domain(x) \in Pow^M(\bigcup\bigcup\bigcup X)$ **if** $M(X)$ **for** X
  proof
    fix x
    assume $x{\in}X$
    moreover
    note ‹$M(X)$›
    moreover from *calculation*

```
    have M(x) by (auto dest:transM)
    ultimately
    show domain(x) ∈ Pow^M(⋃⋃⋃X)
      using mem_Pow_rel_abs[of domain(x) ⋃⋃⋃X]

      apply (auto simp:Pair_def)
      apply (rule_tac x=x in bexI)
       apply (rule_tac x={{xaa}, {xaa, ya}} in bexI)
        apply (rule_tac x={xaa} in bexI)
      by simp_all
  qed
  with assms
  show ?thesis
    using bounded_lam_replacement[of domain λX. Pow^M(⋃⋃⋃X)] by simp
qed
```

— Below we assume the replacement instance for *fst*. Alternatively it follows from the instance of separation assumed in this lemma.

```
lemma lam_replacement_fst':
  assumes ∀A[M]. separation(M, λy. ∃x∈A. y = ⟨x, fst(x)⟩)
  shows lam_replacement(M,fst)
proof -
  have ∀x∈X. fst(x) ∈ {0} ∪ ⋃⋃X if M(X) for X
  proof
    fix x
    assume x∈X
    moreover
    note ‹M(X)›
    moreover from calculation
    have M(x) by (auto dest:transM)
    ultimately
    show fst(x) ∈ {0} ∪ ⋃⋃X unfolding fst_def Pair_def
      by (auto, rule_tac [1] the_0) force— tricky! And slow. It doesn't work for
snd
  qed
  with assms
  show ?thesis
    using bounded_lam_replacement[of fst λX. {0} ∪ ⋃⋃X] by simp
qed

lemma lam_replacement_restrict:
assumes ∀A[M]. separation(M, λy. ∃x∈A. y = ⟨x, restrict(x,B)⟩)  M(B)
shows lam_replacement(M, λr . restrict(r,B))
proof -
  have ∀r∈R. restrict(r,B)∈Pow^M(⋃R) if M(R) for R
  proof -
    {
      fix r
      assume r∈R
```

```
    with ‹M(B)›
    have restrict(r,B)∈Pow(⋃R) M(restrict(r,B))
      using Union_upper subset_Pow_Union subset_trans[OF restrict_subset]
        transM[OF _ ‹M(R)›]
      by simp_all
  } then show ?thesis
    using Pow_rel_char that by simp
 qed
 with assms
 show ?thesis
  using bounded_lam_replacement[of λr . restrict(r,B) λX. Pow^M(⋃X)]
  by simp
qed

end

locale M_replacement = M_basic +
 assumes
  lam_replacement_domain: lam_replacement(M,domain)
  and
  lam_replacement_vimage: lam_replacement(M, λp. fst(p) -`` snd(p))
  and
  lam_replacement_fst: lam_replacement(M,fst)
  and
  lam_replacement_snd: lam_replacement(M,snd)
  and
  lam_replacement_Union: lam_replacement(M,Union)
  and
  id_separation:separation(M, λz. ∃x[M]. z = ⟨x, x⟩)
  and
  middle_separation: separation(M, λx. snd(fst(x))=fst(snd(x)))
  and
 middle_del_replacement: strong_replacement(M, λx y. y=⟨fst(fst(x)),snd(snd(x))⟩)
  and
  product_separation: separation(M, λx. fst(fst(x))=fst(snd(x)))
  and
  product_replacement:
  strong_replacement(M, λx y. y=⟨fst(fst(x)),⟨snd(fst(x)),snd(snd(x))⟩⟩)
  and
  lam_replacement_Upair:lam_replacement(M, λp. Upair(fst(p),snd(p)))
  and
  lam_replacement_Diff:lam_replacement(M, λp. fst(p) - snd(p))
  and
  lam_replacement_Image:lam_replacement(M, λp. fst(p) `` snd(p))
  and
  separation_fst_equal : M(a) ⟹ separation(M,λx . fst(x)=a)
  and
  separation_equal_fst2 : M(a) ⟹ separation(M,λx . fst(fst(x))=a)
  and
```

separation_equal_apply: $M(f) \Longrightarrow M(a) \Longrightarrow$ *separation*$(M,\lambda x.\ f`x=a)$
and
separation_restrict: $M(B) \Longrightarrow \forall\, A[M].$ *separation*$(M,\ \lambda y.\ \exists\, x{\in}A.\ y\ =\ \langle x,$ *restrict*$(x,\ B)\rangle)$
begin

lemmas *lam_replacement_restrict'* = *lam_replacement_restrict*[*OF separation_restrict*]

lemma *lam_replacement_imp_strong_replacement*:
assumes *lam_replacement*$(M,\ f)$
shows *strong_replacement*$(M,\ \lambda x\ y.\ y = f(x))$
proof -
{
fix A
assume $M(A)$
moreover
from *assms*
have *univalent*$(M,A,\lambda x\ y.\ y{=}\langle x,f(x)\rangle)$ **by** *simp*
moreover from *calculation assms*
obtain Y **where** *1*:$M(Y)\ \forall\, b[M].\ b \in Y \longleftrightarrow (\exists\, x[M].\ x \in A \wedge b = \langle x,f(x)\rangle)$
unfolding *lam_replacement_def strong_replacement_def*
by *auto*
moreover from *this*
have $M(\{snd(b)\ .\ b \in Y\})$
using *transM*[*OF* _ ‹$M(Y)$›] *lam_replacement_snd lam_replacement_imp_strong_replacement_aux*
RepFun_closed **by** *simp*
moreover
have $\{snd(b)\ .\ b \in Y\} = \{y\ .\ x{\in}A\ ,\ M(f(x)) \wedge y{=}f(x)\}$ (**is** *?L=?R*)
proof(*intro equalityI subsetI*)
fix x
assume $x{\in}$*?L*
moreover from *this*
obtain b **where** $b{\in}Y\ x{=}snd(b)\ M(b)$
using *transM*[*OF* _ ‹$M(Y)$›] **by** *auto*
moreover from *this 1*
obtain a **where** $a{\in}A\ b{=}\langle a,f(a)\rangle$ **by** *auto*
moreover from *calculation*
have $x{=}f(a)$ **by** *simp*
ultimately show $x{\in}$*?R*
by *auto*
next
fix z
assume $z{\in}$*?R*
moreover from *this*
obtain a **where** $a{\in}A\ z{=}f(a)\ M(a)\ M(f(a))$
using *transM*[*OF* _ ‹$M(A)$›]
by *auto*
moreover from *calculation this 1*
have $z{=}snd(\langle a,f(a)\rangle)\ \langle a,f(a)\rangle \in Y$ **by** *auto*

```
    ultimately
    show z∈?L by force
  qed
  ultimately
  have ∃ Z[M]. ∀ z[M]. z∈Z ⟷ (∃ a[M]. a∈A ∧ z=f(a))
    by (rule_tac rexI[where x={snd(b) . b ∈ Y}],auto)
 }
 then
 show ?thesis unfolding strong_replacement_def by simp
qed

lemma Collect_middle: {p ∈ (λx∈A. f(x)) × (λx∈{f(x) . x∈A}. g(x)) . snd(fst(p))=fst(snd(p))}
    = { ⟨⟨x,f(x)⟩,⟨f(x),g(f(x))⟩⟩ . x∈A }
 by (intro equalityI; auto simp:lam_def)

lemma RepFun_middle_del: { ⟨fst(fst(p)),snd(snd(p))⟩ . p ∈ { ⟨⟨x,f(x)⟩,⟨f(x),g(f(x))⟩⟩
. x∈A }}
      =  { ⟨x,g(f(x))⟩ . x∈A }
 by auto

lemma lam_replacement_imp_RepFun:
 assumes lam_replacement(M, f) M(A)
 shows M({y . x∈A , M(y) ∧ y=f(x)})
proof -
 from assms
 have univalent(M,A,λx y. y=⟨x,f(x)⟩) by simp
 moreover from calculation assms
 obtain Y where 1:M(Y) ∀ b[M]. b ∈ Y ⟷ (∃ x[M]. x ∈ A ∧ b = ⟨x,f(x)⟩)
   unfolding lam_replacement_def strong_replacement_def
   by auto
 moreover from this
 have M({snd(b) . b ∈ Y})
  using transM[OF _ ‹M(Y)›] lam_replacement_snd lam_replacement_imp_strong_replacement_aux
     RepFun_closed by simp
 moreover
 have {snd(b) . b ∈ Y} = {y . x∈A , M(y) ∧ y=f(x)} (is ?L=?R)
 proof(intro equalityI subsetI)
   fix x
   assume x∈?L
   moreover from this
   obtain b where b∈Y x=snd(b) M(b)
     using transM[OF _ ‹M(Y)›] by auto
   moreover from this 1
   obtain a where a∈A b=⟨a,f(a)⟩ by auto
   moreover from calculation
   have x=f(a) by simp
   ultimately show x∈?R
     by auto
 next
```

```
    fix z
    assume z∈?R
    moreover from this
    obtain a where a∈A z=f(a) M(a) M(f(a))
      using transM[OF _ ‹M(A)›]
      by auto
    moreover from calculation this 1
    have z=snd(⟨a,f(a)⟩) ⟨a,f(a)⟩ ∈ Y by auto
    ultimately
    show z∈?L by force
  qed
  ultimately
  show ?thesis by simp
qed

lemma lam_replacement_product:
  assumes lam_replacement(M,f) lam_replacement(M,g)
  shows lam_replacement(M, λx. ⟨f(x),g(x)⟩)
proof -
  {
    fix A
    let ?Y={y . x∈A , M(y) ∧ y=f(x)}
    let ?Y'={y . x∈A ,M(y) ∧  y=⟨x,f(x)⟩}
    let ?Z={y . x∈A , M(y) ∧ y=g(x)}
    let ?Z'={y . x∈A ,M(y) ∧  y=⟨x,g(x)⟩}
   have x∈C ⟹ y∈C ⟹ fst(x) = fst(y) ⟶ M(fst(y)) ∧ M(snd(x)) ∧ M(snd(y))
if M(C) for C y x
      using transM[OF _ that] by auto
    moreover
    note assms
    moreover
    assume M(A)
    moreover from ‹M(A)› assms(1)
    have M(?Y') M(?Y)
      using lam_replacement_imp_RepFun_Lam lam_replacement_imp_RepFun
by auto
    moreover from calculation
    have M(?Z) M(?Z')
      using lam_replacement_imp_RepFun_Lam lam_replacement_imp_RepFun
by auto
    moreover from calculation
    have M(?Y'×?Z')
      by simp
    moreover from this
    have M({p ∈ ?Y'×?Z' . fst(fst(p))=fst(snd(p))}) (is M(?P))
      using product_separation by simp
    moreover from calculation
    have M({ ⟨fst(fst(p)),⟨snd(fst(p)),snd(snd(p))⟩⟩ . p∈?P }) (is M(?R))
      using RepFun_closed[OF product_replacement ‹M(?P)› ] by simp
```

```
    ultimately
    have b ∈ ?R ⟷ (∃ x[M]. x ∈ A ∧ b = ⟨x,⟨f(x),g(x)⟩⟩) if M(b) for b
      using that
      apply(intro iffI)apply(auto)[1]
    proof -
      assume  ∃ x[M]. x ∈ A ∧ b = ⟨x, f(x), g(x)⟩
      moreover from this
      obtain x where M(x) x∈A b= ⟨x, ⟨f(x), g(x)⟩⟩
        by auto
      moreover from calculation that
      have M(⟨x,f(x)⟩) M(⟨x,g(x)⟩) by auto
      moreover from calculation
      have ⟨x,f(x)⟩ ∈ ?Y' ⟨x,g(x)⟩ ∈ ?Z' by auto
      moreover from calculation
      have ⟨⟨x,f(x)⟩,⟨x,g(x)⟩⟩∈?Y'×?Z' by auto
      moreover from calculation
      have ⟨⟨x,f(x)⟩,⟨x,g(x)⟩⟩ ∈ ?P
        (is ?p∈?P)
        by auto
      moreover from calculation
      have b = ⟨fst(fst(?p)),⟨snd(fst(?p)),snd(snd(?p))⟩⟩ by auto
      moreover from calculation
      have ⟨fst(fst(?p)),⟨snd(fst(?p)),snd(snd(?p))⟩⟩∈?R
        by(rule_tac RepFunI[of ?p ?P], simp)
      ultimately show b∈?R by simp
    qed
    with ‹M(?R)›
    have ∃ Y[M]. ∀ b[M]. b ∈ Y ⟷ (∃ x[M]. x ∈ A ∧ b = ⟨x,⟨f(x),g(x)⟩⟩)
      by (rule_tac rexI[where x=?R],simp_all)
  }
  with assms
  show ?thesis using lam_replacement_def strong_replacement_def by simp
qed

lemma lam_replacement_hcomp:
  assumes lam_replacement(M,f) lam_replacement(M,g) ∀ x[M]. M(f(x))
  shows lam_replacement(M, λx. g(f(x)))
proof -
  {
    fix A
    let ?Y={y . x∈A , y=f(x)}
    let ?Y'={y . x∈A , y=⟨x,f(x)⟩}
    have ∀ x∈C. M(⟨fst(fst(x)),snd(snd(x))⟩) if M(C) for C
      using transM[OF _ that] by auto
    moreover
    note assms
    moreover
    assume M(A)
    moreover from assms
```

```
  have eq:?Y = {y . x∈A ,M(y) ∧ y=f(x)}  ?Y' = {y . x∈A ,M(y) ∧ y=⟨x,f(x)⟩}
    using transM[OF _ ‹M(A)›] by auto
   moreover from ‹M(A)› assms(1)
   have M(?Y') M(?Y)
    using lam_replacement_imp_RepFun_Lam lam_replacement_imp_RepFun
eq by auto
   moreover from calculation
   have M({z . y∈?Y , M(z) ∧ z=⟨y,g(y)⟩}) (is M(?Z))
    using lam_replacement_imp_RepFun_Lam by auto
   moreover from calculation
   have M(?Y'×?Z)
    by simp
   moreover from this
   have M({p ∈ ?Y'×?Z . snd(fst(p))=fst(snd(p))}) (is M(?P))
    using middle_separation by simp
   moreover from calculation
   have M({ ⟨fst(fst(p)),snd(snd(p))⟩ . p∈?P }) (is M(?R))
    using RepFun_closed[OF middle_del_replacement ‹M(?P)›] by simp
   ultimately
   have b ∈ ?R ⟷ (∃x[M]. x ∈ A ∧ b = ⟨x,g(f(x))⟩) if M(b) for b
    using that assms(3)
    apply(intro iffI) apply(auto)[1]
   proof -
    assume ∃x[M]. x ∈ A ∧ b = ⟨x, g(f(x))⟩
    moreover from this
    obtain x where M(x) x∈A b= ⟨x, g(f(x))⟩
     by auto
    moreover from calculation that assms(3)
    have M(f(x)) M(g(f(x))) by auto
    moreover from calculation
    have ⟨x,f(x)⟩ ∈ ?Y' by auto
    moreover from calculation
    have ⟨f(x),g(f(x))⟩∈?Z by auto
    moreover from calculation
    have ⟨⟨x,f(x)⟩,⟨f(x),g(f(x))⟩⟩ ∈ ?P
     (is ?p∈?P)
     by auto
    moreover from calculation
    have b = ⟨fst(fst(?p)),snd(snd(?p))⟩ by auto
    moreover from calculation
    have ⟨fst(fst(?p)),snd(snd(?p))⟩∈?R
     by(rule_tac RepFunI[of ?p ?P], simp)
    ultimately show b∈?R by simp
   qed
   with ‹M(?R)›
   have ∃ Y[M]. ∀ b[M]. b ∈ Y ⟷ (∃x[M]. x ∈ A ∧ b = ⟨x,g(f(x))⟩)
    by (rule_tac rexI[where x=?R],simp_all)
  }
  with assms
```

show *?thesis* **using** *lam_replacement_def strong_replacement_def* **by** *simp*
qed

lemma *lam_replacement_Collect* :
assumes $M(A)$ $\forall\, x[M].$ *separation*$(M,F(x))$
separation$(M,\lambda p\ .\ \forall\, x{\in}A.\ x{\in}snd(p) \longleftrightarrow F(fst(p),x))$
shows *lam_replacement*$(M,\lambda x.\ \{y{\in}A\ .\ F(x,y)\})$
proof -
{
fix Z
let $?Y{=}\lambda z.\{x{\in}A\ .\ F(z,x)\}$
assume $M(Z)$
moreover from *this*
have $M(?Y(z))$ **if** $z{\in}Z$ **for** z
using *assms that transM*[*of* _ Z] **by** *simp*
moreover from *this*
have $?Y(z){\in}Pow^M(A)$ **if** $z{\in}Z$ **for** z
using *Pow_rel_char that assms* **by** *auto*
moreover from *calculation* ‹$M(A)$›
have $M(Z{\times}Pow^M(A))$ **by** *simp*
moreover from *this*
have $M(\{p \in Z{\times}Pow^M(A)\ .\ \forall\, x{\in}A.\ x{\in}snd(p) \longleftrightarrow F(fst(p),x)\})$ (**is** $M(?P)$)
using *assms* **by** *simp*
ultimately
have $b \in ?P \longleftrightarrow (\exists\, z[M].\ z{\in}Z \wedge b{=}\langle z,?Y(z)\rangle)$ **if** $M(b)$ **for** b
using *assms*(*1*) *Pow_rel_char*[*OF* ‹$M(A)$›] *that*
by(*intro iffI*,*auto*,*intro equalityI*,*auto*)
with ‹$M(?P)$›
have $\exists\, Y[M].\ \forall\, b[M].\ b \in Y \longleftrightarrow (\exists\, z[M].\ z \in Z \wedge b = \langle z,?Y(z)\rangle)$
by (*rule_tac rexI*[**where** $x{=}?P$],*simp_all*)
}
then
show *?thesis*
unfolding *lam_replacement_def strong_replacement_def*
by *simp*
qed

lemma *lam_replacement_hcomp2*:
assumes *lam_replacement*(M,f) *lam_replacement*(M,g)
$\forall\, x[M].\ M(f(x))$ $\forall\, x[M].\ M(g(x))$
lam_replacement$(M,\ \lambda p.\ h(fst(p),snd(p)))$
$\forall\, x[M].\ \forall\, y[M].\ M(h(x,y))$
shows *lam_replacement*$(M,\ \lambda x.\ h(f(x),g(x)))$
using *assms lam_replacement_product*[*of f g*]
lam_replacement_hcomp[*of* $\lambda x.\ \langle f(x),\ g(x)\rangle$ $\lambda\langle x,y\rangle.\ h(x,y)$]
unfolding *split_def* **by** *simp*

lemma *strong_replacement_separation_aux* :

```
  assumes strong_replacement(M,λ x y . y=f(x)) separation(M,P)
  shows strong_replacement(M, λx y . P(x) ∧ y=f(x))
proof -
  {
    fix A
    let ?Q=λX. ∀ b[M]. b ∈ X ⟷ (∃ x[M]. x ∈ A ∧ P(x) ∧ b = f(x))
    assume M(A)
    moreover from this
    have M({x∈A . P(x)}) (is M(?B)) using assms by simp
    moreover from calculation assms
    obtain Y where M(Y) ∀ b[M]. b ∈ Y ⟷ (∃ x[M]. x ∈ ?B ∧ b = f(x))
      unfolding strong_replacement_def by auto
    then
    have ∃ Y[M]. ∀ b[M]. b ∈ Y ⟷ (∃ x[M]. x ∈ A ∧ P(x) ∧ b = f(x))
      using rexI[of ?Q _ M] by simp
  }
  then
  show ?thesis
    unfolding strong_replacement_def by simp
qed

lemma lam_replacement_separation :
  assumes lam_replacement(M,f) separation(M,P)
  shows strong_replacement(M, λx y . P(x) ∧ y=⟨x,f(x)⟩)
  using strong_replacement_separation_aux assms
  unfolding lam_replacement_def
  by simp

lemmas strong_replacement_separation =
 strong_replacement_separation_aux[OF lam_replacement_imp_strong_replacement]

lemma lam_replacement_identity: lam_replacement(M,λx. x)
proof -
  {
    fix A
    assume M(A)
    moreover from this
    have id(A) = {z∈ A×A. ∃ x[M]. z=⟨x,x⟩}
      unfolding id_def lam_def by (auto dest:transM)
    moreover from calculation
    have M({z∈ A×A. ∃ x[M]. z=⟨x,x⟩})
      using id_separation by simp
    ultimately
    have M(id(A)) by simp
  }
  then
  show ?thesis using lam_replacement_iff_lam_closed
    unfolding id_def by simp
qed
```

lemma *lam_replacement_Un*: *lam_replacement*(*M*, $\lambda p.$ *fst*(*p*) $\cup$ *snd*(*p*))
using *lam_replacement_Upair lam_replacement_Union*
lam_replacement_hcomp[**where** *g*=*Union* **and** *f*=$\lambda p.$ *Upair*(*fst*(*p*),*snd*(*p*))]
unfolding *Un_def* **by** *simp*

lemma *lam_replacement_cons*: *lam_replacement*(*M*, $\lambda p.$ *cons*(*fst*(*p*),*snd*(*p*)))
using *lam_replacement_Upair*
lam_replacement_hcomp2[*of* _ _ ($\cup$)]
lam_replacement_hcomp2[*of fst fst Upair*]
lam_replacement_Un lam_replacement_fst lam_replacement_snd
unfolding *cons_def*
by *auto*

lemma *lam_replacement_sing*: *lam_replacement*(*M*, $\lambda x.$ {*x*})
using *lam_replacement_constant lam_replacement_cons*
lam_replacement_hcomp2[*of* $\lambda x.$ *x* λ_. *0 cons*]
by (*force intro*: *lam_replacement_identity*)

lemmas *tag_replacement* = *lam_replacement_constant*[*unfolded lam_replacement_def*]

lemma *lam_replacement_id2*: *lam_replacement*(*M*, $\lambda x.$ $\langle x, x\rangle$)
using *lam_replacement_identity lam_replacement_product*[*of* $\lambda x.$ *x* $\lambda x.$ *x*]
by *simp*

lemmas *id_replacement* = *lam_replacement_id2*[*unfolded lam_replacement_def*]

lemma *lam_replacement_apply2*:*lam_replacement*(*M*, $\lambda p.$ *fst*(*p*) ‘ *snd*(*p*))
using *lam_replacement_sing lam_replacement_fst lam_replacement_snd*
lam_replacement_Image lam_replacement_Union
unfolding *apply_def*
by (*rule_tac lam_replacement_hcomp*[*of* _ *Union*],
rule_tac lam_replacement_hcomp2[*of* _ _ (‘‘)])
(*force intro*:*lam_replacement_hcomp*)+

definition *map_snd* **where**
map_snd(*X*) = {*snd*(*z*) . *z*$\in$*X*}

lemma *map_sndE*: *y*$\in$*map_snd*(*X*) $\Longrightarrow$ $\exists$ *p*$\in$*X*. *y*=*snd*(*p*)
unfolding *map_snd_def* **by** *auto*

lemma *map_sndI* : $\exists$ *p*$\in$*X*. *y*=*snd*(*p*) $\Longrightarrow$ *y*$\in$*map_snd*(*X*)
unfolding *map_snd_def* **by** *auto*

lemma *map_snd_closed*: *M*(*x*) $\Longrightarrow$ *M*(*map_snd*(*x*))
unfolding *map_snd_def*
using *lam_replacement_imp_strong_replacement*[*OF lam_replacement_snd*]
RepFun_closed snd_closed[*OF transM*[*of* _ *x*]]
by *simp*

lemma *lam_replacement_imp_lam_replacement_RepFun*:
assumes *lam_replacement*(M, f) $\forall\, x[M].\ M(f(x))$
separation(M, $\lambda x.\ ((\forall\, y{\in}snd(x).\ fst(y)\ \in\ fst(x))\ \wedge\ (\forall\, y{\in}fst(x).\ \exists\, u{\in}snd(x).\ y{=}fst(u))))$
and
lam_replacement_RepFun_snd:*lam_replacement*(M,*map_snd*)
shows *lam_replacement*(M, $\lambda x.\ \{f(y)\ .\ y{\in}x\}$)
proof -
have *f_closed*:$M(<fst(z),map\_snd(snd(z))>)$ **if** $M(z)$ **for** z
using *pair_in_M_iff fst_closed snd_closed map_snd_closed that*
by *simp*
have *p_closed*:$M(<x,\{f(y)\ .\ y{\in}x\}>)$ **if** $M(x)$ **for** x
using *pair_in_M_iff RepFun_closed lam_replacement_imp_strong_replacement*
transM[*OF* _ *that*] *that assms* **by** *auto*
{
fix A
assume $M(A)$
then
have $M(\{<y,f(y)>\ .\ y{\in}x\})$ **if** $x{\in}A$ **for** x
using *lam_replacement_iff_lam_closed assms that transM*[*of* _ A]
unfolding *lam_def* **by** *simp*
from *assms* ‹$M(A)$›
have $\forall\, x{\in}\bigcup A.\ M(f(x))$
using *transM*[*of* _ $\bigcup A$] **by** *auto*
with *assms* ‹$M(A)$›
have $M(\{<y,f(y)>\ .\ y\ \in\ \bigcup A\})$ (**is** $M(?fUnA)$)
using *lam_replacement_iff_lam_closed*[*THEN iffD1*,*OF assms*(*2*) *assms*(*1*)]
unfolding *lam_def*
by *simp*
with ‹$M(A)$›
have $M(Pow\_rel(M,?fUnA))$ **by** *simp*
with ‹$M(A)$›
have $M(\{z{\in}A{\times}Pow\_rel(M,?fUnA)\ .\ ((\forall\, y{\in}snd(z).\ fst(y)\in fst(z))\wedge(\forall\, y{\in}fst(z).\ \exists\, u{\in}snd(z).\ y{=}fst(u)))\})$ (**is** $M(?T)$)
using *assms*(*3*) **by** *simp*
then
have *1*:$M(\{<fst(z),map\_snd(snd(z))>\ .\ z{\in}?T\})$ (**is** $M(?Y)$)
using *lam_replacement_product*[*OF lam_replacement_fst*
lam_replacement_hcomp[*OF lam_replacement_snd lam_replacement_RepFun_snd*]]
RepFun_closed lam_replacement_imp_strong_replacement
f_closed[*OF transM*[*OF* _ ‹$M(?T)$›]]
by *simp*
have *2*:$?Y = \{<x,\{f(y)\ .\ y{\in}x\}>\ .\ x{\in}A\}$ (**is** _ = $?R$)
proof(*intro equalityI subsetI*)
fix p
assume $p{\in}?R$
with ‹$M(A)$›
obtain x **where** $x{\in}A$ $p{=}<x,\{f(y)\ .\ y\ \in\ x\}>$ $M(x)$

using *transM*[*OF* _ ‹*M*(*A*)›]
by *auto*
moreover from *calculation*
have *M*({<*y*,*f*(*y*)> . *y*∈*x*}) (**is** *M*(*?Ux*))
using *lam_replacement_iff_lam_closed assms*
unfolding *lam_def* **by** *auto*
moreover from *calculation*
have *?Ux* ⊆ *?fUnA*
by *auto*
moreover from *calculation*
have *?Ux* ∈ *Pow_rel*(*M*,*?fUnA*)
using *Pow_rel_char*[*OF* ‹*M*(*?fUnA*)›] **by** *simp*
moreover from *calculation*
have ∀ *u*∈*x*. ∃ *w*∈*?Ux*. *u*=*fst*(*w*)
by *force*
moreover from *calculation*
have <*x*,*?Ux*> ∈ *?T* **by** *auto*
moreover from *calculation*
have {*f*(*y*).*y*∈*x*} = *map_snd*(*?Ux*)
unfolding *map_snd_def*
by(*intro equalityI*,*auto*)
ultimately
show *p*∈*?Y*
by (*auto*,*rule_tac bexI*[**where** *x*=*x*],*simp_all*,*rule_tac bexI*[**where** *x*=*?Ux*],*simp_all*)
next
fix *u*
assume *u*∈*?Y*
moreover from *this*
obtain *z* **where** *z*∈*?T u*=<*fst*(*z*),*map_snd*(*snd*(*z*))>
by *blast*
moreover from *calculation*
obtain *x U* **where** *1*:*x*∈*A U*∈*Pow_rel*(*M*,*?fUnA*) (∀ *u*∈*U*. *fst*(*u*) ∈ *x*) ∧
(∀ *w*∈*x*. ∃ *v*∈*U*. *w*=*fst*(*v*)) *z*=<*x*,*U*>
by *force*
moreover from *this*
have *fst*(*u*)∈⋃ *A snd*(*u*) = *f*(*fst*(*u*)) **if** *u*∈*U* **for** *u*
using *that Pow_rel_char*[*OF* ‹*M*(*?fUnA*)›]
by *auto*
moreover from *calculation*
have *map_snd*(*U*) = {*f*(*y*) . *y*∈*x*}
unfolding *map_snd_def*
by(*intro equalityI subsetI*,*auto*)
moreover from *calculation*
have *u*=<*x*,*map_snd*(*U*)>
by *simp*
ultimately
show *u*∈*?R*
by (*auto*)
qed

```
    from 1 2
    have M({<x,{f(y) . y∈x}> . x∈A})
      by simp
  }
  then
  have ∀ A[M]. M(λx∈A. {f(y) . y∈x})
    unfolding lam_def by auto
  then
  show ?thesis
    using lam_replacement_iff_lam_closed[THEN iffD2] p_closed
    by simp
qed


lemma lam_replacement_apply:M(S) ⟹ lam_replacement(M, λx.  S ‘ x)
 using lam_replacement_Union lam_replacement_constant lam_replacement_identity
    lam_replacement_Image lam_replacement_cons
     lam_replacement_hcomp2[of _ _ Image] lam_replacement_hcomp2[of λx. x
λ_. 0 cons]
  unfolding apply_def
 by (rule_tac lam_replacement_hcomp[of_ Union]) (force intro:lam_replacement_hcomp)+

lemma apply_replacement:M(S) ⟹ strong_replacement(M, λx y. y = S ‘ x)
   using lam_replacement_apply lam_replacement_imp_strong_replacement by
simp

lemma lam_replacement_id_const: M(b) ⟹ lam_replacement(M, λx. ⟨x, b⟩)
  using lam_replacement_identity lam_replacement_constant
    lam_replacement_product[of λx. x λx. b] by simp

lemmas pospend_replacement = lam_replacement_id_const[unfolded lam_replacement_def]

lemma lam_replacement_const_id: M(b) ⟹ lam_replacement(M, λz. ⟨b, z⟩)
  using lam_replacement_identity lam_replacement_constant
    lam_replacement_product[of λx. b λx. x] by simp

lemmas prepend_replacement = lam_replacement_const_id[unfolded lam_replacement_def]

lemma lam_replacement_apply_const_id: M(f) ⟹ M(z) ⟹
      lam_replacement(M, λx. f ‘ ⟨z, x⟩)
  using lam_replacement_const_id[of z] lam_replacement_apply[of f]
    lam_replacement_hcomp[of λx. ⟨z, x⟩ λx. f‘x] by simp

lemmas apply_replacement2 = lam_replacement_apply_const_id[unfolded lam_replacement_def]

lemma lam_replacement_Inl: lam_replacement(M, Inl)
  using lam_replacement_identity lam_replacement_constant
    lam_replacement_product[of λx. 0 λx. x]
  unfolding Inl_def by simp
```

lemma *lam_replacement_Inr*: *lam_replacement*(*M*, *Inr*)
using *lam_replacement_identity lam_replacement_constant*
lam_replacement_product[*of* λ*x*. *1* λ*x*. *x*]
unfolding *Inr_def* **by** *simp*

lemmas *Inl_replacement1* = *lam_replacement_Inl*[*unfolded lam_replacement_def*]

lemma *lam_replacement_Diff'*: *M*(*X*) ⟹ *lam_replacement*(*M*, λ*x*. *x* - *X*)
using *lam_replacement_Diff*
by (*force intro*: *lam_replacement_hcomp2 lam_replacement_constant*
lam_replacement_identity)+

lemmas *Pair_diff_replacement* = *lam_replacement_Diff'*[*unfolded lam_replacement_def*]

lemma *diff_Pair_replacement*: *M*(*p*) ⟹ *strong_replacement*(*M*, λ*x y* . *y*=⟨*x*,*x*-{*p*}⟩)
using *Pair_diff_replacement* **by** *simp*

lemma *lam_replacement_swap*: *lam_replacement*(*M*, λ*x*. ⟨*snd*(*x*),*fst*(*x*)⟩)
using *lam_replacement_fst lam_replacement_snd*
lam_replacement_product[*of snd fst*] **by** *simp*

lemma *swap_replacement*:*strong_replacement*(*M*, λ*x y*. *y* = ⟨*x*, (λ⟨*x*,*y*⟩. ⟨*y*, *x*⟩)(*x*)⟩)
using *lam_replacement_swap* **unfolding** *lam_replacement_def split_def* **by** *simp*

lemma *lam_replacement_Un_const*:*M*(*b*) ⟹ *lam_replacement*(*M*, λ*x*. *x* ∪ *b*)
using *lam_replacement_Un lam_replacement_hcomp2*[*of* _ _ (∪)]
lam_replacement_constant[*of b*] *lam_replacement_identity* **by** *simp*

lemmas *tag_union_replacement* = *lam_replacement_Un_const*[*unfolded lam_replacement_def*]

lemma *lam_replacement_csquare*: *lam_replacement*(*M*,λ*p*. ⟨*fst*(*p*) ∪ *snd*(*p*), *fst*(*p*), *snd*(*p*)⟩)
using *lam_replacement_Un lam_replacement_fst lam_replacement_snd*
by (*fast intro*: *lam_replacement_product lam_replacement_hcomp2*)

lemma *csquare_lam_replacement*:*strong_replacement*(*M*, λ*x y*. *y* = ⟨*x*, (λ⟨*x*,*y*⟩. ⟨*x* ∪ *y*, *x*, *y*⟩)(*x*)⟩)
using *lam_replacement_csquare* **unfolding** *split_def lam_replacement_def* .

lemma *lam_replacement_assoc*:*lam_replacement*(*M*,λ*x*. ⟨*fst*(*fst*(*x*)), *snd*(*fst*(*x*)), *snd*(*x*)⟩)
using *lam_replacement_fst lam_replacement_snd*
by (*force intro*: *lam_replacement_product lam_replacement_hcomp*)

lemma *assoc_replacement*:*strong_replacement*(*M*, λ*x y*. *y* = ⟨*x*, (λ⟨⟨*x*,*y*⟩,*z*⟩. ⟨*x*, *y*, *z*⟩)(*x*)⟩)
using *lam_replacement_assoc* **unfolding** *split_def lam_replacement_def* .

lemma *lam_replacement_prod_fun*: $M(f) \Longrightarrow M(g) \Longrightarrow$ *lam_replacement*$(M,\lambda x.$
$\langle f\ `\ fst(x),\ g\ `\ snd(x)\rangle)$
using *lam_replacement_fst lam_replacement_snd*
by (*force intro*: *lam_replacement_product lam_replacement_hcomp lam_replacement_apply*)

lemma *prod_fun_replacement*:$M(f) \Longrightarrow M(g) \Longrightarrow$
strong_replacement$(M,\ \lambda x\ y.\ y = \langle x,\ (\lambda\langle w,y\rangle.\ \langle f\ `\ w,\ g\ `\ y\rangle)(x)\rangle)$
using *lam_replacement_prod_fun* **unfolding** *split_def lam_replacement_def* **.**

lemma *lam_replacement_vimage_sing*: *lam_replacement*$(M,\ \lambda p.\ fst(p)\ -``\{snd(p)\})$
using *lam_replacement_hcomp*[*OF lam_replacement_snd lam_replacement_sing*]
lam_replacement_hcomp2[*OF lam_replacement_fst _ _ _ lam_replacement_vimage*]
by *simp*

lemma *lam_replacement_vimage_sing_fun*: $M(f) \Longrightarrow$ *lam_replacement*$(M,\ \lambda x.$
$f\ -``\{x\})$
using *lam_replacement_hcomp2*[*OF lam_replacement_constant*[*of f*]
lam_replacement_identity _ _ lam_replacement_vimage_sing]
by *simp*

lemma *converse_apply_projs*: $\forall\, x[M].\ \bigcup\ (fst(x)\ -``\{snd(x)\}) = converse(fst(x))$
$`\ (snd(x))$
using *converse_apply_eq* **by** *auto*

lemma *lam_replacement_converse_app*: *lam_replacement*$(M,\ \lambda p.\ converse(fst(p))$
$`\ snd(p))$
using *lam_replacement_cong*[*OF _ converse_apply_projs*]
lam_replacement_hcomp[*OF lam_replacement_vimage_sing lam_replacement_Union*]
by *simp*

lemmas *cardinal_lib_assms4* = *lam_replacement_vimage_sing_fun*[*unfolded lam_replacement_def*]

lemma *lam_replacement_sing_const_id*:
$M(x) \Longrightarrow$ *lam_replacement*$(M,\ \lambda y.\ \{\langle x,\ y\rangle\})$
using *lam_replacement_hcomp*[*OF lam_replacement_const_id*[*of x*]]
lam_replacement_sing pair_in_M_iff
by *simp*

lemma *tag_singleton_closed*: $M(x) \Longrightarrow M(z) \Longrightarrow M(\{\{\langle z,\ y\rangle\}\ .\ y \in x\})$
using *RepFun_closed*[**where** $A{=}x$ **and** $f{=}\lambda\ u.\ \{\langle z,u\rangle\}$]
lam_replacement_imp_strong_replacement lam_replacement_sing_const_id
transM[*of _ x*]
by *simp*

lemma *case_closed* :
assumes $\forall\, x[M].\ M(f(x))\ \forall\, x[M].\ M(g(x))$
shows $\forall\, x[M].\ M(case(f,g,x))$
unfolding *case_def split_def cond_def*

using *assms* **by** *simp*

lemma *lam_replacement_case* :
assumes *lam_replacement*(*M*,*f*) *lam_replacement*(*M*,*g*)
$\forall\, x[M].\ M(f(x))\ \forall\, x[M].\ M(g(x))$
shows *lam_replacement*(*M*, λx . *case*(*f*,*g*,*x*))
unfolding *case_def split_def cond_def*
using *lam_replacement_if separation_fst_equal*
lam_replacement_hcomp[*of snd g*]
lam_replacement_hcomp[*of snd f*]
lam_replacement_snd assms
by *simp*

lemma *Pi_replacement1*: $M(x) \Longrightarrow M(y) \Longrightarrow$ *strong_replacement*(*M*, $\lambda ya\ z.\ ya \in y \wedge z = \{\langle x,\ ya\rangle\}$)
using *lam_replacement_imp_strong_replacement*
strong_replacement_separation[*OF lam_replacement_sing_const_id*[*of x*],**where** *P*=λx . *x* ∈*y*]
separation_in
by *simp*

lemma *surj_imp_inj_replacement1*:
$M(f) \Longrightarrow M(x) \Longrightarrow$ *strong_replacement*(*M*, $\lambda y\ z.\ y \in f$ -`` $\{x\} \wedge z = \{\langle x,\ y\rangle\}$)
using *Pi_replacement1 vimage_closed singleton_closed*
by *simp*

lemmas *domain_replacement* = *lam_replacement_domain*[*unfolded lam_replacement_def*]

lemma *domain_replacement_simp*: *strong_replacement*(*M*, $\lambda x\ y.\ y$=*domain*(*x*))
using *lam_replacement_domain lam_replacement_imp_strong_replacement* **by** *simp*

lemma *un_Pair_replacement*: $M(p) \Longrightarrow$ *strong_replacement*(*M*, $\lambda x\ y\ .\ y = x \cup \{p\}$)
using *lam_replacement_Un_const*[*THEN lam_replacement_imp_strong_replacement*]
by *simp*

lemma *restrict_strong_replacement*: $M(A) \Longrightarrow$ *strong_replacement*(*M*, $\lambda x\ y.\ y$=*restrict*(*x*,*A*))
using *lam_replacement_restrict separation_restrict*
lam_replacement_imp_strong_replacement
by *simp*

lemma *diff_replacement*: $M(X) \Longrightarrow$ *strong_replacement*(*M*, $\lambda x\ y.\ y = x$ - *X*)
using *lam_replacement_Diff'*[*THEN lam_replacement_imp_strong_replacement*]
by *simp*

lemma *lam_replacement_succ*:
lam_replacement(*M*,λz . *succ*(*z*))
unfolding *succ_def*
using *lam_replacement_hcomp2*[*of* $\lambda x.\ x\ \lambda x.\ x$ *cons*]

lam_replacement_cons lam_replacement_identity
by *simp*

lemma *lam_replacement_hcomp_Least*:
assumes *lam_replacement*(*M*, *g*) *lam_replacement*(*M*,λ*x*. *μ i*. *x*∈*F*(*i*,*x*))
∀ *x*[*M*]. *M*(*g*(*x*)) ⋀*x i*. *M*(*x*) ⟹ *i* ∈ *F*(*i*, *x*) ⟹ *M*(*i*)
shows *lam_replacement*(*M*,λ*x*. *μ i*. *g*(*x*)∈*F*(*i*,*g*(*x*)))
using *assms*
by (*rule_tac lam_replacement_hcomp*[*of* _ λ*x*. *μ i*. *x*∈*F*(*i*,*x*)])
(*auto intro*:*Least_closed'*)

end

locale *M_replacement_extra* = *M_replacement* +
assumes
lam_replacement_minimum:*lam_replacement*(*M*, λ*p*. *minimum*(*fst*(*p*),*snd*(*p*)))
and
lam_replacement_RepFun_cons:*lam_replacement*(*M*, λ*p*. *RepFun*(*fst*(*p*), λ*x*.
{⟨*snd*(*p*),*x*⟩}))
— This one is too particular: It is for *Sigfun*. I would like greater modularity here.

begin
lemma *lam_replacement_Sigfun*:
assumes *lam_replacement*(*M*,*f*) ∀ *y*[*M*]. *M*(*f*(*y*))
shows *lam_replacement*(*M*, λ*x*. *Sigfun*(*x*,*f*))
using *lam_replacement_Union lam_replacement_identity*
lam_replacement_sing[*THEN lam_replacement_imp_strong_replacement*]
unfolding *Sigfun_def*
apply (*rule_tac lam_replacement_hcomp*[*of* _ *Union*],*simp_all*)
apply (*rule lam_replacement_RepFun_cons*[*THEN* [*5*] *lam_replacement_hcomp2*])
apply(*simp_all add*:*assms*)
using *assms tag_singleton_closed*
by (*simp_all*)

20.2 Particular instances

lemma *surj_imp_inj_replacement2*:
M(*f*) ⟹ *strong_replacement*(*M*, λ*x z*. *z* = *Sigfun*(*x*, λ*y*. *f* -‘‘ {*y*}))
using *lam_replacement_imp_strong_replacement lam_replacement_Sigfun*
lam_replacement_vimage_sing_fun
by *simp*

lemma *lam_replacement_minimum_vimage*:
M(*f*) ⟹ *M*(*r*) ⟹ *lam_replacement*(*M*, λ*x*. *minimum*(*r*, *f* -‘‘ {*x*}))
using *lam_replacement_minimum lam_replacement_vimage_sing_fun lam_replacement_constant*
by (*rule_tac lam_replacement_hcomp2*[*of* _ _ *minimum*])
(*force intro*: *lam_replacement_identity*)+

lemmas *surj_imp_inj_replacement4 = lam_replacement_minimum_vimage*[*unfolded lam_replacement_def*]

lemma *lam_replacement_Pi*: $M(y) \Longrightarrow$ *lam_replacement*$(M,\ \lambda x.\ \bigcup xa{\in}y.\ \{\langle x,\ xa\rangle\})$
using *lam_replacement_Union lam_replacement_identity lam_replacement_constant lam_replacement_sing_const_id*[*THEN lam_replacement_imp_strong_replacement*] *lam_replacement_hcomp2*[*of* $\lambda x.\ \langle \_,x\rangle\ \lambda \_.\ 0$ *cons*, *THEN lam_replacement_imp_strong_replacement*] **unfolding** *apply_def*
apply (*rule_tac lam_replacement_hcomp*[*of* _ *Union*])
apply (*auto intro*:*RepFun_closed dest*:*transM*)
by (*rule lam_replacement_RepFun_cons*[*THEN* [5] *lam_replacement_hcomp2*])
(*auto intro*:*RepFun_closed dest*:*transM*)

lemma *Pi_replacement2*: $M(y) \Longrightarrow$ *strong_replacement*$(M,\ \lambda x\ z.\ z = (\bigcup xa{\in}y.\ \{\langle x,\ xa\rangle\}))$
using *lam_replacement_Pi*[*THEN lam_replacement_imp_strong_replacement*, *of y*]
proof -
assume $M(y)$
then
have $M(x) \Longrightarrow M(\bigcup xa{\in}y.\ \{\langle x,\ xa\rangle\})$ **for** x
using *lam_replacement_sing_const_id*[*THEN lam_replacement_imp_strong_replacement*]
by (*rule_tac Union_closed RepFun_closed*, *simp_all* | *safe*)+
(*force dest*: *transM*) — a bit slow
with ‹$M(y)$›
show *?thesis*
using *lam_replacement_Pi*[*THEN lam_replacement_imp_strong_replacement*, *of y*]
by *blast*
qed

lemma *if_then_Inj_replacement*:
shows $M(A) \Longrightarrow$ *strong_replacement*$(M,\ \lambda x\ y.\ y = \langle x,\ \mathit{if}\ x \in A\ \mathit{then}\ \mathit{Inl}(x)\ \mathit{else}\ \mathit{Inr}(x)\rangle)$
using *lam_replacement_if lam_replacement_Inl lam_replacement_Inr separation_in*
unfolding *lam_replacement_def*
by *simp*

lemma *lam_if_then_replacement*:
$M(b) \Longrightarrow$
$M(a) \Longrightarrow M(f) \Longrightarrow$ *strong_replacement*$(M,\ \lambda y\ ya.\ ya = \langle y,\ \mathit{if}\ y = a\ \mathit{then}\ b\ \mathit{else}\ f\ `\ y\rangle)$
using *lam_replacement_if lam_replacement_apply lam_replacement_constant separation_equal*
unfolding *lam_replacement_def*
by *simp*

lemma *if_then_replacement*:

$M(A) \Longrightarrow M(f) \Longrightarrow M(g) \Longrightarrow strong\_replacement(M, \lambda x\ y.\ y = \langle x, if\ x \in A\ then\ f\ `\ x\ else\ g\ `\ x\rangle)$
using *lam_replacement_if lam_replacement_apply*[*of f*] *lam_replacement_apply*[*of g*]
separation_in
unfolding *lam_replacement_def*
by *simp*

lemma *ifx_replacement*:
$M(f) \Longrightarrow$
$M(b) \Longrightarrow strong\_replacement(M, \lambda x\ y.\ y = \langle x, if\ x \in range(f)\ then\ converse(f)\ `\ x\ else\ b\rangle)$
using *lam_replacement_if lam_replacement_apply lam_replacement_constant*
separation_in
unfolding *lam_replacement_def*
by *simp*

lemma *if_then_range_replacement2*:
$M(A) \Longrightarrow M(C) \Longrightarrow strong\_replacement(M, \lambda x\ y.\ y = \langle x, if\ x = Inl(A)\ then\ C\ else\ x\rangle)$
using *lam_replacement_if lam_replacement_constant lam_replacement_identity*
separation_equal
unfolding *lam_replacement_def*
by *simp*

lemma *if_then_range_replacement*:
$M(u) \Longrightarrow$
$M(f) \Longrightarrow$
strong_replacement
$(M,$
$\lambda z\ y.\ y = \langle z, if\ z = u\ then\ f\ `\ 0\ else\ if\ z \in range(f)\ then\ f\ `\ succ(converse(f)\ `\ z)\ else\ z\rangle)$
using *lam_replacement_if separation_equal separation_in*
lam_replacement_constant lam_replacement_identity
lam_replacement_succ lam_replacement_apply
lam_replacement_hcomp[*of* $\lambda x.\ converse(f)`x\ succ$]
lam_replacement_hcomp[*of* $\lambda x.\ succ(converse(f)`x)\ \lambda x\ .\ f`x$]
unfolding *lam_replacement_def*
by *simp*

lemma *Inl_replacement2*:
$M(A) \Longrightarrow$
$strong\_replacement(M, \lambda x\ y.\ y = \langle x, if\ fst(x) = A\ then\ Inl(snd(x))\ else\ Inr(x)\rangle)$
using *lam_replacement_if separation_fst_equal*
lam_replacement_hcomp[*of snd Inl*]
lam_replacement_Inl lam_replacement_Inr lam_replacement_snd
unfolding *lam_replacement_def*
by *simp*

lemma *case_replacement1*:
strong_replacement(M, $\lambda z\ y.\ y = \langle z, case(Inr, Inl, z)\rangle$)
using *lam_replacement_case lam_replacement_Inl lam_replacement_Inr*
unfolding *lam_replacement_def*
by *simp*

lemma *case_replacement2*:
strong_replacement(M, $\lambda z\ y.\ y = \langle z, case(case(Inl, \lambda y.\ Inr(Inl(y))), \lambda y.\ Inr(Inr(y)), z)\rangle$)
using *lam_replacement_case lam_replacement_hcomp*
case_closed[*of Inl* $\lambda x.\ Inr(Inl(x))$]
lam_replacement_Inl lam_replacement_Inr
unfolding *lam_replacement_def*
by *simp*

lemma *case_replacement4*:
$M(f) \Longrightarrow M(g) \Longrightarrow$ *strong_replacement*(M, $\lambda z\ y.\ y = \langle z, case(\lambda w.\ Inl(f\ `\ w), \lambda y.\ Inr(g\ `\ y), z)\rangle$)
using *lam_replacement_case lam_replacement_hcomp*
lam_replacement_Inl lam_replacement_Inr lam_replacement_apply
unfolding *lam_replacement_def*
by *simp*

lemma *case_replacement5*:
strong_replacement(M, $\lambda x\ y.\ y = \langle x, (\lambda\langle x,z\rangle.\ case(\lambda y.\ Inl(\langle y, z\rangle), \lambda y.\ Inr(\langle y, z\rangle), x))(x)\rangle$)
unfolding *split_def case_def cond_def*
using *lam_replacement_if separation_equal_fst2*
lam_replacement_snd lam_replacement_Inl lam_replacement_Inr
lam_replacement_hcomp[*OF*
lam_replacement_product[*OF*
lam_replacement_hcomp[*OF lam_replacement_fst lam_replacement_snd*]]]
unfolding *lam_replacement_def*
by *simp*

end

— To be used in the relativized treatment of Cohen posets
definition
— "domain collect F"
$dC\_F :: i \Rightarrow i \Rightarrow i$ **where**
$dC\_F(A,d) \equiv \{p \in A.\ domain(p) = d\ \}$

definition
— "domain restrict SepReplace Y"
$drSR\_Y :: i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow i$ **where**
$drSR\_Y(B,D,A,x) \equiv \{domain(r)\ ..\ r{\in}A,\ restrict(r,B) = x \wedge domain(r) \in D\}$

lemma *drSR_Y_equality*: $drSR\_Y(B,D,A,x) = \{\ dr{\in}D\ .\ (\exists\, r{\in}A\ .\ restrict(r,B)$

$= x \wedge dr{=}domain(r))$ }
unfolding *drSR_Y_def SepReplace_def* **by** *auto*

context *M_replacement_extra*
begin

lemma *lam_replacement_drSR_Y*:
assumes
$\bigwedge A\ B.\ M(A) \Longrightarrow M(B) \Longrightarrow \forall\, x[M].\ separation(M,\ \lambda dr.\ \exists\, r{\in}A\ .\ restrict(r,B) = x \wedge dr{=}domain(r))$
$\bigwedge A\ B\ D\ .\ M(A) \Longrightarrow M(B) \Longrightarrow M(D) \Longrightarrow$
separation$(M,\ \lambda p.\ \forall\, x{\in}D.\ x \in snd(p) \longleftrightarrow (\exists\, r{\in}A.\ restrict(r,\ B) = fst(p) \wedge x = domain(r)))$
$M(B)\ M(D)\ M(A)$
shows *lam_replacement*$(M,\ drSR\_Y(B,D,A))$
using *lam_replacement_cong lam_replacement_Collect*[*OF* ‹$M(D)$› *assms*(*1*)[*of A B*]]
assms drSR_Y_equality **by** *simp*

lemma *lam_if_then_apply_replacement*: $M(f) \Longrightarrow M(v) \Longrightarrow M(u) \Longrightarrow$
lam_replacement$(M,\ \lambda x.\ if\ f\ `\ x = v\ then\ f\ `\ u\ else\ f\ `\ x)$
using *lam_replacement_if separation_equal_apply lam_replacement_constant lam_replacement_apply*
by *simp*

lemma *lam_if_then_apply_replacement2*: $M(f) \Longrightarrow M(m) \Longrightarrow M(y) \Longrightarrow$
lam_replacement$(M,\ \lambda z\ .\ if\ f\ `\ z = m\ then\ y\ else\ f\ `\ z)$
using *lam_replacement_if separation_equal_apply lam_replacement_constant lam_replacement_apply*
by *simp*

lemma *lam_if_then_replacement2*: $M(A) \Longrightarrow M(f) \Longrightarrow$
lam_replacement$(M,\ \lambda x\ .\ if\ x \in A\ then\ f\ `\ x\ else\ x)$
using *lam_replacement_if separation_in lam_replacement_identity lam_replacement_apply*
by *simp*

lemma *lam_if_then_replacement_apply*: $M(G) \Longrightarrow lam\_replacement(M,\ \lambda x.\ if\ M(x)\ then\ G\ `\ x\ else\ 0)$
using *lam_replacement_if separation_in lam_replacement_identity lam_replacement_apply lam_replacement_constant*[*of 0*] *separation_univ*
by *simp*

lemma *lam_replacement_dC_F*:
assumes $M(A)$
$\bigwedge d\ .\ M(d) \Longrightarrow separation(M,\ \lambda x\ .\ domain(x) = d)$
$\bigwedge A\ .\ M(A) \Longrightarrow separation(M,\ \lambda p.\ \forall\, x{\in}A.\ x \in snd(p) \longleftrightarrow domain(x) = fst(p))$
shows *lam_replacement*$(M,\ dC\_F(A))$
unfolding *dC_F_def*
using *assms lam_replacement_Collect*[*of A* $\lambda\ d\ x\ .\ domain(x) = d$]

by *simp*

lemma *lam_replacement_min*: $M(f) \Longrightarrow M(r) \Longrightarrow$ *lam_replacement*$(M,\ \lambda x\ .$
minimum$(r,\ f\text{-}``\ \{x\}))$
using *lam_replacement_hcomp2*[*OF lam_replacement_constant*[*of r*] *lam_replacement_vimage_sing_fun*]
lam_replacement_minimum
by *simp*

lemma *lam_replacement_Collect_ball_Pair*:
assumes *separation*$(M,\ \lambda p.\ \forall\, x{\in}G.\ x \in snd(p) \longleftrightarrow (\forall\, s{\in}fst(p).\ \langle s,\ x\rangle \in Q))$
$\bigwedge x.\ M(x) \Longrightarrow$ *separation*$(M,\ \lambda y.\ \forall\, s{\in}x.\ \langle s,\ y\rangle \in Q)\ M(G)$
shows *lam_replacement*$(M,\ \lambda x\ .\ \{a \in G\ .\ \forall\, s{\in}x.\ \langle s,\ a\rangle \in Q\})$
using *assms lam_replacement_Collect* **by** *simp*

lemma *surj_imp_inj_replacement3*:
$(\bigwedge x.\ M(x) \Longrightarrow$ *separation*$(M,\ \lambda y.\ \forall\, s{\in}x.\ \langle s,\ y\rangle \in Q)) \Longrightarrow M(G) \Longrightarrow M(Q) \Longrightarrow$
$M(x) \Longrightarrow$
strong_replacement$(M,\ \lambda y\ z.\ y \in \{a \in G\ .\ \forall\, s{\in}x.\ \langle s,\ a\rangle \in Q\} \wedge z = \{\langle x,\ y\rangle\})$
using *lam_replacement_imp_strong_replacement*
using *lam_replacement_sing_const_id*[*THEN lam_replacement_imp_strong_replacement*,
of x]
unfolding *strong_replacement_def*
by (*simp*, *safe*, *drule_tac x*=$A \cap \{a \in G\ .\ \forall\, s{\in}x.\ \langle s,\ a\rangle \in Q\}$ **in** *rspec*,
simp, *erule_tac rexE*, *rule_tac x*=Y **in** *rexI*) *auto*

lemmas *replacements* = *Pair_diff_replacement id_replacement tag_replacement*
pospend_replacement prepend_replacement
Inl_replacement1 diff_Pair_replacement
swap_replacement tag_union_replacement csquare_lam_replacement
assoc_replacement prod_fun_replacement
cardinal_lib_assms4 domain_replacement
apply_replacement
un_Pair_replacement restrict_strong_replacement diff_replacement
if_then_Inj_replacement lam_if_then_replacement if_then_replacement
ifx_replacement if_then_range_replacement2 if_then_range_replacement
Inl_replacement2
case_replacement1 case_replacement2 case_replacement4 case_replacement5

end

end

21 Relative, Choice-less Cardinal Numbers

theory *Cardinal_Relative*
imports
ZF_Miscellanea
Discipline_Cardinal
Lambda_Replacement

begin

hide_const (**open**) L

definition
$Finite\_rel$ :: $[i \Rightarrow o, i] => o$ **where**
$Finite\_rel(M,A) \equiv \exists\, om[M].\ \exists\, n[M].\ omega(M,om) \wedge n \in om \wedge eqpoll\_rel(M,A,n)$

definition
$banach\_functor$ :: $[i,i,i,i,i] \Rightarrow i$ **where**
$banach\_functor(X,Y,f,g,W) \equiv X$ - g‘‘$(Y$ - f‘‘$W)$

definition
$is\_banach\_functor$ :: $[i \Rightarrow o, i,i,i,i,i,i] \Rightarrow o$ **where**
$is\_banach\_functor(M,X,Y,f,g,W,b) \equiv$
$\exists\, fW[M].\ \exists\, YfW[M].\ \exists\, gYfW[M].\ image(M,f,W,fW) \wedge setdiff(M,Y,fW,YfW)$
$\wedge$
$image(M,g,YfW,gYfW) \wedge setdiff(M,X,gYfW,b)$

lemma (**in** M_basic) $banach\_functor\_abs$:
assumes $M(X)$ $M(Y)$ $M(f)$ $M(g)$
shows $relation1(M, is\_banach\_functor(M,X,Y,f,g), banach\_functor(X,Y,f,g))$
unfolding $relation1\_def$ $is\_banach\_functor\_def$ $banach\_functor\_def$
using $assms$
by $simp$

lemma (**in** M_basic) $banach\_functor\_closed$:
assumes $M(X)$ $M(Y)$ $M(f)$ $M(g)$
shows $\forall\, W[M].\ M(banach\_functor(X,Y,f,g,W))$
unfolding $banach\_functor\_def$ **using** $assms$ $image\_closed$
by $simp$

locale $M\_cardinals = M\_ordertype + M\_trancl + M\_Perm + M\_replacement\_extra$
$+$
assumes
$rvimage\_separation$: $M(f) \Longrightarrow M(r) \Longrightarrow$
$separation(M,\ \lambda z.\ \exists\, x\ y.\ z = \langle x,\ y\rangle \wedge \langle f\ `\ x,\ f\ `\ y\rangle \in r)$
and
$radd\_separation$: $M(R) \Longrightarrow M(S) \Longrightarrow$
$separation(M,\ \lambda z.$
$(\exists\, x\ y.\ z = \langle Inl(x),\ Inr(y)\rangle) \vee$
$(\exists\, x'\ x.\ z = \langle Inl(x'),\ Inl(x)\rangle \wedge \langle x',\ x\rangle \in R) \vee$
$(\exists\, y'\ y.\ z = \langle Inr(y'),\ Inr(y)\rangle \wedge \langle y',\ y\rangle \in S))$
and
$rmult\_separation$: $M(b) \Longrightarrow M(d) \Longrightarrow separation(M,$
$\lambda z.\ \exists\, x'\ y'\ x\ y.\ z = \langle\langle x',\ y'\rangle,\ x,\ y\rangle \wedge (\langle x',\ x\rangle \in b \vee x' = x \wedge \langle y',\ y\rangle \in d))$
and
$banach\_repl\_iter$: $M(X) \Longrightarrow M(Y) \Longrightarrow M(f) \Longrightarrow M(g) \Longrightarrow$

strong_replacement(M, $\lambda x\ y.\ x{\in}nat \wedge y = banach\_functor(X,\ Y,\ f,\ g)\hat{}x\ (0)$)
begin

lemma *radd_closed*[*intro*,*simp*]: $M(a) \Longrightarrow M(b) \Longrightarrow M(c) \Longrightarrow M(d) \Longrightarrow M(radd(a,b,c,d))$
  using *radd_separation* **by** (*auto simp add*: *radd_def*)

lemma *rmult_closed*[*intro*,*simp*]: $M(a) \Longrightarrow M(b) \Longrightarrow M(c) \Longrightarrow M(d) \Longrightarrow M(rmult(a,b,c,d))$
  using *rmult_separation* **by** (*auto simp add*: *rmult_def*)

end

lemma (**in** *M_cardinals*) *is_cardinal_iff_Least*:
  assumes $M(A)$ $M(\kappa)$
  shows $is\_cardinal(M,A,\kappa) \longleftrightarrow \kappa = (\mu\ i.\ M(i) \wedge i \approx^M A)$
  using *is_cardinal_iff assms*
  unfolding *cardinal_rel_def* **by** *simp*

21.1 The Schroeder-Bernstein Theorem

See Davey and Priestly, page 106

context *M_cardinals*
begin

lemma *bnd_mono_banach_functor*: $bnd\_mono(X,\ banach\_functor(X,Y,f,g))$
  unfolding *bnd_mono_def banach_functor_def*
  by *blast*

lemma *inj_Inter*:
  assumes $g \in inj(Y,X)$ $A{\neq}0$ $\forall\, a{\in}A.\ a \subseteq Y$
  shows $g``(\bigcap A) = (\bigcap a{\in}A.\ g``a)$
proof (*intro equalityI subsetI*)
  fix x
  from *assms*
  obtain a **where** $a{\in}A$ **by** *blast*
  moreover
  assume $x \in (\bigcap a{\in}A.\ g\ ``\ a)$
  ultimately
  have *x_in_im*: $x \in g``y$ **if** $y{\in}A$ **for** y
    using *that* **by** *auto*
  have *exists*: $\exists\, z \in y.\ x = g`z$ **if** $y{\in}A$ **for** y
  proof -
    note *that*
    moreover from *this* **and** *x_in_im*
    have $x \in g``y$ **by** *simp*
    moreover from *calculation*
    have $x \in g``y$ **by** *simp*

```
    moreover
    note assms
    ultimately
    show ?thesis
      using image_fun[OF inj_is_fun] by auto
  qed
  with ‹a∈A›
  obtain z where z ∈ a x = g‘z by auto
  moreover
  have z ∈ y if y∈A for y
  proof -
    from that and exists
    obtain w where w ∈ y x = g‘w by auto
    moreover from this ‹x = g‘z› assms that ‹a∈A› ‹z∈a›
    have z = w unfolding inj_def by blast
    ultimately
    show ?thesis by simp
  qed
  moreover
  note assms
  moreover from calculation
  have z ∈ ⋂A by auto
  moreover from calculation
  have z ∈ Y by blast
  ultimately
  show x ∈ g ‘‘ (⋂A)
    using inj_is_fun[THEN funcI, of g] by fast
qed auto

lemma contin_banach_functor:
  assumes g ∈ inj(Y,X)
  shows contin(banach_functor(X,Y,f,g))
  unfolding contin_def
proof (intro allI impI)
  fix A
  assume directed(A)
  then
  have A ≠ 0
    unfolding directed_def ..
  have banach_functor(X, Y, f, g, ⋃A) = X - g‘‘(Y - f‘‘(⋃A))
    unfolding banach_functor_def ..
  also
  have ... = X - g‘‘(Y - (⋃a∈A. f‘‘a))
    by auto
  also from ‹A≠0›
  have ... = X - g‘‘(⋂a∈A. Y-f‘‘a)
    by auto
  also from ‹A≠0› and assms
  have ... = X - (⋂a∈A. g‘‘(Y-f‘‘a))
```

```
    using inj_Inter[of g Y X {Y-f``a. a∈A} ] by fastforce
  also from ‹A≠0›
  have  ... = (⋃a∈A. X - g``(Y-f``a)) by simp
  also
  have  ... = (⋃a∈A. banach_functor(X, Y, f, g, a))
    unfolding banach_functor_def ..
  finally
  show banach_functor(X,Y,f,g,⋃A) = (⋃a∈A. banach_functor(X,Y,f,g,a)) .
qed

lemma lfp_banach_functor:
  assumes g∈inj(Y,X)
  shows lfp(X, banach_functor(X,Y,f,g)) =
       (⋃n∈nat. banach_functor(X,Y,f,g)^n (0))
  using assms lfp_eq_Union bnd_mono_banach_functor contin_banach_functor
  by simp

lemma lfp_banach_functor_closed:
  assumes M(g) M(X) M(Y) M(f) g∈inj(Y,X)
  shows M(lfp(X, banach_functor(X,Y,f,g)))
proof -
  from assms
  have M(banach_functor(X,Y,f,g)^n (0)) if n∈nat for n
    by(rule_tac nat_induct[OF that],simp_all add:banach_functor_closed)
  with assms
  show ?thesis
    using family_union_closed'[OF banach_repl_iter M_nat] lfp_banach_functor
    by simp
qed

lemma banach_decomposition_rel:
  [| M(f); M(g); M(X); M(Y); f ∈ X->Y;  g ∈ inj(Y,X) |] ==>
      ∃XA[M]. ∃XB[M]. ∃YA[M]. ∃YB[M].
         (XA ∩ XB = 0) & (XA ∪ XB = X) &
         (YA ∩ YB = 0) & (YA ∪ YB = Y) &
         f``XA=YA & g``YB=XB
  apply (intro rexI conjI)
          apply (rule_tac [6] Banach_last_equation)
          apply (rule_tac [5] refl)
         apply (assumption |
      rule inj_is_fun Diff_disjoint Diff_partition fun_is_rel
      image_subset lfp_subset)+
  using lfp_banach_functor_closed[of g X Y f]
  unfolding banach_functor_def by simp_all

lemma schroeder_bernstein_closed:
  [| M(f); M(g); M(X); M(Y); f ∈ inj(X,Y);  g ∈ inj(Y,X) |] ==> ∃h[M]. h ∈
bij(X,Y)
  apply (insert banach_decomposition_rel [of f g X Y])
```

apply (*simp add*: *inj_is_fun*)
apply (*auto*)
apply (*rule_tac x*=*restrict*(*f*,*XA*) ∪ *converse*(*restrict*(*g*,*YB*)) **in** *rexI*)
apply (*auto intro*!: *restrict_bij bij_disjoint_Un intro*: *bij_converse_bij*)
done

lemma *mem_Pow_rel*: $M(r) \Longrightarrow a \in Pow\_rel(M,r) \Longrightarrow a \in Pow(r) \wedge M(a)$
using *Pow_rel_char* **by** *simp*

lemma *mem_bij_abs*[*simp*]: $[\![M(f);M(A);M(B)]\!] \Longrightarrow f \in bij^M(A,B) \longleftrightarrow f \in bij(A,B)$
using *bij_rel_char* **by** *simp*

lemma *mem_inj_abs*[*simp*]: $[\![M(f);M(A);M(B)]\!] \Longrightarrow f \in inj^M(A,B) \longleftrightarrow f \in inj(A,B)$
using *inj_rel_char* **by** *simp*

lemma *mem_surj_abs*: $[\![M(f);M(A);M(B)]\!] \Longrightarrow f \in surj^M(A,B) \longleftrightarrow f \in surj(A,B)$
using *surj_rel_char* **by** *simp*

lemma *bij_imp_eqpoll_rel*:
assumes $f \in bij(A,B)$ $M(f)$ $M(A)$ $M(B)$
shows $A \approx^M B$
using *assms* **by** (*auto simp add*:*def_eqpoll_rel*)

lemma *id_closed*: $M(A) \Longrightarrow M(id(A))$
using *lam_replacement_identity lam_replacement_iff_lam_closed*
unfolding *id_def* **by** *simp*

lemma *eqpoll_rel_refl*: $M(A) \Longrightarrow A \approx^M A$
using *bij_imp_eqpoll_rel*[*OF id_bij*, *OF id_closed*] **.**

lemma *eqpoll_rel_sym*: $X \approx^M Y \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow Y \approx^M X$
unfolding *def_eqpoll_rel* **using** *converse_closed*
by (*auto intro*: *bij_converse_bij*)

lemma *eqpoll_rel_trans* [*trans*]:
$[|X \approx^M Y;\ Y \approx^M Z;\ M(X);\ M(Y)\ ;\ M(Z)\ |] ==> X \approx^M Z$
unfolding *def_eqpoll_rel* **by** (*auto intro*: *comp_bij*)

lemma *subset_imp_lepoll_rel*: $X \subseteq Y \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow X \lesssim^M Y$
unfolding *def_lepoll_rel* **using** *id_subset_inj id_closed*
by *simp blast*

lemmas *lepoll_rel_refl* = *subset_refl* [*THEN subset_imp_lepoll_rel*, *simp*]

lemmas *le_imp_lepoll_rel* = *le_imp_subset* [*THEN subset_imp_lepoll_rel*]

lemma *eqpoll_rel_imp_lepoll_rel*: $X \approx^M Y \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow X \lesssim^M Y$
unfolding *def_eqpoll_rel bij_def def_lepoll_rel* **using** *bij_is_inj*
by (*auto*)

lemma *lepoll_rel_trans* [*trans*]:
assumes
$X \lesssim^M Y$ $Y \lesssim^M Z$ $M(X)$ $M(Y)$ $M(Z)$
shows
$X \lesssim^M Z$
using *assms def_lepoll_rel*
by (*auto intro*: *comp_inj*)

lemma *eq_lepoll_rel_trans* [*trans*]:
assumes
$X \approx^M Y$ $Y \lesssim^M Z$ $M(X)$ $M(Y)$ $M(Z)$
shows
$X \lesssim^M Z$
using *assms*
by (*blast intro*: *eqpoll_rel_imp_lepoll_rel lepoll_rel_trans*)

lemma *lepoll_rel_eq_trans* [*trans*]:
assumes $X \lesssim^M Y$ $Y \approx^M Z$ $M(X)$ $M(Y)$ $M(Z)$
shows $X \lesssim^M Z$
using *assms*
eqpoll_rel_imp_lepoll_rel[*of Y Z*] *lepoll_rel_trans*[*of X Y Z*]
by *simp*

lemma *eqpoll_relI*: $[\![X \lesssim^M Y;\ Y \lesssim^M X;\ M(X)\ ;\ M(Y)]\!] \Longrightarrow X \approx^M Y$
unfolding *def_lepoll_rel def_eqpoll_rel* **using** *schroeder_bernstein_closed*
by *auto*

lemma *eqpoll_relE*:
$[|\ X \approx^M Y;\ [|\ X \lesssim^M Y;\ Y \lesssim^M X\ |] \Longrightarrow P\ ;\ \ M(X)\ ;\ M(Y)\ |] \Longrightarrow P$
by (*blast intro*: *eqpoll_rel_imp_lepoll_rel eqpoll_rel_sym*)

lemma *eqpoll_rel_iff*: $M(X) \Longrightarrow M(Y) \Longrightarrow X \approx^M Y \longleftrightarrow X \lesssim^M Y\ \&\ Y \lesssim^M X$
by (*blast intro*: *eqpoll_relI elim*: *eqpoll_relE*)

lemma *lepoll_rel_0_is_0*: $A \lesssim^M 0 \Longrightarrow M(A) \Longrightarrow A = 0$
using *def_lepoll_rel*
by (*cases A=0*) (*auto simp add*: *inj_def*)

lemmas *empty_lepoll_relI* = *empty_subsetI* [*THEN subset_imp_lepoll_rel*, *OF nonempty*]

lemma *lepoll_rel_0_iff*: $M(A) \Longrightarrow A \lesssim^M 0 \longleftrightarrow A=0$

by (*blast intro*: *lepoll_rel_0_is_0 lepoll_rel_refl*)

lemma *Un_lepoll_rel_Un*:
[| $A \lesssim^M B$; $C \lesssim^M D$; $B \cap D = 0$; $M(A)$; $M(B)$; $M(C)$; $M(D)$ |] ==> $A \cup C \lesssim^M B \cup D$
using *def_lepoll_rel* **using** *inj_disjoint_Un*[*of_ A B_ C D*] *if_then_replacement*
apply (*auto*)
apply (*rule*, *assumption*)
apply (*auto intro*!:*lam_closed elim*:*transM*)+
done

lemma *eqpoll_rel_0_is_0*: $A \approx^M 0 \Longrightarrow M(A) \Longrightarrow A = 0$
using *eqpoll_rel_imp_lepoll_rel lepoll_rel_0_is_0 nonempty*
by *blast*

lemma *eqpoll_rel_0_iff*: $M(A) \Longrightarrow A \approx^M 0 \longleftrightarrow A=0$
by (*blast intro*: *eqpoll_rel_0_is_0 eqpoll_rel_refl*)

lemma *eqpoll_rel_disjoint_Un*:
[| $A \approx^M B$; $C \approx^M D$; $A \cap C = 0$; $B \cap D = 0$; $M(A)$; $M(B)$; $M(C)$; $M(D)$ |]
==> $A \cup C \approx^M B \cup D$
by (*auto intro*: *bij_disjoint_Un simp add*:*def_eqpoll_rel*)

21.2 lesspoll_rel: contributions by Krzysztof Grabczewski

lemma *lesspoll_rel_not_refl*: $M(i) \Longrightarrow \sim (i \prec^M i)$
by (*simp add*: *lesspoll_rel_def eqpoll_rel_refl*)

lemma *lesspoll_rel_irrefl*: $i \prec^M i$ ==> $M(i) \Longrightarrow P$
by (*simp add*: *lesspoll_rel_def eqpoll_rel_refl*)

lemma *lesspoll_rel_imp_lepoll_rel*: $[\![A \prec^M B; M(A); M(B)]\!] \Longrightarrow A \lesssim^M B$
by (*unfold lesspoll_rel_def*, *blast*)

lemma *rvimage_closed* [*intro*,*simp*]:
assumes
$M(A)$ $M(f)$ $M(r)$
shows
$M(rvimage(A,f,r))$
unfolding *rvimage_def* **using** *assms rvimage_separation* **by** *auto*

lemma *lepoll_rel_well_ord*: [| $A \lesssim^M B$; *well_ord*(B,r); $M(A)$; $M(B)$; $M(r)$ |]
==> $\exists s[M].$ *well_ord*(A,s)
unfolding *def_lepoll_rel* **by** (*auto intro*:*well_ord_rvimage*)

lemma *lepoll_rel_iff_leqpoll_rel*: $[\![M(A); M(B)]\!] \Longrightarrow A \lesssim^M B \longleftrightarrow A \prec^M B \mid A \approx^M B$
apply (*unfold lesspoll_rel_def*)
apply (*blast intro*: *eqpoll_relI elim*: *eqpoll_relE*)

done

end

context *M_cardinals*
begin

lemma *inj_rel_is_fun_M*: $f \in inj^M(A,B) \Longrightarrow M(f) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow f \in A \rightarrow^M B$
using *inj_is_fun function_space_rel_char* **by** *simp*

— In porting the following theorem, I tried to follow the Discipline strictly, though finally only an approach maximizing the use of absoluteness results ($[\![M(?A); M(?y)]\!] \Longrightarrow ?A \rightarrow^M ?y = \{f \in ?A \rightarrow ?y \,.\, M(f)\}$
$[\![M(?A); M(?B)]\!] \Longrightarrow inj^M(?A,?B) = \{f \in inj(?A, ?B) \,.\, M(f)\}$) was the one paying dividends.
lemma *inj_rel_not_surj_rel_succ*:
notes *mem_inj_abs*[*simp del*]
assumes *fi*: $f \in inj^M(A, succ(m))$ **and** *fns*: $f \notin surj^M(A, succ(m))$
and *types*: $M(f)\ M(A)\ M(m)$
shows $\exists f[M].\ f \in inj^M(A,m)$
proof -
from *fi* [*THEN inj_rel_is_fun_M*] *fns types*
obtain *y* **where** *y*: $y \in succ(m)\ \bigwedge x.\ x{\in}A \Longrightarrow f\ `\ x \neq y\ M(y)$
by (*auto simp add*: *def_surj_rel*)
show *?thesis*
proof
from *types* **and** ‹$M(y)$›
show $M(\lambda z{\in}A.\ if\ f\ `\ z = m\ then\ y\ else\ f\ `\ z)$
using *transM*[*OF* _ ‹$M(A)$›] *lam_if_then_apply_replacement2 lam_replacement_iff_lam_closed*
by (*auto*)
with *types y fi*
have $(\lambda z{\in}A.\ if\ f`z = m\ then\ y\ else\ f`z) \in A{\rightarrow}^M\ m$
using *function_space_rel_char inj_rel_char inj_is_fun*[*of f A succ*(*m*)]
by (*auto intro*!: *if_type* [*THEN lam_type*] *dest*: *apply_funtype*)
with *types y fi*
show $(\lambda z{\in}A.\ if\ f`z = m\ then\ y\ else\ f`z) \in inj^M(A,\ m)$
by (*simp add*: *def_inj_rel*) *blast*
qed
qed

lemma *lesspoll_rel_trans* [*trans*]:
$[\!|\ X \prec^M Y;\ Y \prec^M Z;\ M(X);\ M(Y)\ ;\ M(Z)\ |\!] ==> X \prec^M Z$
apply (*unfold lesspoll_rel_def*)
apply (*blast elim*: *eqpoll_relE intro*: *eqpoll_relI lepoll_rel_trans*)
done

lemma *lesspoll_rel_trans1* [*trans*]:
[| $X \lesssim^M Y$; $Y \prec^M Z$; $M(X)$; $M(Y)$; $M(Z)$ |] ==> $X \prec^M Z$
apply (*unfold lesspoll_rel_def*)
apply (*blast elim*: *eqpoll_relE intro*: *eqpoll_relI lepoll_rel_trans*)
done

lemma *lesspoll_rel_trans2* [*trans*]:
[| $X \prec^M Y$; $Y \lesssim^M Z$; $M(X)$; $M(Y)$; $M(Z)$|] ==> $X \prec^M Z$
apply (*unfold lesspoll_rel_def*)
apply (*blast elim*: *eqpoll_relE intro*: *eqpoll_relI lepoll_rel_trans*)
done

lemma *eq_lesspoll_rel_trans* [*trans*]:
[| $X \approx^M Y$; $Y \prec^M Z$; $M(X)$; $M(Y)$; $M(Z)$ |] ==> $X \prec^M Z$
by (*blast intro*: *eqpoll_rel_imp_lepoll_rel lesspoll_rel_trans1*)

lemma *lesspoll_rel_eq_trans* [*trans*]:
[| $X \prec^M Y$; $Y \approx^M Z$; $M(X)$; $M(Y)$; $M(Z)$ |] ==> $X \prec^M Z$
by (*blast intro*: *eqpoll_rel_imp_lepoll_rel lesspoll_rel_trans2*)

lemma *is_cardinal_cong*:
assumes $X \approx^M Y$ $M(X)$ $M(Y)$
shows $\exists \kappa[M].$ *is_cardinal*(M,X,κ) $\wedge$ *is_cardinal*(M,Y,κ)
proof -
from *assms*
have $(\mu\ i.\ M(i) \wedge i \approx^M X) = (\mu\ i.\ M(i) \wedge i \approx^M Y)$
by (*intro Least_cong*) (*auto intro*: *comp_bij bij_converse_bij simp add*:*def_eqpoll_rel*)
moreover from *assms*
have $M(\mu\ i.\ M(i) \wedge i \approx^M X)$
using *Least_closed'* **by** *fastforce*
moreover
note *assms*
ultimately
show *?thesis*
using *is_cardinal_iff_Least*
by *auto*
qed

— ported from Cardinal
lemma *cardinal_rel_cong*: $X \approx^M Y \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow |X|^M = |Y|^M$
apply (*simp add*: *def_eqpoll_rel cardinal_rel_def*)
apply (*rule Least_cong*)
apply (*auto intro*: *comp_bij bij_converse_bij*)
done

lemma *well_ord_is_cardinal_eqpoll_rel*:
assumes *well_ord*(A,r) **shows** *is_cardinal*(M,A,κ) $\Longrightarrow$ $M(A) \Longrightarrow M(\kappa) \Longrightarrow$
$M(r) \Longrightarrow \kappa \approx^M A$
proof (*subst is_cardinal_iff_Least*[*THEN iffD1*, *of A* κ])

assume $M(A)$ $M(\kappa)$ $M(r)$ *is_cardinal*(M,A,κ)
moreover from *assms* **and** *calculation*
obtain f i **where** $M(f)$ *Ord*(i) $M(i)$ $f \in$ *bij*(A,i)
using *ordertype_exists*[*of* A r] *ord_iso_is_bij* **by** *auto*
moreover
have $M(\mu\ i.\ M(i) \wedge i \approx^M A)$
using *Least_closed'* **by** *fastforce*
ultimately
show $(\mu\ i.\ M(i) \wedge i \approx^M A) \approx^M A$
using *assms*[*THEN well_ord_imp_relativized*]
LeastI[*of* $\lambda i.\ M(i) \wedge i \approx^M A\ i$] *Ord_ordertype*[*OF assms*]
bij_converse_bij[*THEN bij_imp_eqpoll_rel*, *of* f] **by** *simp*
qed

lemmas *Ord_is_cardinal_eqpoll_rel* = *well_ord_Memrel*[*THEN well_ord_is_cardinal_eqpoll_rel*]

22 Porting from *ZF.Cardinal*

The following results were ported more or less directly from *ZF.Cardinal*

— This result relies on various closure properties and thus cannot be translated directly
lemma *well_ord_cardinal_rel_eqpoll_rel*:
assumes r: *well_ord*(A,r) **and** $M(A)$ $M(r)$ **shows** $|A|^M \approx^M A$
using *assms well_ord_is_cardinal_eqpoll_rel is_cardinal_iff*
by *blast*

lemmas *Ord_cardinal_rel_eqpoll_rel* = *well_ord_Memrel*[*THEN well_ord_cardinal_rel_eqpoll_rel*]

lemma *Ord_cardinal_rel_idem*: *Ord*(A) $\Longrightarrow$ $M(A) \Longrightarrow$ $||A|^M|^M = |A|^M$
by (*rule_tac Ord_cardinal_rel_eqpoll_rel* [*THEN cardinal_rel_cong*]) *auto*

lemma *well_ord_cardinal_rel_eqE*:
assumes *woX*: *well_ord*(X,r) **and** *woY*: *well_ord*(Y,s) **and** *eq*: $|X|^M = |Y|^M$
and *types*: $M(X)$ $M(r)$ $M(Y)$ $M(s)$
shows $X \approx^M Y$
proof -
from *types*
have $X \approx^M |X|^M$ **by** (*blast intro*: *well_ord_cardinal_rel_eqpoll_rel* [*OF woX*] *eqpoll_rel_sym*)
also
have ... = $|Y|^M$ **by** (*rule eq*)
also from *types*
have ... $\approx^M Y$ **by** (*rule_tac well_ord_cardinal_rel_eqpoll_rel* [*OF woY*])
finally show *?thesis* **by** (*simp add*:*types*)
qed

lemma *well_ord_cardinal_rel_eqpoll_rel_iff*:

[| *well_ord*(*X*,*r*); *well_ord*(*Y*,*s*); *M*(*X*); *M*(*r*); *M*(*Y*); *M*(*s*) |] ==> $|X|^M = |Y|^M \longleftrightarrow X \approx^M Y$
by (*blast intro*: *cardinal_rel_cong well_ord_cardinal_rel_eqE*)

lemma *Ord_cardinal_rel_le*: *Ord*(*i*) $\Longrightarrow$ *M*(*i*) ==> $|i|^M \leq i$
unfolding *cardinal_rel_def*
using *eqpoll_rel_refl Least_le* **by** *simp*

lemma *Card_rel_cardinal_rel_eq*: $Card^M(K)$ ==> *M*(*K*) $\Longrightarrow$ $|K|^M = K$
apply (*unfold Card_rel_def*)
apply (*erule sym*)
done

lemma *Card_relI*: [| *Ord*(*i*); !!*j*. *j*<*i* $\Longrightarrow$ *M*(*j*) ==> $\sim(j \approx^M i)$; *M*(*i*) |] ==> $Card^M(i)$
apply (*unfold Card_rel_def cardinal_rel_def*)
apply (*subst Least_equality*)
apply (*blast intro*: *eqpoll_rel_refl*)+
done

lemma *Card_rel_is_Ord*: $Card^M(i)$ ==> *M*(*i*) $\Longrightarrow$ *Ord*(*i*)
apply (*unfold Card_rel_def cardinal_rel_def*)
apply (*erule ssubst*)
apply (*rule Ord_Least*)
done

lemma *Card_rel_cardinal_rel_le*: $Card^M(K)$ ==> *M*(*K*) $\Longrightarrow$ $K \leq |K|^M$
apply (*simp* (*no_asm_simp*) *add*: *Card_rel_is_Ord Card_rel_cardinal_rel_eq*)
done

lemma *Ord_cardinal_rel* [*simp*,*intro*!]: *M*(*A*) $\Longrightarrow$ *Ord*($|A|^M$)
apply (*unfold cardinal_rel_def*)
apply (*rule Ord_Least*)
done

lemma *Card_rel_iff_initial*: **assumes** *types*:*M*(*K*)
shows $Card^M(K) \longleftrightarrow$ *Ord*(*K*) & ($\forall j[M].\ j<K \longrightarrow \sim (j \approx^M K)$)
proof -
{ fix *j*
assume *K*: $Card^M(K)$ *M*(*j*) $\wedge$ $j \approx^M K$
assume $j < K$
also have ... = (μ *i*. *M*(*i*) $\wedge$ $i \approx^M K$) **using** *K*
by (*simp add*: *Card_rel_def cardinal_rel_def types*)
finally have $j < (\mu\ i.\ M(i) \wedge i \approx^M K)$ **.**
then have *False* **using** *K*
by (*best intro*: *less_LeastE*[*of* $\lambda j.\ M(j) \wedge j \approx^M K$])
}
with *types*
show *?thesis*

```
    by (blast intro: Card_relI Card_rel_is_Ord)
qed

lemma lt_Card_rel_imp_lesspoll_rel: [| Card^M(a); i<a; M(a); M(i) |] ==> i
≺^M a
  apply (unfold lesspoll_rel_def)
  apply (frule Card_rel_iff_initial [THEN iffD1], assumption)
  apply (blast intro!: leI [THEN le_imp_lepoll_rel])
  done

lemma Card_rel_0: Card^M(0)
  apply (rule Ord_0 [THEN Card_relI])
   apply (auto elim!: ltE)
  done

lemma Card_rel_Un: [| Card^M(K);  Card^M(L); M(K); M(L) |] ==> Card^M(K
∪ L)
  apply (rule Ord_linear_le [of K L])
   apply (simp_all add: subset_Un_iff [THEN iffD1]  Card_rel_is_Ord le_imp_subset
     subset_Un_iff2 [THEN iffD1])
  done

lemma Card_rel_cardinal_rel [iff]: assumes types:M(A) shows Card^M(|A|^M)
  using assms
proof (unfold cardinal_rel_def)
  show Card^M(μ i. M(i) ∧ i ≈^M A)
  proof (cases ∃ i[M]. Ord (i) ∧ i ≈^M A)
    case False thus ?thesis            — degenerate case
      using Least_0[of λi. M(i) ∧ i ≈^M A] Card_rel_0
      by fastforce
  next
    case True                          — real case: A is isomorphic to some ordinal
    then obtain i where i: Ord(i) i ≈^M A M(i) by blast
    show ?thesis
    proof (rule Card_relI [OF Ord_Least], rule notI)
      fix j
      assume j: j < (μ i. M(i) ∧ i ≈^M A) and M(j)
      assume j ≈^M (μ i. M(i) ∧ i ≈^M A)
      also have ... ≈^M A using i LeastI[of λi. M(i) ∧ i ≈^M A] by (auto)
      finally have j ≈^M A
        using Least_closed'[of λi. M(i) ∧ i ≈^M A] by (simp add: ‹M(j)› types)
      thus False
        using ‹M(j)› by (blast intro:less_LeastE [OF _ j])
    qed (auto intro:Least_closed)
  qed
qed

lemma cardinal_rel_eq_lemma:
  assumes i:|i|^M ≤ j and j: j ≤ i and types: M(i) M(j)
```

shows $|j|^M = |i|^M$
proof (*rule eqpoll_relI* [*THEN cardinal_rel_cong*])
show $j \lesssim^M i$ **by** (*rule le_imp_lepoll_rel* [*OF j*]) (*simp_all add*:*types*)
next
have *Oi*: *Ord*(*i*) **using** *j* **by** (*rule le_Ord2*)
with *types*
have $i \approx^M |i|^M$
by (*blast intro*: *Ord_cardinal_rel_eqpoll_rel eqpoll_rel_sym*)
also from *types*
have $... \lesssim^M j$
by (*blast intro*: *le_imp_lepoll_rel i*)
finally show $i \lesssim^M j$ **by** (*simp_all add*:*types*)
qed (*simp_all add*:*types*)

lemma *cardinal_rel_mono*:
assumes *ij*: $i \le j$ **and** *types*:*M*(*i*) *M*(*j*) **shows** $|i|^M \le |j|^M$
using *Ord_cardinal_rel* [*OF* ‹*M*(*i*)›] *Ord_cardinal_rel* [*OF* ‹*M*(*j*)›]
proof (*cases rule*: *Ord_linear_le*)
case *le* **then show** *?thesis* **.**
next
case *ge*
have *i*: *Ord*(*i*) **using** *ij*
by (*simp add*: *lt_Ord*)
have *ci*: $|i|^M \le j$
by (*blast intro*: *Ord_cardinal_rel_le ij le_trans i types*)
have $|i|^M = ||i|^M|^M$
by (*auto simp add*: *Ord_cardinal_rel_idem i types*)
also have $... = |j|^M$
by (*rule cardinal_rel_eq_lemma* [*OF ge ci*]) (*simp_all add*:*types*)
finally have $|i|^M = |j|^M$ **.**
thus *?thesis* **by** (*simp add*:*types*)
qed

lemma *cardinal_rel_lt_imp_lt*: [| $|i|^M < |j|^M$; *Ord*(*i*); *Ord*(*j*); *M*(*i*); *M*(*j*) |]
==> $i < j$
apply (*rule Ord_linear2* [*of i j*], *assumption*+)
apply (*erule lt_trans2* [*THEN lt_irrefl*])
apply (*erule cardinal_rel_mono*, *assumption*+)
done

lemma *Card_rel_lt_imp_lt*: [| $|i|^M < K$; *Ord*(*i*); $Card^M(K)$; *M*(*i*); *M*(*K*)|]
==> $i < K$
by (*simp* (*no_asm_simp*) *add*: *cardinal_rel_lt_imp_lt Card_rel_is_Ord Card_rel_cardinal_rel_eq*)

lemma *Card_rel_lt_iff*: [| *Ord*(*i*); $Card^M(K)$; *M*(*i*); *M*(*K*) |] ==> ($|i|^M < K$)
$\longleftrightarrow$ ($i < K$)
by (*blast intro*: *Card_rel_lt_imp_lt Ord_cardinal_rel_le* [*THEN lt_trans1*])

lemma *Card_rel_le_iff*: [| *Ord*(*i*); $Card^M(K)$; *M*(*i*); *M*(*K*) |] ==> ($K \le |i|^M$)

$\longleftrightarrow$ $(K \leq i)$
by (*simp add*: *Card_rel_lt_iff Card_rel_is_Ord not_lt_iff_le* [*THEN iff_sym*])

lemma *well_ord_lepoll_rel_imp_cardinal_rel_le*:
assumes *wB*: *well_ord*(*B*,*r*) **and** *AB*: $A \lesssim^M B$
and
types: $M(B)$ $M(r)$ $M(A)$
shows $|A|^M \leq |B|^M$
using *Ord_cardinal_rel* [*OF* ‹$M(A)$›] *Ord_cardinal_rel* [*OF* ‹$M(B)$›]
proof (*cases rule*: *Ord_linear_le*)
case *le* **thus** *?thesis* **.**
next
case *ge*
from *lepoll_rel_well_ord* [*OF AB wB*]
obtain *s* **where** *s*: *well_ord*(*A*, *s*) $M(s)$ **by** (*blast intro*:*types*)
have $B \approx^M |B|^M$ **by** (*blast intro*: *wB eqpoll_rel_sym well_ord_cardinal_rel_eqpoll_rel types*)
also have ... $\lesssim^M |A|^M$ **by** (*rule le_imp_lepoll_rel* [*OF ge*]) (*simp_all add*:*types*)
also have ... $\approx^M A$ **by** (*rule well_ord_cardinal_rel_eqpoll_rel* [*OF s*(*1*) _ *s*(*2*)]) (*simp_all add*:*types*)
finally have $B \lesssim^M A$ **by** (*simp_all add*:*types*)
hence $A \approx^M B$ **by** (*blast intro*: *eqpoll_relI AB types*)
hence $|A|^M = |B|^M$ **by** (*rule cardinal_rel_cong*) (*simp_all add*:*types*)
thus *?thesis* **by** (*simp_all add*:*types*)
qed

lemma *lepoll_rel_cardinal_rel_le*: [| $A \lesssim^M i$; *Ord*(*i*); $M(A)$; $M(i)$ |] ==> $|A|^M \leq i$
using *Memrel_closed*
apply (*rule_tac le_trans*)
apply (*erule well_ord_Memrel* [*THEN well_ord_lepoll_rel_imp_cardinal_rel_le*], *assumption*+)
apply (*erule Ord_cardinal_rel_le*, *assumption*)
done

lemma *lepoll_rel_Ord_imp_eqpoll_rel*: [| $A \lesssim^M i$; *Ord*(*i*); $M(A)$; $M(i)$ |] ==> $|A|^M \approx^M A$
by (*blast intro*: *lepoll_rel_cardinal_rel_le well_ord_Memrel well_ord_cardinal_rel_eqpoll_rel dest*!: *lepoll_rel_well_ord*)

lemma *lesspoll_rel_imp_eqpoll_rel*: [| $A \prec^M i$; *Ord*(*i*); $M(A)$; $M(i)$ |] ==> $|A|^M \approx^M A$
using *lepoll_rel_Ord_imp_eqpoll_rel*[*OF lesspoll_rel_imp_lepoll_rel*] **.**

lemma *lesspoll_cardinal_lt_rel*:
shows [| $A \prec^M i$; *Ord*(*i*); $M(i)$; $M(A)$ |] ==> $|A|^M < i$
proof -
assume *assms*:$A \prec^M i$ ‹*Ord*(*i*)› ‹$M(i)$› ‹$M(A)$›
then

```
  have A:Ord(|A|^M) |A|^M ≈^M A M(|A|^M)
    using Ord_cardinal_rel lesspoll_rel_imp_eqpoll_rel
    by simp_all
  with assms
  have |A|^M ≺^M i
    using eq_lesspoll_rel_trans by auto
  consider |A|^M∈i | |A|^M=i | i∈|A|^M
    using Ord_linear[OF ‹Ord(i)› ‹Ord(|A|^M)›] by auto
  then
  have |A|^M < i
  proof(cases)
    case 1
    then show ?thesis using ltI ‹Ord(i)› by simp
  next
    case 2
    with ‹|A|^M ≺^M i› ‹M(i)›
    show ?thesis using lesspoll_rel_irrefl by simp
  next
    case 3
    with ‹Ord(|A|^M)›
    have i<|A|^M using ltI by simp
    with ‹M(A)› A ‹M(i)›
    have i ≺^M |A|^M
      using lt_Card_rel_imp_lesspoll_rel Card_rel_cardinal_rel by simp
    with ‹M(|A|^M)› ‹M(i)›
    show ?thesis
      using lesspoll_rel_irrefl lesspoll_rel_trans[OF ‹|A|^M ≺^M i› ‹i ≺^M _ ›]
      by simp
  qed
  then show ?thesis by simp
qed

lemma cardinal_rel_subset_Ord: [|A<=i; Ord(i); M(A); M(i)|] ==> |A|^M ⊆ i
  apply (drule subset_imp_lepoll_rel [THEN lepoll_rel_cardinal_rel_le])
       apply (auto simp add: lt_def)
  apply (blast intro: Ord_trans)
  done

— The next lemma is the first with several porting issues
lemma cons_lepoll_rel_consD:
  [| cons(u,A) ≲^M cons(v,B);  u∉A;  v∉B; M(u); M(A); M(v); M(B) |] ==> A
≲^M B
apply (simp add: def_lepoll_rel, unfold inj_def, safe)
apply (rule_tac x = λx∈A. if f‘x=v then f‘u else f‘x in rexI)
apply (rule CollectI)

apply (rule if_type [THEN lam_type])
apply (blast dest: apply_funtype)
apply (blast elim!: mem_irrefl dest: apply_funtype)
```

apply (*auto simp add*:*transM*[*of* _ *A*])
using *lam_replacement_iff_lam_closed lam_if_then_apply_replacement*
by *simp*

lemma *cons_eqpoll_rel_consD*: [| *cons*(*u*,*A*) $\approx^M$ *cons*(*v*,*B*); *u*∉*A*; *v*∉*B*; *M*(*u*); *M*(*A*); *M*(*v*); *M*(*B*) |] ==> *A* $\approx^M$ *B*
apply (*simp add*: *eqpoll_rel_iff*)
apply (*blast intro*: *cons_lepoll_rel_consD*)
done

lemma *succ_lepoll_rel_succD*: *succ*(*m*) $\lesssim^M$ *succ*(*n*) ⟹ *M*(*m*) ⟹ *M*(*n*) ==> *m* $\lesssim^M$ *n*
apply (*unfold succ_def*)
apply (*erule cons_lepoll_rel_consD*)
apply (*rule mem_not_refl*)+
apply *assumption*+
done

lemma *nat_lepoll_rel_imp_le*:
m ∈ *nat* ==> *n* ∈ *nat* ⟹ *m* $\lesssim^M$ *n* ⟹ *M*(*m*) ⟹ *M*(*n*) ⟹ *m* ≤ *n*
proof (*induct m arbitrary*: *n rule*: *nat_induct*)
case *0* **thus** *?case* **by** (*blast intro*!: *nat_0_le*)
next
case (*succ m*)
show *?case* **using** ‹*n* ∈ *nat*›
proof (*cases rule*: *natE*)
case *0* **thus** *?thesis* **using** *succ*
by (*simp add*: *def_lepoll_rel inj_def*)
next
case (*succ n'*) **thus** *?thesis* **using** *succ.hyps* ‹ *succ*(*m*) $\lesssim^M$ *n*›
by (*blast dest*!: *succ_lepoll_rel_succD*)
qed
qed

lemma *nat_eqpoll_rel_iff*: [| *m* ∈ *nat*; *n* ∈ *nat*; *M*(*m*); *M*(*n*) |] ==> *m* $\approx^M$ *n* ⟷ *m* = *n*
apply (*rule iffI*)
apply (*blast intro*: *nat_lepoll_rel_imp_le le_anti_sym elim*!: *eqpoll_relE*)
apply (*simp add*: *eqpoll_rel_refl*)
done

lemma *nat_into_Card_rel*:
assumes *n*: *n* ∈ *nat* **and** *types*: *M*(*n*) **shows** $Card^M(n)$
using *types*
apply (*subst Card_rel_def*)
proof (*unfold cardinal_rel_def*, *rule sym*)
have *Ord*(*n*) **using** *n* **by** *auto*
moreover

{ fix i
assume $i < n$ $M(i)$ $i \approx^M n$
hence *False* **using** n
by (*auto simp add*: *lt_nat_in_nat* [*THEN nat_eqpoll_rel_iff*] *types*)
}
ultimately show (μ i. $M(i) \wedge i \approx^M n) = n$ **by** (*auto intro*!: *Least_equality types eqpoll_rel_refl*)
qed

lemmas *cardinal_rel_0* = *nat_0I* [*THEN nat_into_Card_rel*, *THEN Card_rel_cardinal_rel_eq*, *simplified*, *iff*]
lemmas *cardinal_rel_1* = *nat_1I* [*THEN nat_into_Card_rel*, *THEN Card_rel_cardinal_rel_eq*, *simplified*, *iff*]

lemma *succ_lepoll_rel_natE*: [| $succ(n) \lesssim^M n$; $n \in nat$ |] ==> P
by (*rule nat_lepoll_rel_imp_le* [*THEN lt_irrefl*], *auto*)

lemma *nat_lepoll_rel_imp_ex_eqpoll_rel_n*:
[| $n \in nat$; $nat \lesssim^M X$; $M(n)$; $M(X)$|] ==> $\exists Y[M]. Y \subseteq X \,\&\, n \approx^M Y$
apply (*simp add*: *def_lepoll_rel def_eqpoll_rel*)
apply (*fast del*: *subsetI subsetCE*
intro!: *subset_SIs*
dest!: *Ord_nat* [*THEN* [*2*] *OrdmemD*, *THEN* [*2*] *restrict_inj*]
elim!: *restrict_bij*
inj_is_fun [*THEN fun_is_rel*, *THEN image_subset*])
done

lemma *lepoll_rel_succ*: $M(i) \Longrightarrow i \lesssim^M succ(i)$
by (*blast intro*: *subset_imp_lepoll_rel*)

lemma *lepoll_rel_imp_lesspoll_rel_succ*:
assumes *A*: $A \lesssim^M m$ **and** *m*: $m \in nat$
and *types*: $M(A)$ $M(m)$
shows $A \prec^M succ(m)$
proof -
{ assume $A \approx^M succ(m)$
hence $succ(m) \approx^M A$ **by** (*rule eqpoll_rel_sym*) (*auto simp add*:*types*)
also have ... $\lesssim^M m$ **by** (*rule A*)
finally have $succ(m) \lesssim^M m$ **by** (*auto simp add*:*types*)
hence *False* **by** (*rule succ_lepoll_rel_natE*) (*rule m*) **}**
moreover have $A \lesssim^M succ(m)$ **by** (*blast intro*: *lepoll_rel_trans A lepoll_rel_succ types*)
ultimately show *?thesis* **by** (*auto simp add*: *types lesspoll_rel_def*)
qed

lemma *lesspoll_rel_succ_imp_lepoll_rel*:
[| $A \prec^M succ(m)$; $m \in nat$; $M(A)$; $M(m)$ |] ==> $A \lesssim^M m$
proof -
{

assume $m \in nat$ $M(A)$ $M(m)$ $A \lesssim^M succ(m)$
$\forall f{\in}inj^M(A,\ succ(m)).\ f \notin surj^M(A,\ succ(m))$
moreover from *this*
obtain f **where** $M(f)$ $f{\in}inj^M(A,succ(m))$
using *def_lepoll_rel* **by** *auto*
moreover from *calculation*
have $f \notin surj^M(A,\ succ(m))$ **by** *simp*
ultimately
have $\exists f[M].\ f \in inj^M(A,\ m)$
using *inj_rel_not_surj_rel_succ* **by** *auto*
}
from *this*
show $[\![\ A \prec^M succ(m);\ m \in nat;\ M(A);\ M(m)\ ]\!] \Longrightarrow A \lesssim^M m$
unfolding *lepoll_rel_def eqpoll_rel_def bij_rel_def lesspoll_rel_def*
by (*simp del*:*mem_inj_abs*)
qed

lemma *lesspoll_rel_succ_iff*: $m \in nat \Longrightarrow M(A) \Longrightarrow A \prec^M succ(m) \longleftrightarrow A \lesssim^M m$
by (*blast intro*!: *lepoll_rel_imp_lesspoll_rel_succ lesspoll_rel_succ_imp_lepoll_rel*)

lemma *lepoll_rel_succ_disj*: $[|\ A \lesssim^M succ(m);\ \ m \in nat;\ M(A)\ ;\ M(m)|] \Longrightarrow A \lesssim^M m \mid A \approx^M succ(m)$
apply (*rule disjCI*)
apply (*rule lesspoll_rel_succ_imp_lepoll_rel*)
prefer *2* **apply** *assumption*
apply (*simp* (*no_asm_simp*) *add*: *lesspoll_rel_def*, *assumption*+)
done

lemma *lesspoll_rel_cardinal_rel_lt*: $[|\ A \prec^M i;\ Ord(i);\ M(A);\ M(i)\ |] \Longrightarrow |A|^M < i$
apply (*unfold lesspoll_rel_def*, *clarify*)
apply (*frule lepoll_rel_cardinal_rel_le*, *assumption*+) — because of types
apply (*blast intro*: *well_ord_Memrel well_ord_cardinal_rel_eqpoll_rel* [*THEN eqpoll_rel_sym*]
dest: *lepoll_rel_well_ord* *elim*!: *leE*)
done

lemma *lt_not_lepoll_rel*:
assumes *n*: $n<i$ $n \in nat$
and *types*:$M(n)$ $M(i)$ **shows** $\sim\ i \lesssim^M n$
proof -
{ **assume** *i*: $i \lesssim^M n$
have $succ(n) \lesssim^M i$ **using** *n*
by (*elim ltE*, *blast intro*: *Ord_succ_subsetI* [*THEN subset_imp_lepoll_rel*] *types*)
also have ... $\lesssim^M n$ **by** (*rule i*)
finally have $succ(n) \lesssim^M n$ **by** (*simp add*:*types*)

hence *False* **by** (*rule succ_lepoll_rel_natE*) (*rule n*) **}**
thus *?thesis* **by** *auto*
qed

A slightly weaker version of *nat_eqpoll_rel_iff*

lemma *Ord_nat_eqpoll_rel_iff*:
assumes *i*: *Ord*(*i*) **and** *n*: $n \in nat$
and *types*: $M(i)$ $M(n)$
shows $i \approx^M n \longleftrightarrow i{=}n$
using *i nat_into_Ord* [*OF n*]
proof (*cases rule*: *Ord_linear_lt*)
case *lt*
hence $i \in nat$ **by** (*rule lt_nat_in_nat*) (*rule n*)
thus *?thesis* **by** (*simp add*: *nat_eqpoll_rel_iff n types*)
next
case *eq*
thus *?thesis* **by** (*simp add*: *eqpoll_rel_refl types*)
next
case *gt*
hence $\sim i \lesssim^M n$ **using** *n* **by** (*rule lt_not_lepoll_rel*) (*simp_all add*: *types*)
hence $\sim i \approx^M n$ **using** *n* **by** (*blast intro*: *eqpoll_rel_imp_lepoll_rel types*)
moreover have $i \neq n$ **using** ‹*n*<*i*› **by** *auto*
ultimately show *?thesis* **by** *blast*
qed

lemma *Card_rel_nat*: $Card^M(nat)$
proof -
{ fix *i*
assume *i*: $i < nat$ $i \approx^M nat$ $M(i)$
hence $\sim nat \lesssim^M i$
by (*simp add*: *lt_def lt_not_lepoll_rel*)
hence *False* **using** *i*
by (*simp add*: *eqpoll_rel_iff*)
}
hence $(\mu\, i.\ M(i) \wedge i \approx^M nat) = nat$ **by** (*blast intro*: *Least_equality eqpoll_rel_refl*)
thus *?thesis*
by (*auto simp add*: *Card_rel_def cardinal_rel_def*)
qed

lemma *nat_le_cardinal_rel*: $nat \leq i \Longrightarrow M(i) ==> nat \leq |i|^M$
apply (*rule Card_rel_nat* [*THEN Card_rel_cardinal_rel_eq*, *THEN subst*], *simp_all*)
apply (*erule cardinal_rel_mono*, *simp_all*)
done

lemma *n_lesspoll_rel_nat*: $n \in nat ==> n \prec^M nat$
by (*blast intro*: *Card_rel_nat ltI lt_Card_rel_imp_lesspoll_rel*)

lemma *cons_lepoll_rel_cong*:

[| $A \lesssim^M B$; $b \notin B$; $M(A)$; $M(B)$; $M(b)$; $M(a)$ |] ==> $cons(a,A) \lesssim^M cons(b,B)$
apply (*subst* (*asm*) *def_lepoll_rel*, *simp_all*, *subst def_lepoll_rel*, *simp_all*, *safe*)
apply (*rule_tac* $x = \lambda y{\in}cons\ (a,A)$. *if* $y{=}a$ *then* b *else* $f`y$ **in** *rexI*)
apply (*rule_tac* $d = \%z$. *if* $z \in B$ *then converse* (f) `z *else* a **in** *lam_injective*)
apply (*safe elim*!: *consE'*)
apply *simp_all*
apply (*blast intro*: *inj_is_fun* [*THEN apply_type*])+
apply (*auto intro*:*lam_closed lam_if_then_replacement simp add*:*transM*[*of* _ *A*])
done

lemma *cons_eqpoll_rel_cong*:
[| $A \approx^M B$; $a \notin A$; $b \notin B$; $M(A)$; $M(B)$; $M(a)$; $M(b)$ |] ==> $cons(a,A) \approx^M cons(b,B)$
by (*simp add*: *eqpoll_rel_iff cons_lepoll_rel_cong*)

lemma *cons_lepoll_rel_cons_iff*:
[| $a \notin A$; $b \notin B$; $M(a)$; $M(A)$; $M(b)$; $M(B)$ |] ==> $cons(a,A) \lesssim^M cons(b,B) \longleftrightarrow A \lesssim^M B$
by (*blast intro*: *cons_lepoll_rel_cong cons_lepoll_rel_consD*)

lemma *cons_eqpoll_rel_cons_iff*:
[| $a \notin A$; $b \notin B$; $M(a)$; $M(A)$; $M(b)$; $M(B)$ |] ==> $cons(a,A) \approx^M cons(b,B) \longleftrightarrow A \approx^M B$
by (*blast intro*: *cons_eqpoll_rel_cong cons_eqpoll_rel_consD*)

lemma *singleton_eqpoll_rel_1*: $M(a) \Longrightarrow \{a\} \approx^M 1$
apply (*unfold succ_def*)
apply (*blast intro*!: *eqpoll_rel_refl* [*THEN cons_eqpoll_rel_cong*])
done

lemma *cardinal_rel_singleton*: $M(a) \Longrightarrow |\{a\}|^M = 1$
apply (*rule singleton_eqpoll_rel_1* [*THEN cardinal_rel_cong*, *THEN trans*])
apply (*simp* (*no_asm*) *add*: *nat_into_Card_rel* [*THEN Card_rel_cardinal_rel_eq*])
apply *auto*
done

lemma *not_0_is_lepoll_rel_1*: $A \neq 0$ ==> $M(A) \Longrightarrow 1 \lesssim^M A$
apply (*erule not_emptyE*)
apply (*rule_tac* $a = cons\ (x,\ A\text{-}\{x\})$ **in** *subst*)
apply (*rule_tac* [*2*] $a = cons(0,0)$ **and** $P = \%y.\ y \lesssim^M cons\ (x,\ A\text{-}\{x\})$ **in** *subst*)
apply *auto*
proof -
fix x
assume $M(A)$
then
show $x \in A \Longrightarrow \{0\} \lesssim^M cons(x,\ A - \{x\})$
by (*auto intro*: *cons_lepoll_rel_cong transM*[*OF* _ ‹$M(A)$›] *subset_imp_lepoll_rel*)

qed

lemma *succ_eqpoll_rel_cong*: $A \approx^M B \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow succ(A) \approx^M succ(B)$
apply (*unfold succ_def*)
apply (*simp add*: *cons_eqpoll_rel_cong mem_not_refl*)
done

The next result was not straightforward to port, and even a different statement was needed.

lemma *sum_bij_rel*:
[| $f \in bij^M(A,C)$; $g \in bij^M(B,D)$; $M(f)$; $M(A)$; $M(C)$; $M(g)$; $M(B)$; $M(D)$|]
==> $(\lambda z \in A+B.\ case(\%x.\ Inl(f`x),\ \%y.\ Inr(g`y),\ z)) \in bij^M(A+B,\ C+D)$
proof -
assume *asm*:$f \in bij^M(A,C)$ $g \in bij^M(B,D)$ $M(f)$ $M(A)$ $M(C)$ $M(g)$ $M(B)$ $M(D)$
then
have $M(\lambda z \in A+B.\ case(\%x.\ Inl(f`x),\ \%y.\ Inr(g`y),\ z))$
using *transM*[*OF* _ ‹$M(A)$›] *transM*[*OF* _ ‹$M(B)$›]
by (*auto intro*:*case_replacement4*[*THEN lam_closed*])
with *asm*
show *?thesis*
apply *simp*
apply (*rule_tac d* = *case* (%*x*. *Inl* (*converse*(*f*)`*x*), %*y*. *Inr*(*converse*(*g*)`*y*))
in *lam_bijective*)
apply (*typecheck add*: *bij_is_inj inj_is_fun*)
apply (*auto simp add*: *left_inverse_bij right_inverse_bij*)
done
qed

lemma *sum_bij_rel'*:
assumes $f \in bij^M(A,C)$ $g \in bij^M(B,D)$ $M(f)$
$M(A)$ $M(C)$ $M(g)$ $M(B)$ $M(D)$
shows
$(\lambda z \in A+B.\ case(\lambda x.\ Inl(f`x),\ \lambda y.\ Inr(g`y),\ z)) \in bij(A+B,\ C+D)$
$M(\lambda z \in A+B.\ case(\lambda x.\ Inl(f`x),\ \lambda y.\ Inr(g`y),\ z))$
proof -
from *assms*
show $M(\lambda z \in A+B.\ case(\lambda x.\ Inl(f`x),\ \lambda y.\ Inr(g`y),\ z))$
using *transM*[*OF* _ ‹$M(A)$›] *transM*[*OF* _ ‹$M(B)$›]
by (*auto intro*:*case_replacement4*[*THEN lam_closed*])
with *assms*
show $(\lambda z \in A+B.\ case(\lambda x.\ Inl(f`x),\ \lambda y.\ Inr(g`y),\ z)) \in bij(A+B,\ C+D)$
apply *simp*
apply (*rule_tac d* = *case* (%*x*. *Inl* (*converse*(*f*)`*x*), %*y*. *Inr*(*converse*(*g*)`*y*))
in *lam_bijective*)
apply (*typecheck add*: *bij_is_inj inj_is_fun*)
apply (*auto simp add*: *left_inverse_bij right_inverse_bij*)
done

qed

lemma *sum_eqpoll_rel_cong*:
assumes $A \approx^M C\ B \approx^M D\ M(A)\ M(C)\ M(B)\ M(D)$
shows $A+B \approx^M C+D$
using *assms*
proof (*simp add*: *def_eqpoll_rel*, *safe*, *rename_tac g*)
fix *f g*
assume $M(f)\ f \in bij(A,\ C)\ M(g)\ g \in bij(B,\ D)$
with *assms*
obtain *h* **where** $h \in bij(A+B,\ C+D)\ M(h)$
using *sum_bij_rel'*[*of f A C g B D*] **by** *simp*
then
show $\exists f[M].\ f \in bij(A\ +\ B,\ C\ +\ D)$
by *auto*
qed

lemma *prod_bij_rel'*:
assumes $f \in bij^M(A,C)\ g \in bij^M(B,D)\ M(f)$
$M(A)\ M(C)\ M(g)\ M(B)\ M(D)$
shows
$(\lambda<x,y>\in A*B.\ <f`x,\ g`y>) \in bij(A*B,\ C*D)$
$M(\lambda<x,y>\in A*B.\ <f`x,\ g`y>)$
proof -
from *assms*
show $M((\lambda<x,y>\in A*B.\ <f`x,\ g`y>))$
using *transM*[*OF _* ‹*M*(*A*)›] *transM*[*OF _* ‹*M*(*B*)›]
transM[*OF _ cartprod_closed*, *of _ A B*]
by (*auto intro*:*prod_fun_replacement*[*THEN lam_closed*, *of f g A*×*B*])
with *assms*
show $(\lambda<x,y>\in A*B.\ <f`x,\ g`y>) \in bij(A*B,\ C*D)$
apply *simp*
apply (*rule_tac d* = %<*x*,*y*>. <*converse* (*f*) `*x*, *converse* (*g*) `*y*>
in *lam_bijective*)
apply (*typecheck add*: *bij_is_inj inj_is_fun*)
apply (*auto simp add*: *left_inverse_bij right_inverse_bij*)
done
qed

lemma *prod_eqpoll_rel_cong*:
assumes $A \approx^M C\ B \approx^M D\ M(A)\ M(C)\ M(B)\ M(D)$
shows $A\times B \approx^M C\times D$
using *assms*
proof (*simp add*: *def_eqpoll_rel*, *safe*, *rename_tac g*)
fix *f g*
assume $M(f)\ f \in bij(A,\ C)\ M(g)\ g \in bij(B,\ D)$
with *assms*
obtain *h* **where** $h \in bij(A\times B,\ C\times D)\ M(h)$
using *prod_bij_rel'*[*of f A C g B D*] **by** *simp*

```
  then
  show ∃f[M]. f ∈ bij(A × B, C × D)
    by auto
qed

lemma inj_rel_disjoint_eqpoll_rel:
  [| f ∈ inj^M(A,B);  A ∩ B = 0;M(f); M(A);M(B) |] ==> A ∪ (B - range(f))
≈^M B
  apply (simp add: def_eqpoll_rel)
  apply (rule rexI)
  apply (rule_tac c = %x. if x ∈ A then f‘x else x
      and d = %y. if y ∈ range (f) then converse (f) ‘y else y
      in lam_bijective)
     apply (blast intro!: if_type inj_is_fun [THEN apply_type])
    apply (simp (no_asm_simp) add: inj_converse_fun [THEN apply_funtype])
   apply (safe elim!: UnE')
    apply (simp_all add: inj_is_fun [THEN apply_rangeI])
   apply (blast intro: inj_converse_fun [THEN apply_type])
proof -
  assume f ∈ inj(A, B) A ∩ B = 0 M(f) M(A) M(B)
  then
  show M(λx∈A ∪ (B - range(f)). if x ∈ A then f ‘ x else x)
    using  transM[OF _ ‹M(A)›] transM[OF _ ‹M(B)›]
      lam_replacement_iff_lam_closed lam_if_then_replacement2
    by auto
qed

lemma Diff_sing_lepoll_rel:
  [| a ∈ A;  A ≲^M succ(n); M(a); M(A); M(n) |] ==> A - {a} ≲^M n
  apply (unfold succ_def)
  apply (rule cons_lepoll_rel_consD)
   apply (rule_tac [3] mem_not_refl)
  apply (erule cons_Diff [THEN ssubst], simp_all)
  done

lemma lepoll_rel_Diff_sing:
  assumes A: succ(n) ≲^M A
    and types: M(n) M(A) M(a)
  shows n ≲^M A - {a}
proof -
  have cons(n,n) ≲^M A using A
    by (unfold succ_def)
  also from types
  have ... ≲^M cons(a, A-{a})
    by (blast intro: subset_imp_lepoll_rel)
  finally have cons(n,n) ≲^M cons(a, A-{a}) by (simp_all add:types)
  with types
  show ?thesis
    by (blast intro: cons_lepoll_rel_consD mem_irrefl)
```

qed

lemma *Diff_sing_eqpoll_rel*: [| $a \in A$; $A \approx^M succ(n)$; $M(a)$; $M(A)$; $M(n)$ |] ==> $A - \{a\} \approx^M n$
by (*blast intro*!: *eqpoll_relI*
elim!: *eqpoll_relE*
intro: *Diff_sing_lepoll_rel lepoll_rel_Diff_sing*)

lemma *lepoll_rel_1_is_sing*: [| $A \lesssim^M 1$; $a \in A$;$M(a)$; $M(A)$ |] ==> $A = \{a\}$
apply (*frule Diff_sing_lepoll_rel*, *assumption+*, *simp*)
apply (*drule lepoll_rel_0_is_0*, *simp*)
apply (*blast elim*: *equalityE*)
done

lemma *Un_lepoll_rel_sum*: $M(A) \Longrightarrow M(B) \Longrightarrow A \cup B \lesssim^M A+B$
apply (*simp add*: *def_lepoll_rel*)
apply (*rule_tac* $x = \lambda x \in A \cup B$. *if* $x \in A$ *then Inl* (x) *else Inr* (x) **in** *rexI*)
apply (*rule_tac* $d = \%z$. *snd* (z) **in** *lam_injective*)
apply *force*
apply (*simp add*: *Inl_def Inr_def*)
proof -
assume $M(A)$ $M(B)$
then
show $M(\lambda x \in A \cup B$. *if* $x \in A$ *then Inl*(x) *else Inr*$(x))$
using *transM*[*OF* _ ‹$M(A)$›] *transM*[*OF* _ ‹$M(B)$›] *if_then_Inj_replacement*
by (*rule_tac lam_closed*) *auto*
qed

lemma *well_ord_Un_M*:
assumes *well_ord*(X,R) *well_ord*(Y,S)
and *types*: $M(X)$ $M(R)$ $M(Y)$ $M(S)$
shows $\exists T[M]$. *well_ord*($X \cup Y$, T)
using *assms*
by (*erule_tac well_ord_radd* [*THEN* [*3*] *Un_lepoll_rel_sum* [*THEN lepoll_rel_well_ord*]])
(*auto simp add*: *types*)

lemma *disj_Un_eqpoll_rel_sum*: $M(A) \Longrightarrow M(B) \Longrightarrow A \cap B = 0 \Longrightarrow A \cup B \approx^M A + B$
apply (*simp add*: *def_eqpoll_rel*)
apply (*rule_tac* $x = \lambda a \in A \cup B$. *if* $a \in A$ *then Inl* (a) *else Inr* (a) **in** *rexI*)
apply (*rule_tac* $d = \%z$. *case* $(\%x.\ x, \%x.\ x, z)$ **in** *lam_bijective*)
apply *auto*
proof -
assume $M(A)$ $M(B)$
then
show $M(\lambda x \in A \cup B$. *if* $x \in A$ *then Inl*(x) *else Inr*$(x))$
using *transM*[*OF* _ ‹$M(A)$›] *transM*[*OF* _ ‹$M(B)$›] *if_then_Inj_replacement*
by (*rule_tac lam_closed*) *auto*
qed

lemma *eqpoll_rel_imp_Finite_rel_iff*: $A \approx^M B$ ==> $M(A) \implies M(B) \implies$ *Finite_rel*$(M,A) \longleftrightarrow$ *Finite_rel*(M,B)
apply (*unfold Finite_rel_def*)
apply (*blast intro*: *eqpoll_rel_trans eqpoll_rel_sym*)
done

— It seems reasonable to have the absoluteness of *Finite* here, and deduce the rest of the results from this.
Perhaps modularize that proof to have absoluteness of injections and bijections of finite sets (cf. $[\![?A \prec^M succ(?m);\ ?m \in nat;\ M(?A);\ M(?m)]\!] \implies ?A \lesssim^M ?m$.

lemma *Finite_abs*[*simp*]: **assumes** $M(A)$ **shows** *Finite_rel*$(M,A) \longleftrightarrow$ *Finite*(A)
unfolding *Finite_rel_def Finite_def*
proof (*simp*, *intro iffI*)
assume $\exists\, n{\in}nat.\ A \approx^M n$
then
obtain n **where** $A \approx^M n\ n{\in}nat$ **by** *blast*
with *assms*
show $\exists\, n{\in}nat.\ A \approx n$
unfolding *eqpoll_def* **using** *nat_into_M* **by** (*auto simp add*:*def_eqpoll_rel*)
next
fix n
assume $\exists\, n{\in}nat.\ A \approx n$
then
obtain n **where** $A \approx n\ n{\in}nat$ **by** *blast*
moreover from *this*
obtain f **where** $f \in bij(A,n)$ **unfolding** *eqpoll_def* **by** *auto*
moreover
note *assms*
moreover from *calculation*
have *converse*$(f) \in n{\to}A$ **using** *bij_is_fun* **by** *simp*
moreover from *calculation*
have $M(converse(f))$ **using** *transM*[*of* _ $n{\to}A$] **by** *simp*
moreover from *calculation*
have $M(f)$ **using** *bij_is_fun*
fun_is_rel[*of f A* $\lambda\_.\ n$, *THEN converse_converse*]
converse_closed[*of converse*(f)] **by** *simp*
ultimately
show $\exists\, n{\in}nat.\ A \approx^M n$
by (*force dest*:*nat_into_M simp add*:*def_eqpoll_rel*)
qed

lemma *lepoll_rel_nat_imp_Finite_rel*:
assumes *A*: $A \lesssim^M n$ **and** *n*: $n \in nat$
and *types*: $M(A)\ M(n)$
shows *Finite_rel*(M,A)

proof -
  have $A \lesssim^M n \Longrightarrow$ *Finite_rel*(M,A) **using** n
    proof (*induct n*)
      case *0*
      hence $A = 0$ **by** (*rule lepoll_rel_0_is_0*, *simp_all add*:*types*)
      thus *?case* **by** *simp*
    next
      case (*succ n*)
    hence $A \lesssim^M n \vee A \approx^M succ(n)$ **by** (*blast dest*: *lepoll_rel_succ_disj intro*:*types*)
      thus *?case* **using** *succ* **by** (*auto simp add*: *Finite_rel_def types*)
    qed
  thus *?thesis* **using** A **.**
qed

lemma *lesspoll_rel_nat_is_Finite_rel*:
  $A \prec^M nat \Longrightarrow M(A) \Longrightarrow$ *Finite_rel*(M,A)
apply (*unfold Finite_rel_def*)
apply (*auto dest*: *ltD lesspoll_rel_cardinal_rel_lt*
                lesspoll_rel_imp_eqpoll_rel [*THEN eqpoll_rel_sym*])
done

lemma *lepoll_rel_Finite_rel*:
  assumes *Y*: $Y \lesssim^M X$ **and** *X*: *Finite_rel*(M,X)
    and *types*:$M(Y)$ $M(X)$
  shows *Finite_rel*(M,Y)
proof -
  obtain n **where** n: $n \in nat$ $X \approx^M n$ $M(n)$ **using** X
    by (*auto simp add*: *Finite_rel_def*)
  have $Y \lesssim^M X$ **by** (*rule Y*)
  also have ... $\approx^M n$ **by** (*rule n*)
  finally have $Y \lesssim^M n$ **by** (*simp_all add*:*types* ‹$M(n)$›)
  thus *?thesis* **using** n
    by (*simp add*: *lepoll_rel_nat_imp_Finite_rel types* ‹$M(n)$› *del*:*Finite_abs*)
qed

lemma *succ_lepoll_rel_imp_not_empty*: $succ(x) \lesssim^M y$ ==> $M(x) \Longrightarrow M(y)$
$\Longrightarrow y \neq 0$
  by (*fast dest*!: *lepoll_rel_0_is_0*)

lemma *eqpoll_rel_succ_imp_not_empty*: $x \approx^M succ(n)$ ==> $M(x) \Longrightarrow M(n)$
$\Longrightarrow x \neq 0$
  by (*fast elim*!: *eqpoll_rel_sym* [*THEN eqpoll_rel_0_is_0*, *THEN succ_neq_0*])

lemma *Finite_subset_closed*:
  assumes *Finite*(B) $B \subseteq A$ $M(A)$
  shows $M(B)$
proof -
  from ‹*Finite*(B)› ‹$B \subseteq A$›
  show *?thesis*

proof(*induct*,*simp*)
case (*cons x D*)
with *assms*
have *M*(*D*) *x*∈*A*
unfolding *cons_def* **by** *auto*
then
show *?case* **using** *transM*[*OF* _ ‹*M*(*A*)›] **by** *simp*
qed
qed

lemma *Finite_Pow_abs*:
assumes *Finite*(*A*) *M*(*A*)
shows *Pow*(*A*) = *Pow_rel*(*M*,*A*)
using *Finite_subset_closed*[*OF subset_Finite*] *assms Pow_rel_char*
by *auto*

lemma *Finite_Pow_rel*:
assumes *Finite*(*A*) *M*(*A*)
shows *Finite*(*Pow_rel*(*M*,*A*))
using *Finite_Pow Finite_Pow_abs*[*symmetric*] *assms* **by** *simp*

lemma *Pow_rel_0* [*simp*]: *Pow_rel*(*M*,*0*) = {*0*}
using *Finite_Pow_abs*[*of 0*] **by** *simp*

end

end

23 Relative, Choice-less Cardinal Arithmetic

theory *CardinalArith_Relative*
imports
Cardinal_Relative

begin

relativize functional *rvimage rvimage_rel* **external**
relationalize *rvimage_rel is_rvimage*

definition
csquare_lam :: *i*⇒*i* **where**
csquare_lam(*K*) ≡ λ⟨*x*,*y*⟩∈*K*×*K*. ⟨*x* ∪ *y*, *x*, *y*⟩

— Can't do the next thing because split is a missing HOC

relativize_tm <*fst*(*x*) ∪ *snd*(*x*), *fst*(*x*), *snd*(*x*)> *is_csquare_lam_body*

definition
is_csquare_lam :: $[i⇒o,i,i]⇒o$ **where**
is_csquare_lam$(M,K,l) \equiv \exists\, K2[M].\ cartprod(M,K,K,K2) \wedge$
is_lambda$(M,K2,is\_csquare\_lam\_body(M),l)$

definition *jump_cardinal_body* :: $[i⇒o,i] ⇒ i$ **where**
jump_cardinal_body$(M,X) \equiv$
$\{z\ .\ r \in Pow^M(X \times X),\ M(z) \wedge M(r) \wedge well\_ord(X,\ r) \wedge z = ordertype(X,$
$r)\}$

lemma (**in** *M_cardinals*) *csquare_lam_closed*[*intro*,*simp*]: $M(K) \Longrightarrow M(csquare\_lam(K))$
using *csquare_lam_replacement* **unfolding** *csquare_lam_def*
by (*rule lam_closed*) (*auto dest*:*transM*)

locale *M_pre_cardinal_arith* = *M_cardinals* +
assumes
ord_iso_separation: $M(A) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow$
separation$(M,\ \lambda f.\ \forall\, x{\in}A.\ \forall\, y{\in}A.\ \langle x,\ y\rangle \in r \longleftrightarrow \langle f\ `\ x,\ f\ `\ y\rangle \in s)$
and
wfrec_pred_replacement:$M(A) \Longrightarrow M(r) \Longrightarrow$
wfrec_replacement$(M,\ \lambda x\, f\, z.\ z = f\ ``\ Order.pred(A,\ x,\ r),\ r)$

locale *M_cardinal_arith* = *M_pre_cardinal_arith* +
assumes
ordertype_replacement :
$M(X) \Longrightarrow strong\_replacement(M,\lambda\ x\ z\ .\ M(z) \wedge M(x) \wedge x{\in}Pow\_rel(M,X{\times}X)$
$\wedge\ well\_ord(X,\ x) \wedge z{=}ordertype(X,x))$
and
strong_replacement_jc_body :
$strong\_replacement(M,\lambda\ x\ z\ .\ M(z) \wedge M(x) \wedge z = jump\_cardinal\_body(M,x))$
and
surj_imp_inj_replacement:
$M(f) \Longrightarrow M(x) \Longrightarrow strong\_replacement(M,\ \lambda y\ z.\ y \in f\text{-}``\ \{x\} \wedge z = \{\langle x,\ y\rangle\})$
$M(f) \Longrightarrow strong\_replacement(M,\ \lambda x\ z.\ z = Sigfun(x,\ \lambda y.\ f\text{-}``\ \{y\}))$
$M(f) \Longrightarrow strong\_replacement(M,\ \lambda x\ y.\ y = f\text{-}``\ \{x\})$
$M(f) \Longrightarrow M(r) \Longrightarrow strong\_replacement(M,\ \lambda x\ y.\ y = \langle x,\ minimum(r,\ f\text{-}``$
$\{x\})\rangle)$

relativize_tm $\exists\, x'\ y'\ x\ y.\ z = \langle\langle x',\ y'\rangle,\ x,\ y\rangle \wedge (\langle x',\ x\rangle \in r \vee x' = x \wedge \langle y',\ y\rangle \in$
$s)$
is_rmultP

relativize functional *rmult rmult_rel* **external**
relationalize *rmult_rel is_rmult*

lemma (**in** *M_trivial*) *rmultP_abs* [*absolut*]: $[\![\ M(r);\ M(s);\ M(z)\ ]\!] \Longrightarrow is\_rmultP(M,s,r,z)$
$\longleftrightarrow$
$(\exists\, x'\ y'\ x\ y.\ z = \langle\langle x',\ y'\rangle,\ x,\ y\rangle \wedge (\langle x',\ x\rangle \in r \vee x' = x \wedge \langle y',\ y\rangle \in s))$

unfolding *is_rmultP_def* **by** (*auto dest*:*transM*)

definition
is_csquare_rel :: [*i*⇒*o*,*i*,*i*]⇒*o* **where**
is_csquare_rel(*M*,*K*,*cs*) ≡ ∃ *K2*[*M*]. ∃ *la*[*M*]. ∃ *memK*[*M*].
∃ *rmKK*[*M*]. ∃ *rmKK2*[*M*].
cartprod(*M*,*K*,*K*,*K2*) ∧ *is_csquare_lam*(*M*,*K*,*la*) ∧
membership(*M*,*K*,*memK*) ∧ *is_rmult*(*M*,*K*,*memK*,*K*,*memK*,*rmKK*) ∧
is_rmult(*M*,*K*,*memK*,*K2*,*rmKK*,*rmKK2*) ∧ *is_rvimage*(*M*,*K2*,*la*,*rmKK2*,*cs*)

context *M_basic*
begin

lemma *rvimage_abs*[*absolut*]:
assumes *M*(*A*) *M*(*f*) *M*(*r*) *M*(*z*)
shows *is_rvimage*(*M*,*A*,*f*,*r*,*z*) ⟷ *z* = *rvimage*(*A*,*f*,*r*)
using *assms transM*[*OF* _ ‹*M*(*A*)›]
unfolding *is_rvimage_def rvimage_def*
by *auto*

lemma *rmult_abs* [*absolut*]: ⟦ *M*(*A*); *M*(*r*); *M*(*B*); *M*(*s*); *M*(*z*) ⟧ ⟹
is_rmult(*M*,*A*,*r*,*B*,*s*,*z*) ⟷ *z*=*rmult*(*A*,*r*,*B*,*s*)
using *rmultP_abs transM*[*of* _ (*A* × *B*) × *A* × *B*]
unfolding *is_rmultP_def is_rmult_def rmult_def*
by (*auto del*: *iffI*)

lemma *csquare_lam_body_abs*[*absolut*]: *M*(*x*) ⟹ *M*(*z*) ⟹
is_csquare_lam_body(*M*,*x*,*z*) ⟷ *z* = <*fst*(*x*) ∪ *snd*(*x*), *fst*(*x*), *snd*(*x*)>
unfolding *is_csquare_lam_body_def* **by** (*simp add*:*absolut*)

lemma *csquare_lam_abs*[*absolut*]: *M*(*K*) ⟹ *M*(*l*) ⟹
is_csquare_lam(*M*,*K*,*l*) ⟷ *l* = (λ*x*∈*K*×*K*. ⟨*fst*(*x*) ∪ *snd*(*x*), *fst*(*x*), *snd*(*x*)⟩)
unfolding *is_csquare_lam_def*
using *lambda_abs2*[*of* *K*×*K* *is_csquare_lam_body*(*M*)
λ*x*. ⟨*fst*(*x*) ∪ *snd*(*x*), *fst*(*x*), *snd*(*x*)⟩]
unfolding *Relation1_def* **by** (*simp add*:*absolut*)

lemma *csquare_lam_eq_lam*:*csquare_lam*(*K*) = (λ*z*∈*K*×*K*. <*fst*(*z*) ∪ *snd*(*z*),
fst(*z*), *snd*(*z*)>)
proof -
have (λ⟨*x*,*y*⟩∈*K* × *K*. ⟨*x* ∪ *y*, *x*, *y*⟩)‘*z* =
(λ*z*∈*K*×*K*. <*fst*(*z*) ∪ *snd*(*z*), *fst*(*z*), *snd*(*z*)>)‘*z* **if** *z*∈*K*×*K* **for** *z*
using *that* **by** *auto*
then
show *?thesis*
unfolding *csquare_lam_def*
by *simp*
qed

end

context *M_pre_cardinal_arith*
begin

lemma *csquare_rel_closed*[*intro*,*simp*]: $M(K) \Longrightarrow M(csquare\_rel(K))$
using *csquare_lam_replacement* **unfolding** *csquare_rel_def*
by (*intro rvimage_closed lam_closed*) (*auto dest*:*transM*)

lemma *csquare_rel_abs*[*absolut*]: $[\![M(K);\ M(cs)]\!] \Longrightarrow$
$is\_csquare\_rel(M,K,cs) \longleftrightarrow cs = csquare\_rel(K)$
unfolding *is_csquare_rel_def csquare_rel_def*
using *csquare_lam_closed*[*unfolded csquare_lam_eq_lam*]
by (*simp add*:*absolut csquare_lam_eq_lam*[*unfolded csquare_lam_def*])

end

relativize functional *csucc csucc_rel* **external**
relationalize *csucc_rel is_csucc*
synthesize *is_csucc* **from_definition assuming** *nonempty*
arity_theorem for *is_csucc_fm*

abbreviation
$csucc\_r :: [i,i \Rightarrow o] \Rightarrow i$ ($\langle' (\_^+') \text{–}\rangle$) **where**
$csucc\_r(x,M) \equiv csucc\_rel(M,x)$

abbreviation
$csucc\_r\_set :: [i,i] \Rightarrow i$ ($\langle' (\_^+') \text{–}\rangle$) **where**
$csucc\_r\_set(x,M) \equiv csucc\_rel(\#\#M,x)$

context *M_Perm*
begin

rel_closed for *csucc*
using *Least_closed'*[*of* λ *L*. $M(L) \wedge Card^M(L) \wedge K < L$]
unfolding *csucc_rel_def*
by *simp*

is_iff_rel for *csucc*
using *least_abs'*[*of* λ *L*. $M(L) \wedge Card^M(L) \wedge K < L$ *res*]
is_Card_iff
unfolding *is_csucc_def csucc_rel_def*
by (*simp add*:*absolut*)

end

notation *csucc_rel* ($\langle csucc\text{–}'(\_')\rangle$)

context *M_cardinals*
begin

lemma *Card_rel_Union* [*simp*,*intro*,*TC*]:
  assumes *A*: $\bigwedge x.\ x{\in}A \Longrightarrow Card^M(x)$ **and**
    types:$M(A)$
  shows $Card^M(\bigcup(A))$
proof (*rule Card_relI*)
  show *Ord*($\bigcup A$) **using** *A*
    by (*simp add*: *Card_rel_is_Ord types transM*)
next
  fix *j*
  assume *j*: $j < \bigcup A$
  moreover from *this*
  have $M(j)$ **unfolding** *lt_def* **by** (*auto simp add*:*types dest*:*transM*)
  from *j*
  have $\exists\, c{\in}A.\ j \in c \wedge Card^M(c)$ **using** *A types*
    unfolding *lt_def*
    by (*simp*)
  then
  obtain *c* **where** *c*: $c{\in}A$ $j < c$ $Card^M(c)$ $M(c)$
    using *Card_rel_is_Ord types* **unfolding** *lt_def*
    by (*auto dest*:*transM*)
  with ‹$M(j)$›
  have *jls*: $j \prec^M c$
    by (*simp add*: *lt_Card_rel_imp_lesspoll_rel types*)
  { assume *eqp*: $j \approx^M \bigcup A$
    have $c \lesssim^M \bigcup A$ **using** *c*
      by (*blast intro*: *subset_imp_lepoll_rel types*)
    also from *types* ‹$M(j)$›
    have ... $\approx^M j$ **by** (*rule_tac eqpoll_rel_sym* [*OF eqp*]) (*simp_all add*:*types*)
    also have ... $\prec^M c$ **by** (*rule jls*)
    finally have $c \prec^M c$ **by** (*simp_all add*:‹$M(c)$› ‹$M(j)$› *types*)
    with ‹$M(c)$›
    have *False*
      by (*auto dest*:*lesspoll_rel_irrefl*)
  } thus $\neg\ j \approx^M \bigcup A$ **by** *blast*
qed (*simp_all add*:*types*)

lemma *in_Card_imp_lesspoll*: [| $Card^M(K)$; $b \in K$; $M(K)$; $M(b)$ |] ==> $b \prec^M K$
apply (*unfold lesspoll_rel_def*)
apply (*simp add*: *Card_rel_iff_initial*)
apply (*fast intro*!: *le_imp_lepoll_rel ltI leI*)

done

23.1 Cardinal addition

Note (Paulson): Could omit proving the algebraic laws for cardinal addition and multiplication. On finite cardinals these operations coincide with addition and multiplication of natural numbers; on infinite cardinals they coincide with union (maximum). Either way we get most laws for free.

23.1.1 Cardinal addition is commutative

lemma *sum_commute_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow A+B \approx^M B+A$
proof (*simp add*: *def_eqpoll_rel*, *rule rexI*)
show $(\lambda z \in A+B.\ case(Inr,Inl,z)) \in bij(A+B,\ B+A)$
by (*auto intro*: *lam_bijective* [**where** $d = case(Inr,Inl)$])
assume $M(A)$ $M(B)$
then
show $M(\lambda z \in A + B.\ case(Inr,\ Inl,\ z))$
using *case_replacement1*
by (*rule_tac lam_closed*) (*auto dest*:*transM*)
qed

lemma *cadd_rel_commute*: $M(i) \Longrightarrow M(j) \Longrightarrow i \oplus^M j = j \oplus^M i$
apply (*unfold cadd_rel_def*)
apply (*auto intro*: *sum_commute_eqpoll_rel* [*THEN cardinal_rel_cong*])
done

23.1.2 Cardinal addition is associative

lemma *sum_assoc_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (A+B)+C \approx^M A+(B+C)$
apply (*simp add*: *def_eqpoll_rel*)
apply (*rule rexI*)
apply (*rule sum_assoc_bij*)
using *case_replacement2*
by (*rule_tac lam_closed*) (*auto dest*:*transM*)

Unconditional version requires AC

lemma *well_ord_cadd_rel_assoc*:
assumes *i*: *well_ord*(*i*,*ri*) **and** *j*: *well_ord*(*j*,*rj*) **and** *k*: *well_ord*(*k*,*rk*)
and
types: $M(i)$ $M(ri)$ $M(j)$ $M(rj)$ $M(k)$ $M(rk)$
shows $(i \oplus^M j) \oplus^M k = i \oplus^M (j \oplus^M k)$
proof (*simp add*: *assms cadd_rel_def*, *rule cardinal_rel_cong*)
from *types*
have $|i + j|^M + k \approx^M (i + j) + k$
by (*auto intro*!: *sum_eqpoll_rel_cong well_ord_cardinal_rel_eqpoll_rel eqpoll_rel_refl well_ord_radd i j*)

also have ... $\approx^M i + (j + k)$
by (*rule sum_assoc_eqpoll_rel*) (*simp_all add*:*types*)
also
have ... $\approx^M i + |j + k|^M$
proof (*auto intro*!: *sum_eqpoll_rel_cong intro*:*eqpoll_rel_refl simp add*:*types*)
from *types*
have $|j + k|^M \approx^M j + k$
using *well_ord_cardinal_rel_eqpoll_rel*[*OF well_ord_radd*, *OF j k*]
by (*simp*)
with *types*
show $j + k \approx^M |j + k|^M$
using *eqpoll_rel_sym* **by** *simp*
qed
finally show $|i + j|^M + k \approx^M i + |j + k|^M$ **by** (*simp_all add*:*types*)
qed (*simp_all add*:*types*)

23.1.3 0 is the identity for addition

lemma *case_id_eq*: $x \in sum(A,B) \Longrightarrow case(\lambda z\ .\ z,\ \lambda z.\ z\ ,x) = snd(x)$
unfolding *case_def cond_def* **by** (*auto simp*:*Inl_def Inr_def*)

lemma *lam_case_id*: $(\lambda z \in 0 + A.\ case(\lambda x.\ x,\ \lambda y.\ y,\ z)) = (\lambda z \in 0 + A\ .\ snd(z))$
using *case_id_eq* **by** *simp*

lemma *sum_0_eqpoll_rel*: $M(A) \Longrightarrow 0+A \approx^M A$
apply (*simp add*:*def_eqpoll_rel*)
apply (*rule rexI*)
apply (*rule bij_0_sum*,*subst lam_case_id*)
using *lam_replacement_snd*[*unfolded lam_replacement_def*]
by (*rule lam_closed*)
(*auto simp add*:*case_def cond_def Inr_def dest*:*transM*)

lemma *cadd_rel_0* [*simp*]: $Card^M(K) \Longrightarrow M(K) \Longrightarrow 0 \oplus^M K = K$
apply (*simp add*: *cadd_rel_def*)
apply (*simp add*: *sum_0_eqpoll_rel* [*THEN cardinal_rel_cong*] *Card_rel_cardinal_rel_eq*)
done

23.1.4 Addition by another cardinal

lemma *sum_lepoll_rel_self*: $M(A) \Longrightarrow M(B) \Longrightarrow A \lesssim^M A+B$
proof (*simp add*: *def_lepoll_rel*, *rule rexI*)
show $(\lambda x \in A.\ Inl\ (x)) \in inj(A,\ A + B)$
by (*simp add*: *inj_def*)
assume $M(A)$ $M(B)$
then
show $M(\lambda x \in A.\ Inl(x))$
using *Inl_replacement1 transM*[*OF _* ‹$M(A)$›]
by (*rule_tac lam_closed*) (*auto simp add*: *Inl_def*)
qed

lemma *cadd_rel_le_self*:
  assumes K: $Card^M(K)$ **and** L: $Ord(L)$ **and**
    types:$M(K)$ $M(L)$
  shows $K \leq (K \oplus^M L)$
proof (*simp add*:*types cadd_rel_def*)
  have $K \leq |K|^M$
    by (*rule Card_rel_cardinal_rel_le* [*OF K*]) (*simp add*:*types*)
  moreover have $|K|^M \leq |K + L|^M$ **using** K L
    by (*blast intro*: *well_ord_lepoll_rel_imp_cardinal_rel_le sum_lepoll_rel_self*
                  well_ord_radd well_ord_Memrel Card_rel_is_Ord types)
  ultimately show $K \leq |K + L|^M$
    by (*blast intro*: *le_trans*)
qed

23.1.5 Monotonicity of addition

lemma *sum_lepoll_rel_mono*:
    [| $A \lesssim^M C$; $B \lesssim^M D$; $M(A)$; $M(B)$; $M(C)$; $M(D)$ |] ==> $A + B \lesssim^M C + D$
apply (*simp add*: *def_lepoll_rel*)
apply (*elim rexE*)
apply (*rule_tac* $x = \lambda z{\in}A+B.$ *case* (%*w*. *Inl*(*f*‘*w*), %*y*. *Inr*(*fa*‘*y*), *z*) **in** *rexI*)
apply (*rule_tac* d = *case* (%*w*. *Inl*(*converse*(*f*) ‘*w*), %*y*. *Inr*(*converse*(*fa*) ‘ *y*))
      in *lam_injective*)
    apply (*typecheck add*: *inj_is_fun*, *auto*)
  apply (*rule_tac lam_closed*, *auto dest*:*transM intro*:*case_replacement4*)
  done

lemma *cadd_rel_le_mono*:
    [| $K' \leq K$; $L' \leq L$;$M(K')$;$M(K)$;$M(L')$;$M(L)$ |] ==> $(K' \oplus^M L') \leq (K \oplus^M L)$
apply (*unfold cadd_rel_def*)
apply (*safe dest*!: *le_subset_iff* [*THEN iffD1*])
apply (*rule well_ord_lepoll_rel_imp_cardinal_rel_le*)
apply (*blast intro*: *well_ord_radd well_ord_Memrel*)
apply (*auto intro*: *sum_lepoll_rel_mono subset_imp_lepoll_rel*)
done

23.1.6 Addition of finite cardinals is "ordinary" addition

lemma *sum_succ_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow succ(A)+B \approx^M succ(A+B)$
apply (*simp add*:*def_eqpoll_rel*)
apply (*rule rexI*)
apply (*rule_tac* c = %*z*. *if z*=*Inl* (*A*) *then A*+*B else z*
          and d = %*z*. *if z*=*A*+*B then Inl* (*A*) *else z* **in** *lam_bijective*)
  apply *simp_all*
    apply (*blast dest*: *sym* [*THEN eq_imp_not_mem*] *elim*: *mem_irrefl*)+
 apply(*rule_tac lam_closed*, *auto dest*:*transM intro*:*if_then_range_replacement2*)
done

lemma *cadd_succ_lemma*:
  assumes *Ord*(*m*) *Ord*(*n*) **and**
    types: *M*(*m*) *M*(*n*)
  shows *succ*(*m*) $\oplus^M$ *n* = |*succ*(*m* $\oplus^M$ *n*)|^M
  using *types*
proof (*simp add*: *cadd_rel_def*)
  have [*intro*]: *m* + *n* $\approx^M$ |*m* + *n*|^M **using** *assms*
   by (*blast intro*: *eqpoll_rel_sym well_ord_cardinal_rel_eqpoll_rel well_ord_radd well_ord_Memrel*)

  have |*succ*(*m*) + *n*|^M = |*succ*(*m* + *n*)|^M
   by (*rule sum_succ_eqpoll_rel* [*THEN cardinal_rel_cong*]) (*simp_all add*:*types*)
  also have ... = |*succ*(|*m* + *n*|^M)|^M
    by (*blast intro*: *succ_eqpoll_rel_cong cardinal_rel_cong types*)
  finally show |*succ*(*m*) + *n*|^M = |*succ*(|*m* + *n*|^M)|^M **.**
qed

lemma *nat_cadd_rel_eq_add*:
  assumes *m*: *m* ∈ *nat* **and** [*simp*]: *n* ∈ *nat* **shows***m* $\oplus^M$ *n* = *m* #+ *n*
  using *m*
proof (*induct m*)
  case *0* **thus** *?case*
    using *transM*[*OF _ M_nat*]
    by (*auto simp add*: *nat_into_Card_rel*)
next
  case (*succ m*) **thus** *?case*
    using *transM*[*OF _ M_nat*]
   by (*simp add*: *cadd_succ_lemma nat_into_Card_rel Card_rel_cardinal_rel_eq*)
qed

23.2 Cardinal multiplication

23.2.1 Cardinal multiplication is commutative

lemma *prod_commute_eqpoll_rel*: *M*(*A*) ⟹ *M*(*B*) ⟹ *A*∗*B* $\approx^M$ *B*∗*A*
apply (*simp add*: *def_eqpoll_rel*)
apply (*rule rexI*)
apply (*rule_tac c* = %<*x*,*y*>.<*y*,*x*> **and** *d* = %<*x*,*y*>.<*y*,*x*> **in** *lam_bijective*,
      auto)
  apply(*rule_tac lam_closed*, *auto intro*:*swap_replacement dest*:*transM*)
done

lemma *cmult_rel_commute*: *M*(*i*) ⟹ *M*(*j*) ⟹ *i* $\otimes^M$ *j* = *j* $\otimes^M$ *i*
apply (*unfold cmult_rel_def*)
  apply (*rule prod_commute_eqpoll_rel* [*THEN cardinal_rel_cong*], *simp_all*)
done

23.2.2 Cardinal multiplication is associative

lemma *prod_assoc_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (A*B)*C \approx^M A*(B*C)$
apply (*simp add*: *def_eqpoll_rel*)
apply (*rule rexI*)
apply (*rule prod_assoc_bij*)
apply(*rule_tac lam_closed*, *auto intro*:*assoc_replacement dest*:*transM*)
done

Unconditional version requires AC

lemma *well_ord_cmult_rel_assoc*:
assumes *i*: *well_ord*(*i*,*ri*) **and** *j*: *well_ord*(*j*,*rj*) **and** *k*: *well_ord*(*k*,*rk*)
and
types: $M(i)$ $M(ri)$ $M(j)$ $M(rj)$ $M(k)$ $M(rk)$
shows $(i \otimes^M j) \otimes^M k = i \otimes^M (j \otimes^M k)$
proof (*simp add*: *assms cmult_rel_def*, *rule cardinal_rel_cong*)
have $|i * j|^M * k \approx^M (i * j) * k$
by (*auto intro*!: *prod_eqpoll_rel_cong*
well_ord_cardinal_rel_eqpoll_rel eqpoll_rel_refl
well_ord_rmult i j simp add:*types*)
also have ... $\approx^M i * (j * k)$
by (*rule prod_assoc_eqpoll_rel*, *simp_all add*:*types*)
also have ... $\approx^M i * |j * k|^M$
by (*blast intro*: *prod_eqpoll_rel_cong well_ord_cardinal_rel_eqpoll_rel*
eqpoll_rel_refl well_ord_rmult j k eqpoll_rel_sym types)
finally show $|i * j|^M * k \approx^M i * |j * k|^M$ **by** (*simp add*:*types*)
qed (*simp_all add*:*types*)

23.2.3 Cardinal multiplication distributes over addition

lemma *sum_prod_distrib_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (A+B)*C \approx^M (A*C)+(B*C)$
apply (*simp add*: *def_eqpoll_rel*)
apply (*rule rexI*)
apply (*rule sum_prod_distrib_bij*)
apply(*rule_tac lam_closed*, *auto intro*:*case_replacement5 dest*:*transM*)
done

lemma *well_ord_cadd_cmult_distrib*:
assumes *i*: *well_ord*(*i*,*ri*) **and** *j*: *well_ord*(*j*,*rj*) **and** *k*: *well_ord*(*k*,*rk*)
and
types: $M(i)$ $M(ri)$ $M(j)$ $M(rj)$ $M(k)$ $M(rk)$
shows $(i \oplus^M j) \otimes^M k = (i \otimes^M k) \oplus^M (j \otimes^M k)$
proof (*simp add*: *assms cadd_rel_def cmult_rel_def*, *rule cardinal_rel_cong*)
have $|i + j|^M * k \approx^M (i + j) * k$
by (*blast intro*: *prod_eqpoll_rel_cong well_ord_cardinal_rel_eqpoll_rel*
eqpoll_rel_refl well_ord_radd i j types)
also have ... $\approx^M i * k + j * k$

by (*rule sum_prod_distrib_eqpoll_rel*) (*simp_all add*:*types*)
also have ... $\approx^M |i * k|^M + |j * k|^M$
by (*blast intro*: *sum_eqpoll_rel_cong well_ord_cardinal_rel_eqpoll_rel well_ord_rmult i j k eqpoll_rel_sym types*)
finally show $|i + j|^M * k \approx^M |i * k|^M + |j * k|^M$ **by** (*simp add*:*types*)
qed (*simp_all add*:*types*)

23.2.4 Multiplication by 0 yields 0

lemma *prod_0_eqpoll_rel*: $M(A) \Longrightarrow 0*A \approx^M 0$
apply (*simp add*: *def_eqpoll_rel*)
apply (*rule rexI*)
apply (*rule lam_bijective*, *auto*)
done

lemma *cmult_rel_0* [*simp*]: $M(i) \Longrightarrow 0 \otimes^M i = 0$
by (*simp add*: *cmult_rel_def prod_0_eqpoll_rel* [*THEN cardinal_rel_cong*])

23.2.5 1 is the identity for multiplication

lemma *prod_singleton_eqpoll_rel*: $M(x) \Longrightarrow M(A) \Longrightarrow \{x\}*A \approx^M A$
apply (*simp add*: *def_eqpoll_rel*)
apply (*rule rexI*)
apply (*rule singleton_prod_bij* [*THEN bij_converse_bij*])
apply (*rule converse_closed*)
apply(*rule_tac lam_closed*, *auto intro*:*prepend_replacement dest*:*transM*)
done

lemma *cmult_rel_1* [*simp*]: $Card^M(K) \Longrightarrow M(K) \Longrightarrow 1 \otimes^M K = K$
apply (*simp add*: *cmult_rel_def succ_def*)
apply (*simp add*: *prod_singleton_eqpoll_rel*[*THEN cardinal_rel_cong*] *Card_rel_cardinal_rel_eq*)
done

23.3 Some inequalities for multiplication

lemma *prod_square_lepoll_rel*: $M(A) \Longrightarrow A \lesssim^M A*A$
apply (*simp add*:*def_lepoll_rel inj_def*)
apply (*rule_tac* $x = \lambda x \in A.\ \langle x,x\rangle$ **in** *rexI*, *simp*)
apply(*rule_tac lam_closed*, *auto intro*:*id_replacement dest*:*transM*)
done

lemma *cmult_rel_square_le*: $Card^M(K) \Longrightarrow M(K) \Longrightarrow K \le K \otimes^M K$
apply (*unfold cmult_rel_def*)
apply (*rule le_trans*)
apply (*rule_tac* [*2*] *well_ord_lepoll_rel_imp_cardinal_rel_le*)
apply (*rule_tac* [*3*] *prod_square_lepoll_rel*)
apply (*simp add*: *le_refl Card_rel_is_Ord Card_rel_cardinal_rel_eq*)
apply (*blast intro*: *well_ord_rmult well_ord_Memrel Card_rel_is_Ord*)
apply *simp_all*

done

23.3.1 Multiplication by a non-zero cardinal

lemma *prod_lepoll_rel_self*: $b \in B \Longrightarrow M(b) \Longrightarrow M(B) \Longrightarrow M(A) \Longrightarrow A \lesssim^M A*B$
apply (*simp add*: *def_lepoll_rel inj_def*)
apply (*rule_tac* $x = \lambda x \in A.\ <x,b>$ **in** *rexI*, *simp*)
apply(*rule_tac lam_closed*, *auto intro*:*pospend_replacement dest*:*transM*)
done

lemma *cmult_rel_le_self*:
$[|\ Card^M(K);\ \ Ord(L);\ \ 0<L;\ M(K);M(L)\ |] ==> K \le (K \otimes^M L)$
apply (*unfold cmult_rel_def*)
apply (*rule le_trans* [*OF Card_rel_cardinal_rel_le well_ord_lepoll_rel_imp_cardinal_rel_le*])
apply *assumption* **apply** *simp*
apply (*blast intro*: *well_ord_rmult well_ord_Memrel Card_rel_is_Ord*)
apply (*auto intro*: *prod_lepoll_rel_self ltD*)
done

23.3.2 Monotonicity of multiplication

lemma *prod_lepoll_rel_mono*:
$[|\ A \lesssim^M C;\ \ B \lesssim^M D;\ M(A);\ M(B);\ M(C);\ M(D)|] ==> A * B \lesssim^M C * D$
apply (*simp add*:*def_lepoll_rel*)
apply (*elim rexE*)
apply (*rule_tac* $x = lam <w,y>:A*B.\ <f`w,\ fa`y>$ **in** *rexI*)
apply (*rule_tac* $d = \%<w,y>.\ <converse\ (f)\ `w,\ converse\ (fa)\ `y>$
in *lam_injective*)
apply (*typecheck add*: *inj_is_fun*, *auto*)
apply(*rule_tac lam_closed*, *auto intro*:*prod_fun_replacement dest*:*transM*)
done

lemma *cmult_rel_le_mono*:
$[|\ K' \le K;\ \ L' \le L;M(K');M(K);M(L');M(L)\ |] ==> (K' \otimes^M L') \le (K \otimes^M L)$
apply (*unfold cmult_rel_def*)
apply (*safe dest*!: *le_subset_iff* [*THEN iffD1*])
apply (*rule well_ord_lepoll_rel_imp_cardinal_rel_le*)
apply (*blast intro*: *well_ord_rmult well_ord_Memrel*)
apply (*auto intro*: *prod_lepoll_rel_mono subset_imp_lepoll_rel*)
done

23.4 Multiplication of finite cardinals is "ordinary" multiplication

lemma *prod_succ_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow succ(A)*B \approx^M B + A*B$
apply (*simp add*: *def_eqpoll_rel*)
apply (*rule rexI*)
apply (*rule_tac* $c = \lambda p.$ *if* $fst(p)=A$ *then Inl* $(snd(p))$ *else Inr* (p)

```
          and d = case (%y. <A,y>, %z. z) in lam_bijective)
apply safe
        apply (simp_all add: succI2 if_type mem_imp_not_eq)
  apply(rule_tac lam_closed, auto intro:Inl_replacement2 dest:transM)
done


lemma cmult_rel_succ_lemma:
    [| Ord(m);  Ord(n) ; M(m); M(n) |] ==> succ(m) ⊗^M n = n ⊕^M (m ⊗^M n)
apply (simp add: cmult_rel_def cadd_rel_def)
apply (rule prod_succ_eqpoll_rel [THEN cardinal_rel_cong, THEN trans], simp_all)
apply (rule cardinal_rel_cong [symmetric], simp_all)
apply (rule sum_eqpoll_rel_cong [OF eqpoll_rel_refl well_ord_cardinal_rel_eqpoll_rel],
assumption)
       apply (blast intro: well_ord_rmult well_ord_Memrel)
  apply simp_all
done

lemma nat_cmult_rel_eq_mult: [| m ∈ nat;  n ∈ nat |] ==> m ⊗^M n = m#*n
  using transM[OF _ M_nat]
apply (induct_tac m)
apply (simp_all add: cmult_rel_succ_lemma nat_cadd_rel_eq_add)
done

lemma cmult_rel_2: Card^M(n) ⟹ M(n) ⟹ 2 ⊗^M n = n ⊕^M n
  by (simp add: cmult_rel_succ_lemma Card_rel_is_Ord cadd_rel_commute [of
_ 0])

lemma sum_lepoll_rel_prod:
  assumes C: 2 ≲^M C and
    types:M(C) M(B)
  shows B+B ≲^M C*B
proof -
  have B+B ≲^M 2*B
    by (simp add: sum_eq_2_times types)
  also have ... ≲^M C*B
    by (blast intro: prod_lepoll_rel_mono lepoll_rel_refl C types)
  finally show B+B ≲^M C*B by (simp_all add:types)
qed

lemma lepoll_imp_sum_lepoll_prod: [| A ≲^M B; 2 ≲^M A; M(A) ;M(B) |] ==>
A+B ≲^M A*B
by (blast intro: sum_lepoll_rel_mono sum_lepoll_rel_prod lepoll_rel_trans lepoll_rel_refl)

end
```

23.5 Infinite Cardinals are Limit Ordinals

```
context M_pre_cardinal_arith
```

begin

lemma *nat_cons_lepoll_rel*: $nat \lesssim^M A \Longrightarrow M(A) \Longrightarrow M(u) ==> cons(u,A) \lesssim^M A$
apply (*simp add*: *def_lepoll_rel*)
apply (*erule rexE*)
apply (*rule_tac x* =
λ*z*∈*cons* (*u*,*A*).
if z=*u then f*‘*0*
else if z ∈ *range* (*f*) *then f*‘*succ* (*converse* (*f*) ‘*z*) *else z*
in *rexI*)
apply (*rule_tac d* =
%*y*. *if y* ∈ *range*(*f*) *then nat_case* (*u*, %*z*. *f*‘*z*, *converse*(*f*) ‘*y*)
else y
in *lam_injective*)
apply (*fast intro*!: *if_type apply_type intro*: *inj_is_fun inj_converse_fun*)
apply (*simp add*: *inj_is_fun* [*THEN apply_rangeI*]
inj_converse_fun [*THEN apply_rangeI*]
inj_converse_fun [*THEN apply_funtype*])
proof -
fix *f*
assume *M*(*A*) *M*(*f*) *M*(*u*)
then
show *M*(λ*z*∈*cons*(*u*, *A*). *if z* = *u then f* ‘ *0 else if z* ∈ *range*(*f*) *then f* ‘ *succ*(*converse*(*f*) ‘ *z*) *else z*)
using *if_then_range_replacement transM*[*OF* _ ‹*M*(*A*)›]
by (*rule_tac lam_closed*, *auto*)
qed

lemma *nat_cons_eqpoll_rel*: $nat \lesssim^M A ==> M(A) \Longrightarrow M(u) \Longrightarrow cons(u,A) \approx^M A$
apply (*erule nat_cons_lepoll_rel* [*THEN eqpoll_relI*], *assumption*+)
apply (*rule subset_consI* [*THEN subset_imp_lepoll_rel*], *simp_all*)
done

lemma *nat_succ_eqpoll_rel*: $nat \subseteq A ==> M(A) \Longrightarrow succ(A) \approx^M A$
apply (*unfold succ_def*)
apply (*erule subset_imp_lepoll_rel* [*THEN nat_cons_eqpoll_rel*], *simp_all*)
done

lemma *InfCard_rel_nat*: $InfCard^M(nat)$
apply (*simp add*: *InfCard_rel_def*)
apply (*blast intro*: *Card_rel_nat Card_rel_is_Ord*)
done

lemma *InfCard_rel_is_Card_rel*: $M(K) \Longrightarrow InfCard^M(K) \Longrightarrow Card^M(K)$
apply (*simp add*: *InfCard_rel_def*)
done

lemma *InfCard_rel_Un*:
[| $InfCard^M(K)$; $Card^M(L)$; $M(K)$; $M(L)$ |] ==> $InfCard^M(K \cup L)$
apply (*simp add*: *InfCard_rel_def*)
apply (*simp add*: *Card_rel_Un Un_upper1_le* [*THEN* [*2*] *le_trans*] *Card_rel_is_Ord*)
done

lemma *InfCard_rel_is_Limit*: $InfCard^M(K)$ ==> $M(K) \Longrightarrow Limit(K)$
apply (*simp add*: *InfCard_rel_def*)
apply (*erule conjE*)
apply (*frule Card_rel_is_Ord*, *assumption*)
apply (*rule ltI* [*THEN non_succ_LimitI*])
apply (*erule le_imp_subset* [*THEN subsetD*])
apply (*safe dest*!: *Limit_nat* [*THEN Limit_le_succD*])
apply (*unfold Card_rel_def*)
apply (*drule trans*)
apply (*erule le_imp_subset* [*THEN nat_succ_eqpoll_rel*, *THEN cardinal_rel_cong*], *simp_all*)
apply (*erule Ord_cardinal_rel_le* [*THEN lt_trans2*, *THEN lt_irrefl*], *assumption*)
apply (*rule le_eqI*) **prefer** *2*
apply (*rule Ord_cardinal_rel*, *assumption*+)
done

end

— FIXME: Awful proof, it essentially repeats the same argument twice
lemma (**in** *M_ordertype*) *ordertype_abs*[*absolut*]:
[| *wellordered*(*M*,*A*,*r*); $M(A)$; $M(r)$; $M(i)$|] ==>
otype(*M*,*A*,*r*,*i*) $\longleftrightarrow$ i = *ordertype*(*A*,*r*)
proof (*intro iffI*)
assume *wellordered*(M, A, r) $M(A)$ $M(r)$ $M(i)$ *otype*(M, A, r, i)
moreover from *this*
obtain f j **where** $M(f)$ $M(j)$ *Ord*(j) $f \in \langle A, r\rangle \cong \langle j, Memrel(j)\rangle$
using *ordertype_exists*[*of A r*] **by** *auto*
moreover from *calculation*
have $\exists f[M].\ f \in \langle A, r\rangle \cong \langle j, Memrel(j)\rangle$ **by** *auto*
moreover
have $\exists f[M].\ f \in \langle A, r\rangle \cong \langle i, Memrel(i)\rangle$
proof -
note *calculation*
moreover from *this*
obtain h **where** *omap*(M, A, r, h) $M(h)$
using *omap_exists* **by** *auto*
moreover from *calculation*
have $h \in \langle A, r\rangle \cong \langle i, Memrel(i)\rangle$
using *omap_ord_iso obase_equals* **by** *simp*
moreover from *calculation*
have h O *converse*(f) $\in \langle j, Memrel(j)\rangle \cong \langle i, Memrel(i)\rangle$

```
      using ord_iso_sym ord_iso_trans by blast
    moreover from calculation
    have i=j
      using Ord_iso_implies_eq[of j i h O converse(f)]
        Ord_otype[OF _ well_ord_is_trans_on] by simp
    ultimately
    show ?thesis by simp
  qed
  ultimately
  show i = ordertype(A, r)
    by (force intro:ordertypes_are_absolute[of A r _ i]
        simp add:Ord_otype[OF _ well_ord_is_trans_on])
next
  assume wellordered(M,A, r) i = ordertype(A, r)
    M(i) M(A) M(r)
  moreover from this
  obtain h where omap(M, A, r, h) M(h)
    using omap_exists by auto
  moreover from calculation
  obtain j where otype(M,A,r,j) M(j)
    using otype_exists by auto
  moreover from calculation
  have h ∈ ⟨A, r⟩ ≅ ⟨j, Memrel(j)⟩
    using omap_ord_iso_otype by simp
  moreover from calculation
  obtain f where f ∈ ⟨A, r⟩ ≅ ⟨i, Memrel(i)⟩
    using ordertype_ord_iso by auto
  moreover
  have j=i
  proof -
    note calculation
    moreover from this
    have h O converse(f) ∈ ⟨i, Memrel(i)⟩ ≅ ⟨j, Memrel(j)⟩
      using ord_iso_sym ord_iso_trans by blast
    moreover from calculation
    have Ord(i) using Ord_ordertype by simp
    ultimately
    show j=i
      using Ord_iso_implies_eq[of i j h O converse(f)]
        Ord_otype[OF _ well_ord_is_trans_on] by simp
  qed
  ultimately
  show otype(M, A, r, i) by simp
qed

lemma (in M_ordertype) ordertype_closed[intro,simp]: ⟦ wellordered(M,A,r);M(A);M(r)⟧
⟹ M(ordertype(A,r))
  using ordertype_exists ordertypes_are_absolute by blast
```

relationalize *transitive_rel is_transitive* **external**
synthesize *is_transitive* **from_definition assuming** *nonempty*

lemma (**in** *M_trivial*) *is_transitive_iff_transitive_rel*:
M(*A*)⟹ *M*(*r*) ⟹ *transitive_rel*(*M*, *A*, *r*) ⟷ *is_transitive*(*M*,*A*, *r*)
unfolding *transitive_rel_def is_transitive_def* **by** *simp*

relationalize *linear_rel is_linear* **external**
synthesize *is_linear* **from_definition assuming** *nonempty*

lemma (**in** *M_trivial*) *is_linear_iff_linear_rel*:
M(*A*)⟹ *M*(*r*) ⟹ *is_linear*(*M*,*A*, *r*) ⟷ *linear_rel*(*M*, *A*, *r*)
unfolding *linear_rel_def is_linear_def* **by** *simp*

relationalize *wellfounded_on is_wellfounded_on* **external**
synthesize *is_wellfounded_on* **from_definition assuming** *nonempty*

lemma (**in** *M_trivial*) *is_wellfounded_on_iff_wellfounded_on*:
M(*A*)⟹ *M*(*r*) ⟹ *is_wellfounded_on*(*M*,*A*, *r*) ⟷ *wellfounded_on*(*M*, *A*, *r*)
unfolding *wellfounded_on_def is_wellfounded_on_def* **by** *simp*

definition
is_well_ord :: [*i*=>*o*,*i*,*i*]=>*o* **where**
— linear and wellfounded on *A*
is_well_ord(*M*,*A*,*r*) ==
is_transitive(*M*,*A*,*r*) ∧ *is_linear*(*M*,*A*,*r*) ∧ *is_wellfounded_on*(*M*,*A*,*r*)

lemma (**in** *M_trivial*) *is_well_ord_iff_wellordered*:
M(*A*)⟹ *M*(*r*) ⟹ *is_well_ord*(*M*,*A*, *r*) ⟷ *wellordered*(*M*, *A*, *r*)
using *is_wellfounded_on_iff_wellfounded_on is_linear_iff_linear_rel*
is_transitive_iff_transitive_rel
unfolding *wellordered_def is_well_ord_def* **by** *simp*

reldb_add relational *well_ord is_well_ord*
reldb_add functional *well_ord well_ord*
synthesize *is_well_ord* **from_definition assuming** *nonempty*

— One keyword (functional or relational) means going from an absolute term to that kind of term
reldb_add relational *Order.pred pred_set*

— The following form (twice the same argument) is only correct when an "_abs" theorem is available
reldb_add functional *Order.pred Order.pred*
reldb_add functional *Ord Ord*

relativize functional *ord_iso ord_iso_rel* **external**
— The following corresponds to "relativize functional relational"
relationalize *ord_iso_rel is_ord_iso*

context *M_pre_cardinal_arith*
begin

is_iff_rel for *ord_iso*
  using *bij_rel_iff*
  unfolding *is_ord_iso_def ord_iso_rel_def*
  by *simp*

rel_closed for *ord_iso*
  using *ord_iso_separation* **unfolding** *ord_iso_rel_def*
  by *simp*

end

synthesize *is_ord_iso* **from_definition assuming** *nonempty*

lemma *is_lambda_iff_sats*[*iff_sats*]:
  assumes *is_F_iff_sats*:
    !!*a0 a1 a2*.
      [|*a0*∈*Aa*; *a1*∈*Aa*; *a2*∈*Aa*|]
     ==> *is_F*(*a1*, *a0*) ⟷ *sats*(*Aa*, *is_F_fm*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*env*))))
  shows
    nth(*A*, *env*) = *Ab* ⟹
    nth(*r*, *env*) = *ra* ⟹
    A ∈ *nat* ⟹
    r ∈ *nat* ⟹
    env ∈ *list*(*Aa*) ⟹
    is_lambda(##*Aa*, *Ab*, *is_F*, *ra*) ⟷ *Aa*, *env* ⊨ *lambda_fm*(*is_F_fm*,*A*, *r*)
  using *sats_lambda_fm*[*OF assms*, *of A r*] **by** *simp*

— same as ⟦⋀*a0 a1 a2 a3 a4*. ⟦*a0* ∈ *?A*; *a1* ∈ *?A*; *a2* ∈ *?A*; *a3* ∈ *?A*; *a4* ∈ *?A*⟧
⟹ *?MH*(*a2*, *a1*, *a0*) ⟷ *?A*, *Cons*(*a0*, *Cons*(*a1*, *Cons*(*a2*, *Cons*(*a3*, *Cons*(*a4*,
?env))))) ⊨ *?p*; *?x* ∈ *nat*; *?y* < *length*(*?env*); *?z* < *length*(*?env*); *?env* ∈ *list*(*?A*)⟧
⟹ (*?A*, *?env* ⊨ *is_wfrec_fm*(*?p*, *?x*, *?y*, *?z*)) ⟷ *is_wfrec*(##*?A*, *?MH*, *nth*(*?x*,
?env), *nth*(*?y*, *?env*), *nth*(*?z*, *?env*)), but changing length assumptions to *0* being
in the model
lemma *sats_is_wfrec_fm′*:
  assumes *MH_iff_sats*:
    !!*a0 a1 a2 a3 a4*.
      [|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*; *a3*∈*A*; *a4*∈*A*|]
     ==> *MH*(*a2*, *a1*, *a0*) ⟷ *sats*(*A*, *p*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*Cons*(*a3*,*Cons*(*a4*,*env*))))))
  shows
    [|*x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*); *0* ∈ *A*|]
      ==> *sats*(*A*, *is_wfrec_fm*(*p*,*x*,*y*,*z*), *env*) ⟷

is_wfrec(*##A*, *MH*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
using *MH_iff_sats* [*THEN iff_sym*] *nth_closed sats_is_recfun_fm*
by (*simp add*: *is_wfrec_fm_def is_wfrec_def*) *blast*

lemma *is_wfrec_iff_sats'*[*iff_sats*]:
assumes *MH_iff_sats*:
!!*a0 a1 a2 a3 a4*.
[|*a0*∈*Aa*; *a1*∈*Aa*; *a2*∈*Aa*; *a3*∈*Aa*; *a4*∈*Aa*|]
==> *MH*(*a2*, *a1*, *a0*) ⟷ *sats*(*Aa*, *p*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*Cons*(*a3*,*Cons*(*a4*,*env*))))))
x ∈ *nat y* ∈ *nat z* ∈ *nat env* ∈ *list*(*Aa*) *0* ∈ *Aa*
nth(*x*, *env*) = *xx nth*(*y*, *env*) = *yy nth*(*z*, *env*) = *zz*
shows
is_wfrec(*##Aa*, *MH*, *xx*, *yy*, *zz*) ⟷ *Aa*, *env* ⊨ *is_wfrec_fm*(*p*,*x*,*y*,*z*)
using *assms*(*7-9*) *sats_is_wfrec_fm'*[*OF assms*(*1-6*)] **by** *simp*

lemma *is_wfrec_on_iff_sats*[*iff_sats*]:
assumes *MH_iff_sats*:
!!*a0 a1 a2 a3 a4*.
[|*a0*∈*Aa*; *a1*∈*Aa*; *a2*∈*Aa*; *a3*∈*Aa*; *a4*∈*Aa*|]
==> *MH*(*a2*, *a1*, *a0*) ⟷ *sats*(*Aa*, *p*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*Cons*(*a3*,*Cons*(*a4*,*env*))))))
shows
nth(*x*, *env*) = *xx* ⟹
nth(*y*, *env*) = *yy* ⟹
nth(*z*, *env*) = *zz* ⟹
x ∈ *nat* ⟹
y ∈ *nat* ⟹
z ∈ *nat* ⟹
env ∈ *list*(*Aa*) ⟹
0 ∈ *Aa* ⟹ *is_wfrec_on*(*##Aa*, *MH*, *aa*,*xx*, *yy*, *zz*) ⟷ *Aa*, *env* ⊨ *is_wfrec_fm*(*p*,*x*,*y*,*z*)
using *assms sats_is_wfrec_fm'*[*OF assms*] **unfolding** *is_wfrec_on_def* **by** *simp*

lemma *trans_on_iff_trans*: *trans*[*A*](*r*) ⟷ *trans*(*r* ∩ *A*×*A*)
unfolding *trans_on_def trans_def* **by** *auto*

lemma *trans_on_subset*: *trans*[*A*](*r*) ⟹ *B* ⊆ *A* ⟹ *trans*[*B*](*r*)
unfolding *trans_on_def*
by *auto*

lemma *relation_Int*: *relation*(*r* ∩ *B*×*B*)
unfolding *relation_def*
by *auto*

Discipline for *ordermap*

relativize functional *ordermap ordermap_rel* **external**
relationalize *ordermap_rel is_ordermap*

context *M_pre_cardinal_arith*
begin

lemma *wfrec_on_pred_eq*:
assumes $r \in Pow(A\times A)$ $M(A)$ $M(r)$
shows $wfrec[A](r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r)) = wfrec(r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r))$
proof -
from ‹$r \in Pow(A\times A)$›
have $r \cap A\times A = r$ **by** *auto*
moreover from *this*
show *?thesis*
unfolding *wfrec_on_def* **by** *simp*
qed

lemma *wfrec_on_pred_closed*:
assumes $wf[A](r)$ $trans[A](r)$ $r \in Pow(A\times A)$ $M(A)$ $M(r)$ $x \in A$
shows $M(wfrec(r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r)))$
proof -
from *assms*
have $wfrec[A](r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r)) = wfrec(r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r))$
using *wfrec_on_pred_eq* **by** *simp*
moreover from *assms*
have $M(wfrec(r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r)))$
using *wfrec_pred_replacement wf_on_imp_wf trans_on_imp_trans subset_Sigma_imp_relation*
by (*rule_tac MH*=$\lambda x\ f\ b.\ \exists\ a[M].\ image(M,\ f,\ a,\ b) \wedge pred\_set(M,\ A,\ x,\ r,\ a)$
in *trans_wfrec_closed*)
(*auto dest*:*transM simp*:*relation2_def*)
ultimately
show *?thesis* **by** *simp*
qed

lemma *wfrec_on_pred_closed'*:
assumes $wf[A](r)$ $trans[A](r)$ $r \in Pow(A\times A)$ $M(A)$ $M(r)$ $x \in A$
shows $M(wfrec[A](r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r)))$
using *assms wfrec_on_pred_closed wfrec_on_pred_eq* **by** *simp*

lemma *ordermap_rel_closed'*:
assumes $wf[A](r)$ $trans[A](r)$ $r \in Pow(A\times A)$ $M(A)$ $M(r)$
shows $M(ordermap\_rel(M,\ A,\ r))$
proof -
from *assms*
have $r \cap A\times A = r$ **by** *auto*
with *assms* **have** $wf(r)$ $trans(r)$ $relation(r)$
unfolding *wf_on_def* **using** *trans_on_iff_trans relation_def* **by** *auto*
then
have *1*:$\bigwedge\ x\ z\ .\ M(x) \Longrightarrow M(z) \Longrightarrow$
$(\exists\ y[M].\ pair(M,\ x,\ y,\ z) \wedge is\_wfrec(M,\ \lambda x\ f\ z.\ z = f\ ``\ Order.pred(A,\ x,\ r),\ r,\ x,\ y))$
$\longleftrightarrow$

```
    z = <x,wfrec(r,x,λx f. f `` Order.pred(A, x, r))>
    using trans_wfrec_abs[of r,where
        H=λx f. f `` Order.pred(A, x, r) and
        MH=λx f z . z= f `` Order.pred(A, x, r),simplified] assms
      wfrec_pred_replacement unfolding relation2_def
    by auto
  then
  have strong_replacement(M,λx z. z = <x,wfrec(r,x,λx f. f `` Order.pred(A, x,
r))>)
    using strong_replacement_cong[of M,OF 1,THEN iffD1,OF _ _
      wfrec_pred_replacement[unfolded wfrec_replacement_def]] assms by simp
  then show ?thesis
    using Pow_iff assms
    unfolding ordermap_rel_def
    apply(subst lam_cong[OF refl wfrec_on_pred_eq],simp_all)
    using wfrec_on_pred_closed lam_closed
    by simp
qed

lemma ordermap_rel_closed[intro,simp]:
  assumes wf[A](r) trans[A](r) r ∈ Pow(A×A)
  shows M(A) ⟹ M(r) ⟹ M(ordermap_rel(M, A, r))
  using ordermap_rel_closed' assms by simp

lemma is_ordermap_iff:
  assumes r ∈ Pow(A×A) wf[A](r) trans[A](r)
    M(A) M(r) M(res)
  shows is_ordermap(M, A, r, res) ⟷ res = ordermap_rel(M, A, r)
proof -
  from ‹r ∈ Pow(A×A)›
  have r ∩ A×A = r by auto
  with assms have 1:wf(r) trans(r) relation(r)
    unfolding wf_on_def using trans_on_iff_trans relation_def by auto
  from assms
  have r ∩ A×A = r r ⊆ A×A <x,y> ∈ r ⟹ x∈A ∧ y∈A for x y by auto
  then
  show ?thesis
 using ordermap_rel_closed[of r A] assms wfrec_on_pred_closed wfrec_pred_replacement
1
  unfolding is_ordermap_def ordermap_rel_def
  apply (rule_tac lambda_abs2)
     apply (simp_all add:Relation1_def)
  apply clarify
  apply (rule trans_wfrec_on_abs)
           apply (auto dest:transM simp add: relation_Int relation2_def)
  by(rule_tac wfrec_on_pred_closed'[of A r],auto)
qed

end
```

synthesize *is_ordermap* **from_definition assuming** *nonempty*

Discipline for *ordertype*

relativize functional *ordertype ordertype_rel* **external**
relationalize *ordertype_rel is_ordertype*

context *M_pre_cardinal_arith*
begin

lemma *is_ordertype_iff*:
assumes $r \in Pow(A\times A)$ *wf*[*A*](*r*) *trans*[*A*](*r*)
shows $M(A) \Longrightarrow M(r) \Longrightarrow M(res) \Longrightarrow is\_ordertype(M,\ A,\ r,\ res) \longleftrightarrow res = ordertype\_rel(M,\ A,\ r)$
using *assms is_ordermap_iff*[*of r A*] *trans_on_iff_trans ordermap_rel_closed*[*of A r*]
unfolding *is_ordertype_def ordertype_rel_def wf_on_def* **by** *simp*

lemma *is_ordertype_iff′*:
assumes $r \in Pow\_rel(M,A\times A)$ *well_ord*(*A*,*r*)
shows $M(A) \Longrightarrow M(r) \Longrightarrow M(res) \Longrightarrow is\_ordertype(M,\ A,\ r,\ res) \longleftrightarrow res = ordertype\_rel(M,\ A,\ r)$
using *assms is_ordertype_iff Pow_rel_char*
unfolding *well_ord_def part_ord_def tot_ord_def* **by** *simp*

lemma *is_ordertype_iff′′*:
assumes *well_ord*(*A*,*r*) $r\subseteq A\times A$
shows $M(A) \Longrightarrow M(r) \Longrightarrow M(res) \Longrightarrow is\_ordertype(M,\ A,\ r,\ res) \longleftrightarrow res = ordertype\_rel(M,\ A,\ r)$
using *assms is_ordertype_iff*
unfolding *well_ord_def part_ord_def tot_ord_def* **by** *simp*

end

synthesize *is_ordertype* **from_definition assuming** *nonempty*

— NOTE: not quite the same as *jump_cardinal*, note $Pow(X \times X)$.
definition
jump_cardinal′ :: $i\Rightarrow i$ **where**
jump_cardinal′(*K*) ≡
$\bigcup X\in Pow(K).\ \{z.\ r \in Pow(X*X),\ well\_ord(X,r)\ \&\ z = ordertype(X,r)\}$

relativize functional *jump_cardinal′ jump_cardinal′_rel* **external**
relationalize *jump_cardinal′_rel is_jump_cardinal′*
synthesize *is_jump_cardinal′* **from_definition assuming** *nonempty*
arity_theorem for *is_jump_cardinal′_fm*
definition *jump_cardinal_body′* **where**
$jump\_cardinal\_body'(X) \equiv \{z\ .\ r \in Pow(X \times X),\quad well\_ord(X,\ r) \wedge z = ordertype(X,\ r)\}$

relativize functional *jump_cardinal_body′ jump_cardinal_body′_rel* **external**
relationalize *jump_cardinal_body′_rel is_jump_cardinal_body′*
synthesize *is_jump_cardinal_body′* **from_definition assuming** *nonempty*
arity_theorem for *is_jump_cardinal_body′_fm*

context *M_pre_cardinal_arith*
begin

lemma *ordertype_rel_closed′*:
  assumes *wf*[*A*](*r*) *trans*[*A*](*r*) *r* ∈ *Pow*(*A*×*A*) *M*(*r*) *M*(*A*)
  shows *M*(*ordertype_rel*(*M*,*A*,*r*))
    unfolding *ordertype_rel_def*
  using *ordermap_rel_closed image_closed assms* **by** *simp*

lemma *ordertype_rel_closed*[*intro*,*simp*]:
  assumes *well_ord*(*A*,*r*) *r* ∈ *Pow_rel*(*M*,*A*×*A*) *M*(*A*)
  shows *M*(*ordertype_rel*(*M*,*A*,*r*))
    using *assms Pow_rel_char ordertype_rel_closed′*
    unfolding *well_ord_def tot_ord_def part_ord_def*
    by *simp*

lemma *ordertype_rel_abs*:
  assumes *wellordered*(*M*,*X*,*r*) *M*(*X*) *M*(*r*)
  shows *ordertype_rel*(*M*,*X*,*r*) = *ordertype*(*X*,*r*)
  using *assms ordertypes_are_absolute*[*of X r*]
  unfolding *ordertype_def ordertype_rel_def ordermap_rel_def ordermap_def*
  by *simp*

lemma *univalent_aux1*: *M*(*X*) ⟹ *univalent*(*M*,*Pow_rel*(*M*,*X*×*X*),
 λ*r z*. *M*(*z*) ∧ *M*(*r*) ∧ *r*∈*Pow_rel*(*M*,*X*×*X*) ∧ *is_well_ord*(*M*, *X*, *r*) ∧ *is_ordertype*(*M*,
X, *r*, *z*))
  using *is_well_ord_iff_wellordered*
    is_ordertype_iff[*of _ X*]
    trans_on_subset[*OF well_ord_is_trans_on*]
     well_ord_is_wf[*THEN wf_on_subset_A*] *mem_Pow_rel_abs*
  unfolding *univalent_def*
  by (*simp*)

lemma *jump_cardinal_body_eq* :
  M(*X*) ⟹ *jump_cardinal_body*(*M*,*X*) = *jump_cardinal_body′_rel*(*M*,*X*)
  unfolding *jump_cardinal_body_def jump_cardinal_body′_rel_def*
  using *ordertype_rel_abs*
  by *auto*

end

context *M_cardinal_arith*
begin

lemma *jump_cardinal_closed_aux1*:
assumes $M(X)$
shows
$M(jump\_cardinal\_body(M,X))$
unfolding *jump_cardinal_body_def*
using ‹$M(X)$› *ordertype_rel_abs*
ordertype_replacement[*OF* ‹$M(X)$›] *univalent_aux1*[*OF* ‹$M(X)$›]
strong_replacement_closed[**where** A=$Pow^M(X \times X)$ **and**
P=$\lambda\ r\ z\ .\ M(z) \wedge M(r) \wedge\ \ r \in Pow^M(X \times X) \wedge well\_ord(X,\ r) \wedge z = ordertype(X,\ r)$]
by *auto*

lemma *univalent_jc_body*: $M(X) \Longrightarrow univalent(M,X,\lambda\ x\ z\ .\ M(z) \wedge M(x) \wedge z = jump\_cardinal\_body(M,x))$
using *transM*[*of* _ X] *jump_cardinal_closed_aux1* **by** *auto*

lemma *jump_cardinal_body_closed*:
assumes $M(K)$
shows $M(\{a\ .\ X \in Pow^M(K),\ M(a) \wedge M(X) \wedge a = jump\_cardinal\_body(M,X)\})$
using *assms univalent_jc_body jump_cardinal_closed_aux1 strong_replacement_jc_body*
by *simp*

rel_closed for *jump_cardinal'*
using *jump_cardinal_body_closed ordertype_rel_abs*
unfolding *jump_cardinal_body_def jump_cardinal'_rel_def*
by *simp*

is_iff_rel for *jump_cardinal'*
proof -
assume *types*: $M(K)$ $M(res)$
have *is_Replace*(M, *Pow_rel*(M,$X\times X$), $\lambda r\ z.\ M(z) \wedge M(r) \wedge is\_well\_ord(M,\ X,\ r) \wedge is\_ordertype(M,\ X,\ r,\ z)$,
$a) \longleftrightarrow a = \{z\ .\ r \in Pow\_rel(M,X\times X),\ M(z) \wedge M(r) \wedge is\_well\_ord(M,X,r) \wedge is\_ordertype(M,\ X,\ r,\ z)\}$
if $M(X)$ $M(a)$ **for** X a
using *that univalent_aux1*
by (*rule_tac Replace_abs*) (*simp_all*)
then
have *is_Replace*(M, *Pow_rel*(M,$X\times X$), $\lambda r\ z.\ M(z) \wedge M(r) \wedge is\_well\_ord(M,\ X,\ r) \wedge is\_ordertype(M,\ X,\ r,\ z)$,
$a) \longleftrightarrow a = \{z\ .\ r \in Pow\_rel(M,X\times X),\ M(z) \wedge M(r) \wedge well\_ord(X,\ r) \wedge z = ordertype\_rel(M,\ X,\ r)\}$
if $M(X)$ $M(a)$ **for** X a
using *that univalent_aux1 is_ordertype_iff' is_well_ord_iff_wellordered well_ord_abs*
by *auto*
moreover
have *is_Replace*(M, d, $\lambda X\ a.\ M(a) \wedge M(X) \wedge$
$a = \{z\ .\ r \in Pow^M(X \times X),\ M(z) \wedge M(r) \wedge well\_ord(X,\ r) \wedge z = ordertype(X,\ r)\},\ e$)

```
    ⟷
    e ={a . X ∈ d, M(a) ∧ M(X) ∧ a = jump_cardinal_body(M,X)}
    if M(d) M(e) for d e
    using jump_cardinal_closed_aux1 that
    unfolding jump_cardinal_body_def
    by (rule_tac Replace_abs) simp_all
  ultimately
  show ?thesis
    using Pow_rel_iff jump_cardinal_body_closed[of K] ordertype_rel_abs
   unfolding is_jump_cardinal'_def jump_cardinal'_rel_def jump_cardinal_body_def
    by (simp add: types)
qed

end

locale M_jump_cardinal = M_ordertype

context M_cardinal_arith
begin

lemma (in M_ordertype) ordermap_closed[intro,simp]:
  assumes wellordered(M,A,r) and types:M(A) M(r)
  shows M(ordermap(A,r))
proof -
  note assms
  moreover from this
  obtain i f where Ord(i) f ∈ ord_iso(A, r, i, Memrel(i))
    M(i) M(f) using ordertype_exists by blast
  moreover from calculation
  have i = ordertype(A,r) using ordertypes_are_absolute by force
  moreover from calculation
  have ordermap(A,r) ∈ ord_iso(A, r, i, Memrel(i))
    using ordertype_ord_iso by simp
  ultimately
  have f = ordermap(A,r) using well_ord_iso_unique by fastforce
  with ‹M(f)›
  show ?thesis by simp
qed



lemma ordermap_eqpoll_pred:
     [| well_ord(A,r);  x ∈ A ; M(A);M(r);M(x)|] ==> ordermap(A,r)‘x ≈^M
Order.pred(A,x,r)
apply (simp add: def_eqpoll_rel)
apply (rule rexI)
apply (simp add: ordermap_eq_image well_ord_is_wf)
apply (erule ordermap_bij [THEN bij_is_inj, THEN restrict_bij,
                           THEN bij_converse_bij])
```

apply (*rule pred_subset*, *simp*)
done

Kunen: "each $\langle x,\, y\rangle \in K \times K$ has no more than $z \times z$ predecessors..." (page 29)

lemma *ordermap_csquare_le*:
assumes *K*: *Limit*(*K*) **and** *x*: $x<K$ **and** *y*: $y<K$
and *types*: *M*(*K*) *M*(*x*) *M*(*y*)
shows $|ordermap(K \times K,\, csquare\_rel(K))\ `\ \langle x,y\rangle|^M \le |succ(succ(x \cup y))|^M \otimes^M |succ(succ(x \cup y))|^M$
using *types*
proof (*simp add*: *cmult_rel_def*, *rule_tac well_ord_lepoll_rel_imp_cardinal_rel_le*)
let *?z*=*succ*($x \cup y$)
show *well_ord*($|succ(?z)|^M \times |succ(?z)|^M$,
rmult($|succ(?z)|^M$, *Memrel*($|succ(?z)|^M$), $|succ(?z)|^M$, *Memrel*($|succ(?z)|^M$)))
by (*blast intro*: *well_ord_Memrel well_ord_rmult types*)
next
let *?z*=*succ*($x \cup y$)
have *zK*: $?z<K$ **using** *x y K*
by (*blast intro*: *Un_least_lt Limit_has_succ*)
hence *oz*: *Ord*(*?z*) **by** (*elim ltE*)
from *assms*
have *Mom*:*M*(*ordermap*($K \times K$, *csquare_rel*(*K*)))
using *well_ord_csquare Limit_is_Ord* **by** *fastforce*
then
have $ordermap(K \times K,\, csquare\_rel(K))\ `\ \langle x,y\rangle \lesssim^M ordermap(K \times K,\, csquare\_rel(K))\ `\ \langle ?z,?z\rangle$
by (*blast intro*: *ordermap_z_lt leI le_imp_lepoll_rel K x y types*)
also have ... $\approx^M$ *Order.pred*($K \times K$, $\langle ?z,?z\rangle$, *csquare_rel*(*K*))
proof (*rule ordermap_eqpoll_pred*)
show *well_ord*($K \times K$, *csquare_rel*(*K*)) **using** *K*
by (*rule Limit_is_Ord* [*THEN well_ord_csquare*])
next
show $\langle ?z,\, ?z\rangle \in K \times K$ **using** *zK*
by (*blast intro*: *ltD*)
qed (*simp_all add*:*types*)
also have ... $\lesssim^M$ *succ*(*?z*) $\times$ *succ*(*?z*) **using** *zK*
by (*rule_tac pred_csquare_subset* [*THEN subset_imp_lepoll_rel*]) (*simp_all add*:*types*)
also have ... $\approx^M |succ(?z)|^M \times |succ(?z)|^M$ **using** *oz*
by (*blast intro*: *prod_eqpoll_rel_cong Ord_cardinal_rel_eqpoll_rel eqpoll_rel_sym types*)
finally show $ordermap(K \times K,\, csquare\_rel(K))\ `\ \langle x,y\rangle \lesssim^M |succ(?z)|^M \times |succ(?z)|^M$
by (*simp_all add*:*types Mom*)
from *Mom*
show *M*(*ordermap*($K \times K$, *csquare_rel*(*K*)) ` $\langle x,\, y\rangle$) **by** (*simp_all add*:*types*)
qed (*simp_all add*:*types*)

Kunen: "... so the order type is ≤ K"

lemma *ordertype_csquare_le_M*:
assumes *IK*: $InfCard^M(K)$ **and** *eq*: $\bigwedge y.\ y{\in}K \Longrightarrow InfCard^M(y) \Longrightarrow M(y) \Longrightarrow y \otimes^M y = y$
— Note the weakened hypothesis $[\![?y \in K;\ InfCard^M(?y);\ M(?y)]\!] \Longrightarrow ?y \otimes^M ?y = ?y$
and *types*: $M(K)$
shows $ordertype(K{*}K,\ csquare\_rel(K)) \leq K$
proof -
have *CK*: $Card^M(K)$ **using** *IK* **by** (*rule_tac InfCard_rel_is_Card_rel*) (*simp_all add*:*types*)
hence *OK*: $Ord(K)$ **by** (*rule Card_rel_is_Ord*) (*simp_all add*:*types*)
moreover have $Ord(ordertype(K \times K,\ csquare\_rel(K)))$ **using** *OK*
by (*rule well_ord_csquare* [*THEN Ord_ordertype*])
ultimately show *?thesis*
proof (*rule all_lt_imp_le*)
fix *i*
assume *i*:$i < ordertype(K \times K,\ csquare\_rel(K))$
hence *Oi*: $Ord(i)$ **by** (*elim ltE*)
obtain *x y* **where** *x*: $x \in K$ **and** *y*: $y \in K$
and *ieq*: $i = ordermap(K \times K,\ csquare\_rel(K))\ `\langle x,y\rangle$
using *i* **by** (*auto simp add*: *ordertype_unfold elim*: *ltE*)
hence *xy*: $Ord(x)\ Ord(y)\ x < K\ y < K$ **using** *OK*
by (*blast intro*: *Ord_in_Ord ltI*)+
hence *ou*: $Ord(x \cup y)$
by (*simp*)
from *OK types*
have $M(ordertype(K \times K,\ csquare\_rel(K)))$
using *well_ord_csquare* **by** *fastforce*
with *i x y types*
have *types'*: $M(K)\ M(i)\ M(x)\ M(y)$
using *types* **by** (*auto dest*:*transM ltD*)
show $i < K$
proof (*rule Card_rel_lt_imp_lt* [*OF _ Oi CK*])
have $|i|^M \leq |succ(succ(x \cup y))|^M \otimes^M |succ(succ(x \cup y))|^M$ **using** *IK xy*
by (*auto simp add*: *ieq types intro*: *InfCard_rel_is_Limit* [*THEN ordermap_csquare_le*] *types'*)
moreover have $|succ(succ(x \cup y))|^M \otimes^M |succ(succ(x \cup y))|^M < K$
proof (*cases rule*: *Ord_linear2* [*OF ou Ord_nat*])
assume $x \cup y < nat$
hence $|succ(succ(x \cup y))|^M \otimes^M |succ(succ(x \cup y))|^M \in nat$
by (*simp add*: *lt_def nat_cmult_rel_eq_mult nat_succI*
nat_into_Card_rel [*THEN Card_rel_cardinal_rel_eq*] *types'*)
also have ... $\subseteq K$ **using** *IK*
by (*simp add*: *InfCard_rel_def le_imp_subset types*)
finally show $|succ(succ(x \cup y))|^M \otimes^M |succ(succ(x \cup y))|^M < K$
by (*simp add*: *ltI OK*)
next
assume *natxy*: $nat \leq x \cup y$

```
          hence seq: |succ(succ(x ∪ y))|^M = |x ∪ y|^M using xy
        by (simp add: le_imp_subset nat_succ_eqpoll_rel [THEN cardinal_rel_cong]
le_succ_iff types')
          also have ... < K using xy
              by (simp add: Un_least_lt Ord_cardinal_rel_le [THEN lt_trans1]
types')
          finally have |succ(succ(x ∪ y))|^M < K .
          moreover have InfCard^M(|succ(succ(x ∪ y))|^M) using xy natxy
            by (simp add: seq InfCard_rel_def nat_le_cardinal_rel types')
          ultimately show ?thesis by (simp add: eq ltD types')
         qed
       ultimately show |i|^M < K by (blast intro: lt_trans1)
     qed (simp_all add:types')
  qed
qed


lemma InfCard_rel_csquare_eq:
  assumes IK: InfCard^M(K) and
  types: M(K)
  shows K ⊗^M K = K
proof -
  have OK: Ord(K) using IK by (simp add: Card_rel_is_Ord InfCard_rel_is_Card_rel
types)
  from OK assms
  show K ⊗^M K = K
  proof (induct rule: trans_induct)
    case (step i)
    note types = ‹M(K)› ‹M(i)›
    show i ⊗^M i = i
    proof (rule le_anti_sym)
      from step types
    have Mot:M(ordertype(i × i, csquare_rel(i))) M(ordermap(i × i, csquare_rel(i)))
        using well_ord_csquare Limit_is_Ord by simp_all
      then
      have |i × i|^M = |ordertype(i × i, csquare_rel(i))|^M
        by (rule_tac cardinal_rel_cong,
          simp_all add: step.hyps well_ord_csquare [THEN ordermap_bij, THEN
bij_imp_eqpoll_rel] types)
      with Mot
      have i ⊗^M i ≤ ordertype(i × i, csquare_rel(i))
     by (simp add: step.hyps cmult_rel_def Ord_cardinal_rel_le well_ord_csquare
[THEN Ord_ordertype] types)
      moreover
      have ordertype(i × i, csquare_rel(i)) ≤ i using step
        by (rule_tac ordertype_csquare_le_M) (simp add: types)
      ultimately show i ⊗^M i ≤ i by (rule le_trans)
    next
      show i ≤ i ⊗^M i using step
```

```
      by (blast intro: cmult_rel_square_le InfCard_rel_is_Card_rel)
    qed
  qed
qed



lemma well_ord_InfCard_rel_square_eq:
  assumes r: well_ord(A,r) and I: InfCard^M(|A|^M) and
    types: M(A) M(r)
  shows A × A ≈^M A
proof -
  have A × A ≈^M |A|^M × |A|^M
   by (blast intro: prod_eqpoll_rel_cong well_ord_cardinal_rel_eqpoll_rel eqpoll_rel_sym
r types)
  also have ... ≈^M A
    proof (rule well_ord_cardinal_rel_eqE [OF _ r])
    show well_ord(|A|^M × |A|^M, rmult(|A|^M, Memrel(|A|^M), |A|^M, Memrel(|A|^M)))
       by (blast intro: well_ord_rmult well_ord_Memrel r types)
    next
      show ||A|^M × |A|^M|^M = |A|^M using InfCard_rel_csquare_eq I
        by (simp add: cmult_rel_def types)
    qed (simp_all add:types)
  finally show ?thesis by (simp_all add:types)
qed

lemma InfCard_rel_square_eqpoll:
  assumes InfCard^M(K) and types:M(K) shows K × K ≈^M K
  using assms
apply (rule_tac well_ord_InfCard_rel_square_eq)
 apply (erule InfCard_rel_is_Card_rel [THEN Card_rel_is_Ord, THEN well_ord_Memrel])
apply (simp_all add: InfCard_rel_is_Card_rel [THEN Card_rel_cardinal_rel_eq]
types)
done

lemma Inf_Card_rel_is_InfCard_rel: [| Card^M(i); ~ Finite_rel(M,i) ; M(i) |]
==> InfCard^M(i)
  by (simp add: InfCard_rel_def Card_rel_is_Ord [THEN nat_le_infinite_Ord])
```

23.5.1 Toward's Kunen's Corollary 10.13 (1)

```
lemma InfCard_rel_le_cmult_rel_eq: [| InfCard^M(K);  L ≤ K;  0<L; M(K) ;
M(L) |] ==> K ⊗^M L = K
apply (rule le_anti_sym)
 prefer 2
 apply (erule ltE, blast intro: cmult_rel_le_self InfCard_rel_is_Card_rel)
apply (frule InfCard_rel_is_Card_rel [THEN Card_rel_is_Ord, THEN le_refl])
prefer 3
apply (rule cmult_rel_le_mono [THEN le_trans], assumption+)
```

apply (*simp_all add*: *InfCard_rel_csquare_eq*)
done

lemma *InfCard_rel_cmult_rel_eq*: [| $InfCard^M(K)$; $InfCard^M(L)$; $M(K)$; $M(L)$ |] ==> $K \otimes^M L = K \cup L$
apply (*rule_tac* $i = K$ **and** $j = L$ **in** *Ord_linear_le*)
apply (*typecheck add*: *InfCard_rel_is_Card_rel Card_rel_is_Ord*)
apply (*rule cmult_rel_commute* [*THEN ssubst*]) **prefer** *3*
apply (*rule Un_commute* [*THEN ssubst*])
apply (*simp_all add*: *InfCard_rel_is_Limit* [*THEN Limit_has_0*] *InfCard_rel_le_cmult_rel_eq*
subset_Un_iff2 [*THEN iffD1*] *le_imp_subset*)
done

lemma *InfCard_rel_cdouble_eq*: $InfCard^M(K) \Longrightarrow M(K) \Longrightarrow K \oplus^M K = K$
apply (*simp add*: *cmult_rel_2* [*symmetric*] *InfCard_rel_is_Card_rel cmult_rel_commute*)
apply (*simp add*: *InfCard_rel_le_cmult_rel_eq InfCard_rel_is_Limit Limit_has_0*
Limit_has_succ)
done

lemma *InfCard_rel_le_cadd_rel_eq*: [| $InfCard^M(K)$; $L \leq K$; $M(K)$; $M(L)$|]
==> $K \oplus^M L = K$
apply (*rule le_anti_sym*)
prefer *2*
apply (*erule ltE*, *blast intro*: *cadd_rel_le_self InfCard_rel_is_Card_rel*)
apply (*frule InfCard_rel_is_Card_rel* [*THEN Card_rel_is_Ord*, *THEN le_refl*])
prefer *3*
apply (*rule cadd_rel_le_mono* [*THEN le_trans*], *assumption*+)
apply (*simp_all add*: *InfCard_rel_cdouble_eq*)
done

lemma *InfCard_rel_cadd_rel_eq*: [| $InfCard^M(K)$; $InfCard^M(L)$; $M(K)$; $M(L)$ |] ==> $K \oplus^M L = K \cup L$
apply (*rule_tac* $i = K$ **and** $j = L$ **in** *Ord_linear_le*)
apply (*typecheck add*: *InfCard_rel_is_Card_rel Card_rel_is_Ord*)
apply (*rule cadd_rel_commute* [*THEN ssubst*]) **prefer** *3*
apply (*rule Un_commute* [*THEN ssubst*])
apply (*simp_all add*: *InfCard_rel_le_cadd_rel_eq subset_Un_iff2* [*THEN iffD1*]
le_imp_subset)
done

end

23.6 For Every Cardinal Number There Exists A Greater One

This result is Kunen's Theorem 10.16, which would be trivial using AC

locale *M_cardinal_arith_jump* = *M_cardinal_arith* + *M_jump_cardinal*
begin

lemma *well_ord_restr*: *well_ord*(X, r) $\Longrightarrow$ *well_ord*(X, $r \cap X\times X$)
proof -
  have $r \cap X\times X \cap X\times X = r \cap X\times X$ **by** *auto*
  moreover
  assume *well_ord*(X, r)
  ultimately
  show *?thesis*
  unfolding *well_ord_def tot_ord_def part_ord_def linear_def*
    irrefl_def wf_on_def
    by *simp_all* (*simp only*: *trans_on_def*, *blast*)
qed

lemma *ordertype_restr_eq* :
  assumes *well_ord*(X,r)
  shows *ordertype*(X, r) = *ordertype*(X, $r \cap X\times X$)
  using *ordermap_restr_eq assms* **unfolding** *ordertype_def*
  by *simp*

lemma *def_jump_cardinal_rel_aux*:
  $X \in Pow^M(K) \Longrightarrow$ *well_ord*(X, w) $\Longrightarrow$ $M(K) \Longrightarrow$
  $\{z\ .\ r \in Pow^M(X \times X),\ M(z) \wedge$ *well_ord*(X, r) $\wedge\ z =$ *ordertype*(X, r)$\} =$
  $\{z\ .\ r \in Pow^M(K \times K),\ M(z) \wedge$ *well_ord*(X, r) $\wedge\ z =$ *ordertype*(X, r)$\}$
proof(*rule*,*auto simp*:*Pow_rel_char dest*:*transM*)
  let *?L*=$\{z\ .\ r \in Pow^M(X \times X),\ M(z) \wedge$ *well_ord*(X, r) $\wedge\ z =$ *ordertype*(X, r)$\}$
  let *?R*=$\{z\ .\ r \in Pow^M(K \times K),\ M(z) \wedge$ *well_ord*(X, r) $\wedge\ z =$ *ordertype*(X, r)$\}$
  show *ordertype*(X, r) $\in \{y\ .\ x \in \{x \in Pow(X \times X)\ .\ M(x)\},\ M(y) \wedge$ *well_ord*(X, x) $\wedge\ y =$ *ordertype*(X, x)$\}$
    if $M(K)$ $M(r)$ $r\subseteq K\times K$ $X\subseteq K$ $M(X)$ *well_ord*(X,r) **for** r
  proof -
    from *that*
    have *ordertype*(X,r) = *ordertype*(X,$r\cap X\times X$) ($r\cap X\times X$)$\subseteq X\times X$ $M(r\cap X\times X)$
      well_ord(X,$r\cap X\times X$) *wellordered*(M,X,$r\cap X\times X$)
      using *well_ord_restr ordertype_restr_eq* **by** *auto*
    moreover from *this*
    have *ordertype*(X,$r\cap X\times X$) $\in$ *?L*
      using *that Pow_rel_char*
          ReplaceI[*of* $\lambda\ z\ r\ .\ M(z) \wedge$ *well_ord*(X, r) $\wedge\ z =$ *ordertype*(X, r) *ordertype*(X,$r\cap X\times X$)]
      by *auto*
    ultimately
    show *?thesis* **using** *Pow_rel_char* **by** *auto*
  qed

qed

lemma *def_jump_cardinal_rel*:
assumes $M(K)$
shows *jump_cardinal'_rel*$(M,K) =$
$(\bigcup X{\in}Pow\_rel(M,K).\ \{z.\ r \in Pow\_rel(M,K{*}K),\ well\_ord(X,r)\ \&\ z = ordertype(X,r)\})$
proof -
have $M(\{z\ .\ r \in Pow^M(X \times X),\ M(z) \wedge well\_ord(X,\ r) \wedge z = ordertype(X,\ r)\})$
(**is** $M(Replace(\_,?P))$)
if $M(X)$ **for** X
using *that jump_cardinal_closed_aux1*[*of X*] *ordertype_rel_abs*[*of X*]
jump_cardinal_body_def
by (*subst Replace_cong*[**where** *P=?P*
and $Q{=}\lambda r\ z.\ M(z) \wedge M(r) \wedge well\_ord(X,\ r) \wedge z = ordertype\_rel(M,X,r)$,
OF refl, *of* $Pow^M(X \times X)$]) (*auto dest*:*transM*)
then
have $M(\{z\ .\ r \in Pow^M(Y \times Y),\ M(z) \wedge well\_ord(X,\ r) \wedge z = ordertype(X,\ r)\})$
if $M(Y)$ $M(X)$ $X \in Pow^M(Y)$ *well_ord*(X,r) **for** Y X r
using *that def_jump_cardinal_rel_aux*[*of X Y r*, *symmetric*] **by** *simp*
moreover from ‹$M(K)$›
have $R \in Pow^M(X \times X) \Longrightarrow X \in Pow^M(K) \Longrightarrow R \in Pow^M(K \times K)$
for X R **using** *mem_Pow_rel_abs transM*[*OF _ Pow_rel_closed*, *of R X*×*X*]
transM[*OF _ Pow_rel_closed*, *of X K*] **by** *auto*
ultimately
show *?thesis*
using *assms is_ordertype_iff is_well_ord_iff_wellordered*
ordertype_rel_abs transM[*of _* $Pow^M(K)$] *transM*[*of _* $Pow^M(K{\times}K)$]
def_jump_cardinal_rel_aux
unfolding *jump_cardinal'_rel_def*
apply (*intro equalityI*)
apply (*auto dest*:*transM*)
apply (*rename_tac X R*)
apply (*rule_tac x=X* **in** *bexI*)
apply (*rule_tac x=R* **in** *ReplaceI*)
apply *auto*
apply (*rule_tac x=*$\{y\ .\ xa \in Pow^M(K \times K),\ M(y) \wedge M(xa) \wedge well\_ord(X,\ xa) \wedge y = ordertype(X,\ xa)\}$ **in** *bexI*)
apply *auto*
by (*rule_tac x=X* **in** *ReplaceI*) *auto*
qed

notation *jump_cardinal'_rel* (‹*jump'_cardinal'_rel*›)

lemma *Ord_jump_cardinal_rel*: $M(K) \Longrightarrow Ord(jump\_cardinal\_rel(M,K))$
apply (*unfold def_jump_cardinal_rel*)
apply (*rule Ord_is_Transset* [*THEN* [*2*] *OrdI*])

prefer *2* **apply** (*blast intro*!: *Ord_ordertype*)
apply (*unfold Transset_def*)
apply (*safe del*: *subsetI*)
apply (*subst ordertype_pred_unfold*, *simp*, *safe*)
apply (*rule UN_I*)
apply (*rule_tac* [*2*] *ReplaceI*)
prefer *4* **apply** (*blast intro*: *well_ord_subset elim*!: *predE*, *simp_all*)
prefer *2* **apply** (*blast intro*: *well_ord_subset elim*!: *predE*)
proof -
fix *X r xb*
assume $M(K)$ $X \in Pow^M(K)$ $r \in Pow^M(K \times K)$ *well_ord*(X, r) $xb \in X$
moreover from *this*
have $M(X)$ $M(r)$
using *cartprod_closed trans_Pow_rel_closed* **by** *auto*
moreover from *this*
have $M(xb)$ **using** *transM*[*OF* ‹*xb*∈*X*›] **by** *simp*
ultimately
show *Order.pred*(X, xb, r) $\in Pow^M(K)$
using *def_Pow_rel* **by** (*auto dest*:*predE*)
qed

declare *conj_cong* [*cong del*]
— incompatible with some of the proofs of the original theory

lemma *jump_cardinal_rel_iff_old*:
$M(i) \Longrightarrow M(K) \Longrightarrow i \in$ *jump_cardinal_rel*(M,K) $\longleftrightarrow$
($\exists\, r[M].\ \exists\, X[M].\ r \subseteq K*K$ & $X \subseteq K$ & *well_ord*(X,r) & $i =$ *ordertype*(X,r))
apply (*unfold def_jump_cardinal_rel*)
apply (*auto del*: *subsetI*)
apply (*rename_tac y r*)
apply (*rule_tac x*=*r* **in** *rexI*, *intro conjI*) **prefer** *2*
apply (*rule_tac x*=*y* **in** *rexI*, *intro conjI*)
apply (*auto dest*:*mem_Pow_rel transM*)
apply (*rule_tac A*=*r* **in** *rev_subsetD*, *assumption*)
defer
apply (*rename_tac r y*)
apply (*rule_tac x*=*y* **in** *bexI*)
apply (*rule_tac x*=*r* **in** *ReplaceI*, *auto*)
using *def_Pow_rel*
apply (*force*+)[*2*]
apply (*rule_tac A*=*r* **in** *rev_subsetD*, *assumption*)
using *mem_Pow_rel*[*THEN conjunct1*]
apply *auto*
done

lemma *K_lt_jump_cardinal_rel*: *Ord*(K) ==> $M(K) \Longrightarrow K <$ *jump_cardinal_rel*(M,K)
apply (*rule Ord_jump_cardinal_rel* [*THEN* [*2*] *ltI*])
apply (*rule jump_cardinal_rel_iff_old* [*THEN iffD2*], *assumption*+)

apply (*rule_tac x=Memrel(K)* **in** *rexI*)
apply (*rule_tac x=K* **in** *rexI*)
  apply (*simp add*: *ordertype_Memrel well_ord_Memrel*)
 using *Memrel_closed*
apply (*simp_all add*: *Memrel_def subset_iff*)
done

lemma *Card_rel_jump_cardinal_rel_lemma*:
  [| *well_ord*(*X*,*r*); $r \subseteq K * K$; $X \subseteq K$;
    $f \in bij(ordertype(X,r), jump\_cardinal\_rel(M,K))$;
    M(*X*); *M*(*r*); *M*(*K*); *M*(*f*) |]
  ==> $jump\_cardinal\_rel(M,K) \in jump\_cardinal\_rel(M,K)$
apply (*subgoal_tac f O ordermap* (*X*,*r*) ∈ *bij* (*X*, *jump_cardinal_rel* (*M*,*K*)))
 prefer *2* **apply** (*blast intro*: *comp_bij ordermap_bij*)
apply (*rule jump_cardinal_rel_iff_old* [*THEN iffD2*], *simp*+)
apply (*intro rexI conjI*)
apply (*rule subset_trans* [*OF rvimage_type Sigma_mono*], *assumption*+)
apply (*erule bij_is_inj* [*THEN well_ord_rvimage*])
  apply (*rule Ord_jump_cardinal_rel* [*THEN well_ord_Memrel*])
apply (*simp_all add*: *well_ord_Memrel* [*THEN* [*2*] *bij_ordertype_vimage*]
          ordertype_Memrel Ord_jump_cardinal_rel)
done

lemma *Card_rel_jump_cardinal_rel*: $M(K) \Longrightarrow Card\_rel(M,jump\_cardinal\_rel(M,K))$
 apply (*rule Ord_jump_cardinal_rel* [*THEN Card_relI*])
  apply (*simp_all add*: *def_eqpoll_rel*)
 apply (*drule_tac i1=j* **in** *jump_cardinal_rel_iff_old* [*THEN iffD1*, *OF _ _ ltD*,
of _ K], *safe*)
 apply (*blast intro*: *Card_rel_jump_cardinal_rel_lemma* [*THEN mem_irrefl*])
 done

23.7 Basic Properties of Successor Cardinals

lemma *csucc_rel_basic*: *Ord*(*K*) ==> $M(K) \Longrightarrow Card\_rel(M,csucc\_rel(M,K))$
& $K < csucc\_rel(M,K)$
apply (*unfold csucc_rel_def*)
apply (*rule LeastI*[*of* $\lambda i.\ M(i) \wedge Card\_rel(M,i) \wedge K < i$, *THEN conjunct2*])
apply (*blast intro*: *Card_rel_jump_cardinal_rel K_lt_jump_cardinal_rel Ord_jump_cardinal_rel*)+
done

lemmas *Card_rel_csucc_rel* = *csucc_rel_basic* [*THEN conjunct1*]

lemmas *lt_csucc_rel* = *csucc_rel_basic* [*THEN conjunct2*]

lemma *Ord_0_lt_csucc_rel*: *Ord*(*K*) ==> $M(K) \Longrightarrow 0 < csucc\_rel(M,K)$
by (*blast intro*: *Ord_0_le lt_csucc_rel lt_trans1*)

lemma *csucc_rel_le*: [| *Card_rel*(*M*,*L*); *K*<*L*; *M*(*K*); *M*(*L*) |] ==> *csucc_rel*(*M*,*K*) ≤ *L*
apply (*unfold csucc_rel_def*)
apply (*rule Least_le*)
apply (*blast intro*: *Card_rel_is_Ord*)+
done

lemma *lt_csucc_rel_iff*: [| *Ord*(*i*); *Card_rel*(*M*,*K*); *M*(*K*); *M*(*i*)|] ==> *i* < *csucc_rel*(*M*,*K*) ⟷ |*i*|^M ≤ *K*
apply (*rule iffI*)
apply (*rule_tac* [*2*] *Card_rel_lt_imp_lt*)
apply (*erule_tac* [*2*] *lt_trans1*)
apply (*simp_all add*: *lt_csucc_rel Card_rel_csucc_rel Card_rel_is_Ord*)
apply (*rule notI* [*THEN not_lt_imp_le*])
apply (*rule Card_rel_cardinal_rel* [*THEN csucc_rel_le*, *THEN lt_trans1*, *THEN lt_irrefl*], *simp_all*+)
apply (*rule Ord_cardinal_rel_le* [*THEN lt_trans1*])
apply (*simp_all add*: *Card_rel_is_Ord*)
done

lemma *Card_rel_lt_csucc_rel_iff*:
  [| *Card_rel*(*M*,*K'*); *Card_rel*(*M*,*K*); *M*(*K'*); *M*(*K*) |] ==> *K'* < *csucc_rel*(*M*,*K*) ⟷ *K'* ≤ *K*
by (*simp add*: *lt_csucc_rel_iff Card_rel_cardinal_rel_eq Card_rel_is_Ord*)

lemma *InfCard_rel_csucc_rel*: *InfCard_rel*(*M*,*K*) ⟹ *M*(*K*) ==> *InfCard_rel*(*M*,*csucc_rel*(*M*,*K*))
by (*simp add*: *InfCard_rel_def Card_rel_csucc_rel Card_rel_is_Ord*
        lt_csucc_rel [*THEN leI*, *THEN* [*2*] *le_trans*])

23.7.1 Theorems by Krzysztof Grabczewski, proofs by lcp

lemma *nat_sum_eqpoll_rel_sum*:
  assumes *m*: *m* ∈ *nat* **and** *n*: *n* ∈ *nat* **shows** *m* + *n* ≈^{M} *m* #+ *n*
proof -
  have *m* + *n* ≈^{M} |*m*+*n*|^M **using** *m n*
   by (*blast intro*: *nat_implies_well_ord well_ord_radd well_ord_cardinal_rel_eqpoll_rel eqpoll_rel_sym*)
  also have ... = *m* #+ *n* **using** *m n*
    by (*simp add*: *nat_cadd_rel_eq_add* [*symmetric*] *cadd_rel_def transM*[*OF* _ *M_nat*])
  finally show *?thesis* **.**
qed

lemma *Ord_nat_subset_into_Card_rel*: [| *Ord*(*i*); *i* ⊆ *nat* |] ==> *Card*^{M}(*i*)
by (*blast dest*: *Ord_subset_natD intro*: *Card_rel_nat nat_into_Card_rel*)

end
end
theory *Aleph_Relative*

imports
CardinalArith_Relative
begin

definition
HAleph :: [*i*,*i*] ⇒ *i* **where**
HAleph(*i*,*r*) ≡ *if*(¬(*Ord*(*i*)),*i*,*if*(*i*=*0*, *nat*, *if*(¬*Limit*(*i*) ∧ *i*≠*0*,
csucc(*r*‘(⋃ *i*)),
⋃*j*∈*i*. *r*‘*j*)))

reldb_add functional *Limit Limit*
relationalize *Limit is_Limit* **external**
synthesize *is_Limit* **from_definition**
arity_theorem for *is_Limit_fm*

relativize functional *HAleph HAleph_rel*
relationalize *HAleph_rel is_HAleph*

synthesize *is_HAleph* **from_definition assuming** *nonempty*
arity_theorem for *is_HAleph_fm*

definition
Aleph′ :: *i* => *i* **where**
Aleph′(*a*) == *transrec*(*a*,λ*i r*. *HAleph*(*i*,*r*))

relativize functional *Aleph′ Aleph_rel*
relationalize *Aleph_rel is_Aleph*

The extra assumptions *a* < *length*(*env*) and *c* < *length*(*env*) in this schematic goal (and the following results on synthesis that depend on it) are imposed by ⟦⋀*a0 a1 a2 a3 a4 a5 a6 a7*. ⟦*a0* ∈ *?A*; *a1* ∈ *?A*; *a2* ∈ *?A*; *a3* ∈ *?A*; *a4* ∈ *?A*; *a5* ∈ *?A*; *a6* ∈ *?A*; *a7* ∈ *?A*⟧ ⟹ *?MH*(*a2*, *a1*, *a0*) ⟷ *?A*, *Cons*(*a0*, *Cons*(*a1*, *Cons*(*a2*, *Cons*(*a3*, *Cons*(*a4*, *Cons*(*a5*, *Cons*(*a6*, *Cons*(*a7*, *?env*)))))))) ⊨ *?p*; *nth*(*?i*, *?env*) = *?x*; *nth*(*?k*, *?env*) = *?z*; *?i* < *length*(*?env*); *?k* < *length*(*?env*); *?env* ∈ *list*(*?A*)⟧ ⟹ *is_transrec*(*##?A*, *?MH*, *?x*, *?z*) ⟷ *?A*, *?env* ⊨ *is_transrec_fm*(*?p*, *?i*, *?k*).

schematic_goal *sats_is_Aleph_fm_auto*:
a ∈ *nat* ⟹ *c* ∈ *nat* ⟹ *env* ∈ *list*(*A*) ⟹
a < *length*(*env*) ⟹ *c* < *length*(*env*) ⟹ *0* ∈ *A* ⟹
is_Aleph(*##A*, *nth*(*a*, *env*), *nth*(*c*, *env*)) ⟷ *A*, *env* ⊨ *?fm*(*a*, *c*)
unfolding *is_Aleph_def*
proof (*rule is_transrec_iff_sats*, *rule_tac* [*1*] *is_HAleph_iff_sats*)
fix *a0 a1 a2 a3 a4 a5 a6 a7*
show *nth*(*2*, *Cons*(*a0*, *Cons*(*a1*, *Cons*(*a2*, *Cons*(*a3*, *Cons*(*a4*, *Cons*(*a5*, *Cons*(*a6*, *Cons*(*a7*, *env*)))))))) = *a2*
nth(*1*, *Cons*(*a0*, *Cons*(*a1*, *Cons*(*a2*, *Cons*(*a3*, *Cons*(*a4*, *Cons*(*a5*, *Cons*(*a6*, *Cons*(*a7*, *env*)))))))) = *a1*
nth(*0*, *Cons*(*a0*, *Cons*(*a1*, *Cons*(*a2*, *Cons*(*a3*, *Cons*(*a4*, *Cons*(*a5*, *Cons*(*a6*, *Cons*(*a7*, *env*)))))))) = *a0*

$nth(c,\ env) = nth(c,\ env)$
by *simp_all*
qed *simp_all*

synthesize_notc *is_Aleph* **from_schematic**

notation *is_Aleph_fm* (‹·א′(_′) *is* _·›)

lemma *is_Aleph_fm_type* [*TC*]: $a \in nat \Longrightarrow c \in nat \Longrightarrow is\_Aleph\_fm(a,\ c) \in formula$
unfolding *is_Aleph_fm_def* **by** *simp*

lemma *sats_is_Aleph_fm*:
assumes $f{\in}nat\ r{\in}nat\ env \in list(A)\ 0{\in}A\ f < length(env)\ r{<}\ length(env)$
shows $is\_Aleph(\#\#A,\ nth(f,\ env),\ nth(r,\ env)) \longleftrightarrow A,\ env \models is\_Aleph\_fm(f,r)$
using *assms sats_is_Aleph_fm_auto* **unfolding** *is_Aleph_def is_Aleph_fm_def*
by *simp*

lemma *is_Aleph_iff_sats* [*iff_sats*]:
assumes
$nth(f,\ env) = fa\ nth(r,\ env) = ra\ f < length(env)\ r{<}\ length(env)$
$f \in nat\ r \in nat\ env \in list(A)\ 0{\in}A$
shows $is\_Aleph(\#\#A,fa,ra) \longleftrightarrow A,\ env \models is\_Aleph\_fm(f,r)$
using *assms sats_is_Aleph_fm*[*of f r env A*] **by** *simp*

arity_theorem for *is_Aleph_fm*

context *M_cardinal_arith_jump*
begin

lemma *is_Limit_iff*:
assumes $M(a)$
shows $is\_Limit(M,a) \longleftrightarrow Limit(a)$
unfolding *is_Limit_def Limit_def* **using** *lt_abs transM*[*OF ltD* ‹$M(a)$›] *assms*
by *auto*

end

lemma *HAleph_eq_Aleph_recursive*:
$Ord(i) \Longrightarrow HAleph(i,r) = (if\ i = 0\ then\ nat$
$else\ if\ \exists j.\ i = succ(j)\ then\ csucc(r\ `\ (THE\ j.\ i = succ(j)))\ else\ \bigcup j{<}i.\ r\ `\ j)$
proof -
assume $Ord(i)$
moreover from *this*
have $i = succ(j) \Longrightarrow (\bigcup succ(j)) = j$ **for** j
using *Ord_Union_succ_eq* **by** *simp*
moreover from ‹$Ord(i)$›
have $(\exists j.\ i = succ(j)) \longleftrightarrow \neg Limit(i) \wedge i \neq 0$

```
    using Ord_cases_disj by auto
  ultimately
  show ?thesis
    unfolding HAleph_def OUnion_def
    by auto
qed

lemma Aleph'_eq_Aleph: Ord(a) ⟹ Aleph'(a) = Aleph(a)
  unfolding Aleph'_def Aleph_def transrec2_def
  using HAleph_eq_Aleph_recursive
  by (intro transrec_equal_on_Ord) auto

reldb_rem functional Aleph'
reldb_rem relational is_Aleph
reldb_add functional Aleph Aleph_rel
reldb_add relational Aleph is_Aleph

abbreviation
  Aleph_r :: [i,i⇒o] ⇒ i (‹ℵ_—›) where
  Aleph_r(a,M) ≡ Aleph_rel(M,a)

abbreviation
  Aleph_r_set :: [i,i] ⇒ i (‹ℵ_—›) where
  Aleph_r_set(a,M) ≡ Aleph_rel(##M,a)

lemma Aleph_rel_def': Aleph_rel(M,a) ≡ transrec(a, λi r. HAleph_rel(M, i, r))
  unfolding Aleph_rel_def .

lemma succ_mem_Limit: Limit(j) ⟹ i ∈ j ⟹ succ(i) ∈ j
  using Limit_has_succ[THEN ltD] ltI Limit_is_Ord by auto

locale M_pre_aleph = M_eclose + M_cardinal_arith_jump +
  assumes
   haleph_transrec_replacement: M(a) ⟹ transrec_replacement(M,is_HAleph(M),a)

begin
lemma aux:
  assumes M(a) M(f)
  shows ∃x[M]. is_Replace(M, a, λj y. f ‘ j = y, x)
proof -
  have {f‘j . j∈a} = {y . j∈a , f ‘ j=y}
    {y . j∈a , f ‘ j=y} = {y . j∈a , y =f ‘ j}
    by auto
  moreover
  note assms
  moreover
  have 1:univalent(M, a, λj y. y = f ‘ j)
    using univalent_triv[of M a λj .f ‘ j] by auto
  moreover from calculation
```

```
  have x ∈ a ⟹ y = f ‘x ⟹ M(y) for x y
    using transM[OF _ ‹M(a)›] by auto
  moreover from this assms
  have M({f‘j . j∈a})
    using RepFun_closed[OF apply_replacement] by simp
  ultimately
  have 2:is_Replace(M, a, λj y. y = f ‘ j, {f‘j . j∈a})
   using Replace_abs[of _ _ λj y. y = f ‘ j,OF ‹M(a)› _ 1,THEN iffD2, simplified]
     by auto
  with ‹M({f‘j . j∈a})›
  show ?thesis
    using
      is_Replace_cong[of _ _ M λj y. y = f ‘ j λj y. f ‘ j = y, THEN iffD1,OF _
_ _ 2]
    by auto
qed

lemma is_HAleph_zero:
  assumes M(f)
  shows is_HAleph(M,0,f,res) ⟷ res = nat
  unfolding is_HAleph_def
  using Ord_0 If_abs is_Limit_iff is_csucc_iff assms aux
  by auto

lemma is_HAleph_succ:
  assumes M(f) M(x) Ord(x) M(res)
  shows is_HAleph(M,succ(x),f,res) ⟷ res = csucc_rel(M,f‘( ⋃ succ(x) ))
  unfolding is_HAleph_def
  using assms is_Limit_iff is_csucc_iff aux If_abs
  by simp

lemma is_HAleph_limit:
  assumes M(f) M(x) Limit(x) M(res)
  shows is_HAleph(M,x,f,res) ⟷ res = (⋃{y . i∈x ,M(i) ∧ M(y) ∧ y = f‘i})
proof -
  from assms
  have univalent(M, x, λj y. y = f ‘ j  )
    (⋀xa y. xa ∈ x ⟹ f ‘ xa = y ⟹ M(y))
    {y . x ∈ x, f ‘ x = y} = {y . i∈x ,M(i) ∧ M(y) ∧ y = f‘i}
    using univalent_triv[of M x λj .f ‘ j] transM[OF _ ‹M(x)›] by auto
  moreover
  from this
  have A:univalent(M, x, λj y. f ‘ j = y )
     by (rule_tac univalent_cong[of x x M  λj y. y = f ‘ j  λj y. f ‘ j=y,THEN
iffD1], auto)
  moreover
  from this
  have univalent(M, x, λj y. M(j) ∧ M(y) ∧ f ‘ j = y ) by auto
  ultimately
```

```
  show ?thesis
    unfolding is_HAleph_def
    using assms is_Limit_iff Limit_is_Ord zero_not_Limit If_abs is_csucc_iff
      Replace_abs[OF ‹M(x)› _ A] apply_replacement
    by auto
qed

lemma is_HAleph_iff:
  assumes M(a) M(f) M(res)
  shows is_HAleph(M, a, f, res) ⟷ res = HAleph_rel(M, a, f)
proof(cases Ord(a))
  case True
  note Ord_cases[OF ‹Ord(a)›]
  then
  show ?thesis
  proof(cases )
    case 1
    with True assms
    show ?thesis
      using is_HAleph_zero unfolding HAleph_rel_def
      by simp
  next
    case (2 j)
    with True assms
    show ?thesis
      using is_HAleph_succ unfolding HAleph_rel_def
      by simp
  next
    case 3
    with assms
    show ?thesis
      using is_HAleph_limit zero_not_Limit Limit_is_Ord
      unfolding HAleph_rel_def
      by auto
  qed
next
  case False
  then
  have ¬Limit(a) a≠0 ⋀ x . Ord(x) ⟹ a≠succ(x)
    using Limit_is_Ord by auto
  with False
  show ?thesis
    unfolding is_HAleph_def HAleph_rel_def
    using assms is_Limit_iff If_abs is_csucc_iff aux
    by auto
qed

lemma HAleph_rel_closed [intro,simp]:
  assumes function(f) M(a) M(f)
```

shows $M(HAleph\_rel(M,a,f))$
unfolding *HAleph_rel_def SepReplace_def*
using *assms apply_replacement*
by *simp*

lemma *Aleph_rel_closed*[*intro*, *simp*]:
assumes $Ord(a)$ $M(a)$
shows $M(Aleph\_rel(M,a))$
proof -
have *relation2*(M, *is_HAleph*(M), *HAleph_rel*(M))
unfolding *relation2_def* **using** *is_HAleph_iff assms* **by** *simp*
moreover
have $\forall\, x[M].\ \forall\, g[M].\ function(g) \longrightarrow M(HAleph\_rel(M,\ x,\ g))$
using *HAleph_rel_closed* **by** *simp*
moreover
note *assms*
ultimately
show *?thesis*
unfolding *Aleph_rel_def*
using *transrec_closed*[*of is_HAleph*(M) *a HAleph_rel*(M)]
haleph_transrec_replacement **by** *simp*
qed

lemma *Aleph_rel_zero*: ${\aleph_0}^M = nat$
using *def_transrec* [*OF Aleph_rel_def'*,*of* _ *0*]
unfolding *HAleph_rel_def* **by** *simp*

lemma *Aleph_rel_succ*: $Ord(\alpha) \Longrightarrow M(\alpha) \Longrightarrow {\aleph_{succ(\alpha)}}^M = ({\aleph_\alpha}^{M+})^M$
using *Ord_Union_succ_eq*
by (*subst def_transrec* [*OF Aleph_rel_def'*])
(*simp add*:*HAleph_rel_def*)

lemma *Aleph_rel_limit*:
assumes $Limit(\alpha)$ $M(\alpha)$
shows ${\aleph_\alpha}^M = \bigcup\{{\aleph_j}^M\ .\ j \in \alpha\}$
proof -
note *trans*=*transM*[*OF* _ ‹$M(\alpha)$›]
from ‹$M(\alpha)$›
have ${\aleph_\alpha}^M = HAleph\_rel(M,\ \alpha,\ \lambda x{\in}\alpha.\ {\aleph_x}^M)$
using *def_transrec* [*OF Aleph_rel_def'*,*of M* α] **by** *simp*
also
have $... = \bigcup\{a\ .\ j \in \alpha,\ M(a) \wedge a = {\aleph_j}^M\}$
unfolding *HAleph_rel_def*
using *assms zero_not_Limit Limit_is_Ord trans* **by** *auto*
also
have $... = \bigcup\{{\aleph_j}^M\ .\ j \in \alpha\}$
using *Aleph_rel_closed*[*OF* _ *trans*] *Ord_in_Ord Limit_is_Ord*[*OF* ‹$Limit(\alpha)$›]
by *auto*
finally

show *?thesis* **.**
qed

lemma *is_Aleph_iff*:
assumes *Ord*(*a*) *M*(*a*) *M*(*res*)
shows *is_Aleph*(*M*, *a*, *res*) $\longleftrightarrow$ *res* = $\aleph_a{}^M$
proof -
have *relation2*(*M*, *is_HAleph*(*M*), *HAleph_rel*(*M*))
unfolding *relation2_def* **using** *is_HAleph_iff assms* **by** *simp*
moreover
have $\forall\, x[M].\ \forall\, g[M].$ *function*(*g*) $\longrightarrow$ *M*(*HAleph_rel*(*M*, *x*, *g*))
using *HAleph_rel_closed* **by** *simp*
ultimately
show *?thesis*
using *assms transrec_abs haleph_transrec_replacement*
unfolding *is_Aleph_def Aleph_rel_def*
by *simp*
qed

end

locale *M_aleph* = *M_pre_aleph* +
assumes
aleph_rel_replacement: *strong_replacement*(*M*, $\lambda x\ y.$ *Ord*(*x*) $\wedge$ *y* = $\aleph_x{}^M$)
begin

lemma *Aleph_rel_cont*: *Limit*(*l*) $\Longrightarrow$ *M*(*l*) $\Longrightarrow$ $\aleph_l{}^M$ = $(\bigcup i{<}l.\ \aleph_i{}^M)$
using *Limit_is_Ord Aleph_rel_limit*
by (*simp add*:*OUnion_def*)

lemma *Ord_Aleph_rel*:
assumes *Ord*(*a*)
shows *M*(*a*) $\Longrightarrow$ *Ord*($\aleph_a{}^M$)
using ‹*Ord*(*a*)›
proof(*induct a rule*:*trans_induct3*)
case *0*
show *?case* **using** *Aleph_rel_zero* **by** *simp*
next
case (*succ x*)
with ‹*Ord*(*x*)›
have *M*(*x*) *Ord*($\aleph_x{}^M$) **by** *simp_all*
with ‹*Ord*(*x*)›
have *Ord*(*csucc_rel*(*M*,$\aleph_x{}^M$))
using *Card_rel_is_Ord Card_rel_csucc_rel*
by *simp*
with ‹*Ord*(*x*)› ‹*M*(*x*)›
show *?case* **using** *Aleph_rel_succ* **by** *simp*
next
case (*limit x*)

note *trans=transM*[*OF* _ ‹$M(x)$›]
from *limit*
have $\aleph_x{}^M = (\bigcup i{\in}x.\ \aleph_i{}^M)$
using *Aleph_rel_cont OUnion_def Limit_is_Ord*
by *auto*
with *limit*
show *?case* **using** *Ord_UN trans* **by** *auto*
qed

lemma *Card_rel_Aleph_rel* [*simp*, *intro*]:
assumes *Ord*(a) **and** *types*: $M(a)$ **shows** $Card^M(\aleph_a{}^M)$
using *assms*
proof (*induct rule:trans_induct3*)
case *0*
then
show *?case*
using *Aleph_rel_zero Card_rel_nat* **by** *simp*
next
case (*succ x*)
then
show *?case*
using *Card_rel_csucc_rel Ord_Aleph_rel Aleph_rel_succ*
by *simp*
next
case (*limit x*)
moreover
from *this*
have $M(\{y\ .\ z \in x,\ M(y) \wedge M(z) \wedge Ord(z) \wedge y = \aleph_z{}^M\})$
using *aleph_rel_replacement*
by *auto*
moreover
have $\{y\ .\ z \in x,\ M(y) \wedge M(z) \wedge y = \aleph_z{}^M\} = \{y\ .\ z \in x,\ M(y) \wedge M(z) \wedge Ord(z) \wedge y = \aleph_z{}^M\}$
using *Ord_in_Ord Limit_is_Ord*[*OF limit(1)*] **by** *simp*
ultimately
show *?case*
using *Ord_Aleph_rel Card_nat Limit_is_Ord Card_relI*
by (*subst def_transrec* [*OF Aleph_rel_def'*])
(*auto simp add:HAleph_rel_def*)
qed

lemma *Aleph_rel_increasing*:
assumes *ab*: $a < b$ **and** *types*: $M(a)$ $M(b)$
shows $\aleph_a{}^M < \aleph_b{}^M$
proof -
{ fix x
have *Ord*(b) **using** *ab* **by** (*blast intro*: *lt_Ord2*)
moreover
assume $M(x)$

```
    moreover
    note ‹M(b)›
    ultimately
    have x < b ⟹ ℵ_x^M < ℵ_b^M
    proof (induct b arbitrary: x rule: trans_induct3)
      case 0 thus ?case by simp
    next
      case (succ b)
      then
      show ?case
      using Card_rel_csucc_rel Ord_Aleph_rel Ord_Union_succ_eq lt_csucc_rel
          lt_trans[of _ ℵ_b^M csucc^M(ℵ_b^M)]
        by (subst (2) def_transrec[OF Aleph_rel_def'])
          (auto simp add: le_iff HAleph_rel_def)
    next
      case (limit l)
      then
      have sc: succ(x) < l
        by (blast intro: Limit_has_succ)
      then
      have ℵ_x^M < (⋃j<l. ℵ_j^M)
        using limit Ord_Aleph_rel Ord_OUN
        apply (rule_tac OUN_upper_lt)
          apply (blast intro: Card_rel_is_Ord ltD lt_Ord)
      proof -
        from ‹x<l› ‹Limit(l)›
        have Ord(x)
          using Limit_is_Ord Ord_in_Ord
          by (auto dest!:ltD)
        with ‹M(x)›
        show ℵ_x^M < ℵ_succ(x)^M
          using Card_rel_csucc_rel Ord_Aleph_rel lt_csucc_rel
            ltD[THEN [2] Ord_in_Ord] succ_in_MI[OF ‹M(x)›]
            Aleph_rel_succ[of x]
          by (simp)
      next
        from ‹M(l)› ‹Limit(l)›
        show Ord(⋃j<l. ℵ_j^M)
          using Ord_Aleph_rel lt_Ord Limit_is_Ord Ord_in_Ord
          by (rule_tac Ord_OUN)
            (auto dest:transM ltD intro!:Ord_Aleph_rel)
      qed
      then
      show ?case using limit Aleph_rel_cont by simp
    qed
  }
  with types
  show ?thesis using ab by simp
qed
```

end

end

24 Cohen forcing notions

theory *Cohen_Posets*
  imports
    Forcing_Notions
    Names — only for *SepReplace*
    Recursion_Thms — only for the definition of *Rrel*
    Delta_System_Lemma.ZF_Library
begin

lemmas *app_fun* = *apply_iff*[*THEN iffD1*]

definition
  Fn :: [*i*,*i*,*i*] ⇒ *i* **where**
  Fn(κ,*I*,*J*) ≡ ⋃{(*d*→*J*) .. *d* ∈ *Pow*(*I*), *d*≺κ}

lemma *FnI*[*intro*]:
  assumes *p* : *d* → *J* *d* ⊆ *I* *d* ≺ κ
  shows *p* ∈ *Fn*(κ,*I*,*J*)
  using *assms Sep_and_Replace*
  unfolding *Fn_def* **by** *auto*

lemma *FnD*[*dest*]:
  assumes *p* ∈ *Fn*(κ,*I*,*J*)
  shows ∃ *d*. *p* : *d* → *J* ∧ *d* ⊆ *I* ∧ *d* ≺ κ
  using *assms Sep_and_Replace*
  unfolding *Fn_def* **by** *auto*

lemma *Fn_is_function*: *p* ∈ *Fn*(κ,*I*,*J*) ⟹ *function*(*p*)
  unfolding *Fn_def* **using** *Sep_and_Replace fun_is_function* **by** *auto*

lemma *Fn_csucc*:
  assumes *Ord*(κ)
  shows *Fn*(*csucc*(κ),*I*,*J*) = ⋃{(*d*→*J*) .. *d* ∈ *Pow*(*I*), *d*≲κ}
  using *assms*
  unfolding *Fn_def* **using** *lesspoll_csucc* **by** (*simp*)

lemma *Finite_imp_lesspoll_nat*:
  assumes *Finite*(*A*)
  shows *A* ≺ *nat*
  using *assms subset_imp_lepoll*[*OF naturals_subset_nat*] *eq_lepoll_trans*
    n_lesspoll_nat eq_lesspoll_trans
  unfolding *Finite_def lesspoll_def* **by** *auto*

```
lemma Fn_nat_eq_FiniteFun: Fn(nat,I,J) = I -||> J
  unfolding Fn_def
proof (intro equalityI subsetI)
  fix x
  assume x ∈ I -||> J
  then
  show x ∈ ⋃{(d→J) .. d ∈ Pow(I), d≺nat}
  proof (induct)
    case emptyI
    have 0: 0→J by simp
    moreover
    have |0|<nat using ltI by simp
    ultimately
    show ?case using lt_Card_imp_lesspoll Card_nat
      by (simp,rule_tac x=0→J in bexI)
        (auto | rule_tac x=0 in bexI)+
  next
    case (consI a b h)
    then
    obtain d where h:d→J d≺nat d⊆I by auto
    moreover from this
    have Finite(d)
      using lesspoll_nat_is_Finite by simp
    ultimately
    have h : d -||> J
      using fun_FiniteFunI Finite_into_Fin by blast
    note ‹h:d→J›
    moreover from this
    have domain(cons(⟨a, b⟩, h)) = cons(a,d) (is domain(?h) = ?d)
      and domain(h) = d
      using domain_of_fun by simp_all
    moreover
    note consI
    moreover from calculation
    have cons(⟨a, b⟩, h) ∈ cons(a,d) → J
      using fun_extend3 by simp
    moreover from ‹Finite(d)›
    have Finite(cons(a,d)) by simp
    moreover from this
    have cons(a,d) ≺ nat using Finite_imp_lesspoll_nat by simp
    ultimately
    show ?case by (simp,rule_tac x=?d→J in bexI)
        (force dest:app_fun)+
  qed
next
  fix x
  assume x ∈ ⋃{(d→J) .. d ∈ Pow(I),  d≺nat}
  then
```

obtain d **where** $x{:}d \to J$ $d \in Pow(I)$ $d \prec nat$ **by** *auto*
moreover from *this*
have $Finite(d)$
using *lesspoll_nat_is_Finite* **by** *simp*
moreover from *calculation*
have $d \in Fin(I)$
using *Finite_into_Fin*[*of d*] *Fin_mono* **by** *auto*
ultimately
show $x \in I \text{ -||> } J$ **using** *fun_FiniteFunI FiniteFun_mono* **by** *blast*
qed

definition
$FnleR :: i \Rightarrow i \Rightarrow o$ (**infixl** $\langle\supseteq\rangle$ *50*) **where**
$f \supseteq g \equiv g \subseteq f$

lemma *FnleR_iff_subset* [*iff*]: $f \supseteq g \longleftrightarrow g \subseteq f$
unfolding *FnleR_def* **..**

definition
$Fnlerel :: i \Rightarrow i$ **where**
$Fnlerel(A) \equiv Rrel(\lambda x\ y.\ x \supseteq y, A)$

definition
$Fnle :: [i,i,i] \Rightarrow i$ **where**
$Fnle(\kappa,I,J) \equiv Fnlerel(Fn(\kappa,I,J))$

lemma *FnleI*[*intro*]:
assumes $p \in Fn(\kappa,I,J)$ $q \in Fn(\kappa,I,J)$ $p \supseteq q$
shows $<p,q> \in Fnle(\kappa,I,J)$
using *assms* **unfolding** *Fnlerel_def Fnle_def FnleR_def Rrel_def*
by *auto*

lemma *FnleD*[*dest*]:
assumes $<p,q> \in Fnle(\kappa,I,J)$
shows $p \in Fn(\kappa,I,J)$ $q \in Fn(\kappa,I,J)$ $p \supseteq q$
using *assms* **unfolding** *Fnlerel_def Fnle_def FnleR_def Rrel_def*
by *auto*

locale *cohen_data* =
fixes κ I $J{::}i$
assumes *zero_lt_kappa*: $0<\kappa$
begin

lemmas *zero_lesspoll_kappa* = *zero_lesspoll*[*OF zero_lt_kappa*]

end

sublocale *cohen_data* $\subseteq$ *forcing_notion* $Fn(\kappa,I,J)$ $Fnle(\kappa,I,J)$ *0*
proof

show $0 \in Fn(\kappa,\ I,\ J)$
unfolding *Fn_def*
by (*simp*,*rule_tac* $x=0 \to J$ **in** *bexI*, *auto*)
(*rule_tac* $x=0$ **in** *bexI*, *auto intro*:*zero_lesspoll_kappa*)
then
show $\forall\, p{\in}Fn(\kappa,\ I,\ J).\ \langle p,\ 0\rangle \in Fnle(\kappa,\ I,\ J)$
unfolding *preorder_on_def refl_def trans_on_def*
by *auto*
next
show *preorder_on*($Fn(\kappa,\ I,\ J)$, $Fnle(\kappa,\ I,\ J)$)
unfolding *preorder_on_def refl_def trans_on_def*
by *blast*
qed

24.1 MOVE THIS to an appropriate place

definition
antichain :: $i{\Rightarrow}i{\Rightarrow}i{\Rightarrow}o$ **where**
$antichain(P,leq,A) \equiv A{\subseteq}P \wedge (\forall\, p{\in}A.\ \forall\, q{\in}A.$
$p{\neq}q \longrightarrow \neg compat\_in(P,leq,p,q))$
definition
ccc :: $i \Rightarrow i \Rightarrow o$ **where**
$ccc(P,leq) \equiv \forall\, A.\ antichain(P,leq,A) \longrightarrow |A| \le nat$

24.2 Combinatorial results on Cohen posets

context *cohen_data*
begin

lemma *restrict_eq_imp_compat*:
assumes $f \in Fn(nat,\ I,\ J)$ $g \in Fn(nat,\ I,\ J)$ $InfCard(nat)$
$restrict(f,\ domain(f) \cap domain(g)) = restrict(g,\ domain(f) \cap domain(g))$
shows $f \cup g \in Fn(nat,\ I,\ J)$
proof -
from *assms*
obtain *d1 d2* **where** $f : d1 \to J$ $d1 \in Pow(I)$ $d1 \prec nat$
$g : d2 \to J$ $d2 \in Pow(I)$ $d2 \prec nat$
by *blast*
with *assms*
show *?thesis*
using *domain_of_fun*
restrict_eq_imp_Un_into_Pi[*of f d1* $\lambda\_.\ J$ *g d2* $\lambda\_.\ J$]
by (*auto dest*!:*lesspoll_nat_is_Finite intro*!:*Finite_imp_lesspoll_nat*)
qed

lemma *compat_imp_Un_is_function*:
assumes $G \subseteq Fn(\kappa,\ I,\ J)$ $\bigwedge p\ q.\ p \in G \Longrightarrow q \in G \Longrightarrow compat(p,q)$
shows *function*($\bigcup G$)

unfolding *function_def*
proof (*intro allI ballI impI*)
fix x y y'
assume $\langle x, y\rangle \in \bigcup G$ $\langle x, y'\rangle \in \bigcup G$
then
obtain p q **where** $\langle x, y\rangle \in p$ $\langle x, y'\rangle \in q$ $p \in G$ $q \in G$
by *auto*
moreover from *this* **and** *assms*
obtain r **where** $r \in Fn(\kappa, I, J)$ $r \supseteq p$ $r \supseteq q$
using *compatD*[*of p q*] **by** *force*
moreover from *this*
have *function*(r)
using *Fn_is_function* **by** *simp*
ultimately
show $y = y'$
unfolding *function_def* **by** *fastforce*
qed

lemma *filter_subset_notion*: *filter*(G) $\Longrightarrow$ $G \subseteq Fn(\kappa, I, J)$
unfolding *filter_def* **by** *simp*

lemma *Un_filter_is_function*: *filter*(G) $\Longrightarrow$ *function*($\bigcup G$)
using *compat_imp_Un_is_function filter_imp_compat*[*of G*]
filter_subset_notion **by** *simp*

end

locale *add_reals* = *cohen_data nat _ 2*

end

25 Relativization of Finite Functions

theory *FiniteFun_Relative*
imports
Synthetic_Definition
Delta_System_Lemma.ZF_Library
Discipline_Function
Lambda_Replacement
Cohen_Posets

begin

25.1 The set of finite binary sequences

notation *nat* (‹ω›) — TODO: already in ZF Library

We implement the poset for adding one Cohen real, the set $2^{<\omega}$ of finite

binary sequences.

definition
seqspace :: $[i,i] \Rightarrow i$ ($\langle\_^{<\_}\rangle$ $[100,1]100$) **where**
$B^{<\alpha} \equiv \bigcup n{\in}\alpha.\ (n{\to}B)$

lemma *seqspaceI*[*intro*]: $n{\in}\alpha \Longrightarrow f{:}n{\to}B \Longrightarrow f{\in}B^{<\alpha}$
unfolding *seqspace_def* **by** *blast*

lemma *seqspaceD*[*dest*]: $f{\in}B^{<\alpha} \Longrightarrow \exists\, n{\in}\alpha.\ f{:}n{\to}B$
unfolding *seqspace_def* **by** *blast*

— FIXME: Now this is too particular (only for ω-sequences). A relative definition for *seqspace* would be appropriate.

locale *M_seqspace* = *M_trancl* + *M_replacement* +
assumes
seqspace_replacement: $M(B) \Longrightarrow$ *strong_replacement*(M,$\lambda n\ z.\ n{\in}nat \wedge$ *is_funspace*(M,n,B,z))
begin

lemma *seqspace_closed*:
$M(B) \Longrightarrow M(B^{<\omega})$
unfolding *seqspace_def* **using** *seqspace_replacement*[*of B*] *RepFun_closed2*
by *simp*

end

schematic_goal *seqspace_fm_auto*:
assumes
$i \in nat\ j \in nat\ h{\in}nat\ env \in list(A)$
shows
$(\exists\, om{\in}A.\ omega(\#\#A,om) \wedge nth(i,env) \in om \wedge is\_funspace(\#\#A,\ nth(i,env),$
$nth(h,env),\ nth(j,env))) \longleftrightarrow (A,\ env \models (?sqsprp(i,j,h)))$
unfolding *is_funspace_def*
by (*insert assms* ; (*rule iff_sats* | *simp*)+)
synthesize *seqspace_rel* **from_schematic** *seqspace_fm_auto*
arity_theorem for *seqspace_rel_fm*

25.2 Representation of finite functions

A function $f \in A \to_{fin} B$ can be represented by a function $g \in |f| \to A \times B$. It is clear that f can be represented by any $g' = g{\cdot}\pi$, where π is a permutation $\pi \in dom(g) \to dom(g)$. We use this representation of $A \to_{fin} B$ to prove that our model is closed under $\_ \to_{fin} \_$.

A function $g \in n \to A \times B$ that is functional in the first components.

definition *cons_like* :: $i \Rightarrow o$ **where**

cons_like(f) ≡ ∀ *i*∈*domain(f)* . ∀ *j*∈*i* . *fst(f‘i)* ≠ *fst(f‘j)*

relativize *cons_like cons_like_rel*

lemma (**in** *M_seqspace*) *cons_like_abs*:
M(f) ⟹ *cons_like(f)* ⟷ *cons_like_rel(M,f)*
unfolding *cons_like_def cons_like_rel_def*
using *fst_abs*
by *simp*

definition *FiniteFun_iso* :: [*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
FiniteFun_iso(A,B,n,g,f) ≡ (∀ *i*∈*n* . *g‘i* ∈ *f*) ∧ (∀ *ab*∈*f*. (∃ *i*∈*n*. *g‘i*=*ab*))

From a function $g \in n \to A \times B$ we obtain a finite function in A -||> B.

definition *to_FiniteFun* :: *i* ⇒ *i* **where**
to_FiniteFun(f) ≡ {*f‘i*. *i*∈*domain(f)*}

definition *FiniteFun_Repr* :: [*i*,*i*] ⇒ *i* **where**
FiniteFun_Repr(A,B) ≡ {*f* ∈ (*A*×*B*)^<ω . *cons_like(f)* }

locale *M_FiniteFun* = *M_seqspace* +
assumes
cons_like_separation : *separation(M,*λ*f*. *cons_like_rel(M,f))*
and
to_finiteFun_replacement: *strong_replacement(M,* λ*x y*. *y* = *range(x))*
and
supset_separation: *separation(M,* λ *x*. ∃ *a*. ∃ *b*. *x* = ⟨*a*,*b*⟩ ∧ *b* ⊆ *a)*
begin

lemma *fun_range_eq*: *f*∈*A*→*B* ⟹ {*f‘i* . *i*∈*domain(f)* } = *range(f)*
using *range_eq_image*[*of f*] *domain_of_fun image_fun func.apply_rangeI*
by *auto*

lemma *FiniteFun_fst_type*:
assumes *h*∈*A*-||>*B p*∈*h*
shows *fst(p)*∈*domain(h)*
using *assms*
by(*induct h*, *auto*)

lemma *FinFun_closed*:
M(A) ⟹ *M(B)* ⟹ *M*(⋃{*n*→*A*×*B* . *n*∈ω})
using *cartprod_closed seqspace_closed*
unfolding *seqspace_def* **by** *simp*

lemma *cons_like_lt* :
assumes *n*∈ω *f*∈*succ(n)*→*A*×*B cons_like(f)*
shows *restrict(f,n)*∈*n*→*A*×*B cons_like(restrict(f,n))*
using *assms*
proof (*auto simp add*: *le_imp_subset restrict_type2*)

```
  from ‹f∈_›
  have D:domain(restrict(f,n)) = n domain(f) = succ(n)
    using domain_of_fun domain_restrict by auto
  {
    fix i j
    assume i∈domain(restrict(f,n)) (is i∈?D) j∈i
    with ‹n∈_› D
    have j∈?D i∈n j∈n using Ord_trans[of j] by simp_all
    with D ‹cons_like(f)› ‹j∈n› ‹i∈n› ‹j∈i›
    have fst(restrict(f,n)‘i) ≠ fst(restrict(f,n)‘j)
      using restrict_if unfolding cons_like_def by auto
  }
  then show cons_like(restrict(f,n))
    unfolding cons_like_def by auto
qed
```

A finite function $f \in A$ -||> B can be represented by a function $g \in n \to A \times B$, with $n = |f|$.

```
lemma FiniteFun_iso_intro1:
  assumes f ∈ (A -||> B)
  shows ∃ n∈ω . ∃ g∈n→A×B. FiniteFun_iso(A,B,n,g,f) ∧ cons_like(g)
  using assms
proof(induct f,force simp add:emptyI FiniteFun_iso_def cons_like_def)
  case (consI a b h)
  then obtain n g where
    HI: n∈ω g∈n→A×B FiniteFun_iso(A,B,n,g,h) cons_like(g) by auto
  let ?G=λ i ∈ succ(n) . if i=n then <a,b> else g‘i
  from HI ‹a∈_› ‹b∈_›
  have G: ?G ∈ succ(n)→A×B
    by (auto intro:lam_type)
  have FiniteFun_iso(A,B,succ(n),?G,cons(<a,b>,h))
    unfolding FiniteFun_iso_def
  proof(intro conjI)
    {
      fix i
      assume i∈succ(n)
      then consider i=n | i∈n∧i≠n by auto
      then have ?G ‘ i ∈ cons(<a,b>,h)
        using HI
        by(cases,simp;auto simp add:HI FiniteFun_iso_def)
    }
    then show ∀ i∈succ(n). ?G ‘ i ∈ cons(⟨a, b⟩, h) ..
  next
    { fix ab′
      assume ab′ ∈ cons(<a,b>,h)
      then
      consider  ab′ = <a,b> | ab′ ∈ h using cons_iff by auto
      then
      have ∃ i ∈ succ(n) . ?G‘i = ab′ unfolding FiniteFun_iso_def
```

```
      proof(cases,simp)
        case 2
        with HI obtain i
          where i∈n g‘i=ab' unfolding FiniteFun_iso_def by auto
        with HI show ?thesis using  ltI[OF ‹i∈_›] by auto
      qed
    }
    then
    show ∀ ab∈cons(⟨a, b⟩, h). ∃ i∈succ(n). ?G‘i = ab  ..
  qed
  with HI G
 have 1: ?G∈succ(n)→A×B FiniteFun_iso(A,B,succ(n),?G,cons(<a,b>,h)) succ(n)∈ω
by simp_all
  have cons_like(?G)
  proof -
    from ‹?G∈_› ‹g∈_›
    have domain(g) = n using domain_of_fun by simp
    {
      fix i j
      assume i∈domain(?G) j∈i
      with ‹n∈_›
      have j∈n using Ord_trans[of j _ n] by auto
      from ‹i∈_› consider (a) i=n ∧ i∉n | (b) i∈n by auto
      then
      have  fst(?G‘i) ≠ fst(?G‘j)
      proof(cases)
        case a
        with ‹j∈n› HI
        have ?G‘i=<a,b> ?G‘j=g‘j g‘j∈h
          unfolding FiniteFun_iso_def by auto
        with ‹a∉_› ‹h∈_›
        show ?thesis using  FiniteFun_fst_type by auto
      next
        case b
        with ‹i∈n› ‹j∈i› ‹j∈n› HI ‹domain(g) = n›
        show ?thesis unfolding cons_like_def
          using mem_not_refl by auto
      qed
    }
    then show ?thesis unfolding cons_like_def by auto
  qed
  with 1 show ?case by auto
qed
```

All the representations of $f \in A$ -||> B are equal.

```
lemma FiniteFun_isoD :
  assumes n∈ω g∈n→A×B f∈A-||>B FiniteFun_iso(A,B,n,g,f)
  shows to_FiniteFun(g) = f
proof
```

```
  show to_FiniteFun(g) ⊆ f
  proof
    fix ab
    assume ab∈to_FiniteFun(g)
    moreover
    note assms
    moreover from calculation
    obtain i where i∈n g‘i=ab ab∈A×B
      unfolding to_FiniteFun_def using domain_of_fun by auto
    ultimately
    show ab∈f unfolding FiniteFun_iso_def by auto
  qed
next
  show f ⊆ to_FiniteFun(g)
  proof
    fix ab
    assume ab∈f
    with assms
    obtain i where i∈n g‘i=ab ab∈A×B
      unfolding FiniteFun_iso_def by auto
    with assms
    show ab ∈ to_FiniteFun(g)
      unfolding to_FiniteFun_def
      using domain_of_fun by auto
  qed
qed

lemma to_FiniteFun_succ_eq :
  assumes n∈ω f∈succ(n) → A
  shows to_FiniteFun(f) = cons(f‘n,to_FiniteFun(restrict(f,n)))
  using assms domain_restrict domain_of_fun
  unfolding to_FiniteFun_def by auto
```

If $g \in n \to A \times B$ is *cons_like*, then it is a representation of *to_FiniteFun*(g).

```
lemma FiniteFun_iso_intro_to:
  assumes n∈ω g∈n→A×B cons_like(g)
  shows to_FiniteFun(g) ∈ (A -||> B) ∧ FiniteFun_iso(A,B,n,g,to_FiniteFun(g))
  using assms
proof(induct n  arbitrary:g rule:nat_induct)
  case 0
  fix g
  assume g∈0→A×B
  then
  have g=0 by simp
  then have to_FiniteFun(g)=0 unfolding to_FiniteFun_def by simp
 then show to_FiniteFun(g) ∈ (A -||> B) ∧ FiniteFun_iso(A,B,0,g,to_FiniteFun(g))
    using emptyI unfolding FiniteFun_iso_def by simp
next
  case (succ x)
```

fix *g*
let *?g′=restrict(g,x)*
assume *g∈succ(x)→A×B cons_like(g)*
with *succ.hyps* ‹*g∈_*›
have *cons_like(?g′) ?g′ ∈ x→A×B g‘x∈A×B domain(g) = succ(x)*
using *cons_like_lt succI1 apply_funtype domain_of_fun* **by** *simp_all*
with *succ.hyps* ‹*?g′∈_*› ‹*x∈ω*›
have *HI*:
to_FiniteFun(?g′) ∈ A -||> B (**is** *(?h) ∈ _*)
FiniteFun_iso(A,B,x,?g′,to_FiniteFun(?g′))
by *simp_all*
then
have *fst(g‘x) ∉ domain(?h)*
proof -
{
assume *fst(g‘x) ∈ domain(?h)*
with *HI* ‹*x∈_*›
obtain *i b*
where *i∈x <fst(?g′‘i),b>∈?h i<x fst(g‘x) = fst(?g′‘i)*
unfolding *FiniteFun_iso_def* **using** *ltI* **by** *auto*
with ‹*cons_like(g)*› ‹*domain(g) = _*›
have *False*
unfolding *cons_like_def* **by** *auto*
}
then show *?thesis* **..**
qed
with *HI assms* ‹*g‘x∈_*›
have *cons(g‘x,?h) ∈ A-||>B* (**is** *?h′ ∈_*) **using** *consI* **by** *auto*
have *FiniteFun_iso(A,B,succ(x),g,?h′)*
unfolding *FiniteFun_iso_def*
proof
{ **fix** *i*
assume *i∈succ(x)*
with ‹*x∈_*› **consider** (*a*) *i=x*| (*b*) *i∈x∧i≠x* **by** *auto*
then have *g‘i∈ ?h′*
proof(*cases,simp*)
case *b*
with ‹*FiniteFun_iso(_,_,_,?g′,?h)*›
show *?thesis* **unfolding** *FiniteFun_iso_def* **by** *simp*
qed
}
then show ∀ *i∈succ(x). g ‘ i ∈ cons(g ‘ x, ?h)* **..**
next
{
fix *ab*
assume *ab∈?h′*
then consider *ab=g‘x* | *ab ∈ ?h* **using** *cons_iff* **by** *auto*
then
have ∃ *i ∈ succ(x) . g‘i = ab* **unfolding** *FiniteFun_iso_def*

```
      proof(cases,simp)
        case 2
        with HI obtain i
          where 2:i∈x ?g'‘i=ab  unfolding FiniteFun_iso_def by auto
        with ‹x∈_›
        have i≠x i∈succ(x) using  ltI[OF ‹i∈_›] by auto
        with 2 HI show ?thesis by auto
      qed
    } then show ∀ ab∈cons(g ‘ x, ?h). ∃ i∈succ(x). g ‘ i = ab ..
  qed
  with ‹?h'∈_›
 show to_FiniteFun(g) ∈ A -||>B ∧ FiniteFun_iso(A,B,succ(x),g,to_FiniteFun(g))
    using to_FiniteFun_succ_eq[OF ‹x∈_› ‹g∈_›,symmetric] by auto
qed

lemma FiniteFun_iso_intro2:
  assumes n∈ω f∈n→A×B cons_like(f)
  shows ∃ g ∈ (A -||> B) . FiniteFun_iso(A,B,n,f,g)
  using assms FiniteFun_iso_intro_to by blast

lemma FiniteFun_eq_range_Repr :
   shows {range(h) . h ∈ FiniteFun_Repr(A,B) } = {to_FiniteFun(h) . h ∈
FiniteFun_Repr(A,B) }
  unfolding FiniteFun_Repr_def to_FiniteFun_def seqspace_def
  using fun_range_eq
  by(intro equalityI subsetI,auto)


lemma FiniteFun_eq_to_FiniteFun_Repr :
  shows A-||>B = {to_FiniteFun(h) . h ∈ FiniteFun_Repr(A,B) }
    (is ?Y=?X)
proof
  {
    fix f
    assume f∈A-||>B
    then obtain n g where
      1: n∈ω g∈n→A×B FiniteFun_iso(A,B,n,g,f) cons_like(g)
      using FiniteFun_iso_intro1 by blast
    with ‹f∈_›
   have cons_like(g) f=to_FiniteFun(g) domain(g) = n g∈FiniteFun_Repr(A,B)
      using FiniteFun_isoD domain_of_fun
      unfolding FiniteFun_Repr_def
      by auto
    with 1 have f∈?X
      by auto
  } then show ?Y⊆?X ..
next
  {
    fix f
```

```
    assume f∈?X
    then obtain g where
      A:g∈FiniteFun_Repr(A,B) f=to_FiniteFun(g) cons_like(g)
      using RepFun_iff unfolding FiniteFun_Repr_def by auto
    then obtain n where n∈ω g∈n→A×B domain(g) = n
      unfolding FiniteFun_Repr_def using domain_of_fun by force
    with A
    have f∈?Y
      using FiniteFun_iso_intro_to by simp
  } then show ?X⊆?Y ..
qed

lemma FiniteFun_Repr_closed :
  assumes M(A) M(B)
  shows M(FiniteFun_Repr(A,B))
  unfolding FiniteFun_Repr_def
  using assms cartprod_closed
    seqspace_closed separation_closed cons_like_abs cons_like_separation
  by simp

lemma to_FiniteFun_closed:
  assumes M(A) f∈A
  shows M(range(f))
  using assms transM[of _ A] by simp

lemma To_FiniteFun_Repr_closed :
  assumes M(A) M(B)
  shows M({range(h) . h ∈ FiniteFun_Repr(A,B) })
  using assms FiniteFun_Repr_closed
    RepFun_closed  to_finiteFun_replacement
    to_FiniteFun_closed[OF FiniteFun_Repr_closed]
  by simp

lemma FiniteFun_closed[intro,simp] :
  assumes M(A) M(B)
  shows M(A -||> B)
  using assms To_FiniteFun_Repr_closed FiniteFun_eq_to_FiniteFun_Repr
    FiniteFun_eq_range_Repr
  by simp

lemma Fnle_nat_closed[intro,simp]:
  assumes M(I) M(J)
  shows M(Fnle(ω,I,J))
  unfolding Fnle_def Fnlerel_def Rrel_def
 using supset_separation FiniteFun_closed Fn_nat_eq_FiniteFun assms by simp

end

end
```

26 Relative, Cardinal Arithmetic Using AC

theory *Cardinal_AC_Relative*
imports
ZF_Miscellanea
Interface
CardinalArith_Relative

begin

locale *M_AC* =
fixes *M*
assumes
choice_ax: *choice_ax*(*M*)

locale *M_cardinal_AC* = *M_cardinal_arith* + *M_AC*
begin

lemma *well_ord_surj_imp_lepoll_rel*:
assumes *well_ord*(*A*,*r*) *h* ∈ *surj*(*A*,*B*) **and**
types:*M*(*A*) *M*(*r*) *M*(*h*) *M*(*B*)
shows *B* $\lesssim^{M}$ *A*
proof -
note *eq*=*vimage_fun_sing*[*OF surj_is_fun*[*OF* ‹*h*∈_›]]
from *assms*
have (λ*b*∈*B*. *minimum*(*r*, {*a*∈*A*. *h*‘*a*=*b*})) ∈ *inj*(*B*,*A*) (**is** *?f*∈_)
using *well_ord_surj_imp_inj_inverse assms*(*1*,*2*) **by** *simp*
with *assms*
have *M*(*?f*‘*b*) **if** *b*∈*B* **for** *b*
using *apply_type*[*OF inj_is_fun*[*OF* ‹*?f*∈_›]] *that transM*[*OF* _ ‹*M*(*A*)›] **by**
simp
with *assms*
have *M*(*?f*)
using *lam_closed surj_imp_inj_replacement*(*4*) *eq* **by** *auto*
with ‹*?f*∈_› *assms*
have *?f* ∈ inj^{M}(*B*,*A*)
using *mem_inj_abs* **by** *simp*
with ‹*M*(*?f*)›
show *?thesis* **unfolding** *lepoll_rel_def* **by** *auto*
qed

lemma *surj_imp_well_ord_M*:
assumes *wos*: *well_ord*(*A*,*r*) *h* ∈ *surj*(*A*,*B*)
and
types: *M*(*A*) *M*(*r*) *M*(*h*) *M*(*B*)
shows ∃ *s*[*M*]. *well_ord*(*B*,*s*)
using *assms lepoll_rel_well_ord*
well_ord_surj_imp_lepoll_rel **by** *fast*

lemma *choice_ax_well_ord*: $M(S) \Longrightarrow \exists\, r[M].\ well\_ord(S,r)$
using *choice_ax well_ord_Memrel*[*THEN surj_imp_well_ord_M*]
unfolding *choice_ax_def* **by** *auto*

end

locale *M_Pi_assumptions_choice* = *M_Pi_assumptions* + *M_cardinal_AC* +
assumes
B_replacement: *strong_replacement*(M, $\lambda x\ y.\ y = B(x)$)
and
— The next one should be derivable from (some variant) of B_replacement. Proving both instances each time seems inconvenient.
minimum_replacement: $M(r) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ y.\ y = \langle x, minimum(r, B(x))\rangle$)
begin

lemma *AC_M*:
assumes $a \in A$ $\bigwedge x.\ x \in A \Longrightarrow \exists\, y.\ y \in B(x)$
shows $\exists\, z[M].\ z \in Pi^M(A,\ B)$
proof -
have $M(\bigcup x{\in}A.\ B(x))$ **using** *assms family_union_closed Pi_assumptions B_replacement* **by** *simp*
then
obtain r **where** *well_ord*($\bigcup x{\in}A.\ B(x)$,r) $M(r)$
using *choice_ax_well_ord* **by** *blast*
let *?f*=$\lambda x{\in}A.\ minimum(r,B(x))$
have $M(minimum(r,\ B(x)))$ **if** $x{\in}A$ **for** x
proof -
from ‹*well_ord*(_,r)› ‹$x{\in}A$›
have *well_ord*($B(x)$,r) **using** *well_ord_subset UN_upper* **by** *simp*
with *assms* ‹$x{\in}A$› ‹$M(r)$›
show *?thesis* **using** *Pi_assumptions* **by** *blast*
qed
with *assms* **and** ‹$M(r)$›
have $M(?f)$
using *Pi_assumptions minimum_replacement lam_closed*
by *simp*
moreover from *assms* **and** *calculation*
have *?f* $\in Pi^M(A,B)$
using *lam_type*[*OF minimum_in*, *OF* ‹*well_ord*($\bigcup x{\in}A.\ B(x)$,r)›, *of A B*]
Pi_rel_char **by** *auto*
ultimately
show *?thesis* **by** *blast*
qed

lemma *AC_Pi_rel*: **assumes** $\bigwedge x.\ x \in A \Longrightarrow \exists\, y.\ y \in B(x)$
shows $\exists\, z[M].\ z \in Pi^M(A,\ B)$
proof (*cases A=0*)

interpret *Pi0*:*M_Pi_assumptions_0*
using *Pi_assumptions* **by** *unfold_locales auto*
case *True*
then
show *?thesis* **using** *assms* **by** *simp*
next
case *False*
then
obtain *a* **where** $a \in A$ **by** *auto*
— It is noteworthy that without obtaining an element of A, the final step won't work
with *assms*
show *?thesis* **by** (*blast intro*!: *AC_M*)
qed

end

context *M_cardinal_AC*
begin

26.1 Strengthened Forms of Existing Theorems on Cardinals

lemma *cardinal_rel_eqpoll_rel*: $M(A) \Longrightarrow |A|^M \approx^M A$
apply (*rule choice_ax_well_ord* [*THEN rexE*])
apply (*auto intro*:*well_ord_cardinal_rel_eqpoll_rel*)
done

lemmas *cardinal_rel_idem* = *cardinal_rel_eqpoll_rel* [*THEN cardinal_rel_cong*, *simp*]

lemma *cardinal_rel_eqE*: $|X|^M = |Y|^M ==> M(X) \Longrightarrow M(Y) \Longrightarrow X \approx^M Y$
apply (*rule choice_ax_well_ord* [*THEN rexE*], *assumption*)
apply (*rule choice_ax_well_ord* [*THEN rexE*, *of Y*], *assumption*)
apply (*rule well_ord_cardinal_rel_eqE*, *assumption*+)
done

lemma *cardinal_rel_eqpoll_rel_iff*: $M(X) \Longrightarrow M(Y) \Longrightarrow |X|^M = |Y|^M \longleftrightarrow X \approx^M Y$
by (*blast intro*: *cardinal_rel_cong cardinal_rel_eqE*)

lemma *cardinal_rel_disjoint_Un*:
[| $|A|^M$=$|B|^M$; $|C|^M$=$|D|^M$; $A \cap C = 0$; $B \cap D = 0$; $M(A)$; $M(B)$; $M(C)$; $M(D)$|]
==> $|A \cup C|^M = |B \cup D|^M$
by (*simp add*: *cardinal_rel_eqpoll_rel_iff eqpoll_rel_disjoint_Un*)

lemma *lepoll_rel_imp_cardinal_rel_le*: $A \lesssim^M B ==> M(A) \Longrightarrow M(B) \Longrightarrow |A|^M \leq |B|^M$

apply (*rule choice_ax_well_ord* [*THEN rexE*]) **prefer** *2*
apply (*erule well_ord_lepoll_rel_imp_cardinal_rel_le*, *assumption*+)
done

lemma *cadd_rel_assoc*: $[\![M(i);\ M(j);\ M(k)]\!] \Longrightarrow (i \oplus^M j) \oplus^M k = i \oplus^M (j \oplus^M k)$
apply (*rule choice_ax_well_ord* [*THEN rexE*]) **prefer** *2*
apply (*rule choice_ax_well_ord* [*THEN rexE*]) **prefer** *2*
apply (*rule choice_ax_well_ord* [*THEN rexE*]) **prefer** *2*
apply (*rule well_ord_cadd_rel_assoc*, *assumption*+)
done

lemma *cmult_rel_assoc*: $[\![M(i);\ M(j);\ M(k)]\!] \Longrightarrow (i \otimes^M j) \otimes^M k = i \otimes^M (j \otimes^M k)$
apply (*rule choice_ax_well_ord* [*THEN rexE*]) **prefer** *2*
apply (*rule choice_ax_well_ord* [*THEN rexE*]) **prefer** *2*
apply (*rule choice_ax_well_ord* [*THEN rexE*]) **prefer** *2*
apply (*rule well_ord_cmult_rel_assoc*, *assumption*+)
done

lemma *cadd_cmult_distrib*: $[\![M(i);\ M(j);\ M(k)]\!] \Longrightarrow (i \oplus^M j) \otimes^M k = (i \otimes^M k) \oplus^M (j \otimes^M k)$
apply (*rule choice_ax_well_ord* [*THEN rexE*]) **prefer** *2*
apply (*rule choice_ax_well_ord* [*THEN rexE*]) **prefer** *2*
apply (*rule choice_ax_well_ord* [*THEN rexE*]) **prefer** *2*
apply (*rule well_ord_cadd_cmult_distrib*, *assumption*+)
done

lemma *InfCard_rel_square_eq*: $InfCard^M(|A|^M) \Longrightarrow M(A) \Longrightarrow A \times A \approx^M A$
apply (*rule choice_ax_well_ord* [*THEN rexE*]) **prefer** *2*
apply (*erule well_ord_InfCard_rel_square_eq*, *assumption*, *simp_all*)
done

26.2 The relationship between cardinality and le-pollence

lemma *Card_rel_le_imp_lepoll_rel*:
assumes $|A|^M \leq |B|^M$
and *types*: $M(A)$ $M(B)$
shows $A \lesssim^M B$
proof -
have $A \approx^M |A|^M$
by (*rule cardinal_rel_eqpoll_rel* [*THEN eqpoll_rel_sym*], *simp_all add*:*types*)
also have ... $\lesssim^M |B|^M$
by (*rule le_imp_subset* [*THEN subset_imp_lepoll_rel*]) (*rule assms*, *simp_all add*:*types*)
also have ... $\approx^M B$
by (*rule cardinal_rel_eqpoll_rel*, *simp_all add*:*types*)
finally show *?thesis* **by** (*simp_all add*:*types*)

qed

lemma *le_Card_rel_iff*: $Card^M(K) ==> M(K) \Longrightarrow M(A) \Longrightarrow |A|^M \leq K \longleftrightarrow A \lesssim^M K$
apply (*erule Card_rel_cardinal_rel_eq* [*THEN subst*], *assumption*, *rule iffI*,
    erule Card_rel_le_imp_lepoll_rel, *assumption*+)
apply (*erule lepoll_rel_imp_cardinal_rel_le*, *assumption*+)
done

lemma *cardinal_rel_0_iff_0* [*simp*]: $M(A) \Longrightarrow |A|^M = 0 \longleftrightarrow A = 0$
  using *cardinal_rel_0 eqpoll_rel_0_iff* [*THEN iffD1*]
    cardinal_rel_eqpoll_rel_iff [*THEN iffD1*, *OF _ nonempty*]
  by *auto*

lemma *cardinal_rel_lt_iff_lesspoll_rel*:
  assumes *i*: *Ord*(*i*) **and**
    types: $M(i)$ $M(A)$
  shows $i < |A|^M \longleftrightarrow i \prec^M A$
proof
  assume $i < |A|^M$
  hence $i \prec^M |A|^M$
    by (*blast intro*: *lt_Card_rel_imp_lesspoll_rel types*)
  also have ... $\approx^M A$
    by (*rule cardinal_rel_eqpoll_rel*) (*simp_all add*:*types*)
  finally show $i \prec^M A$ **by** (*simp_all add*:*types*)
next
  assume $i \prec^M A$
  also have ... $\approx^M |A|^M$
    by (*blast intro*: *cardinal_rel_eqpoll_rel eqpoll_rel_sym types*)
  finally have $i \prec^M |A|^M$ **by** (*simp_all add*:*types*)
  thus $i < |A|^M$ **using** *i types*
    by (*force intro*: *cardinal_rel_lt_imp_lt lesspoll_rel_cardinal_rel_lt*)
qed

lemma *cardinal_rel_le_imp_lepoll_rel*: $i \leq |A|^M ==> M(i) \Longrightarrow M(A) \Longrightarrow i \lesssim^M A$
  by (*blast intro*: *lt_Ord Card_rel_le_imp_lepoll_rel Ord_cardinal_rel_le le_trans*)

26.3 Other Applications of AC

We have an example of instantiating a locale involving higher order variables inside a proof, by using the assumptions of the first order, active locale.

lemma *surj_rel_implies_inj_rel*:
  assumes *f*: $f \in surj^M(X,Y)$ **and**
    types: $M(f)$ $M(X)$ $M(Y)$
  shows $\exists\, g[M].\ g \in inj^M(Y,X)$
proof -
  from *types*
  interpret *M_Pi_assumptions_choice _ Y* $\lambda y.$ *f-‘‘*$\{y\}$

by *unfold_locales* (*auto intro*:*surj_imp_inj_replacement dest*:*transM*)
from *f AC_Pi_rel*
obtain *z* **where** *z*: $z \in Pi^M(Y,\ \lambda y.\ f\ \text{-``}\ \{y\})$
— In this and the following ported result, it is not clear how uniformly are "_char" theorems to be used
using *surj_rel_char*
by (*auto simp add*: *surj_def types*) (*fast dest*: *apply_Pair*)
show *?thesis*
proof
show $z \in inj^M(Y,\ X)\ M(z)$
using *z surj_is_fun*[*of f X Y*] *f Pi_rel_char*
by (*auto dest*: *apply_type Pi_memberD*
intro: *apply_equality Pi_type f_imp_injective*
simp add:*types mem_surj_abs*)
qed
qed

Kunen's Lemma 10.20

lemma *surj_rel_implies_cardinal_rel_le*:
assumes *f*: $f \in surj^M(X,Y)$ **and**
types:$M(f)\ M(X)\ M(Y)$
shows $|Y|^M \le |X|^M$
proof (*rule lepoll_rel_imp_cardinal_rel_le*)
from *f* [*THEN surj_rel_implies_inj_rel*]
obtain *g* **where** $g \in inj^M(Y,X)$
by (*blast intro*:*types*)
then
show $Y \lesssim^M X$
using *inj_rel_char*
by (*auto simp add*: *def_lepoll_rel types*)
qed (*simp_all add*:*types*)

end

The set-theoretic universe.

abbreviation
Universe :: *i*⇒*o* ($\langle\mathcal{V}\rangle$) **where**
$\mathcal{V}(x) \equiv True$

lemma *separation_absolute*: *separation*($\mathcal{V}$, *P*)
unfolding *separation_def*
by (*rule rallI*, *rule_tac x*=$\{x\in\_\ .\ P(x)\}$ **in** *rexI*) *auto*

lemma *univalent_absolute*:
assumes *univalent*($\mathcal{V}$, *A*, *P*) $P(x,\ b)\ x \in A$
shows $P(x,\ y) \Longrightarrow y = b$
using *assms*
unfolding *univalent_def* **by** *force*

lemma *replacement_absolute*: *strong_replacement*($\mathcal{V}$, P)
unfolding *strong_replacement_def*
proof (*intro rallI impI*)
fix A
assume *univalent*($\mathcal{V}$, A, P)
then
show $\exists\, Y[\mathcal{V}].\ \forall\, b[\mathcal{V}].\ b \in Y \longleftrightarrow (\exists\, x[\mathcal{V}].\ x \in A \wedge P(x,\ b))$
by (*rule_tac* x=$\{y.\ x$∈$A\ ,\ P(x,y)\}$ **in** *rexI*)
(*auto dest*:*univalent_absolute*[*of* _ P])
qed

lemma *Union_ax_absolute*: *Union_ax*($\mathcal{V}$)
unfolding *Union_ax_def big_union_def*
by (*auto intro*:*rexI*[*of* _ $\bigcup$_])

lemma *upair_ax_absolute*: *upair_ax*($\mathcal{V}$)
unfolding *upair_ax_def upair_def rall_def rex_def*
by (*auto*)

lemma *power_ax_absolute*:*power_ax*($\mathcal{V}$)
proof -
{
fix x
have $\forall\, y[\mathcal{V}].\ y \in Pow(x) \longleftrightarrow (\forall\, z[\mathcal{V}].\ z \in y \longrightarrow z \in x)$
by *auto*
}
then
show *power_ax*($\mathcal{V}$)
unfolding *power_ax_def powerset_def subset_def* **by** *blast*
qed

locale *M_cardinal_UN* = *M_Pi_assumptions_choice* _ K X **for** K X +
assumes
— The next assumption is required by ($\bigwedge x.\ [\![?Q(x);\ Ord(x)]\!] \Longrightarrow \exists\, y[M].\ ?Q(y)$
$\wedge$ $Ord(y)) \Longrightarrow M(\mu\ x.\ ?Q(x))$
X_witness_in_M: $w \in X(x) \Longrightarrow M(x)$
and
lam_m_replacement:$M(f) \Longrightarrow$ *strong_replacement*(M,
$\lambda x\ y.\ y = \langle x,\ \mu\ i.\ x \in X(i),\ f\ `\ (\mu\ i.\ x \in X(i))\ `\ x\rangle$)
and
inj_replacement:
$M(x) \Longrightarrow$ *strong_replacement*(M, $\lambda y\ z.\ y \in inj^M(X(x),\ K) \wedge z = \{\langle x,\ y\rangle\}$)
strong_replacement(M, $\lambda x\ y.\ y = inj^M(X(x),\ K)$)
strong_replacement(M,
$\lambda x\ z.\ z = Sigfun(x,\ \lambda i.\ inj^M(X(i),\ K))$)
$M(r) \Longrightarrow$ *strong_replacement*(M,
$\lambda x\ y.\ y = \langle x,\ minimum(r,\ inj^M(X(x),\ K))\rangle$)

begin

lemma *UN_closed*: $M(\bigcup i{\in}K.\ X(i))$
using *family_union_closed B_replacement Pi_assumptions* **by** *simp*

Kunen's Lemma 10.21

lemma *cardinal_rel_UN_le*:
assumes K: $\mathit{InfCard}^M(K)$
shows $(\bigwedge i.\ i{\in}K \Longrightarrow |X(i)|^M \le K) \Longrightarrow |\bigcup i{\in}K.\ X(i)|^M \le K$
proof (*simp add*: *K InfCard_rel_is_Card_rel le_Card_rel_iff Pi_assumptions*)
have $M(f) \Longrightarrow M(\lambda x{\in}(\bigcup x{\in}K.\ X(x)).\ \langle \mu\ i.\ x \in X(i),\ f\ `\ (\mu\ i.\ x \in X(i))\ `\ x\rangle)$
for f
using *lam_m_replacement X_witness_in_M Least_closed' Pi_assumptions UN_closed*
by (*rule_tac lam_closed*) (*auto dest*:*transM*)
note *types* = *this Pi_assumptions UN_closed*
have [*intro*]: $\mathit{Ord}(K)$ **by** (*blast intro*: *InfCard_rel_is_Card_rel Card_rel_is_Ord K types*)
interpret *pii*:*M_Pi_assumptions_choice* _ K $\lambda i.\ \mathit{inj}^M(X(i),\ K)$
using *inj_replacement Pi_assumptions transM*[*of* _ K]
by *unfold_locales* (*simp_all del*:*mem_inj_abs*)
assume *asm*:$\bigwedge i.\ i{\in}K \Longrightarrow X(i) \lesssim^M K$
then
have $\bigwedge i.\ i{\in}K \Longrightarrow M(\mathit{inj}^M(X(i),\ K))$
by (*auto simp add*: *types*)
interpret *V*:*M_N_Perm M* $\mathcal{V}$
using *separation_absolute replacement_absolute Union_ax_absolute power_ax_absolute upair_ax_absolute*
by *unfold_locales auto*
note *bad_simps*[*simp del*] = *V.N.Forall_in_M_iff V.N.Equal_in_M_iff V.N.nonempty*
have *abs*:*inj_rel*($\mathcal{V}$,x,y) = *inj*(x,y) **for** x y
using *V.N.inj_rel_char* **by** *simp*
from *asm*
have $\bigwedge i.\ i{\in}K \Longrightarrow \exists f[M].\ f \in \mathit{inj}^M(X(i),\ K)$
by (*simp add*: *types def_lepoll_rel*)
then
obtain f **where** $f \in (\prod i{\in}K.\ \mathit{inj}^M(X(i),\ K))\ M(f)$
using *pii.AC_Pi_rel pii.Pi_rel_char* **by** *auto*
with *abs*
have *f*:$f \in (\prod i{\in}K.\ \mathit{inj}(X(i),\ K))$
using *Pi_weaken_type*[*OF* _ *V.inj_rel_transfer*, *of f K X* λ_. K] *Pi_assumptions* **by** *simp*
{ fix z
assume z: $z \in (\bigcup i{\in}K.\ X(i))$
then obtain i **where** i: $i \in K$ $\mathit{Ord}(i)$ $z \in X(i)$
by (*blast intro*: *Ord_in_Ord* [*of K*])
hence $(\mu\ i.\ z \in X(i)) \le i$ **by** (*fast intro*: *Least_le*)
hence $(\mu\ i.\ z \in X(i)) < K$ **by** (*best intro*: *lt_trans1 ltI i*)
hence $(\mu\ i.\ z \in X(i)) \in K$ **and** $z \in X(\mu\ i.\ z \in X(i))$

```
    by (auto intro: LeastI ltD i)
  } note mems = this
  have (⋃i∈K. X(i)) ≲^M K × K
    proof (simp add:types def_lepoll_rel)
      show ∃f[M]. f ∈ inj(⋃x∈K. X(x), K × K)
        apply (rule rexI)
        apply (rule_tac c = λz. ⟨μ i. z ∈ X(i), f ‘ (μ i. z ∈ X(i)) ‘ z⟩
                    and d = λ⟨i,j⟩. converse (f‘i) ‘ j in lam_injective)
          apply (force intro: f inj_is_fun mems apply_type Perm.left_inverse)+
        apply (simp add:types ‹M(f)›)
        done
    qed
    also have ... ≈^M K
    by (simp add: K InfCard_rel_square_eq InfCard_rel_is_Card_rel
        Card_rel_cardinal_rel_eq types)
  finally have (⋃i∈K. X(i)) ≲^M K by (simp_all add:types)
  then
  show ?thesis
    by (simp add: K InfCard_rel_is_Card_rel le_Card_rel_iff types)
qed



end

end
```

27 Library of basic *ZF* results

```
theory ZF_Library_Relative
  imports
    Delta_System_Lemma.ZF_Library
    ZF-Constructible.Normal
    Aleph_Relative— must be before Cardinal_AC_Relative!
    Lambda_Replacement
    Cardinal_AC_Relative
    FiniteFun_Relative
begin

lemma (in M_cardinal_arith_jump) csucc_rel_cardinal_rel:
  assumes Ord(κ) M(κ)
  shows (|κ|^M+)^M = (κ^+)^M
proof (intro le_anti_sym)— show both inequalities
  from assms
  have hips:M((κ^+)^M) Ord((κ^+)^M) κ < (κ^+)^M
    using Card_rel_csucc_rel[THEN Card_rel_is_Ord]
      csucc_rel_basic by simp_all
  then
  show (|κ|^M+)^M ≤ (κ^+)^M
```

```
    using Ord_cardinal_rel_le[THEN lt_trans1]
      Card_rel_csucc_rel
    unfolding csucc_rel_def
    by (rule_tac Least_antitone) (assumption, simp_all add:assms)
  from assms
  have κ < L if Card^M(L) |κ|^M < L M(L) for L
    using  that
      Card_rel_is_Ord leI Card_rel_le_iff[of κ L]
    by (rule_tac ccontr, auto dest:not_lt_imp_le) (fast dest: le_imp_not_lt)
  with hips
  show (κ⁺)^M ≤ (|κ|^M+)^M
    using Ord_cardinal_rel_le[THEN lt_trans1] Card_rel_csucc_rel
    unfolding csucc_rel_def
    by (rule_tac Least_antitone) (assumption, auto simp add:assms)
qed

lemma (in M_cardinal_arith_jump) csucc_rel_le_mono:
  assumes κ ≤ ν M(κ) M(ν)
  shows (κ⁺)^M ≤ (ν⁺)^M
proof (cases κ = ν)
  case True
  with assms
  show ?thesis using Card_rel_csucc_rel [THEN Card_rel_is_Ord] by simp
next
  case False
  with assms
  have κ < ν using le_neq_imp_lt by simp
  show ?thesis— by way of contradiction
  proof (rule ccontr)
    assume ¬ (κ⁺)^M ≤ (ν⁺)^M
    with assms
    have (ν⁺)^M < (κ⁺)^M
      using Card_rel_csucc_rel[THEN Card_rel_is_Ord] le_Ord2 lt_Ord
      by (intro not_le_iff_lt[THEN iffD1]) auto
    with assms
    have (ν⁺)^M ≤ |κ|^M
      using le_Ord2[THEN Card_rel_csucc_rel, of κ ν]
        Card_rel_lt_csucc_rel_iff[of (ν⁺)^M |κ|^M, THEN iffD1]
        csucc_rel_cardinal_rel[OF lt_Ord] leI[of (ν⁺)^M (κ⁺)^M]
      by simp
    with assms
    have (ν⁺)^M ≤ κ
      using Ord_cardinal_rel_le[OF lt_Ord] le_trans by auto
    with assms
    have ν < κ
    using csucc_rel_basic le_Ord2[THEN Card_rel_csucc_rel, of κ ν] Card_rel_is_Ord
        le_Ord2
      by (rule_tac j=(ν⁺)^M in lt_trans2) simp_all
    with ‹κ < ν›
```

show *False* **using** *le_imp_not_lt leI* **by** *blast*
qed
qed

lemma (**in** *M_cardinal_AC*) *cardinal_rel_succ_not_0*: $|A|^M = succ(n) \Longrightarrow M(A) \Longrightarrow M(n) \Longrightarrow A \neq 0$
by *auto*

reldb_add functional *Finite Finite* — wrongly done? Finite is absolute
relativize functional *Finite_to_one Finite_to_one_rel* **external**

notation *Finite_to_one_rel* (‹*Finite'_to'_one–'*(_,_')›)

abbreviation
Finite_to_one_r_set :: [*i*,*i*,*i*] ⇒ *i* (‹*Finite'_to'_one–'*(_,_')›) **where**
$Finite\_to\_one^M(X,Y) \equiv Finite\_to\_one\_rel(\#\#M,X,Y)$

locale *M_ZF_library* = *M_cardinal_arith* + *M_aleph* + *M_FiniteFun* + *M_replacement_extra*
begin

lemma *Finite_Collect_imp*: $Finite(\{x\in X\ .\ Q(x)\}) \Longrightarrow Finite(\{x\in X\ .\ M(x) \wedge Q(x)\})$
(**is** $Finite(?A) \Longrightarrow Finite(?B)$)
using *subset_Finite*[*of ?B ?A*] **by** *auto*

lemma *Finite_to_one_relI*[*intro*]:
assumes $f{:}X\to^M Y$ $\bigwedge y.\ y\in Y \Longrightarrow Finite(\{x\in X\ .\ f`x = y\})$
and *types*:$M(f)$ $M(X)$ $M(Y)$
shows $f \in Finite\_to\_one^M(X,Y)$
using *assms Finite_Collect_imp* **unfolding** *Finite_to_one_rel_def*
by (*simp*)

lemma *Finite_to_one_relI'*[*intro*]:
assumes $f{:}X\to^M Y$ $\bigwedge y.\ y\in Y \Longrightarrow Finite(\{x\in X\ .\ M(x) \wedge f`x = y\})$
and *types*:$M(f)$ $M(X)$ $M(Y)$
shows $f \in Finite\_to\_one^M(X,Y)$
using *assms* **unfolding** *Finite_to_one_rel_def*
by (*simp*)

lemma *Finite_to_one_relD*[*dest*]:
$f \in Finite\_to\_one^M(X,Y) \Longrightarrow f{:}X\to^M Y$
$f \in Finite\_to\_one^M(X,Y) \Longrightarrow y\in Y \Longrightarrow M(Y) \Longrightarrow Finite(\{x\in X\ .\ M(x) \wedge f`x = y\})$
unfolding *Finite_to_one_rel_def* **by** *simp_all*

lemma *Diff_bij_rel*:
assumes $\forall A\in F.\ X \subseteq A$

```
    and types: M(F) M(X) shows (λA∈F. A-X) ∈ bij^M(F, {A-X. A∈F})
  using assms  def_inj_rel def_surj_rel unfolding bij_rel_def
  apply (auto)
   apply (subgoal_tac M(λA∈F. A - X) M({A - X . A ∈ F}))
    apply (auto simp add:mem_function_space_rel_abs)
     apply (rule_tac lam_type, auto)
    prefer 4
    apply (subgoal_tac M(λA∈F. A - X) M({A - X . A ∈ F}))
      apply(tactic ‹distinct_subgoals_tac›)
    apply (auto simp add:mem_function_space_rel_abs)
    apply (rule_tac lam_type, auto) prefer 3
   apply (subst subset_Diff_Un[of X])
    apply auto
proof -
  from types
  show M({A - X . A ∈ F})
    using diff_replacement
    by (rule_tac RepFun_closed) (auto dest:transM[of _ F])
  from types
  show M(λA∈F. A - X)
    using Pair_diff_replacement
    by (rule_tac lam_closed, auto dest:transM)
qed

lemma function_space_rel_nonempty:
  assumes b∈B  and types: M(B) M(A)
  shows (λx∈A. b) : A →^M B
proof -
  note assms
  moreover from this
  have M(λx∈A. b)
    using tag_replacement by (rule_tac lam_closed, auto dest:transM)
  ultimately
  show ?thesis
    by (simp add:mem_function_space_rel_abs)
qed

lemma mem_function_space_rel:
  assumes f ∈ A →^M y M(A) M(y)
  shows  f ∈ A → y
  using assms function_space_rel_char by simp

lemmas range_fun_rel_subset_codomain = range_fun_subset_codomain[OF mem_function_space_rel]

end

context M_Pi_assumptions
begin
```

lemma *mem_Pi_rel*: $f \in Pi^M(A,B) \Longrightarrow f \in Pi(A,\ B)$
  using *trans_closed mem_Pi_rel_abs*
  by *force*

lemmas $Pi\_rel\_rangeD = Pi\_rangeD[OF\ mem\_Pi\_rel]$

lemmas $rel\_apply\_Pair = apply\_Pair[OF\ mem\_Pi\_rel]$

lemmas $rel\_apply\_rangeI = apply\_rangeI[OF\ mem\_Pi\_rel]$

lemmas $Pi\_rel\_range\_eq = Pi\_range\_eq[OF\ mem\_Pi\_rel]$

lemmas $Pi\_rel\_vimage\_subset = Pi\_vimage\_subset[OF\ mem\_Pi\_rel]$

end

context *M_ZF_library*
begin

lemma *mem_bij_rel*: $[\![f \in bij^M(A,B);\ M(A);\ M(B)]\!] \Longrightarrow f \in bij(A,B)$
  using *bij_rel_char* **by** *simp*

lemma *mem_inj_rel*: $[\![f \in inj^M(A,B);\ M(A);\ M(B)]\!] \Longrightarrow f \in inj(A,B)$
  using *inj_rel_char* **by** *simp*

lemma *mem_surj_rel*: $[\![f \in surj^M(A,B);\ M(A);\ M(B)]\!] \Longrightarrow f \in surj(A,B)$
  using *surj_rel_char* **by** *simp*

lemmas $rel\_apply\_in\_range = apply\_in\_range[OF\ \_\ \_\ mem\_function\_space\_rel]$

lemmas $rel\_range\_eq\_image = ZF\_Library.range\_eq\_image[OF\ mem\_function\_space\_rel]$

lemmas $rel\_Image\_sub\_codomain = Image\_sub\_codomain[OF\ mem\_function\_space\_rel]$

lemma *rel_inj_to_Image*: $[\![f{:}A \to^M B;\ f \in inj^M(A,B);\ M(A);\ M(B)]\!] \Longrightarrow f \in inj^M(A, f``A)$
  using *inj_to_Image*[*OF mem_function_space_rel mem_inj_rel*]
    transM[*OF _ function_space_rel_closed*] **by** *simp*

lemma *inj_rel_imp_surj_rel*:
  fixes *f b*
  defines [*simp*]: *ifx*$(x) \equiv$ *if* $x \in range(f)$ *then* $converse(f)`x$ *else b*
  assumes $f \in inj^M(B,A)$ $b \in B$ **and** *types*: $M(f)$ $M(B)$ $M(A)$
  shows $(\lambda x \in A.\ ifx(x)) \in surj^M(A,B)$
proof -
  from *types* **and** ‹$b \in B$›
  have $M(\lambda x \in A.\ ifx(x))$
    using *ifx_replacement* **by** (*rule_tac lam_closed*) (*auto dest*:*transM*)
  with *assms*(*2*-)

show *?thesis*
using *inj_imp_surj mem_surj_abs* **by** *simp*
qed

lemma *function_space_rel_disjoint_Un*:
assumes $f \in A \to^M B$ $g \in C \to^M D$ $A \cap C = 0$
and *types*:$M(A)$ $M(B)$ $M(C)$ $M(D)$
shows $f \cup g \in (A \cup C) \to^M (B \cup D)$
using *assms fun_Pi_disjoint_Un*[*OF mem_function_space_rel*
mem_function_space_rel, *OF assms*(*1*) _ _ *assms*(*2*)]
function_space_rel_char **by** *auto*

lemma *restrict_eq_imp_Un_into_function_space_rel*:
assumes $f \in A \to^M B$ $g \in C \to^M D$ $restrict(f, A \cap C) = restrict(g, A \cap C)$
and *types*:$M(A)$ $M(B)$ $M(C)$ $M(D)$
shows $f \cup g \in (A \cup C) \to^M (B \cup D)$
using *assms restrict_eq_imp_Un_into_Pi*[*OF mem_function_space_rel*
mem_function_space_rel, *OF assms*(*1*) _ _ *assms*(*2*)]
function_space_rel_char **by** *auto*

lemma *lepoll_relD*[*dest*]: $A \lesssim^M B \Longrightarrow \exists f[M].\ f \in inj^M(A, B)$
unfolding *lepoll_rel_def* **.**

— Should the assumptions be on f or on A and B? Should BOTH be intro rules?
lemma *lepoll_relI*[*intro*]: $f \in inj^M(A, B) \Longrightarrow M(f) \Longrightarrow A \lesssim^M B$
unfolding *lepoll_rel_def* **by** *blast*

lemma *eqpollD*[*dest*]: $A \approx^M B \Longrightarrow \exists f[M].\ f \in bij^M(A, B)$
unfolding *eqpoll_rel_def* **.**

— Same as $[\![?f \in inj^M(?A, ?B);\ M(?f)]\!] \Longrightarrow ?A \lesssim^M ?B$
lemma *bij_rel_imp_eqpoll_rel*[*intro*]: $f \in bij^M(A,B) \Longrightarrow M(f) \Longrightarrow A \approx^M B$
unfolding *eqpoll_rel_def* **by** *blast*

lemma *restrict_bij_rel*:— Unused
assumes $f \in inj^M(A,B)$ $C \subseteq A$
and *types*:$M(A)$ $M(B)$ $M(C)$
shows $restrict(f,C) \in bij^M(C, f``C)$
using *assms restrict_bij inj_rel_char bij_rel_char* **by** *auto*

lemma *range_of_subset_eqpoll_rel*:
assumes $f \in inj^M(X,Y)$ $S \subseteq X$
and *types*:$M(X)$ $M(Y)$ $M(S)$
shows $S \approx^M f``S$
using *assms restrict_bij bij_rel_char*
trans_inj_rel_closed[*OF* ‹$f \in inj^M(X,Y)$›]
unfolding *eqpoll_rel_def*
by (*rule_tac* x=$restrict(f,S)$ **in** *rexI*) *auto*

end

relativize functional *cexp cexp_rel* **external**
relationalize *cexp_rel is_cexp*

context *M_ZF_library*
begin

— MOVE THIS to an appropriate place
is_iff_rel for *function_space*
proof -
  assume $M(A)$ $M(B)$ $M(res)$
  then
  show *?thesis*
  using *function_space_rel_char mem_function_space_rel_abs*
  unfolding *is_funspace_def is_function_space_def*
  by (*safe*, *intro equalityI*; *clarsimp*) (*auto dest:transM*)
qed

is_iff_rel for *cexp*
 using *is_cardinal_iff is_function_space_iff* **unfolding** *cexp_rel_def is_cexp_def*
  by (*simp*)

rel_closed for *cexp* **unfolding** *cexp_rel_def* **by** *simp*

end

synthesize *is_cexp* **from_definition assuming** *nonempty*
notation *is_cexp_fm* ($\langle \cdot \_^{\uparrow} - \ is\ \_ \cdot \rangle$)
arity_theorem for *is_cexp_fm*

abbreviation
  cexp_r :: $[i,i,i \Rightarrow o] \Rightarrow i$ ($\langle \_^{\uparrow} - , - \rangle$) **where**
  $cexp\_r(x,y,M) \equiv cexp\_rel(M,x,y)$

abbreviation
  cexp_r_set :: $[i,i,i] \Rightarrow i$ ($\langle \_^{\uparrow} - , - \rangle$) **where**
  $cexp\_r\_set(x,y,M) \equiv cexp\_rel(\#\#M,x,y)$

context *M_ZF_library*
begin

lemma *Card_cexp*: $M(\kappa) \Longrightarrow M(\nu) \Longrightarrow Card^M(\kappa^{\uparrow \nu, M})$
  unfolding *cexp_rel_def* **by** *simp*

— Restoring congruence rule, but NOTE: beware
declare *conj_cong*[*cong*]

lemma *eq_csucc_rel_ord*:
Ord(*i*) ⟹ *M*(*i*) ⟹ $(i^+)^M = (|i|^{M+})^M$
using *Card_rel_lt_iff Least_cong* **unfolding** *csucc_rel_def* **by** *auto*

lemma *lesspoll_succ_rel*:
assumes *Ord*(κ) *M*(κ)
shows $\kappa \lesssim^M (\kappa^+)^M$
using *csucc_rel_basic assms lt_Card_rel_imp_lesspoll_rel*
Card_rel_csucc_rel lepoll_rel_iff_leqpoll_rel
by *auto*

lemma *lesspoll_rel_csucc_rel*:
assumes *Ord*(κ)
and *types*:*M*(κ) *M*(*d*)
shows $d \prec^M (\kappa^+)^M \longleftrightarrow d \lesssim^M \kappa$
proof
assume $d \prec^M (\kappa^+)^M$
moreover
note *Card_rel_csucc_rel assms Card_rel_is_Ord*
moreover from *calculation*
have $Card^M((\kappa^+)^M)$ $M((\kappa^+)^M)$ $Ord((\kappa^+)^M)$
using *Card_rel_is_Ord* **by** *simp_all*
moreover from *calculation*
have $d \prec^M (|\kappa|^{M+})^M$ $d \approx^M |d|^M$
using *eq_csucc_rel_ord*[*OF* _ ‹*M*(κ)›]
lesspoll_rel_imp_eqpoll_rel eqpoll_rel_sym **by** *simp_all*
moreover from *calculation*
have $|d|^M < (|\kappa|^{M+})^M$
using *lesspoll_cardinal_lt_rel* **by** *simp*
moreover from *calculation*
have $|d|^M \lesssim^M |\kappa|^M$
using *Card_rel_lt_csucc_rel_iff le_imp_lepoll_rel* **by** *simp*
moreover from *calculation*
have $|d|^M \lesssim^M \kappa$
using *Ord_cardinal_rel_eqpoll_rel lepoll_rel_eq_trans*
by *simp*
ultimately
show $d \lesssim^M \kappa$
using *eq_lepoll_rel_trans* **by** *simp*
next
from ‹*Ord*(κ)›
have $\kappa < (\kappa^+)^M$ $Card^M((\kappa^+)^M)$ $M((\kappa^+)^M)$
using *Card_rel_csucc_rel lt_csucc_rel_iff types eq_csucc_rel_ord*[*OF* _
‹*M*(κ)›]
by *simp_all*
then
have $\kappa \prec^M (\kappa^+)^M$
using *lt_Card_rel_imp_lesspoll_rel*[*OF* _ ‹κ <_›] *types* **by** *simp*
moreover

```
  assume d \<lesssim^M \<kappa>
  ultimately
  have d \<lesssim^M (\<kappa>^+)^M
    using Card_rel_csucc_rel types lesspoll_succ_rel lepoll_rel_trans ‹Ord(\<kappa>)›
    by simp
  moreover
  from ‹d \<lesssim^M \<kappa>› ‹Ord(\<kappa>)›
  have (\<kappa>^+)^M \<lesssim^M \<kappa> if d \<approx>^M (\<kappa>^+)^M
    using eqpoll_rel_sym[OF that] types eq_lepoll_rel_trans[OF _ ‹d\<lesssim^M\<kappa>›]
    by simp
  moreover from calculation ‹\<kappa> \<prec>^M (\<kappa>^+)^M›
  have False if d \<approx>^M (\<kappa>^+)^M
    using lesspoll_rel_irrefl[OF _ ‹M((\<kappa>^+)^M)›] lesspoll_rel_trans1 types that
    by auto
  ultimately
  show d \<prec>^M (\<kappa>^+)^M
    unfolding lesspoll_rel_def by auto
qed

lemma Infinite_imp_nats_lepoll:
  assumes Infinite(X) n \<in> \<omega>
  shows n \<lesssim> X
  using ‹n \<in> \<omega>›
proof (induct)
  case 0
  then
  show ?case using empty_lepollI by simp
next
  case (succ x)
  show ?case
  proof -
    from ‹Infinite(X)› and ‹x \<in> \<omega>›
    have \<not> (x \<approx> X)
      using eqpoll_sym unfolding Finite_def by auto
    with ‹x \<lesssim> X›
    obtain f where f \<in> inj(x,X) f \<notin> surj(x,X)
      unfolding bij_def eqpoll_def by auto
    moreover from this
    obtain b where b \<in> X \<forall> a\<in>x. f`a \<noteq> b
      using inj_is_fun unfolding surj_def by auto
    ultimately
    have f \<in> inj(x,X-{b})
      unfolding inj_def by (auto intro:Pi_type)
    then
    have cons(\<langle>x, b\<rangle>, f) \<in> inj(succ(x), cons(b, X - {b}))
      using inj_extend[of f x X-{b} x b] unfolding succ_def
      by (auto dest:mem_irrefl)
    moreover from ‹b\<in>X›
    have cons(b, X - {b}) = X by auto
```

```
    ultimately
    show succ(x) ≲ X by auto
  qed
qed

lemma nepoll_imp_nepoll_rel :
  assumes ¬ x ≈ X M(x) M(X)
  shows ¬ (x ≈^M X)
  using assms unfolding eqpoll_def eqpoll_rel_def by simp

lemma Infinite_imp_nats_lepoll_rel:
  assumes Infinite(X) n ∈ ω
    and types: M(X)
  shows n ≲^M X
  using ‹n ∈ ω›
proof (induct)
  case 0
  then
  show ?case using empty_lepoll_relI types by simp
next
  case (succ x)
  show ?case
  proof -
    from ‹Infinite(X)› and ‹x ∈ ω›
    have ¬ (x ≈ X) M(x) M(succ(x))
      using eqpoll_sym unfolding Finite_def by auto
    then
    have ¬ (x ≈^M X)
      using nepoll_imp_nepoll_rel types by simp
    with ‹x ≲^M X›
    obtain f where f ∈ inj^M(x,X) f ∉ surj^M(x,X) M(f)
      unfolding bij_rel_def eqpoll_rel_def by auto
    with ‹M(X)› ‹M(x)›
    have f∉surj(x,X) f∈inj(x,X)
      using surj_rel_char by simp_all
    moreover
    from this
    obtain b where b ∈ X ∀ a∈x. f‘a ≠ b
      using inj_is_fun unfolding surj_def by auto
    moreover
    from this calculation ‹M(x)›
    have f ∈ inj(x,X-{b}) M(<x,b>)
      unfolding inj_def using transM[OF _ ‹M(X)›]
      by (auto intro:Pi_type)
    moreover
    from this
    have cons(⟨x, b⟩, f) ∈ inj(succ(x), cons(b, X - {b})) (is ?g∈_)
      using inj_extend[of f x X-{b} x b] unfolding succ_def
      by (auto dest:mem_irrefl)
```

```
    moreover
    note ‹M(<x,b>)› ‹M(f)› ‹b∈X› ‹M(X)› ‹M(succ(x))›
    moreover from this
    have M(?g) cons(b, X - {b}) = X by auto
    moreover from calculation
    have ?g∈inj_rel(M,succ(x),X)
      using mem_inj_abs by simp
    with ‹M(?g)›
    show succ(x) ≲^M X using lepoll_relI by simp
  qed
qed

lemma lepoll_rel_imp_lepoll: A ≲^M B ⟹ M(A) ⟹ M(B) ⟹ A ≲ B
  unfolding lepoll_rel_def by auto

lemma zero_lesspoll_rel: assumes 0<κ M(κ) shows 0 ≺^M κ
  using assms eqpoll_rel_0_iff[THEN iffD1, of κ] eqpoll_rel_sym
  unfolding lesspoll_rel_def lepoll_rel_def
  by (auto simp add:inj_def)

lemma lepoll_rel_nat_imp_Infinite: ω ≲^M X ⟹ M(X) ⟹ Infinite(X)
  using  lepoll_nat_imp_Infinite lepoll_rel_imp_lepoll by simp

lemma InfCard_rel_imp_Infinite: InfCard^M(κ) ⟹ M(κ) ⟹ Infinite(κ)
  using le_imp_lepoll_rel[THEN lepoll_rel_nat_imp_Infinite, of κ]
  unfolding InfCard_rel_def by simp

lemma lt_surj_rel_empty_imp_Card_rel:
  assumes Ord(κ) ⋀α. α < κ ⟹ surj^M(α,κ) = 0
    and types:M(κ)
  shows Card^M(κ)
proof -
  {
    define min where min≡μ x. ∃f[M]. f ∈ bij^M(x,κ)
    moreover
    note ‹Ord(κ)› ‹M(κ)›
    moreover
    assume |κ|^M < κ
    moreover from calculation
    have ∃f. f ∈ bij^M(min,κ)
      using LeastI[of λi. i ≈^M κ κ, OF eqpoll_rel_refl]
      unfolding Card_rel_def cardinal_rel_def eqpoll_rel_def
      by (auto)
    moreover from calculation
    have min < κ
      using lt_trans1[of min μ i. M(i) ∧ (∃f[M]. f ∈ bij^M(i, κ)) κ]
       Least_le[of λi. i ≈^M κ |κ|^M, OF Ord_cardinal_rel_eqpoll_rel]
      unfolding Card_rel_def cardinal_rel_def eqpoll_rel_def
      by (simp)
```

```
    moreover
    note ‹min < κ ⟹ surj^M(min,κ) = 0›
    ultimately
    have False
      unfolding bij_rel_def by simp
  }
  with assms
  show ?thesis
    using Ord_cardinal_rel_le[of κ] not_lt_imp_le[of |κ|^M κ] le_anti_sym
    unfolding Card_rel_def by auto
qed

end

relativize functional mono_map mono_map_rel external
relationalize mono_map_rel is_mono_map

notation mono_map_rel (‹mono'_map⇘_'(_,_,_,_')›)

abbreviation
  mono_map_r_set :: [i,i,i,i,i]⇒i  (‹mono'_map⇘_'(_,_,_,_')›) where
  mono_map^M(a,r,b,s) ≡ mono_map_rel(##M,a,r,b,s)

context M_ZF_library
begin

lemma mono_map_rel_char:
  assumes M(a) M(b)
  shows mono_map^M(a,r,b,s) = {f∈mono_map(a,r,b,s) . M(f)}
 using assms function_space_rel_char unfolding mono_map_rel_def mono_map_def
  by auto
```

Just a sample of porting results on *mono_map*

```
lemma mono_map_rel_mono:
  assumes
    f ∈ mono_map^M(A,r,B,s) B ⊆ C
    and types:M(A) M(B) M(C)
  shows
    f ∈ mono_map^M(A,r,C,s)
  using assms mono_map_mono mono_map_rel_char by auto

lemma nats_le_InfCard_rel:
  assumes n ∈ ω InfCard^M(κ)
  shows n ≤ κ
  using assms Ord_is_Transset
    le_trans[of n ω κ, OF le_subset_iff[THEN iffD2]]
  unfolding InfCard_rel_def Transset_def by simp

lemma nat_into_InfCard_rel:
```

assumes $n \in \omega$ $InfCard^M(\kappa)$
shows $n \in \kappa$
using *assms le_imp_subset*[*of* ω κ]
unfolding *InfCard_rel_def* **by** *auto*

lemma *Finite_cardinal_rel_in_nat* [*simp*]:
assumes *Finite*(A) $M(A)$ **shows** $|A|^M \in \omega$
proof -
note *assms*
moreover from *this*
obtain n **where** $n \in \omega$ $M(n)$ $A \approx n$
unfolding *Finite_def* **by** *auto*
moreover from *calculation*
obtain f **where** $f \in bij(A,n)$ f: A-||>n
using *Finite_Fin*[*THEN fun_FiniteFunI*, *OF _ subset_refl*] *bij_is_fun*
unfolding *eqpoll_def* **by** *auto*
ultimately
have $A \approx^M n$ **unfolding** *eqpoll_rel_def* **by** (*auto dest*:*transM*)
with *assms* **and** ‹$M(n)$›
have $n \approx^M A$ **using** *eqpoll_rel_sym* **by** *simp*
moreover
note ‹$n\in\omega$› ‹$M(n)$›
ultimately
show *?thesis*
using *assms Least_le*[*of* $\lambda i.\ M(i) \wedge i \approx^M A$ n]
lt_trans1[*of _ n* ω, *THEN ltD*]
unfolding *cardinal_rel_def Finite_def*
by (*auto dest*!:*naturals_lt_nat*)
qed

lemma *Finite_cardinal_rel_eq_cardinal*:
assumes *Finite*(A) $M(A)$ **shows** $|A|^M = |A|$
proof -
— Copy-paste from ⟦*Finite*(*?A*); M(*?A*)⟧ ⟹ $|?A|^M \in \omega$
note *assms*
moreover from *this*
obtain n **where** $n \in \omega$ $M(n)$ $A \approx n$
unfolding *Finite_def* **by** *auto*
moreover from *this*
have $|A| = n$
using *cardinal_cong*[*of A n*]
nat_into_Card[*THEN Card_cardinal_eq*, *of n*] **by** *simp*
moreover from *calculation*
obtain f **where** $f \in bij(A,n)$ f: A-||>n
using *Finite_Fin*[*THEN fun_FiniteFunI*, *OF _ subset_refl*] *bij_is_fun*
unfolding *eqpoll_def* **by** *auto*
ultimately
have $A \approx^M n$ **unfolding** *eqpoll_rel_def* **by** (*auto dest*:*transM*)
with *assms* **and** ‹$M(n)$› ‹$n\in\omega$›

have $|A|^M = n$
using *cardinal_rel_cong*[*of A n*]
nat_into_Card_rel[*THEN Card_rel_cardinal_rel_eq*, *of n*]
by *simp*
with ‹$|A| = n$›
show *?thesis* **by** *simp*
qed

lemma *Finite_imp_cardinal_rel_cons*:
assumes *FA*: *Finite*(A) **and** *a*: $a \notin A$ **and** *types*:$M(A)$ $M(a)$
shows $|cons(a,A)|^M = succ(|A|^M)$
using *assms Finite_imp_cardinal_cons Finite_cardinal_rel_eq_cardinal* **by** *simp*

lemma *Finite_imp_succ_cardinal_rel_Diff*:
assumes *Finite*(A) $a \in A$ $M(A)$
shows $succ(|A\text{-}\{a\}|^M) = |A|^M$
proof -
from *assms*
have *inM*: $M(A\text{-}\{a\})$ $M(a)$ $M(A)$ **by** (*auto dest*:*transM*)
with ‹*Finite*(A)›
have $succ(|A\text{-}\{a\}|^M) = succ(|A\text{-}\{a\}|)$
using *Diff_subset*[*THEN subset_Finite*,
THEN Finite_cardinal_rel_eq_cardinal, *of A* $\{a\}$] **by** *simp*
also from *assms*
have $\ldots = |A|$
using *Finite_imp_succ_cardinal_Diff* **by** *simp*
also from *assms*
have $\ldots = |A|^M$ **using** *Finite_cardinal_rel_eq_cardinal* **by** *simp*
finally
show *?thesis* **.**
qed

lemma *InfCard_rel_Aleph_rel*:
notes *Aleph_rel_zero*[*simp*]
assumes *Ord*(α)
and *types*: $M(\alpha)$
shows $InfCard^M(\aleph_\alpha{}^M)$
proof -
have $\neg$ $(\aleph_\alpha{}^M \in \omega)$
proof (*cases* α=*0*)
case *True*
then show *?thesis* **using** *mem_irrefl* **by** *auto*
next
case *False*
with *assms*
have $\omega \in \aleph_\alpha{}^M$ **using** *Ord_0_lt*[*of* α] *ltD* **by** (*auto dest*:*Aleph_rel_increasing*)
then show *?thesis* **using** *foundation* **by** *blast*
qed

with *assms*
have $\neg$ $(|\aleph_\alpha{}^M|^M \in \omega)$
using *Card_rel_cardinal_rel_eq* **by** *auto*
with *assms*
have *Infinite*$(\aleph_\alpha{}^M)$ **using** *Ord_Aleph_rel* **by** *clarsimp*
with *assms*
show *?thesis*
using *Inf_Card_rel_is_InfCard_rel* **by** *simp*
qed

lemmas *Limit_Aleph_rel* = *InfCard_rel_Aleph_rel*[*THEN InfCard_rel_is_Limit*]

bundle *Ord_dests* = *Limit_is_Ord*[*dest*] *Card_rel_is_Ord*[*dest*]
bundle *Aleph_rel_dests* = *Aleph_rel_cont*[*dest*]
bundle *Aleph_rel_intros* = *Aleph_rel_increasing*[*intro*!]
bundle *Aleph_rel_mem_dests* = *Aleph_rel_increasing*[*OF ltI*, *THEN ltD*, *dest*]

end

end

28 Cardinal Arithmetic under Choice

theory *Replacement_Lepoll*
imports
ZF_Library_Relative
Lambda_Replacement
begin

definition
lepoll_assumptions1 :: [*i*⇒*o*,*i*,[*i*,*i*]⇒*i*,*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
lepoll_assumptions1(*M*,*A*,*F*,*S*,*fa*,*K*,*x*,*f*,*r*) ≡ ∀ *x*∈*S*. *strong_replacement*(*M*, λ*y z*. *y* ∈ *F*(*A*, *x*) ∧ *z* = {⟨*x*, *y*⟩})

definition
lepoll_assumptions2 :: [*i*⇒*o*,*i*,[*i*,*i*]⇒*i*,*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
lepoll_assumptions2(*M*,*A*,*F*,*S*,*fa*,*K*,*x*,*f*,*r*) ≡ *strong_replacement*(*M*, λ*x z*. *z* = *Sigfun*(*x*, *F*(*A*)))

definition
lepoll_assumptions3 :: [*i*⇒*o*,*i*,[*i*,*i*]⇒*i*,*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
lepoll_assumptions3(*M*,*A*,*F*,*S*,*fa*,*K*,*x*,*f*,*r*) ≡ *strong_replacement*(*M*, λ*x y*. *y* = *F*(*A*, *x*))

definition
lepoll_assumptions4 :: [*i*⇒*o*,*i*,[*i*,*i*]⇒*i*,*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
lepoll_assumptions4(*M*,*A*,*F*,*S*,*fa*,*K*,*x*,*f*,*r*) ≡ *strong_replacement*(*M*, λ*x y*. *y* = ⟨*x*, *minimum*(*r*, *F*(*A*, *x*))⟩)

definition
lepoll_assumptions5 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions5$(M,A,F,S,fa,K,x,f,r) \equiv$
strong_replacement$(M,\ \lambda x\ y.\ y = \langle x,\ \mu\ i.\ x \in F(A,\ i),\ f\ `\ (\mu\ i.\ x \in F(A,\ i))\ `\ x\rangle)$

definition
lepoll_assumptions6 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions6$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M,\ \lambda y\ z.\ y \in inj^M(F(A,\ x),S) \wedge z = \{\langle x,\ y\rangle\})$

definition
lepoll_assumptions7 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions7$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M,\ \lambda x\ y.\ y = inj^M(F(A,\ x),S))$

definition
lepoll_assumptions8 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions8$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M,\ \lambda x\ z.\ z = Sigfun(x,\ \lambda i.\ inj^M(F(A,\ i),S)))$

definition
lepoll_assumptions9 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions9$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M,\ \lambda x\ y.\ y = \langle x,\ minimum(r,\ inj^M(F(A,\ x),S))\rangle)$

definition
lepoll_assumptions10 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions10$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*
$(M,\ \lambda x\ z.\ z = Sigfun(x,\ \lambda k.\ if\ k \in range(f)\ then\ F(A,\ converse(f)\ `\ k)\ else\ 0))$

definition
lepoll_assumptions11 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions11$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M,\ \lambda x\ y.\ y = (if\ x \in range(f)\ then\ F(A,\ converse(f)\ `\ x)\ else\ 0))$

definition
lepoll_assumptions12 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions12$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M,\ \lambda y\ z.\ y \in F(A,\ converse(f)\ `\ x) \wedge z = \{\langle x,\ y\rangle\})$

definition
lepoll_assumptions13 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions13$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*
$(M,\ \lambda x\ y.\ y = \langle x,\ minimum(r,\ if\ x \in range(f)\ then\ F(A,converse(f)\ `\ x)\ else\ 0)\rangle)$

definition

lepoll_assumptions14 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions14$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*
$(M, \lambda x\ y.\ y = \langle x, \mu\ i.\ x \in (if\ i \in range(f)\ then\ F(A, converse(f)\ `\ i)\ else\ 0),$
$fa\ `\ (\mu\ i.\ x \in (if\ i \in range(f)\ then\ F(A, converse(f)\ `\ i)\ else\ 0))\ `\ x\rangle)$

definition
lepoll_assumptions15 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions15$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*
$(M, \lambda y\ z.\ y \in inj^M(if\ x \in range(f)\ then\ F(A, converse(f)\ `\ x)\ else\ 0,K) \wedge z = \{\langle x, y\rangle\})$

definition
lepoll_assumptions16 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions16$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M, \lambda x\ y.\ y = inj^M(if\ x \in range(f)\ then\ F(A, converse(f)\ `\ x)\ else\ 0,K))$

definition
lepoll_assumptions17 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions17$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*
$(M, \lambda x\ z.\ z = Sigfun(x, \lambda i.\ inj^M(if\ i \in range(f)\ then\ F(A, converse(f)\ `\ i)\ else\ 0,K)))$

definition
lepoll_assumptions18 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions18$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*
$(M, \lambda x\ y.\ y = \langle x, minimum(r, inj^M(if\ x \in range(f)\ then\ F(A, converse(f)\ `\ x)\ else\ 0,K))\rangle)$

lemmas *lepoll_assumptions_defs*[*simp*] = *lepoll_assumptions1_def*
lepoll_assumptions2_def lepoll_assumptions3_def lepoll_assumptions4_def
lepoll_assumptions5_def lepoll_assumptions6_def lepoll_assumptions7_def
lepoll_assumptions8_def lepoll_assumptions9_def lepoll_assumptions10_def
lepoll_assumptions11_def lepoll_assumptions12_def lepoll_assumptions13_def
lepoll_assumptions14_def lepoll_assumptions15_def lepoll_assumptions16_def
lepoll_assumptions17_def lepoll_assumptions18_def

definition *if_range_F* **where**
[*simp*]: *if_range_F*$(H,f,i) \equiv$ *if* $i \in range(f)$ *then* $H(converse(f)\ `\ i)$ *else* 0

definition *if_range_F_else_F* **where**
if_range_F_else_F$(H,b,f,i) \equiv$ *if* $b=0$ *then if_range_F*(H,f,i) *else* $H(i)$

lemma (**in** *M_basic*) *lam_Least_assumption_general*:
assumes
separations:
$\forall\ A'[M].\ separation(M, \lambda y.\ \exists\ x{\in}A'.\ y = \langle x, \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i)\rangle)$
and

mem_F_bound:$\bigwedge x\ c.\ x{\in}F(A,c) \Longrightarrow c \in range(f) \cup U(A)$
and
types:$M(A)\ M(b)\ M(f)\ M(U(A))$
shows *lam_replacement*$(M,\lambda x\ .\ \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i))$
proof -
have $\forall\, x{\in}X.\ (\mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i)) \in$
$Pow^M(\bigcup(X \cup range(f) \cup U(A)))$ **if** $M(X)$ **for** X
proof
fix x
assume $x{\in}X$
moreover
note ‹$M(X)$›
moreover from *calculation*
have $M(x)$ **by** (*auto dest*:*transM*)
moreover
note *assms*
ultimately
show $(\mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i)) \in$
$Pow^M(\bigcup(X \cup range(f) \cup U(A)))$
proof (*rule_tac Least_in_Pow_rel_Union*, *cases b=0*, *simp_all*)
case *True*
fix c
assume *asm*:$x \in if\_range\_F\_else\_F(F(A),\ 0,\ f,\ c)$
with *mem_F_bound*
show $c{\in}X \vee c \in range(f) \vee c \in U(A)$
unfolding *if_range_F_else_F_def if_range_F_def* **by** (*cases c∈range(f)*)
auto
next
case *False*
fix c
assume $x \in if\_range\_F\_else\_F(F(A),\ b,\ f,\ c)$
with *False mem_F_bound*[*of x c*]
show $c{\in}X \vee c \in range(f) \vee c{\in}U(A)$
unfolding *if_range_F_else_F_def if_range_F_def* **by** *auto*
qed
qed
with *assms*
show *?thesis*
using *bounded_lam_replacement*[*of* $\lambda x.(\mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i))$
$\lambda X.\ Pow^M(\bigcup(X \cup range(f) \cup U(A)))$] **by** *simp*
qed

lemma (**in** *M_basic*) *lam_Least_assumption_ifM_b0*:
fixes F
defines $F \equiv \lambda\_\ x.\ if\ M(x)\ then\ x\ else\ 0$
assumes
separations:
$\forall\, A'[M].\ separation(M, \lambda y.\ \exists\, x{\in}A'.\ y = \langle x, \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),0,f,i)\rangle)$
and

```
    types:M(A) M(f)
  shows lam_replacement(M,λx . μ i. x ∈ if_range_F_else_F(F(A),0,f,i))
  (is lam_replacement(M,λx . Least(?P(x))))
proof -
  {
    fix x X
    assume M(X) x∈X (μ i. ?P(x,i)) ≠ 0
    moreover from this
    obtain m where Ord(m) ?P(x,m)
      using Least_0[of ?P(_)] by auto
    moreover
    note assms
    moreover
     have ?P(x,i) ⟷ (M(converse(f) ‘ i) ∧ i ∈ range(f) ∧ x ∈ converse(f) ‘ i)
for i
      unfolding F_def if_range_F_else_F_def if_range_F_def by auto
    ultimately
    have (μ i. ?P(x,i)) ∈ range (f)
      unfolding F_def if_range_F_else_F_def if_range_F_def
      by (rule_tac LeastI2) auto
  }
  with assms
  show ?thesis
    by (rule_tac bounded_lam_replacement[of _ λX. range(f) ∪ {0}]) auto
qed

lemma (in M_replacement_extra) lam_Least_assumption_ifM_bnot0:
  fixes F
  defines F ≡ λ_ x. if M(x) then x else 0
  assumes
    separations:
  ∀ A'[M]. separation(M, λy. ∃ x∈A'. y = ⟨x, μ i. x ∈ if_range_F_else_F(F(A),b,f,i)⟩)
    separation(M,Ord)
    and
    types:M(A) M(f)
    and
    b≠0
  shows lam_replacement(M,λx . μ i. x ∈ if_range_F_else_F(F(A),b,f,i))
  (is lam_replacement(M,λx . Least(?P(x))))
proof -
  have M(x) ⟹(μ i. (M(i) ⟶ x ∈ i) ∧ M(i)) = (if Ord(x) then succ(x) else 0)
for x
    using Ord_in_Ord
    apply (auto intro:Least_0, rule_tac Least_equality, simp_all)
    by (frule lt_Ord) (auto dest:le_imp_not_lt[of _ x] intro:ltI[of x])
  moreover
  have lam_replacement(M, λx. if Ord(x) then succ(x) else 0)
    using lam_replacement_if[OF _ _ separations(2)] lam_replacement_identity
    lam_replacement_constant lam_replacement_hcomp lam_replacement_succ
```

```
    by simp
  moreover
  note types ‹b≠0›
  ultimately
  show ?thesis
    using lam_replacement_cong
    unfolding F_def if_range_F_else_F_def if_range_F_def
    by auto
qed

lemma (in M_replacement_extra) lam_Least_assumption_drSR_Y:
  fixes F r' D
  defines F ≡ drSR_Y(r',D)
 assumes ∀ A'[M]. separation(M, λy. ∃ x∈A'. y = ⟨x, μ i. x ∈ if_range_F_else_F(F(A),b,f,i)⟩)
    M(A) M(b) M(f) M(r')
  shows lam_replacement(M,λx . μ i. x ∈ if_range_F_else_F(F(A),b,f,i))
proof -
  from assms(2-)
  have [simp]: M(X) ⟹ M(X ∪ range(f) ∪ {domain(x) . x ∈ A})
    M(r') ⟹ M(X) ⟹ M({restrict(x,r') . x ∈ A})
    for X r'
   using lam_replacement_domain[THEN lam_replacement_imp_strong_replacement,
        THEN RepFun_closed, of A]
    lam_replacement_restrict'[THEN lam_replacement_imp_strong_replacement,
        THEN RepFun_closed, of r' A] by (auto dest:transM)
  have ∀ x∈X. (μ i. x ∈ if_range_F_else_F(F(A),b,f,i)) ∈
   Pow^M(⋃(X ∪ range(f) ∪ {domain(x). x∈A} ∪ {restrict(x,r'). x∈A} ∪ domain(A)
∪ range(A) ∪ ⋃A)) if M(X) for X
  proof
    fix x
    assume x∈X
    moreover
    note ‹M(X)›
    moreover from calculation
    have M(x) by (auto dest:transM)
    moreover
    note assms(2-)
    ultimately
    show (μ i. x ∈ if_range_F_else_F(F(A),b,f,i)) ∈
         Pow^M(⋃(X ∪ range(f) ∪ {domain(x). x∈A} ∪ {restrict(x,r'). x∈A} ∪
domain(A) ∪ range(A) ∪ ⋃A))
      unfolding if_range_F_else_F_def if_range_F_def
      apply (rule_tac Least_in_Pow_rel_Union, simp_all)
    proof (cases b=0, simp_all)
      case True
      fix c
      assume asm:x ∈ (if c ∈ range(f) then F(A, converse(f) ‘ c) else 0)
      then
    show c∈X ∨ c∈range(f) ∨ (∃ x∈A. c = domain(x)) ∨ (∃ x∈A. c = restrict(x,r'))
```

$\vee$ $c \in domain(A) \vee c \in range(A) \vee (\exists\, x{\in}A.\ c{\in}x)$ **by** *auto*
next
case *False*
fix c
assume $x \in F(A,\ c)$
then
show $c{\in}X \vee c{\in}range(f) \vee (\exists\, x{\in}A.\ c = domain(x)) \vee (\exists\, x{\in}A.\ c = restrict(x,r'))$
$\vee$ $c \in domain(A) \vee c \in range(A) \vee (\exists\, x{\in}A.\ c{\in}x)$
using *apply_0*
by (*cases* $M(c)$, *auto simp*:*F_def drSR_Y_def dC_F_def*)
qed
qed
with *assms*(*2-*)
show *?thesis*
using *bounded_lam_replacement*[*of* $\lambda x.(\mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i))$
$\lambda X.\ Pow^M(\bigcup(X \cup range(f) \cup \{domain(x).\ x{\in}A\} \cup \{restrict(x,r').\ x{\in}A\} \cup$
$domain(A) \cup range(A) \cup \bigcup A))$] **by** *simp*
qed

locale *M_replacement_lepoll* = *M_replacement_extra* + *M_inj* +
fixes F
assumes
F_type[*simp*]: $M(A) \Longrightarrow \forall\, x[M].\ M(F(A,x))$
and
lam_lepoll_assumption_F:$M(A) \Longrightarrow lam\_replacement(M,F(A))$
and
— Here b is a Boolean.
lam_Least_assumption:$M(A) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow$
$lam\_replacement(M,\lambda x\ .\ \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i))$
and
F_args_closed: $M(A) \Longrightarrow M(x) \Longrightarrow x \in F(A,i) \Longrightarrow M(i)$
and
lam_replacement_inj_rel:$lam\_replacement(M,\ \lambda p.\ inj^M(fst(p),snd(p)))$
begin

declare *if_range_F_else_F_def*[*simp*]

lemma *lepoll_assumptions1*:
assumes *types*[*simp*]:$M(A)$ $M(S)$
shows *lepoll_assumptions1*(M,A,F,S,fa,K,x,f,r)
using *strong_replacement_separation*[*OF lam_replacement_sing_const_id separation_in*]
transM[*of* _ S]
by *simp*

lemma *lepoll_assumptions2*:
assumes *types*[*simp*]:$M(A)$ $M(S)$
shows *lepoll_assumptions2*(M,A,F,S,fa,K,x,f,r)
using *lam_replacement_Sigfun lam_replacement_imp_strong_replacement*
assms lam_lepoll_assumption_F

by *simp*

lemma *lepoll_assumptions3*:
assumes *types*[*simp*]:$M(A)$
shows *lepoll_assumptions3*(M,A,F,S,fa,K,x,f,r)
using *lam_lepoll_assumption_F*[*THEN lam_replacement_imp_strong_replacement*]
by *simp*

lemma *lepoll_assumptions4*:
assumes *types*[*simp*]:$M(A)$ $M(r)$
shows *lepoll_assumptions4*(M,A,F,S,fa,K,x,f,r)
using *lam_replacement_minimum lam_replacement_constant lam_lepoll_assumption_F*
unfolding *lepoll_assumptions_defs*
lam_replacement_def[*symmetric*]
by (*rule_tac lam_replacement_hcomp2*[*of* _ _ *minimum*])
(*force intro*: *lam_replacement_identity*)+

lemma *lam_Least_closed* :
assumes $M(A)$ $M(b)$ $M(f)$
shows $\forall\, x[M].\ M(\mu\ i.\ x \in \mathit{if\_range\_F\_else\_F}(F(A),b,f,i))$
proof -
have $x \in (\mathit{if}\ i \in \mathit{range}(f)\ \mathit{then}\ F(A,\ \mathit{converse}(f)\ `\ i)\ \mathit{else}\ 0) \Longrightarrow M(i)$ **for** $x\ i$
proof (*cases* $i \in \mathit{range}(f)$)
case *True*
with ‹$M(f)$›
show *?thesis* **by** (*auto dest*:*transM*)
next
case *False*
moreover
assume $x \in (\mathit{if}\ i \in \mathit{range}(f)\ \mathit{then}\ F(A,\ \mathit{converse}(f)\ `\ i)\ \mathit{else}\ 0)$
ultimately
show *?thesis*
by *auto*
qed
with *assms*
show *?thesis*
using *F_args_closed*[*of A*] **unfolding** *if_range_F_else_F_def if_range_F_def*
by (*clarify*, *rule_tac Least_closed'*, *cases* b=0) *simp_all*
qed

lemma *lepoll_assumptions5*:
assumes
types[*simp*]:$M(A)$ $M(f)$
shows *lepoll_assumptions5*(M,A,F,S,fa,K,x,f,r)
using
lam_replacement_apply2[*THEN* [5] *lam_replacement_hcomp2*]
lam_replacement_hcomp[*OF* _ *lam_replacement_apply*[*of f*]]
lam_replacement_identity
lam_replacement_product lam_Least_closed[**where** b=*1*]

```
    assms lam_Least_assumption[where b=1,OF ‹M(A)› _ ‹M(f)›]
  unfolding lepoll_assumptions_defs
    lam_replacement_def[symmetric]
  by simp

lemma lepoll_assumptions6:
  assumes types[simp]:M(A) M(S) M(x)
  shows lepoll_assumptions6(M,A,F,S,fa,K,x,f,r)
  using strong_replacement_separation[OF lam_replacement_sing_const_id separation_in]
     lam_replacement_inj_rel
  by simp

lemma lepoll_assumptions7:
  assumes types[simp]:M(A) M(S) M(x)
  shows lepoll_assumptions7(M,A,F,S,fa,K,x,f,r)
  using lam_replacement_constant lam_lepoll_assumption_F lam_replacement_inj_rel
  unfolding lepoll_assumptions_defs
  by (rule_tac lam_replacement_imp_strong_replacement)
    (rule_tac lam_replacement_hcomp2[of _ _ inj_rel(M)], simp_all)

lemma lepoll_assumptions8:
  assumes types[simp]:M(A) M(S)
  shows lepoll_assumptions8(M,A,F,S,fa,K,x,f,r)
  using lam_replacement_Sigfun lam_replacement_imp_strong_replacement
    lam_replacement_inj_rel lam_replacement_constant
   lam_replacement_hcomp2[of _ _ inj_rel(M),OF lam_lepoll_assumption_F[of
A]]
  by simp

lemma lepoll_assumptions9:
  assumes types[simp]:M(A) M(S) M(r)
  shows lepoll_assumptions9(M,A,F,S,fa,K,x,f,r)
  using lam_replacement_minimum lam_replacement_constant lam_lepoll_assumption_F
   lam_replacement_hcomp2[of _ _ inj_rel(M)] lam_replacement_inj_rel lepoll_assumptions4
  unfolding lepoll_assumptions_defs lam_replacement_def[symmetric]
  by (rule_tac lam_replacement_hcomp2[of _ _ minimum])
    (force intro: lam_replacement_identity)+

lemma lepoll_assumptions10:
  assumes types[simp]:M(A) M(f)
  shows lepoll_assumptions10(M,A,F,S,fa,K,x,f,r)
  using lam_replacement_Sigfun lam_replacement_imp_strong_replacement
    lam_replacement_constant[OF nonempty]
    lam_replacement_if[OF _ _ separation_in]
    lam_replacement_hcomp
      lam_replacement_apply[OF converse_closed[OF ‹M(f)›]]
      lam_lepoll_assumption_F[of A]
  by simp
```

lemma *lepoll_assumptions11*:
assumes *types*[*simp*]:$M(A)$ $M(f)$
shows *lepoll_assumptions11*(M, A, F, S, fa, K, x, f, r)
using *lam_replacement_imp_strong_replacement*
lam_replacement_if[*OF _ _ separation_in*[*of range*(*f*)]]
lam_replacement_constant
lam_replacement_hcomp lam_replacement_apply
lam_lepoll_assumption_F
by *simp*

lemma *lepoll_assumptions12*:
assumes *types*[*simp*]:$M(A)$ $M(x)$ $M(f)$
shows *lepoll_assumptions12*(M,A,F,S,fa,K,x,f,r)
using *strong_replacement_separation*[*OF lam_replacement_sing_const_id separation_in*]
by *simp*

lemma *lepoll_assumptions13*:
assumes *types*[*simp*]:$M(A)$ $M(r)$ $M(f)$
shows *lepoll_assumptions13*(M,A,F,S,fa,K,x,f,r)
using *lam_replacement_constant*[*OF nonempty*] *lam_lepoll_assumption_F*
lam_replacement_hcomp lam_replacement_apply
lam_replacement_hcomp2[*OF lam_replacement_constant*[*OF* ‹$M(r)$›]
lam_replacement_if[*OF _ _ separation_in*[*of range*(*f*)]] _ _
lam_replacement_minimum] *assms*
unfolding *lepoll_assumptions_defs*
lam_replacement_def[*symmetric*]
by *simp*

lemma *lepoll_assumptions14*:
assumes *types*[*simp*]:$M(A)$ $M(f)$ $M(fa)$
shows *lepoll_assumptions14*(M,A,F,S,fa,K,x,f,r)
using
lam_replacement_apply2[*THEN* [*5*] *lam_replacement_hcomp2*]
lam_replacement_hcomp[*OF _ lam_replacement_apply*[*of fa*]]
lam_replacement_identity
lam_replacement_product lam_Least_closed[**where** *b=0*]
assms lam_Least_assumption[**where** *b=0*,*OF* ‹$M(A)$› _ ‹$M(f)$›]
unfolding *lepoll_assumptions_defs*
lam_replacement_def[*symmetric*]
by *simp*

lemma *lepoll_assumptions15*:
assumes *types*[*simp*]:$M(A)$ $M(x)$ $M(f)$ $M(K)$
shows *lepoll_assumptions15*(M,A,F,S,fa,K,x,f,r)
using *strong_replacement_separation*[*OF lam_replacement_sing_const_id separation_in*]
by *simp*

lemma *lepoll_assumptions16*:
assumes *types*[*simp*]:$M(A)$ $M(f)$ $M(K)$

shows *lepoll_assumptions16*(*M*,*A*,*F*,*S*,*fa*,*K*,*x*,*f*,*r*)
using *lam_replacement_imp_strong_replacement*
lam_replacement_inj_rel lam_replacement_constant
lam_replacement_hcomp2[*of* _ _ *inj_rel*(*M*)]
lam_replacement_constant[*OF nonempty*]
lam_replacement_if[*OF* _ _ *separation_in*]
lam_replacement_hcomp
lam_replacement_apply[*OF converse_closed*[*OF* ‹*M*(*f*)›]]
lam_lepoll_assumption_F[*of A*]
by *simp*

lemma *lepoll_assumptions17*:
assumes *types*[*simp*]:*M*(*A*) *M*(*f*) *M*(*K*)
shows *lepoll_assumptions17*(*M*,*A*,*F*,*S*,*fa*,*K*,*x*,*f*,*r*)
using *lam_replacement_Sigfun lam_replacement_imp_strong_replacement*
lam_replacement_inj_rel lam_replacement_constant
lam_replacement_hcomp2[*of* _ _ *inj_rel*(*M*)]
lam_replacement_constant[*OF nonempty*]
lam_replacement_if[*OF* _ _ *separation_in*]
lam_replacement_hcomp
lam_replacement_apply[*OF converse_closed*[*OF* ‹*M*(*f*)›]]
lam_lepoll_assumption_F[*of A*]
by *simp*

lemma *lepoll_assumptions18*:
assumes *types*[*simp*]:*M*(*A*) *M*(*K*) *M*(*f*) *M*(*r*)
shows *lepoll_assumptions18*(*M*,*A*,*F*,*S*,*fa*,*K*,*x*,*f*,*r*)
using *lam_replacement_constant lam_replacement_inj_rel lam_lepoll_assumption_F*
lam_replacement_minimum lam_replacement_identity lam_replacement_apply2
separation_in
unfolding *lepoll_assumptions18_def lam_replacement_def*[*symmetric*]
apply (*rule_tac lam_replacement_hcomp2*[*of* _ _ *minimum*], *simp_all*)
by (*rule_tac lam_replacement_hcomp2*[*of* _ _ *inj_rel*(*M*)], *simp_all*)
(*rule_tac lam_replacement_if*, *rule_tac lam_replacement_hcomp*[*of* _ *F*(*A*)],
rule_tac lam_replacement_hcomp2[*of* _ _ (‘)], *simp_all*)

lemmas *lepoll_assumptions* = *lepoll_assumptions1 lepoll_assumptions2*
lepoll_assumptions3 lepoll_assumptions4 lepoll_assumptions5
lepoll_assumptions6 lepoll_assumptions7 lepoll_assumptions8
lepoll_assumptions9 lepoll_assumptions10 lepoll_assumptions11
lepoll_assumptions12 lepoll_assumptions13 lepoll_assumptions14
lepoll_assumptions15 lepoll_assumptions16
lepoll_assumptions17 lepoll_assumptions18

end

end
theory *Separation_Instances*
imports

Discipline_Function
Forcing_Data
FiniteFun_Relative
Cardinal_Relative
Replacement_Lepoll
begin

28.1 More Instances of Separation

The following instances are mostly the same repetitive task; and we just copied and pasted, tweaking some lemmas if needed (for example, we might have needed to use some closedness results).

declare *Inl_iff_sats* [*iff_sats*]
declare *Inr_iff_sats* [*iff_sats*]
arity_theorem for *Inl_fm*
arity_theorem for *Inr_fm*

arity_theorem for *injection_fm*
arity_theorem for *surjection_fm*
arity_theorem for *bijection_fm*
arity_theorem for *order_isomorphism_fm*
arity_theorem for *pred_set_fm*

lemma *iff_sym* : $P(x,a) \longleftrightarrow a = f(x) \Longrightarrow P(x,a) \longleftrightarrow f(x) = a$
  by *auto*

lemma *subset_iff_sats* [*iff_sats*]:
      [| $nth(i,env) = x$; $nth(k,env) = z$;
         $i \in nat$; $k \in nat$; $env \in list(A)$|]
       ==> $subset(\#\#A,\ x,\ z) \longleftrightarrow sats(A,\ subset\_fm(i,k),\ env)$
  unfolding *subset_def subset_fm_def*
  by *simp*

definition *radd_body* :: $[i,i,i] \Rightarrow o$ **where**
  $radd\_body(R,S) \equiv \lambda z.\ (\exists\, x\ y.\ z = \langle Inl(x),\ Inr(y)\rangle) \vee$
                $(\exists\, x'\ x.\ z = \langle Inl(x'),\ Inl(x)\rangle \wedge \langle x',\ x\rangle \in R) \vee$
                $(\exists\, y'\ y.\ z = \langle Inr(y'),\ Inr(y)\rangle \wedge \langle y',\ y\rangle \in S)$

relativize functional *radd_body radd_body_rel*
relationalize *radd_body_rel is_radd_body*

synthesize *is_radd_body* **from_definition**
arity_theorem for *is_radd_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_radd_body*:
 $(\#\#M)(r) \Longrightarrow (\#\#M)(A) \Longrightarrow separation(\#\#M,\ is\_radd\_body(\#\#M,A,r))$

apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*,*A*,*r*] ⊨ *is_radd_body_fm*(*1*,*2*,*0*),*THEN iffD1*])
apply(*rule_tac is_radd_body_iff_sats*[**where** *env*=[_,*A*,*r*],*symmetric*])
apply(*simp_all*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*r*],*simplified*])
apply(*simp_all add*:*arity_is_radd_body_fm ord_simp_union is_radd_body_fm_type*)
done

lemma (**in** *M_ZF_trans*) *radd_body_abs*:
assumes (##*M*)(*R*) (##*M*)(*S*) (##*M*)(*x*)
shows *is_radd_body*(##*M*,*R*,*S*,*x*) ⟷ *radd_body*(*R*,*S*,*x*)
using *assms pair_in_M_iff Inl_in_M_iff Inr_in_M_iff*
unfolding *radd_body_def is_radd_body_def*
by (*auto*)

lemma (**in** *M_ZF_trans*) *separation_radd_body*:
(##*M*)(*R*) ⟹ (##*M*)(*S*) ⟹ *separation*
(##*M*, λ*z*. (∃ *x y*. *z* = ⟨*Inl*(*x*), *Inr*(*y*)⟩) ∨
(∃ *x*′ *x*. *z* = ⟨*Inl*(*x*′), *Inl*(*x*)⟩ ∧ ⟨*x*′, *x*⟩ ∈ *R*) ∨
(∃ *y*′ *y*. *z* = ⟨*Inr*(*y*′), *Inr*(*y*)⟩ ∧ ⟨*y*′, *y*⟩ ∈ *S*))
using *separation_is_radd_body radd_body_abs*
unfolding *radd_body_def*
by (*rule_tac separation_cong*[
where *P*=*is_radd_body*(##*M*,*R*,*S*),*THEN iffD1*])

definition *well_ord_body* :: [*i*⇒*o*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
well_ord_body(*N*,*A*,*f*,*r*,*x*) ≡ *x* ∈ *A* ⟶ (∃ *y*[*N*]. ∃ *p*[*N*]. *is_apply*(*N*, *f*, *x*, *y*) ∧
pair(*N*, *y*, *x*, *p*) ∧ *p* ∈ *r*)

synthesize *well_ord_body* **from_definition**
arity_theorem for *well_ord_body_fm*

lemma (**in** *M_ZF_trans*) *separation_well_ord_body*:
(##*M*)(*f*) ⟹ (##*M*)(*r*) ⟹ (##*M*)(*A*) ⟹ *separation*(##*M*, *well_ord_body*(##*M*,*A*,*f*,*r*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*,*A*,*f*,*r*] ⊨ *well_ord_body_fm*(*1*,*2*,*3*,*0*),*THEN iffD1*])
apply(*rule_tac well_ord_body_iff_sats*[**where** *env*=[_,*A*,*f*,*r*],*symmetric*])
apply(*simp_all*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*f*,*r*],*simplified*])
apply(*simp_all add*:*arity_well_ord_body_fm ord_simp_union well_ord_body_fm_type*)
done

lemma (**in** *M_ZF_trans*) *separation_well_ord*:
(##*M*)(*f*) ⟹ (##*M*)(*r*) ⟹ (##*M*)(*A*) ⟹ *separation*
(##*M*, λ*x*. *x* ∈ *A* ⟶ (∃ *y*[##*M*]. ∃ *p*[##*M*]. *is_apply*(##*M*, *f*, *x*, *y*) ∧
pair(##*M*, *y*, *x*, *p*) ∧ *p* ∈ *r*))
using *separation_well_ord_body well_ord_body_def* **by** *simp*

definition *is_obase_body* :: $[i⇒o,i,i,i] ⇒ o$ **where**
is_obase_body$(N,A,r,x) ≡ x ∈ A ⟶$
$¬ (∃\, y[N].$
$∃\, g[N].$
ordinal$(N,\ y) ∧$
$(∃\, my[N].$
$∃\, pxr[N].$
membership$(N,\ y,\ my) ∧$
pred_set$(N,\ A,\ x,\ r,\ pxr) ∧$
order_isomorphism$(N,\ pxr,\ r,\ y,\ my,\ g)))$

synthesize *is_obase_body* **from_definition**
arity_theorem for *is_obase_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_obase_body*:
$(##M)(r) ⟹ (##M)(A) ⟹$ *separation*$(##M,$ *is_obase_body*$(##M,A,r))$
apply(*rule_tac separation_cong*[
where $P=λ\ x\ .\ M,[x,A,r] ⊨$ *is_obase_body_fm*$(1,2,0)$,*THEN iffD1*])
apply(*rule_tac is_obase_body_iff_sats*[**where** *env*=[_,*A*,*r*],*symmetric*])
apply(*simp_all*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*r*],*simplified*])
apply(*simp_all add*:*arity_is_obase_body_fm ord_simp_union is_obase_body_fm_type*)
done

lemma (**in** *M_ZF_trans*) *separation_is_obase*:
$(##M)(f) ⟹ (##M)(r) ⟹ (##M)(A) ⟹$ *separation*
$(##M,\ λx.\ x ∈ A ⟶$
$¬ (∃\, y[##M].$
$∃\, g[##M].$
ordinal$(##M,\ y) ∧$
$(∃\, my[##M].$
$∃\, pxr[##M].$
membership$(##M,\ y,\ my) ∧$
pred_set$(##M,\ A,\ x,\ r,\ pxr) ∧$
order_isomorphism$(##M,\ pxr,\ r,\ y,\ my,\ g))))$
using *separation_is_obase_body is_obase_body_def* **by** *simp*

definition *is_obase_equals* :: $[i⇒o,i,i,i] ⇒ o$ **where**
is_obase_equals$(N,A,r,a) ≡ ∃\, x[N].$
$∃\, g[N].$
$∃\, mx[N].$
$∃\, par[N].$
ordinal$(N,\ x) ∧$
membership$(N,\ x,\ mx) ∧$
pred_set$(N,\ A,\ a,\ r,\ par) ∧$ *order_isomorphism*$(N,\ par,$
$r,\ x,\ mx,\ g)$

synthesize *is_obase_equals* **from_definition**
arity_theorem for *is_obase_equals_fm*

lemma (**in** *M_ZF_trans*) *separation_obase_equals_aux*:
$(\#\#M)(r) \Longrightarrow (\#\#M)(A) \Longrightarrow$ *separation*$(\#\#M,$ *is_obase_equals*$(\#\#M,A,r))$
apply(*rule_tac separation_cong*[
where $P=\lambda\ x\ .\ M,[x,A,r] \models$ *is_obase_equals_fm*(*1*,*2*,*0*),*THEN iffD1*])
apply(*rule_tac is_obase_equals_iff_sats*[**where** *env*=[_,*A*,*r*],*symmetric*])
apply(*simp_all*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*r*],*simplified*])
apply(*simp_all add*:*arity_is_obase_equals_fm ord_simp_union is_obase_equals_fm_type*)
done

lemma (**in** *M_ZF_trans*) *separation_obase_equals*:
$(\#\#M)(f) \Longrightarrow (\#\#M)(r) \Longrightarrow (\#\#M)(A) \Longrightarrow$ *separation*
$(\#\#M,\ \lambda a.\ \exists\, x[\#\#M].$
$\exists\, g[\#\#M].$
$\exists\, mx[\#\#M].$
$\exists\, par[\#\#M].$
ordinal$(\#\#M,\ x)\ \wedge$
membership$(\#\#M,\ x,\ mx)\ \wedge$
pred_set$(\#\#M,\ A,\ a,\ r,\ par)\ \wedge$ *order_isomorphism*$(\#\#M,$
$par,\ r,\ x,\ mx,\ g))$
using *separation_obase_equals_aux is_obase_equals_def* **by** (*simp*)

definition *id_body* :: $[i,i] \Rightarrow o$ **where**
id_body$(A) \equiv \lambda z.\ \exists\, x{\in}A.\ z = \langle x,\ x\rangle$
relativize *id_body is_id_body*
synthesize *is_id_body* **from_definition assuming** *nonempty*
arity_theorem for *is_id_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_id_body*:
$(\#\#M)(A) \Longrightarrow$ *separation*$(\#\#M,$ *is_id_body*$(\#\#M,A))$
apply(*rule_tac separation_cong*[
where $P=\lambda\ x\ .\ M,[x,A] \models$ *is_id_body_fm*(*1*,*0*),*THEN iffD1*])
apply(*rule_tac is_id_body_iff_sats*[**where** *env*=[_,*A*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*],*simplified*])
apply(*simp_all add*:*arity_is_id_body_fm ord_simp_union is_id_body_fm_type*)
done

lemma (**in** *M_ZF_trans*) *id_body_abs*:
assumes $(\#\#M)(A)$ $(\#\#M)(x)$
shows *is_id_body*$(\#\#M,A,x) \longleftrightarrow$ *id_body*(A,x)
using *assms zero_in_M pair_in_M_iff* **unfolding** *id_body_def is_id_body_def*
by *auto*

lemma (**in** *M_ZF_trans*) *separation_id_body*:
$(\#\#M)(A) \Longrightarrow$ *separation*
$(\#\#M,\ \lambda z.\ \exists\, x{\in}A.\ z = \langle x,\ x\rangle)$

using *id_body_abs separation_is_id_body*
unfolding *id_body_def*
by (*rule_tac separation_cong*[**where** *P=is_id_body(##M,A),THEN iffD1*])

definition *rvimage_body* :: [*i*,*i*,*i*] ⇒ *o* **where**
rvimage_body(f,r) ≡ λ*z*. ∃ *x y*. *z* = ⟨*x*, *y*⟩ ∧ ⟨*f* ‘ *x*, *f* ‘ *y*⟩ ∈ *r*

relativize functional *rvimage_body rvimage_body_rel*
relationalize *rvimage_body_rel is_rvimage_body*

synthesize *is_rvimage_body* **from_definition**
arity_theorem for *is_rvimage_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_rvimage_body*:
(##*M*)(*f*) ⟹ (##*M*)(*r*) ⟹ *separation*(##*M*, *is_rvimage_body*(##*M*,*f*,*r*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*,*f*,*r*] ⊨ *is_rvimage_body_fm*(*1*,*2*,*0*),*THEN iffD1*])
apply(*rule_tac is_rvimage_body_iff_sats*[**where** *env*=[_,*f*,*r*],*symmetric*])
apply(*simp_all*)
apply(*rule_tac separation_ax*[**where** *env*=[*f*,*r*],*simplified*])
apply(*simp_all add*:*arity_is_rvimage_body_fm ord_simp_union is_rvimage_body_fm_type*)
done

lemma (**in** *M_ZF_trans*) *rvimage_body_abs*:
assumes (##*M*)(*R*) (##*M*)(*S*) (##*M*)(*x*)
shows *is_rvimage_body*(##*M*,*R*,*S*,*x*) ⟷ *rvimage_body*(*R*,*S*,*x*)
using *assms pair_in_M_iff apply_closed*
unfolding *rvimage_body_def is_rvimage_body_def*
by (*auto*)

lemma (**in** *M_ZF_trans*) *separation_rvimage_body*:
(##*M*)(*f*) ⟹ (##*M*)(*r*) ⟹ *separation*
(##*M*, λ*z*. ∃ *x y*. *z* = ⟨*x*, *y*⟩ ∧ ⟨*f* ‘ *x*, *f* ‘ *y*⟩ ∈ *r*)
using *separation_is_rvimage_body rvimage_body_abs*
unfolding *rvimage_body_def*
by (*rule_tac separation_cong*[
where *P=is_rvimage_body(##M,f,r),THEN iffD1*])

definition *rmult_body* :: [*i*,*i*,*i*] ⇒ *o* **where**
rmult_body(b,d) ≡ λ*z*. ∃ *x′ y′ x y*. *z* = ⟨⟨*x′*, *y′*⟩, *x*, *y*⟩ ∧ (⟨*x′*, *x*⟩ ∈ *b* ∨ *x′* = *x* ∧
⟨*y′*, *y*⟩ ∈ *d*)

relativize functional *rmult_body rmult_body_rel*
relationalize *rmult_body_rel is_rmult_body*

synthesize *is_rmult_body* **from_definition**
arity_theorem for *is_rmult_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_rmult_body*:
(##*M*)(*b*) ⟹ (##*M*)(*d*) ⟹ *separation*(##*M*, *is_rmult_body*(##*M*,*b*,*d*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*,*b*,*d*] ⊨ *is_rmult_body_fm*(*1*,*2*,*0*),*THEN iffD1*])
apply(*rule_tac is_rmult_body_iff_sats*[**where** *env*=[_,*b*,*d*],*symmetric*])
apply(*simp_all*)
apply(*rule_tac separation_ax*[**where** *env*=[*b*,*d*],*simplified*])
apply(*simp_all add*:*arity_is_rmult_body_fm ord_simp_union is_rmult_body_fm_type*)
done

lemma (**in** *M_ZF_trans*) *rmult_body_abs*:
assumes (##*M*)(*b*) (##*M*)(*d*) (##*M*)(*x*)
shows *is_rmult_body*(##*M*,*b*,*d*,*x*) ⟷ *rmult_body*(*b*,*d*,*x*)
using *assms pair_in_M_iff apply_closed*
unfolding *rmult_body_def is_rmult_body_def*
by (*auto*)

lemma (**in** *M_ZF_trans*) *separation_rmult_body*:
(##*M*)(*b*) ⟹ (##*M*)(*d*) ⟹ *separation*
(##*M*, λ*z*. ∃ *x′ y′ x y*. *z* = ⟨⟨*x′*, *y′*⟩, *x*, *y*⟩ ∧ (⟨*x′*, *x*⟩ ∈ *b* ∨ *x′* = *x* ∧ ⟨*y′*, *y*⟩
∈ *d*))
using *separation_is_rmult_body rmult_body_abs*
separation_cong[**where** *P*=*is_rmult_body*(##*M*,*b*,*d*) **and** *M*=##*M*,*THEN*
iffD1]
unfolding *rmult_body_def*
by *simp*

definition *ord_iso_body* :: [*i*,*i*,*i*,*i*] ⇒ *o* **where**
ord_iso_body(*A*,*r*,*s*) ≡ λ*f*. ∀ *x*∈*A*. ∀ *y*∈*A*. ⟨*x*, *y*⟩ ∈ *r* ⟷ ⟨*f* ‘ *x*, *f* ‘ *y*⟩ ∈ *s*

relativize functional *ord_iso_body ord_iso_body_rel*
relationalize *ord_iso_body_rel is_ord_iso_body*

synthesize *is_ord_iso_body* **from_definition assuming** *nonempty*
arity_theorem for *is_ord_iso_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_ord_iso_body*:
(##*M*)(*A*) ⟹ (##*M*)(*r*) ⟹ (##*M*)(*s*) ⟹ *separation*(##*M*, *is_ord_iso_body*(##*M*,*A*,*r*,*s*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*,*A*,*r*,*s*] ⊨ *is_ord_iso_body_fm*(*1*,*2*,*3*,*0*),*THEN iffD1*])
apply(*rule_tac is_ord_iso_body_iff_sats*[**where** *env*=[_,*A*,*r*,*s*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*r*,*s*],*simplified*])
apply(*simp_all add*:*arity_is_ord_iso_body_fm ord_simp_union is_ord_iso_body_fm_type*)
done

lemma (**in** *M_ZF_trans*) *ord_iso_body_abs*:
assumes (##*M*)(*A*) (##*M*)(*r*) (##*M*)(*s*) (##*M*)(*x*)

shows *is_ord_iso_body(##M,A,r,s,x)* $\longleftrightarrow$ *ord_iso_body(A,r,s,x)*
using *assms pair_in_M_iff apply_closed zero_in_M transitivity*[*of* _ *A*]
unfolding *ord_iso_body_def is_ord_iso_body_def*
by *auto*

lemma (**in** *M_ZF_trans*) *separation_ord_iso_body*:
(##*M*)(*A*) $\Longrightarrow$ (##*M*)(*r*) $\Longrightarrow$ (##*M*)(*s*) $\Longrightarrow$ *separation*
(##*M*, $\lambda f.\ \forall x{\in}A.\ \forall y{\in}A.\ \langle x,\ y\rangle \in r \longleftrightarrow \langle f\ `\ x,\ f\ `\ y\rangle \in s$)
using *separation_is_ord_iso_body ord_iso_body_abs*
separation_cong[**where** *P=is_ord_iso_body(##M,A,r,s)* **and** *M=##M*,*THEN iffD1*]
unfolding *ord_iso_body_def*
by *simp*

synthesize *PiP_rel* **from_definition assuming** *nonempty*
arity_theorem for *PiP_rel_fm*

lemma (**in** *M_ZF_trans*) *separation_PiP_rel*:
(##*M*)(*A*) $\Longrightarrow$ *separation*(##*M*, *PiP_rel*(##*M*,*A*))
apply(*rule_tac separation_cong*[
where *P=*λ *x . M*,[*x*,*A*] $\models$ *PiP_rel_fm*(*1*,*0*),*THEN iffD1*])
apply(*rule_tac PiP_rel_iff_sats*[**where** *env=*[_,*A*],*symmetric*])
apply(*simp_all add:zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env=*[*A*],*simplified*])
apply(*simp_all add:arity_PiP_rel_fm ord_simp_union PiP_rel_fm_type*)
done

synthesize *injP_rel* **from_definition assuming** *nonempty*
arity_theorem for *injP_rel_fm*

lemma (**in** *M_ZF_trans*) *separation_injP_rel*:
(##*M*)(*A*) $\Longrightarrow$ *separation*(##*M*, *injP_rel*(##*M*,*A*))
apply(*rule_tac separation_cong*[
where *P=*λ *x . M*,[*x*,*A*] $\models$ *injP_rel_fm*(*1*,*0*),*THEN iffD1*])
apply(*rule_tac injP_rel_iff_sats*[**where** *env=*[_,*A*],*symmetric*])
apply(*simp_all add:zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env=*[*A*],*simplified*])
apply(*simp_all add:arity_injP_rel_fm ord_simp_union injP_rel_fm_type*)
done

synthesize *surjP_rel* **from_definition assuming** *nonempty*
arity_theorem for *surjP_rel_fm*

lemma (**in** *M_ZF_trans*) *separation_surjP_rel*:
(##*M*)(*A*) $\Longrightarrow$ (##*M*)(*B*) $\Longrightarrow$ *separation*(##*M*, *surjP_rel*(##*M*,*A*,*B*))
apply(*rule_tac separation_cong*[
where *P=*λ *x . M*,[*x*,*A*,*B*] $\models$ *surjP_rel_fm*(*1*,*2*,*0*),*THEN iffD1*])
apply(*rule_tac surjP_rel_iff_sats*[**where** *env=*[_,*A*,*B*],*symmetric*])

apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*B*],*simplified*])
apply(*simp_all add*:*arity_surjP_rel_fm ord_simp_union surjP_rel_fm_type*)
done

synthesize *cons_like_rel* **from_definition assuming** *nonempty*
arity_theorem for *cons_like_rel_fm*

lemma (**in** *M_ZF_trans*) *separation_cons_like_rel*:
separation(##*M*, *cons_like_rel*(##*M*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*] ⊨ *cons_like_rel_fm*(*0*),*THEN iffD1*])
apply(*rule_tac cons_like_rel_iff_sats*[**where** *env*=[_],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[],*simplified*])
apply(*simp_all add*:*arity_cons_like_rel_fm ord_simp_union cons_like_rel_fm_type*)
done

definition *supset_body* :: [*i*] ⇒ *o* **where**
supset_body ≡ λ *x*. ∃ *a*. ∃ *b*. *x* = ⟨*a*,*b*⟩ ∧ *b* ⊆ *a*

relativize functional *supset_body supset_body_rel*
relationalize *supset_body_rel is_supset_body*

synthesize *is_supset_body* **from_definition**
arity_theorem for *is_supset_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_supset_body*:
separation(##*M*, *is_supset_body*(##*M*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*] ⊨ *is_supset_body_fm*(*0*),*THEN iffD1*])
apply(*rule_tac is_supset_body_iff_sats*[**where** *env*=[_],*symmetric*])
apply(*simp_all*)
apply(*rule_tac separation_ax*[**where** *env*=[],*simplified*])
apply(*simp_all add*:*arity_is_supset_body_fm ord_simp_union is_supset_body_fm_type*)
done

lemma (**in** *M_ZF_trans*) *supset_body_abs*:
assumes (##*M*)(*x*)
shows *is_supset_body*(##*M*,*x*) ⟷ *supset_body*(*x*)
using *assms pair_in_M_iff apply_closed*
unfolding *supset_body_def is_supset_body_def*
by (*auto*)

lemma (**in** *M_ZF_trans*) *separation_supset_body*:
separation(##*M*, λ *x*. ∃ *a*. ∃ *b*. *x* = ⟨*a*,*b*⟩ ∧ *b* ⊆ *a*)
using *separation_is_supset_body supset_body_abs*
separation_cong[**where** *P*=*is_supset_body*(##*M*) **and** *M*=##*M*,*THEN iffD1*]
unfolding *supset_body_def*

by *simp*

lemma (**in** *M_ZF_trans*) *separation_is_fst*:
(##*M*)(*a*) ⟹ *separation*(##*M*, λ*x* . *is_fst*(##*M*,*x*,*a*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*,*a*] ⊨ *fst_fm*(*0*,*1*),*THEN iffD1*])
apply(*rule_tac fst_iff_sats*[**where** *env*=[_,*a*],*symmetric*])
apply(*simp_all*)
apply(*rule_tac separation_ax*[**where** *env*=[*a*],*simplified*])
apply(*simp_all add*:*arity_fst_fm ord_simp_union fst_fm_type*)
done

lemma (**in** *M_ZF_trans*) *separation_fst_equal*:
(##*M*)(*a*) ⟹ *separation*(##*M*, λ*x*. *fst*(*x*) = *a*)
using *separation_cong*[*THEN iffD1*,*OF* _ *separation_is_fst*[*of a*]]
iff_sym[*of is_fst*(##*M*) _ *a fst*, *OF fst_abs*]
by *auto*

lemma (**in** *M_ZF_trans*) *separation_is_apply*:
(##*M*)(*f*) ⟹ (##*M*)(*a*) ⟹ *separation*(##*M*, λ*x* . *is_apply*(##*M*,*f*,*x*,*a*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*,*f*,*a*] ⊨ *fun_apply_fm*(*1*,*0*,*2*),*THEN iffD1*])
apply(*rule_tac fun_apply_iff_sats*[**where** *env*=[_,*f*,*a*],*symmetric*])
apply(*simp_all*)
apply(*rule_tac separation_ax*[**where** *env*=[*f*,*a*],*simplified*])
apply(*simp_all add*:*arity_fun_apply_fm ord_simp_union*)
done

lemma (**in** *M_ZF_trans*) *separation_equal_apply*:
(##*M*)(*f*) ⟹ (##*M*)(*a*) ⟹ *separation*(##*M*, λ*x*. *f*‘*x* = *a*)
using *separation_cong*[*THEN iffD1*,*OF* _ *separation_is_apply*[*of f a*]] *apply_abs*
by *force*

definition *id_rel* :: [*i*⇒*o*,*i*] ⇒ *o* **where**
id_rel(*A*) ≡ λ*z*. ∃ *x*[*A*]. *z* = ⟨*x*, *x*⟩
relativize *id_rel is_id_rel*
synthesize *is_id_rel* **from_definition assuming** *nonempty*
arity_theorem for *is_id_rel_fm*

lemma (**in** *M_ZF_trans*) *separation_is_id_rel*:
separation(##*M*, *is_id_rel*(##*M*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*] ⊨ *is_id_rel_fm*(*0*),*THEN iffD1*])
apply(*rule_tac is_id_rel_iff_sats*[**where** *env*=[_],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[],*simplified*])
apply(*simp_all add*:*arity_is_id_rel_fm ord_simp_union is_id_rel_fm_type*)
done

lemma (**in** *M_ZF_trans*) *id_rel_abs*:
assumes (*##M*)(*x*)
shows *is_id_rel*(*##M*,*x*) ⟷ *id_rel*(*##M*,*x*)
using *assms zero_in_M pair_in_M_iff* **unfolding** *id_rel_def is_id_rel_def*
by *auto*

lemma (**in** *M_ZF_trans*) *separation_id_rel*:
separation(*##M*, λ*z*. ∃ *x*[*##M*]. *z* = ⟨*x*, *x*⟩)
using *id_rel_abs separation_is_id_rel*
unfolding *id_rel_def*
by (*rule_tac separation_cong*[**where** *P*=*is_id_rel*(*##M*),*THEN iffD1*])

definition *sndfst_eq_fstsnd* :: [*i*] ⇒ *o* **where**
sndfst_eq_fstsnd ≡ λ*x*. *snd*(*fst*(*x*)) = *fst*(*snd*(*x*))
relativize *sndfst_eq_fstsnd is_sndfst_eq_fstsnd*
synthesize *is_sndfst_eq_fstsnd* **from_definition assuming** *nonempty*
arity_theorem for *is_sndfst_eq_fstsnd_fm*

lemma (**in** *M_ZF_trans*) *separation_is_sndfst_eq_fstsnd*:
separation(*##M*, *is_sndfst_eq_fstsnd*(*##M*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*] ⊨ *is_sndfst_eq_fstsnd_fm*(*0*),*THEN iffD1*])
apply(*rule_tac is_sndfst_eq_fstsnd_iff_sats*[**where** *env*=[_],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[],*simplified*])
apply(*simp_all add*:*arity_is_sndfst_eq_fstsnd_fm ord_simp_union is_sndfst_eq_fstsnd_fm_type*)
done

lemma (**in** *M_ZF_trans*) *sndfst_eq_fstsnd_abs*:
assumes (*##M*)(*x*)
shows *is_sndfst_eq_fstsnd*(*##M*,*x*) ⟷ *sndfst_eq_fstsnd*(*x*)
using *assms pair_in_M_iff fst_abs snd_abs fst_snd_closed*
unfolding *sndfst_eq_fstsnd_def is_sndfst_eq_fstsnd_def*
by *auto*

lemma (**in** *M_ZF_trans*) *separation_sndfst_eq_fstsnd*:
separation(*##M*, λ*x*. *snd*(*fst*(*x*)) = *fst*(*snd*(*x*)))
using *sndfst_eq_fstsnd_abs separation_is_sndfst_eq_fstsnd*
unfolding *sndfst_eq_fstsnd_def*
by (*rule_tac separation_cong*[**where** *P*=*is_sndfst_eq_fstsnd*(*##M*),*THEN iffD1*])

definition *fstfst_eq_fstsnd* :: [*i*] ⇒ *o* **where**
fstfst_eq_fstsnd ≡ λ*x*. *fst*(*fst*(*x*)) = *fst*(*snd*(*x*))
relativize *fstfst_eq_fstsnd is_fstfst_eq_fstsnd*

synthesize *is_fstfst_eq_fstsnd* **from_definition assuming** *nonempty*
arity_theorem for *is_fstfst_eq_fstsnd_fm*

lemma (**in** *M_ZF_trans*) *separation_is_fstfst_eq_fstsnd*:
separation(##*M*, *is_fstfst_eq_fstsnd*(##*M*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*] ⊨ *is_fstfst_eq_fstsnd_fm*(*0*),*THEN iffD1*])
apply(*rule_tac is_fstfst_eq_fstsnd_iff_sats*[**where** *env*=[_],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[],*simplified*])
apply(*simp_all add*:*arity_is_fstfst_eq_fstsnd_fm ord_simp_union is_fstfst_eq_fstsnd_fm_type*)
done

lemma (**in** *M_ZF_trans*) *fstfst_eq_fstsnd_abs*:
assumes (##*M*)(*x*)
shows *is_fstfst_eq_fstsnd*(##*M*,*x*) ⟷ *fstfst_eq_fstsnd*(*x*)
using *assms pair_in_M_iff fst_abs snd_abs fst_snd_closed*
unfolding *fstfst_eq_fstsnd_def is_fstfst_eq_fstsnd_def*
by *auto*

lemma (**in** *M_ZF_trans*) *separation_fstfst_eq_fstsnd*:
separation(##*M*, λ*x*. *fst*(*fst*(*x*)) = *fst*(*snd*(*x*)))
using *fstfst_eq_fstsnd_abs separation_is_fstfst_eq_fstsnd*
unfolding *fstfst_eq_fstsnd_def*
by (*rule_tac separation_cong*[**where** *P*=*is_fstfst_eq_fstsnd*(##*M*),*THEN iffD1*])

definition *fstfst_eq* :: [*i*,*i*] ⇒ *o* **where**
fstfst_eq(*a*) ≡ λ*x*. *fst*(*fst*(*x*)) = *a*

relativize *fstfst_eq is_fstfst_eq*
synthesize *is_fstfst_eq* **from_definition assuming** *nonempty*
arity_theorem for *is_fstfst_eq_fm*

lemma (**in** *M_ZF_trans*) *separation_is_fstfst_eq*:
(##*M*)(*a*) ⟹ *separation*(##*M*, *is_fstfst_eq*(##*M*,*a*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*,*a*] ⊨ *is_fstfst_eq_fm*(*1*,*0*),*THEN iffD1*])
apply(*rule_tac is_fstfst_eq_iff_sats*[**where** *env*=[_,*a*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*a*],*simplified*])
apply(*simp_all add*:*arity_is_fstfst_eq_fm ord_simp_union is_fstfst_eq_fm_type*)
done

lemma (**in** *M_ZF_trans*) *fstfst_eq_abs*:
assumes (##*M*)(*a*) (##*M*)(*x*)
shows *is_fstfst_eq*(##*M*,*a*,*x*) ⟷ *fstfst_eq*(*a*,*x*)
using *assms pair_in_M_iff fst_abs snd_abs fst_snd_closed*

unfolding *fstfst_eq_def is_fstfst_eq_def*
by *auto*

lemma (**in** *M_ZF_trans*) *separation_fstfst_eq*:
$(\#\#M)(a) \Longrightarrow$ *separation*$(\#\#M, \lambda x.\ fst(fst(x)) = a)$
using *fstfst_eq_abs separation_is_fstfst_eq*
unfolding *fstfst_eq_def*
by (*rule_tac separation_cong*[**where** *P=is_fstfst_eq*$(\#\#M,a)$,*THEN iffD1*])

definition *restrict_elem* :: $[i,i,i] \Rightarrow o$ **where**
$restrict\_elem(B,A) \equiv \lambda y.\ \exists x{\in}A.\ y = \langle x,\ restrict(x,\ B)\rangle$

relativize *restrict_elem is_restrict_elem*
synthesize *is_restrict_elem* **from_definition assuming** *nonempty*
arity_theorem for *is_restrict_elem_fm*

lemma (**in** *M_ZF_trans*) *separation_is_restrict_elem*:
$(\#\#M)(B) \Longrightarrow (\#\#M)(A) \Longrightarrow$ *separation*$(\#\#M,\ is\_restrict\_elem(\#\#M,B,A))$
apply(*rule_tac separation_cong*[
where $P{=}\lambda\ x\ .\ M,[x,A,B] \models$ *is_restrict_elem_fm*$(2,1,0)$,*THEN iffD1*])
apply(*rule_tac is_restrict_elem_iff_sats*[**where** *env*=[_,*A*,*B*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*B*],*simplified*])
apply(*simp_all add*:*arity_is_restrict_elem_fm ord_simp_union is_restrict_elem_fm_type*)
done

lemma (**in** *M_ZF_trans*) *restrict_elem_abs*:
assumes $(\#\#M)(B)$ $(\#\#M)(A)$ $(\#\#M)(x)$
shows *is_restrict_elem*$(\#\#M,B,A,x) \longleftrightarrow$ *restrict_elem*(B,A,x)
using *assms pair_in_M_iff fst_abs snd_abs fst_snd_closed*
unfolding *restrict_elem_def is_restrict_elem_def*
by *auto*

lemma (**in** *M_ZF_trans*) *separation_restrict_elem_aux*:
$(\#\#M)(B) \Longrightarrow (\#\#M)(A) \Longrightarrow$ *separation*$(\#\#M,\ \lambda y.\ \exists x{\in}A.\ y = \langle x,\ restrict(x,\ B)\rangle)$
using *restrict_elem_abs separation_is_restrict_elem*
unfolding *restrict_elem_def*
by (*rule_tac separation_cong*[**where** *P=is_restrict_elem*$(\#\#M,B,\_)$,*THEN iffD1*])

lemma (**in** *M_ZF_trans*) *separation_restrict_elem*:
$(\#\#M)(B) \Longrightarrow \forall\, A[\#\#M].$ *separation*$(\#\#M,\ \lambda y.\ \exists x{\in}A.\ y = \langle x,\ restrict(x,\ B)\rangle)$
using *separation_restrict_elem_aux* **by** *simp*

lemma (**in** *M_ZF_trans*) *separation_ordinal*:
separation$(\#\#M,\ ordinal(\#\#M))$

apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*] ⊨ *ordinal_fm*(*0*),*THEN iffD1*])
apply(*rule_tac ordinal_iff_sats*[**where** *env*=[_],*symmetric*])
apply(*simp_all*)
apply(*rule_tac separation_ax*[**where** *env*=[],*simplified*])
apply(*simp_all add*:*ord_simp_union*)
done

lemma (**in** *M_ZF_trans*) *separation_Ord*:
separation(##*M*, *Ord*)
using *separation_ordinal ordinal_abs*
separation_cong[**where** *P*=*ordinal*(##*M*) **and** *M*=##*M*,*THEN iffD1*]
unfolding *Ord_def*
by *simp*

definition *insnd_ballPair* :: [*i*,*i*,*i*] ⇒ *o* **where**
insnd_ballPair(*B*,*A*) ≡ λ*p*. ∀ *x*∈*B*. *x* ∈ *snd*(*p*) ⟷ (∀ *s*∈*fst*(*p*). ⟨*s*, *x*⟩ ∈ *A*)

relativize *insnd_ballPair is_insnd_ballPair*
synthesize *is_insnd_ballPair* **from_definition assuming** *nonempty*
arity_theorem for *is_insnd_ballPair_fm*

lemma (**in** *M_ZF_trans*) *separation_is_insnd_ballPair*:
(##*M*)(*B*) ⟹ (##*M*)(*A*) ⟹ *separation*(##*M*, *is_insnd_ballPair*(##*M*,*B*,*A*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*,*A*,*B*] ⊨ *is_insnd_ballPair_fm*(*2*,*1*,*0*),*THEN iffD1*])
apply(*rule_tac is_insnd_ballPair_iff_sats*[**where** *env*=[_,*A*,*B*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*B*],*simplified*])
apply(*simp_all add*:*arity_is_insnd_ballPair_fm ord_simp_union is_insnd_ballPair_fm_type*)
done

lemma (**in** *M_ZF_trans*) *insnd_ballPair_abs*:
assumes (##*M*)(*B*) (##*M*)(*A*) (##*M*)(*x*)
shows *is_insnd_ballPair*(##*M*,*B*,*A*,*x*) ⟷ *insnd_ballPair*(*B*,*A*,*x*)
using *assms pair_in_M_iff fst_abs snd_abs fst_snd_closed*
transM[*of* _ *B*] *transM*[*of* _ *snd*(*x*)] *transM*[*of* _ *fst*(*x*)]
unfolding *insnd_ballPair_def is_insnd_ballPair_def*
by (*auto*)

lemma (**in** *M_ZF_trans*) *separation_insnd_ballPair_aux*:
(##*M*)(*B*) ⟹ (##*M*)(*A*) ⟹ *separation*(##*M*, λ*p*. ∀ *x*∈*B*. *x* ∈ *snd*(*p*) ⟷
(∀ *s*∈*fst*(*p*). ⟨*s*, *x*⟩ ∈ *A*))
using *insnd_ballPair_abs separation_is_insnd_ballPair*
unfolding *insnd_ballPair_def*
by (*rule_tac separation_cong*[**where** *P*=*is_insnd_ballPair*(##*M*,*B*,_),*THEN*
iffD1])

lemma (**in** *M_ZF_trans*) *separation_insnd_ballPair*:
$(\#\#M)(B) \Longrightarrow \forall\, A[\#\#M].\ separation(\#\#M,\ \lambda p.\ \forall\, x{\in}B.\ x \in snd(p) \longleftrightarrow (\forall\, s{\in}fst(p).\ \langle s,\ x\rangle \in A))$
using *separation_insnd_ballPair_aux* **by** *simp*

definition *bex_restrict_eq_dom* :: $[i,i,i,i] \Rightarrow o$ **where**
bex_restrict_eq_dom$(B,A,x) \equiv \lambda dr.\ \exists\, r{\in}A\ .\ restrict(r,B) = x \wedge dr{=}domain(r)$

relativize *bex_restrict_eq_dom is_bex_restrict_eq_dom*
synthesize *is_bex_restrict_eq_dom* **from_definition assuming** *nonempty*
arity_theorem for *is_bex_restrict_eq_dom_fm*

lemma (**in** *M_ZF_trans*) *separation_is_bex_restrict_eq_dom*:
$(\#\#M)(B) \Longrightarrow (\#\#M)(A) \Longrightarrow (\#\#M)(x) \Longrightarrow separation(\#\#M,\ is\_bex\_restrict\_eq\_dom(\#\#M,B,A,x))$
apply(*rule_tac separation_cong*[
where $P{=}\lambda\ dr\ .\ M,[dr,x,A,B] \models is\_bex\_restrict\_eq\_dom\_fm(3,2,1,0)$, *THEN iffD1*])
apply(*rule_tac is_bex_restrict_eq_dom_iff_sats*[**where** *env*=[_,*x*,*A*,*B*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*x*,*A*,*B*],*simplified*])
apply(*simp_all add*:*arity_is_bex_restrict_eq_dom_fm ord_simp_union is_bex_restrict_eq_dom_fm_ty*
done
lemma (**in** *M_ZF_trans*) *bex_restrict_eq_dom_abs*:
assumes $(\#\#M)(B)$ $(\#\#M)(A)$ $(\#\#M)(x)$ $(\#\#M)(dr)$
shows *is_bex_restrict_eq_dom*$(\#\#M,B,A,x,dr) \longleftrightarrow bex\_restrict\_eq\_dom(B,A,x,dr)$
using *assms transM*[*of* _ *B*] *transM*[*of* _ *A*]
unfolding *bex_restrict_eq_dom_def is_bex_restrict_eq_dom_def*
by (*auto*)

lemma (**in** *M_ZF_trans*) *separation_restrict_eq_dom_eq_aux*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow (\#\#M)(x) \Longrightarrow separation(\#\#M,\ \lambda dr.\ \exists\, r{\in}A\ .\ restrict(r,B) = x \wedge dr{=}domain(r))$
using *bex_restrict_eq_dom_abs separation_is_bex_restrict_eq_dom*
unfolding *bex_restrict_eq_dom_def*
by (*rule_tac separation_cong*[**where** $P{=}is\_bex\_restrict\_eq\_dom(\#\#M,B,A,x)$, *THEN iffD1*])

lemma (**in** *M_ZF_trans*) *separation_restrict_eq_dom_eq*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow \forall\, x[\#\#M].\ separation(\#\#M,\ \lambda dr.\ \exists\, r{\in}A\ .\ restrict(r,B) = x \wedge dr{=}domain(r))$
using *separation_restrict_eq_dom_eq_aux* **by** *simp*

definition *insnd_restrict_eq_dom* :: $[i,i,i,i] \Rightarrow o$ **where**
insnd_restrict_eq_dom$(B,A,D) \equiv \lambda p.\ \forall\, x{\in}D.\ x \in snd(p) \longleftrightarrow (\exists\, r{\in}A.\ restrict(r, B) = fst(p) \wedge x = domain(r))$

definition *is_insnd_restrict_eq_dom* :: $[i{\Rightarrow}o,i,i,i,i] \Rightarrow o$ **where**
is_insnd_restrict_eq_dom$(N,B,A,D,p) \equiv$

$\exists\, c[N].$
$\exists\, a[N].$
$(\forall\, x[N].\ x \in D \longrightarrow x \in a \longleftrightarrow (\exists\, r[N].\ \exists\, b[N].\ (r \in A \wedge restriction(N, r, B, b)) \wedge b{=}c \wedge is\_domain(N, r, x))) \wedge$
$is\_snd(N, p, a) \wedge is\_fst(N, p, c)$

synthesize *is_insnd_restrict_eq_dom* **from_definition assuming** *nonempty*
arity_theorem for *is_insnd_restrict_eq_dom_fm*

lemma (**in** *M_ZF_trans*) *separation_is_insnd_restrict_eq_dom*:
$(\#\#M)(B) \Longrightarrow (\#\#M)(A) \Longrightarrow (\#\#M)(D) \Longrightarrow separation(\#\#M, is\_insnd\_restrict\_eq\_dom(\#\#M,B,A,D)$
apply(*rule_tac separation_cong*[
where $P{=}\lambda\, x\, .\ M,[x,D,A,B] \models is\_insnd\_restrict\_eq\_dom\_fm(3,2,1,0)$,*THEN iffD1*])
apply(*rule_tac is_insnd_restrict_eq_dom_iff_sats*[**where** *env*=[_,*D*,*A*,*B*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*D*,*A*,*B*],*simplified*])
apply(*simp_all add*:*arity_is_insnd_restrict_eq_dom_fm ord_simp_union is_insnd_restrict_eq_dom_fm_type*)
done

lemma (**in** *M_basic*) *insnd_restrict_eq_dom_abs*:
assumes $(M)(B)$ $(M)(A)$ $(M)(D)$ $(M)(x)$
shows $is\_insnd\_restrict\_eq\_dom(M,B,A,D,x) \longleftrightarrow insnd\_restrict\_eq\_dom(B,A,D,x)$
using *assms pair_in_M_iff fst_abs snd_abs fst_snd_closed domain_closed transM*[*of* _ *B*] *transM*[*of* _ *D*] *transM*[*of* _ *A*] *transM*[*of* _ *snd*(*x*)] *transM*[*of* _ *fst*(*x*)]
unfolding *insnd_restrict_eq_dom_def is_insnd_restrict_eq_dom_def*
by *simp*

lemma (**in** *M_ZF_trans*) *separation_restrict_eq_dom_eq_pair_aux*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow (\#\#M)(D) \Longrightarrow$
$separation(\#\#M, \lambda p.\ \forall\, x{\in}D.\ x \in snd(p) \longleftrightarrow (\exists\, r{\in}A.\ restrict(r, B) = fst(p) \wedge x = domain(r)))$
using *separation_is_insnd_restrict_eq_dom insnd_restrict_eq_dom_abs*
unfolding *insnd_restrict_eq_dom_def*
by (*rule_tac separation_cong*[**where** $P{=}is\_insnd\_restrict\_eq\_dom(\#\#M,B,A,D)$,*THEN iffD1*])

lemma (**in** *M_ZF_trans*) *separation_restrict_eq_dom_eq_pair*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow$
$\forall\, D[\#\#M].\ separation(\#\#M, \lambda p.\ \forall\, x{\in}D.\ x \in snd(p) \longleftrightarrow (\exists\, r{\in}A.\ restrict(r, B) = fst(p) \wedge x = domain(r)))$
using *separation_restrict_eq_dom_eq_pair_aux* **by** *simp*

synthesize *is_converse* **from_definition assuming** *nonempty*

arity_theorem for *is_converse_fm*

definition *ifrFb_body* **where**
ifrFb_body(*M*,*b*,*f*,*x*,*i*) ≡ *x* ∈
(*if b = 0 then if i* ∈ *range*(*f*) *then*
if M(*converse*(*f*) ‘ *i*) *then converse*(*f*) ‘ *i else 0 else 0 else if M*(*i*) *then i else 0*)

relativize functional *ifrFb_body ifrFb_body_rel*
relationalize *ifrFb_body_rel is_ifrFb_body*

synthesize *is_ifrFb_body* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrFb_body_fm*

definition *ifrangeF_body* :: [*i*⇒*o*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
ifrangeF_body(*M*,*A*,*b*,*f*) ≡ λ*y*. ∃ *x*∈*A*. *y* = ⟨*x*,μ *i*. *ifrFb_body*(*M*,*b*,*f*,*x*,*i*)⟩

relativize functional *ifrangeF_body ifrangeF_body_rel*
relationalize *ifrangeF_body_rel is_ifrangeF_body*

synthesize *is_ifrangeF_body* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrangeF_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_ifrangeF_body*:
(##*M*)(*A*) ⟹ (##*M*)(*r*) ⟹ (##*M*)(*s*) ⟹ *separation*(##*M*, *is_ifrangeF_body*(##*M*,*A*,*r*,*s*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*,*A*,*r*,*s*] ⊨ *is_ifrangeF_body_fm*(*1*,*2*,*3*,*0*),*THEN iffD1*])
apply(*rule_tac is_ifrangeF_body_iff_sats*[**where** *env*=[_,*A*,*r*,*s*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*r*,*s*],*simplified*])
apply(*simp_all add*:*arity_is_ifrangeF_body_fm ord_simp_union is_ifrangeF_body_fm_type*)
done

lemma (**in** *M_basic*) *is_ifrFb_body_closed*: *M*(*r*) ⟹ *M*(*s*) ⟹ *is_ifrFb_body*(*M*,
r, *s*, *x*, *i*) ⟹ *M*(*i*)
unfolding *ifrangeF_body_def is_ifrangeF_body_def is_ifrFb_body_def If_abs*
by (*cases i*∈*range*(*s*); *cases r=0*; *auto dest*:*transM*)

lemma (**in** *M_ZF_trans*) *ifrangeF_body_abs*:
assumes (##*M*)(*A*) (##*M*)(*r*) (##*M*)(*s*) (##*M*)(*x*)
shows *is_ifrangeF_body*(##*M*,*A*,*r*,*s*,*x*) ⟷ *ifrangeF_body*(##*M*,*A*,*r*,*s*,*x*)
proof -
{
fix *a*
assume *a*∈*M*
with *assms*
have (μ *i*. *i*∈ *M* ∧ *is_ifrFb_body*(##*M*, *r*, *s*, *z*, *i*))= (μ *i*. *is_ifrFb_body*(##*M*,
r, *s*, *z*, *i*)) **for** *z*
using *is_ifrFb_body_closed*[*of r s z*]
by (*rule_tac Least_cong*[*of* λ*i*. *i*∈*M* ∧ *is_ifrFb_body*(##*M*,*r*,*s*,*z*,*i*)]) *auto*

```
    moreover
    have (μ i. is_ifrFb_body(##M, r, s, z, i))= (μ i. ifrFb_body(##M, r, s, z,
i)) for z
   proof (rule_tac Least_cong[of λi. is_ifrFb_body(##M,r,s,z,i) λi. ifrFb_body(##M,r,s,z,i)])
     fix y
     from assms ‹a∈M›
     show is_ifrFb_body(##M, r, s, z, y) ⟷ ifrFb_body(##M, r, s, z, y)
       using If_abs apply_0
       unfolding ifrFb_body_def is_ifrFb_body_def
       by (cases y∈M; cases y∈range(s); cases converse(s)‘y ∈ M;
           auto dest:transM split del: split_if del:iffI)
         (auto simp flip:setclass_iff; (force simp only:setclass_iff))+
    qed
    moreover from ‹a∈M›
    have least(##M, λi. i ∈ M ∧ is_ifrFb_body(##M, r, s, z, i), a)
     ⟷ a = (μ i.  i∈ M ∧ is_ifrFb_body(##M, r, s, z,i)) for z
     using If_abs least_abs'[of λi. (##M)(i) ∧ is_ifrFb_body(##M,r,s,z,i) a]
     by simp
    ultimately
    have least(##M, λi. i ∈ M ∧ is_ifrFb_body(##M, r, s, z, i), a)
     ⟷ a = (μ i. ifrFb_body(##M, r, s, z,i)) for z
     by simp
  }
  with assms
  show ?thesis
    using  pair_in_M_iff apply_closed zero_in_M transitivity[of _ A]
    unfolding ifrangeF_body_def is_ifrangeF_body_def
    by (auto dest:transM)
qed

lemma (in M_ZF_trans) separation_ifrangeF_body:
  (##M)(A) ⟹ (##M)(b) ⟹ (##M)(f) ⟹ separation
       (##M, λy.  ∃x∈A. y = ⟨x, μ i. x ∈ if_range_F_else_F(λx. if (##M)(x)
then x else 0, b, f, i)⟩)
  using separation_is_ifrangeF_body ifrangeF_body_abs
   separation_cong[where P=is_ifrangeF_body(##M,A,b,f) and M=##M,THEN
iffD1]
 unfolding ifrangeF_body_def if_range_F_def if_range_F_else_F_def ifrFb_body_def
  by simp



definition ifrFb_body2 where
  ifrFb_body2(M,G,b,f,x,i) ≡ x ∈
  (if b = 0 then if i ∈ range(f) then
  if M(converse(f) ‘ i) then G‘(converse(f) ‘ i) else 0 else 0 else if M(i) then G‘i
else 0)

relativize functional ifrFb_body2 ifrFb_body2_rel
```

relationalize *ifrFb_body2_rel is_ifrFb_body2*

synthesize *is_ifrFb_body2* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrFb_body2_fm*

definition *ifrangeF_body2* :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
ifrangeF_body2(*M*,*A*,*G*,*b*,*f*) ≡ λ*y*. ∃ *x*∈*A*. *y* = ⟨*x*,μ *i*. *ifrFb_body2*(*M*,*G*,*b*,*f*,*x*,*i*)⟩

relativize functional *ifrangeF_body2 ifrangeF_body2_rel*
relationalize *ifrangeF_body2_rel is_ifrangeF_body2*

synthesize *is_ifrangeF_body2* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrangeF_body2_fm*

lemma (**in** *M_ZF_trans*) *separation_is_ifrangeF_body2*:
(##*M*)(*A*) ⟹ (##*M*)(*G*) ⟹ (##*M*)(*r*) ⟹ (##*M*)(*s*) ⟹ *separation*(##*M*,
is_ifrangeF_body2(##*M*,*A*,*G*,*r*,*s*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*,*A*,*G*,*r*,*s*] ⊨ *is_ifrangeF_body2_fm*(*1*,*2*,*3*,*4*,*0*),*THEN*
iffD1])
apply(*rule_tac is_ifrangeF_body2_iff_sats*[**where** *env*=[_,*A*,*G*,*r*,*s*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*G*,*r*,*s*],*simplified*])
apply(*simp_all add*:*arity_is_ifrangeF_body2_fm ord_simp_union is_ifrangeF_body2_fm_type*)
done

lemma (**in** *M_basic*) *is_ifrFb_body2_closed*: *M*(*G*) ⟹ *M*(*r*) ⟹ *M*(*s*) ⟹
is_ifrFb_body2(*M*, *G*, *r*, *s*, *x*, *i*) ⟹ *M*(*i*)
unfolding *ifrangeF_body2_def is_ifrangeF_body2_def is_ifrFb_body2_def If_abs*
by (*cases i*∈*range*(*s*); *cases r=0*; *auto dest*:*transM*)

lemma (**in** *M_ZF_trans*) *ifrangeF_body2_abs*:
assumes (##*M*)(*A*) (##*M*)(*G*) (##*M*)(*r*) (##*M*)(*s*) (##*M*)(*x*)
shows *is_ifrangeF_body2*(##*M*,*A*,*G*,*r*,*s*,*x*) ⟷ *ifrangeF_body2*(##*M*,*A*,*G*,*r*,*s*,*x*)
proof -
{
fix *a*
assume *a*∈*M*
with *assms*
have (μ *i*. *i*∈ *M* ∧ *is_ifrFb_body2*(##*M*, *G*, *r*, *s*, *z*, *i*))= (μ *i*. *is_ifrFb_body2*(##*M*,
G, *r*, *s*, *z*, *i*)) **for** *z*
using *is_ifrFb_body2_closed*[*of G r s z*]
by (*rule_tac Least_cong*[*of* λ*i*. *i*∈*M* ∧ *is_ifrFb_body2*(##*M*,*G*,*r*,*s*,*z*,*i*)])
auto
moreover
have (μ *i*. *is_ifrFb_body2*(##*M*, *G*, *r*, *s*, *z*, *i*))= (μ *i*. *ifrFb_body2*(##*M*, *G*,
r, *s*, *z*, *i*)) **for** *z*
proof (*rule_tac Least_cong*[*of* λ*i*. *is_ifrFb_body2*(##*M*,*G*,*r*,*s*,*z*,*i*) λ*i*. *ifrFb_body2*(##*M*,*G*,*r*,*s*,*z*,*i*)])
fix *y*

```
      from assms ‹a∈M›
      show is_ifrFb_body2(##M, G, r, s, z, y) ⟷ ifrFb_body2(##M, G, r, s,
z, y)
        using If_abs apply_0
        unfolding ifrFb_body2_def is_ifrFb_body2_def
        by (cases y∈M; cases y∈range(s); cases converse(s)‘y ∈ M;
            auto dest:transM split del: split_if del:iffI)
          (auto simp flip:setclass_iff; (force simp only:setclass_iff))+
    qed
    moreover from ‹a∈M›
    have least(##M, λi. i ∈ M ∧ is_ifrFb_body2(##M, G, r, s, z, i), a)
      ⟷ a = (μ i.  i∈ M ∧ is_ifrFb_body2(##M, G, r, s, z,i)) for z
      using If_abs least_abs'[of λi. (##M)(i) ∧ is_ifrFb_body2(##M,G,r,s,z,i)
a]
      by simp
    ultimately
    have least(##M, λi. i ∈ M ∧ is_ifrFb_body2(##M, G, r, s, z, i), a)
      ⟷ a = (μ i. ifrFb_body2(##M, G, r, s, z,i)) for z
      by simp
  }
  with assms
  show ?thesis
    using  pair_in_M_iff apply_closed zero_in_M transitivity[of _ A]
    unfolding ifrangeF_body2_def is_ifrangeF_body2_def
    by (auto dest:transM)
qed

lemma (in M_ZF_trans) separation_ifrangeF_body2:
  (##M)(A) ⟹ (##M)(G) ⟹ (##M)(b) ⟹ (##M)(f) ⟹
      separation
       (##M,
        λy. ∃x∈A.
              y =
              ⟨x, μ i. x ∈
                    if_range_F_else_F(λa. if (##M)(a) then G ‘ a else 0, b, f,
i)⟩)
  using separation_is_ifrangeF_body2 ifrangeF_body2_abs
   separation_cong[where P=is_ifrangeF_body2(##M,A,G,b,f) and M=##M,THEN
iffD1]
 unfolding ifrangeF_body2_def if_range_F_def if_range_F_else_F_def ifrFb_body2_def
  by simp



definition ifrFb_body3 where
  ifrFb_body3(M,G,b,f,x,i) ≡ x ∈
  (if b = 0 then if i ∈ range(f) then
  if M(converse(f) ‘ i) then G-‘‘{converse(f) ‘ i} else 0 else 0 else if M(i) then
G-‘‘{i} else 0)
```

relativize functional *ifrFb_body3 ifrFb_body3_rel*
relationalize *ifrFb_body3_rel is_ifrFb_body3*

synthesize *is_ifrFb_body3* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrFb_body3_fm*

definition *ifrangeF_body3* :: $[i⇒o,i,i,i,i] ⇒ o$ **where**
ifrangeF_body3$(M,A,G,b,f) ≡ λy.\ ∃x∈A.\ y = ⟨x,μ\ i.\ ifrFb\_body3(M,G,b,f,x,i)⟩$

relativize functional *ifrangeF_body3 ifrangeF_body3_rel*
relationalize *ifrangeF_body3_rel is_ifrangeF_body3*

synthesize *is_ifrangeF_body3* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrangeF_body3_fm*

lemma (**in** *M_ZF_trans*) *separation_is_ifrangeF_body3*:
$(##M)(A) ⟹ (##M)(G) ⟹ (##M)(r) ⟹ (##M)(s) ⟹ separation(##M,$
is_ifrangeF_body3$(##M,A,G,r,s))$
apply(*rule_tac separation_cong*[
where $P=λ\ x\ .\ M,[x,A,G,r,s] ⊨ is\_ifrangeF\_body3\_fm(1,2,3,4,0),THEN$
iffD1])
apply(*rule_tac is_ifrangeF_body3_iff_sats*[**where** *env*=[_,*A*,*G*,*r*,*s*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*G*,*r*,*s*],*simplified*])
apply(*simp_all add*:*arity_is_ifrangeF_body3_fm ord_simp_union is_ifrangeF_body3_fm_type*)
done

lemma (**in** *M_basic*) *is_ifrFb_body3_closed*: $M(G) ⟹ M(r) ⟹ M(s) ⟹$
is_ifrFb_body3$(M,\ G,\ r,\ s,\ x,\ i) ⟹ M(i)$
unfolding *ifrangeF_body3_def is_ifrangeF_body3_def is_ifrFb_body3_def If_abs*
by (*cases* $i∈range(s)$; *cases* $r=0$; *auto dest*:*transM*)

lemma (**in** *M_ZF_trans*) *ifrangeF_body3_abs*:
assumes $(##M)(A)\ (##M)(G)\ (##M)(r)\ (##M)(s)\ (##M)(x)$
shows *is_ifrangeF_body3*$(##M,A,G,r,s,x) ⟷ ifrangeF\_body3(##M,A,G,r,s,x)$
proof -
{
fix *a*
assume $a∈M$
with *assms*
have $(μ\ i.\ i∈ M ∧ is\_ifrFb\_body3(##M,\ G,\ r,\ s,\ z,\ i))= (μ\ i.\ is\_ifrFb\_body3(##M,$
$G,\ r,\ s,\ z,\ i))$ **for** *z*
using *is_ifrFb_body3_closed*[*of G r s z*]
by (*rule_tac Least_cong*[*of* $λi.\ i∈M\ ∧\ is\_ifrFb\_body3(##M,G,r,s,z,i)$])
auto
moreover
have $(μ\ i.\ is\_ifrFb\_body3(##M,\ G,\ r,\ s,\ z,\ i))= (μ\ i.\ ifrFb\_body3(##M,\ G,$
$r,\ s,\ z,\ i))$ **for** *z*

```
  proof (rule_tac Least_cong[of λi. is_ifrFb_body3(##M,G,r,s,z,i) λi. ifrFb_body3(##M,G,r,s,z,i)])
    fix y
    from assms ‹a∈M›
    show is_ifrFb_body3(##M, G, r, s, z, y) ⟷ ifrFb_body3(##M, G, r, s,
z, y)
      using If_abs apply_0
      unfolding ifrFb_body3_def is_ifrFb_body3_def
      by (cases y∈M; cases y∈range(s); cases converse(s)‘y ∈ M;
          auto dest:transM split del: split_if del:iffI)
        (auto simp flip:setclass_iff; (force simp only:setclass_iff))+
   qed
   moreover from ‹a∈M›
   have least(##M, λi. i ∈ M ∧ is_ifrFb_body3(##M, G, r, s, z, i), a)
     ⟷ a = (μ i.  i∈ M ∧ is_ifrFb_body3(##M, G, r, s, z,i)) for z
     using If_abs least_abs'[of λi. (##M)(i) ∧ is_ifrFb_body3(##M,G,r,s,z,i)
a]
     by simp
   ultimately
   have least(##M, λi. i ∈ M ∧ is_ifrFb_body3(##M, G, r, s, z, i), a)
     ⟷ a = (μ i. ifrFb_body3(##M, G, r, s, z,i)) for z
     by simp
 }
 with assms
 show ?thesis
   using  pair_in_M_iff apply_closed zero_in_M transitivity[of _ A]
   unfolding ifrangeF_body3_def is_ifrangeF_body3_def
   by (auto dest:transM)
qed

lemma (in M_ZF_trans) separation_ifrangeF_body3:
 (##M)(A) ⟹ (##M)(G) ⟹ (##M)(b) ⟹ (##M)(f) ⟹
     separation
      (##M,
       λy. ∃x∈A.
             y =
             ⟨x, μ i. x ∈
                    if_range_F_else_F(λa. if (##M)(a) then G-‘‘{a} else 0, b,
f, i)⟩)
 using separation_is_ifrangeF_body3 ifrangeF_body3_abs
  separation_cong[where P=is_ifrangeF_body3(##M,A,G,b,f) and M=##M,THEN
iffD1]
 unfolding ifrangeF_body3_def if_range_F_def if_range_F_else_F_def ifrFb_body3_def
 by simp



definition ifrFb_body4 where
 ifrFb_body4(G,b,f,x,i) ≡ x ∈
 (if b = 0 then if i ∈ range(f) then G‘(converse(f) ‘ i) else 0 else G‘i)
```

relativize functional *ifrFb_body4 ifrFb_body4_rel*
relationalize *ifrFb_body4_rel is_ifrFb_body4*

synthesize *is_ifrFb_body4* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrFb_body4_fm*

definition *ifrangeF_body4* :: $[i \Rightarrow o,i,i,i,i] \Rightarrow o$ **where**
$ifrangeF\_body4(M,A,G,b,f) \equiv \lambda y.\ \exists x{\in}A.\ y = \langle x,\mu\ i.\ ifrFb\_body4(G,b,f,x,i)\rangle$

relativize functional *ifrangeF_body4 ifrangeF_body4_rel*
relationalize *ifrangeF_body4_rel is_ifrangeF_body4*

synthesize *is_ifrangeF_body4* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrangeF_body4_fm*

lemma (**in** *M_ZF_trans*) *separation_is_ifrangeF_body4*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(G) \Longrightarrow (\#\#M)(r) \Longrightarrow (\#\#M)(s) \Longrightarrow separation(\#\#M,$
$is\_ifrangeF\_body4(\#\#M,A,G,r,s))$
apply(*rule_tac separation_cong*[
where $P{=}\lambda\ x\ .\ M,[x,A,G,r,s] \models is\_ifrangeF\_body4\_fm(1,2,3,4,0)$, *THEN*
iffD1])
apply(*rule_tac is_ifrangeF_body4_iff_sats*[**where** *env*=[_,*A*,*G*,*r*,*s*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*G*,*r*,*s*],*simplified*])
apply(*simp_all add*:*arity_is_ifrangeF_body4_fm ord_simp_union is_ifrangeF_body4_fm_type*)
done

lemma (**in** *M_basic*) *is_ifrFb_body4_closed*: $M(G) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow$
$is\_ifrFb\_body4(M,\ G,\ r,\ s,\ x,\ i) \Longrightarrow M(i)$
using *If_abs*
unfolding *ifrangeF_body4_def is_ifrangeF_body4_def is_ifrFb_body4_def fun_apply_def*
by (*cases* $i{\in}range(s)$; *cases* $r{=}0$; *auto dest*:*transM*)

lemma (**in** *M_ZF_trans*) *ifrangeF_body4_abs*:
assumes $(\#\#M)(A)$ $(\#\#M)(G)$ $(\#\#M)(r)$ $(\#\#M)(s)$ $(\#\#M)(x)$
shows $is\_ifrangeF\_body4(\#\#M,A,G,r,s,x) \longleftrightarrow ifrangeF\_body4(\#\#M,A,G,r,s,x)$
proof -
{
fix *a*
assume $a{\in}M$
with *assms*
have $(\mu\ i.\ i{\in}\ M \wedge is\_ifrFb\_body4(\#\#M,\ G,\ r,\ s,\ z,\ i)){=}(\mu\ i.\ is\_ifrFb\_body4(\#\#M,$
$G,\ r,\ s,\ z,\ i))$ **for** *z*
using *is_ifrFb_body4_closed*[*of G r s z*]
by (*rule_tac Least_cong*[*of* $\lambda i.\ i{\in}M\ \wedge\ is\_ifrFb\_body4(\#\#M,G,r,s,z,i)$])
auto
moreover
have $(\mu\ i.\ is\_ifrFb\_body4(\#\#M,\ G,\ r,\ s,\ z,\ i)){=}(\mu\ i.\ ifrFb\_body4(G,\ r,\ s,\ z,$

```
i)) if z∈M for z
  proof (rule_tac Least_cong[of λi. is_ifrFb_body4(##M,G,r,s,z,i) λi. ifrFb_body4(G,r,s,z,i)])
    fix y
    from assms ‹a∈M› ‹z∈M›
    show is_ifrFb_body4(##M, G, r, s, z, y) ⟷ ifrFb_body4(G, r, s, z, y)
      using If_abs apply_0
      unfolding ifrFb_body4_def is_ifrFb_body4_def
      apply (cases y∈M; cases y∈range(s); cases r=0; cases y∈domain(G);
          auto dest:transM split del: split_if del:iffI)
    by (auto simp flip:setclass_iff; (force simp only: fun_apply_def setclass_iff))
       (auto simp flip:setclass_iff simp: fun_apply_def )
  qed
  moreover from ‹a∈M›
  have least(##M, λi. i ∈ M ∧ is_ifrFb_body4(##M, G, r, s, z, i), a)
    ⟷ a = (μ i.  i∈ M ∧ is_ifrFb_body4(##M, G, r, s, z,i)) for z
    using If_abs least_abs'[of λi. (##M)(i) ∧ is_ifrFb_body4(##M,G,r,s,z,i)
a]
    by simp
  ultimately
  have z∈M ⟹ least(##M, λi. i ∈ M ∧ is_ifrFb_body4(##M, G, r, s, z, i),
a)
    ⟷ a = (μ i. ifrFb_body4(G, r, s, z,i)) for z
    by simp
 }
 with assms
 show ?thesis
  using  pair_in_M_iff apply_closed zero_in_M transitivity[of _ A]
  unfolding ifrangeF_body4_def is_ifrangeF_body4_def
  by (auto dest:transM)
qed

lemma (in M_ZF_trans) separation_ifrangeF_body4:
 (##M)(A) ⟹ (##M)(G) ⟹ (##M)(b) ⟹ (##M)(f) ⟹
      separation(##M, λy. ∃x∈A. y = ⟨x, μ i. x ∈ if_range_F_else_F((‘)(G),
b, f, i)⟩)
 using separation_is_ifrangeF_body4 ifrangeF_body4_abs
  separation_cong[where P=is_ifrangeF_body4(##M,A,G,b,f) and M=##M,THEN
iffD1]
 unfolding ifrangeF_body4_def if_range_F_def if_range_F_else_F_def ifrFb_body4_def
 by simp



definition ifrFb_body5 where
 ifrFb_body5(G,b,f,x,i) ≡ x ∈
 (if b = 0 then if i ∈ range(f) then {xa ∈ G . converse(f) ‘ i ∈ xa} else 0 else {xa
∈ G . i ∈ xa})

relativize functional ifrFb_body5 ifrFb_body5_rel
```

relationalize *ifrFb_body5_rel is_ifrFb_body5*

synthesize *is_ifrFb_body5* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrFb_body5_fm*

definition *ifrangeF_body5* :: $[i⇒o,i,i,i,i] ⇒ o$ **where**
$ifrangeF\_body5(M,A,G,b,f) ≡ λy.\ ∃x∈A.\ y = ⟨x,μ\ i.\ ifrFb\_body5(G,b,f,x,i)⟩$

relativize functional *ifrangeF_body5 ifrangeF_body5_rel*
relationalize *ifrangeF_body5_rel is_ifrangeF_body5*

synthesize *is_ifrangeF_body5* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrangeF_body5_fm*

lemma (**in** *M_ZF_trans*) *separation_is_ifrangeF_body5*:
$(##M)(A) ⟹ (##M)(G) ⟹ (##M)(r) ⟹ (##M)(s) ⟹ separation(##M,$
$is\_ifrangeF\_body5(##M,A,G,r,s))$
apply(*rule_tac separation_cong*[
where $P=λ\ x\ .\ M,[x,A,G,r,s] ⊨ is\_ifrangeF\_body5\_fm(1,2,3,4,0),THEN$
iffD1])
apply(*rule_tac is_ifrangeF_body5_iff_sats*[**where** *env*=[_,*A*,*G*,*r*,*s*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*G*,*r*,*s*],*simplified*])
apply(*simp_all add*:*arity_is_ifrangeF_body5_fm ord_simp_union is_ifrangeF_body5_fm_type*)
done

lemma (**in** *M_basic*) *is_ifrFb_body5_closed*: $M(G) ⟹ M(r) ⟹ M(s) ⟹$
$is\_ifrFb\_body5(M,\ G,\ r,\ s,\ x,\ i) ⟹ M(i)$
using *If_abs*
unfolding *ifrangeF_body5_def is_ifrangeF_body5_def is_ifrFb_body5_def fun_apply_def*
by (*cases* $i∈range(s)$; *cases* $r=0$; *auto dest*:*transM*)

definition *toplevel6_body* :: $[i,i] ⇒ o$ **where**
$toplevel6\_body(R) ≡ λx.\ domain(x) = R$

relativize functional *toplevel6_body toplevel6_body_rel*
relationalize *toplevel6_body_rel is_toplevel6_body*

synthesize *is_toplevel6_body* **from_definition assuming** *nonempty*
arity_theorem for *is_toplevel6_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_toplevel6_body*:
$(##M)(A) ⟹ separation(##M,\ is\_toplevel6\_body(##M,A))$
apply(*rule_tac separation_cong*[
where $P=λ\ x\ .\ M,[x,A] ⊨ is\_toplevel6\_body\_fm(1,0),THEN\ iffD1$])
apply(*rule_tac is_toplevel6_body_iff_sats*[**where** *env*=[_,*A*],*symmetric*])
apply(*simp_all add*:*nonempty*[*simplified*])
apply(*rule_tac separation_ax*[**where** *env*=[*A*],*simplified*])

apply(*simp_all add*:*arity_is_toplevel6_body_fm ord_simp_union is_toplevel6_body_fm_type*)
done

lemma (**in** *M_ZF_trans*) *toplevel6_body_abs*:
assumes (##*M*)(*R*) (##*M*)(*x*)
shows *is_toplevel6_body*(##*M*,*R*,*x*) ⟷ *toplevel6_body*(*R*,*x*)
using *assms pair_in_M_iff is_Int_abs*
unfolding *toplevel6_body_def is_toplevel6_body_def*
by (*auto simp*:*domain_closed*[*simplified*])

lemma (**in** *M_ZF_trans*) *separation_toplevel6_body*:
(##*M*)(*R*) ⟹ *separation*
(##*M*, λ*x*. *domain*(*x*) = *R*)
using *separation_is_toplevel6_body toplevel6_body_abs*
unfolding *toplevel6_body_def*
by (*rule_tac separation_cong*[
where *P*=*is_toplevel6_body*(##*M*,*R*),*THEN iffD1*])

lemma (**in** *M_ZF_trans*) *ifrangeF_body5_abs*:
assumes (##*M*)(*A*) (##*M*)(*G*) (##*M*)(*r*) (##*M*)(*s*) (##*M*)(*x*)
shows *is_ifrangeF_body5*(##*M*,*A*,*G*,*r*,*s*,*x*) ⟷ *ifrangeF_body5*(##*M*,*A*,*G*,*r*,*s*,*x*)
proof -
{
fix *a*
assume *a*∈*M*
with *assms*
have (μ *i*. *i*∈ *M* ∧ *is_ifrFb_body5*(##*M*, *G*, *r*, *s*, *z*, *i*))= (μ *i*. *is_ifrFb_body5*(##*M*, *G*, *r*, *s*, *z*, *i*)) **for** *z*
using *is_ifrFb_body5_closed*[*of G r s z*]
by (*rule_tac Least_cong*[*of* λ*i*. *i*∈*M* ∧ *is_ifrFb_body5*(##*M*,*G*,*r*,*s*,*z*,*i*)])
auto
moreover
have (μ *i*. *is_ifrFb_body5*(##*M*, *G*, *r*, *s*, *z*, *i*))= (μ *i*. *ifrFb_body5*(*G*, *r*, *s*, *z*, *i*)) **if** *z*∈*M* **for** *z*
proof (*rule_tac Least_cong*[*of* λ*i*. *is_ifrFb_body5*(##*M*,*G*,*r*,*s*,*z*,*i*) λ*i*. *ifrFb_body5*(*G*,*r*,*s*,*z*,*i*)])
fix *y*
from *assms* ‹*a*∈*M*› ‹*z*∈*M*›
show *is_ifrFb_body5*(##*M*, *G*, *r*, *s*, *z*, *y*) ⟷ *ifrFb_body5*(*G*, *r*, *s*, *z*, *y*)
using *If_abs apply_0 separation_in separation_in_rev*
unfolding *ifrFb_body5_def is_ifrFb_body5_def*
apply (*cases y*∈*M*; *cases y*∈*range*(*s*); *cases r*=*0*; *cases y*∈*domain*(*G*);
auto dest:*transM split del*: *split_if del*:*iffI*)
apply (*auto simp flip*:*setclass_iff*; (*force simp only*: *fun_apply_def setclass_iff*))
apply (*auto simp flip*:*setclass_iff simp*: *fun_apply_def*)
apply (*auto dest*:*transM*)
done
qed
moreover from ‹*a*∈*M*›
have *least*(##*M*, λ*i*. *i* ∈ *M* ∧ *is_ifrFb_body5*(##*M*, *G*, *r*, *s*, *z*, *i*), *a*)

$\longleftrightarrow$ $a = (\mu\ i.\ \ i{\in}\ M \wedge is\_ifrFb\_body5(\#\#M,\ G,\ r,\ s,\ z,i))$ **for** z
using *If_abs least_abs'*[*of* $\lambda i.\ (\#\#M)(i) \wedge is\_ifrFb\_body5(\#\#M,G,r,s,z,i)$ a]
by *simp*
ultimately
have $z{\in}M \Longrightarrow least(\#\#M,\ \lambda i.\ i \in M \wedge is\_ifrFb\_body5(\#\#M,\ G,\ r,\ s,\ z,\ i),\ a)$
$\longleftrightarrow$ $a = (\mu\ i.\ ifrFb\_body5(G,\ r,\ s,\ z,i))$ **for** z
by *simp*
}
with *assms*
show *?thesis*
using *pair_in_M_iff apply_closed zero_in_M transitivity*[*of* _ A]
unfolding *ifrangeF_body5_def is_ifrangeF_body5_def*
by (*auto dest*:*transM*)
qed

lemma (**in** *M_ZF_trans*) *separation_ifrangeF_body5*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(G) \Longrightarrow (\#\#M)(b) \Longrightarrow (\#\#M)(f) \Longrightarrow$
$separation(\#\#M,\ \lambda y.\ \exists\, x{\in}A.\ y = \langle x,\ \mu\ i.\ x \in if\_range\_F\_else\_F(\lambda x.\ \{xa \in G\ .\ x \in xa\},\ b,\ f,\ i)\rangle)$
using *separation_is_ifrangeF_body5 ifrangeF_body5_abs*
separation_cong[**where** $P{=}is\_ifrangeF\_body5(\#\#M,A,G,b,f)$ **and** $M{=}\#\#M$,*THEN iffD1*]
unfolding *ifrangeF_body5_def if_range_F_def if_range_F_else_F_def ifrFb_body5_def*
by *simp*

definition *ifrFb_body6* **where**
$ifrFb\_body6(G,b,f,x,i) \equiv x \in$
(*if* $b = 0$ *then if* $i \in range(f)$ *then* $\{p{\in}G\ .\ domain(p) = converse(f)\ `\ i\}$ *else* 0 *else* $\{p{\in}G\ .\ domain(p) = i\}$)

relativize functional *ifrFb_body6 ifrFb_body6_rel*
relationalize *ifrFb_body6_rel is_ifrFb_body6*

synthesize *is_ifrFb_body6* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrFb_body6_fm*

definition *ifrangeF_body6* :: $[i{\Rightarrow}o,i,i,i,i] \Rightarrow o$ **where**
$ifrangeF\_body6(M,A,G,b,f) \equiv \lambda y.\ \exists\, x{\in}A.\ y = \langle x,\mu\ i.\ ifrFb\_body6(G,b,f,x,i)\rangle$

relativize functional *ifrangeF_body6 ifrangeF_body6_rel*
relationalize *ifrangeF_body6_rel is_ifrangeF_body6*

synthesize *is_ifrangeF_body6* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrangeF_body6_fm*

lemma (**in** *M_ZF_trans*) *separation_is_ifrangeF_body6*:
 $(\#\#M)(A) \Longrightarrow (\#\#M)(G) \Longrightarrow (\#\#M)(r) \Longrightarrow (\#\#M)(s) \Longrightarrow$ *separation*$(\#\#M,$
is_ifrangeF_body6$(\#\#M,A,G,r,s))$
 apply(*rule_tac separation_cong*[
 where $P=\lambda\ x\ .\ M,[x,A,G,r,s] \models$ *is_ifrangeF_body6_fm*$(1,2,3,4,0)$,*THEN*
iffD1])
 apply(*rule_tac is_ifrangeF_body6_iff_sats*[**where** *env*=[_,*A*,*G*,*r*,*s*],*symmetric*])
 apply(*simp_all add*:*zero_in_M*)
 apply(*rule_tac separation_ax*[**where** *env*=[*A*,*G*,*r*,*s*],*simplified*])
 apply(*simp_all add*:*arity_is_ifrangeF_body6_fm ord_simp_union is_ifrangeF_body6_fm_type*)
 done

lemma (**in** *M_basic*) *ifrFb_body6_closed*: $M(G) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow$ *ifrFb_body6*$(G,$
$r,\ s,\ x,\ i) \longleftrightarrow\ M(i) \wedge$ *ifrFb_body6*$(G,\ r,\ s,\ x,\ i)$
 using *If_abs*
 unfolding *ifrangeF_body6_def is_ifrangeF_body6_def ifrFb_body6_def fun_apply_def*
 by (*cases* $i\in$*range*(s); *cases* $r=0$; *auto dest*:*transM*)

lemma (**in** *M_basic*) *is_ifrFb_body6_closed*: $M(G) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow$
is_ifrFb_body6$(M,\ G,\ r,\ s,\ x,\ i) \Longrightarrow M(i)$
 using *If_abs*
 unfolding *ifrangeF_body6_def is_ifrangeF_body6_def is_ifrFb_body6_def fun_apply_def*
 by (*cases* $i\in$*range*(s); *cases* $r=0$; *auto dest*:*transM*)

lemmas (**in** *M_ZF_trans*) a = *separation_toplevel6_body*
 separation_cong[*OF eq_commute*,*THEN iffD1*,*OF separation_toplevel6_body*]

lemma (**in** *M_ZF_trans*) *ifrangeF_body6_abs*:
 assumes $(\#\#M)(A)\ (\#\#M)(G)\ (\#\#M)(r)\ (\#\#M)(s)\ (\#\#M)(x)$
 shows *is_ifrangeF_body6*$(\#\#M,A,G,r,s,x) \longleftrightarrow$ *ifrangeF_body6*$(\#\#M,A,G,r,s,x)$
proof -
 {
 fix a
 assume $a\in M$
 with *assms*
 have $(\mu\ i.\ i\in M \wedge$ *is_ifrFb_body6*$(\#\#M,\ G,\ r,\ s,\ z,\ i)) = (\mu\ i.$ *is_ifrFb_body6*$(\#\#M,$
$G,\ r,\ s,\ z,\ i))$ **for** z
 using *is_ifrFb_body6_closed*[*of G r s z*]
 by (*rule_tac Least_cong*[*of* $\lambda i.\ i\in M\ \wedge$ *is_ifrFb_body6*$(\#\#M,G,r,s,z,i)$])
auto
 moreover
 have $(\mu\ i.\ i\in M\ \wedge$ *is_ifrFb_body6*$(\#\#M,\ G,\ r,\ s,\ z,\ i)) = (\mu\ i.\ i\in M\ \wedge$
ifrFb_body6$(G,\ r,\ s,\ z,\ i))$ **if** $z\in M$ **for** z
 proof (*rule_tac Least_cong*[*of* $\lambda i.\ i\in M \wedge$ *is_ifrFb_body6*$(\#\#M,G,r,s,z,i)$ $\lambda i.$
$i\in M \wedge$ *ifrFb_body6*(G,r,s,z,i)])
 fix y
 from *assms* ‹$a\in M$› ‹$z\in M$›
 show $y\in M \wedge$ *is_ifrFb_body6*$(\#\#M,\ G,\ r,\ s,\ z,\ y) \longleftrightarrow y\in M \wedge$ *ifrFb_body6*$(G,$
$r,\ s,\ z,\ y)$

using *If_abs apply_0 separation_in transitivity*[*of* _ *G*]
separation_closed converse_closed apply_closed range_closed zero_in_M
separation_cong[*OF eq_commute*,*THEN iffD1*,*OF separation_toplevel6_body*]
unfolding *ifrFb_body6_def is_ifrFb_body6_def*
by *auto*
qed
moreover from ‹*a*∈*M*›
have *least*(##*M*, λ*i*. *i* ∈ *M* ∧ *is_ifrFb_body6*(##*M*, *G*, *r*, *s*, *z*, *i*), *a*)
⟷ *a* = (μ *i*. *i*∈ *M* ∧ *is_ifrFb_body6*(##*M*, *G*, *r*, *s*, *z*,*i*)) **for** *z*
using *If_abs least_abs'*[*of* λ*i*. (##*M*)(*i*) ∧ *is_ifrFb_body6*(##*M*,*G*,*r*,*s*,*z*,*i*)
a]
by *simp*
ultimately
have *z*∈*M* ⟹ *least*(##*M*, λ*i*. *i* ∈ *M* ∧ *is_ifrFb_body6*(##*M*, *G*, *r*, *s*, *z*, *i*),
a)
⟷ *a* = (μ *i*. *ifrFb_body6*(*G*, *r*, *s*, *z*,*i*)) **for** *z*
using *Least_cong*[*OF ifrFb_body6_closed*[*of G r s*]] *assms*
by *simp*
}
with *assms*
show *?thesis*
using *pair_in_M_iff apply_closed zero_in_M transitivity*[*of* _ *A*]
unfolding *ifrangeF_body6_def is_ifrangeF_body6_def*
by (*auto dest*:*transM*)
qed

lemma (**in** *M_ZF_trans*) *separation_ifrangeF_body6*:
(##*M*)(*A*) ⟹ (##*M*)(*G*) ⟹ (##*M*)(*b*) ⟹ (##*M*)(*f*) ⟹
separation(##*M*,
λ*y*. ∃ *x*∈*A*. *y* = ⟨*x*, μ *i*. *x* ∈ *if_range_F_else_F*(λ*a*. {*p* ∈ *G* . *domain*(*p*) =
a}, *b*, *f*, *i*)⟩)
using *separation_is_ifrangeF_body6 ifrangeF_body6_abs*
separation_cong[**where** *P*=*is_ifrangeF_body6*(##*M*,*A*,*G*,*b*,*f*) **and** *M*=##*M*,*THEN*
iffD1]
unfolding *ifrangeF_body6_def if_range_F_def if_range_F_else_F_def ifrFb_body6_def*
by *simp*

definition *ifrFb_body7* **where**
ifrFb_body7(*B*,*D*,*A*,*b*,*f*,*x*,*i*) ≡ *x* ∈
(*if b* = *0 then if i* ∈ *range*(*f*) *then*
{*d* ∈ *D* . ∃ *r*∈*A*. *restrict*(*r*, *B*) = *converse*(*f*) ‘ *i* ∧ *d* = *domain*(*r*)} *else 0*
else {*d* ∈ *D* . ∃ *r*∈*A*. *restrict*(*r*, *B*) = *i* ∧ *d* = *domain*(*r*)})

relativize functional *ifrFb_body7 ifrFb_body7_rel*
relationalize *ifrFb_body7_rel is_ifrFb_body7*

synthesize *is_ifrFb_body7* **from_definition assuming** *nonempty*

arity_theorem for *is_ifrFb_body7_fm*

definition *ifrangeF_body7* :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
ifrangeF_body7(*M*,*A*,*B*,*D*,*G*,*b*,*f*) ≡ λ*y*. ∃ *x*∈*A*. *y* = ⟨*x*,μ *i*. *ifrFb_body7*(*B*,*D*,*G*,*b*,*f*,*x*,*i*)⟩

relativize functional *ifrangeF_body7* *ifrangeF_body7_rel*
relationalize *ifrangeF_body7_rel* *is_ifrangeF_body7*

synthesize *is_ifrangeF_body7* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrangeF_body7_fm*

lemma (**in** *M_ZF_trans*) *separation_is_ifrangeF_body7*:
(##*M*)(*A*) ⟹ (##*M*)(*B*) ⟹ (##*M*)(*D*) ⟹ (##*M*)(*G*) ⟹ (##*M*)(*r*)
⟹ (##*M*)(*s*) ⟹ *separation*(##*M*, *is_ifrangeF_body7*(##*M*,*A*,*B*,*D*,*G*,*r*,*s*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*,*A*,*B*,*D*,*G*,*r*,*s*] ⊨ *is_ifrangeF_body7_fm*(*1*,*2*,*3*,*4*,*5*,*6*,*0*),*THEN*
iffD1])
apply(*rule_tac is_ifrangeF_body7_iff_sats*[**where** *env*=[_,*A*,*B*,*D*,*G*,*r*,*s*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*B*,*D*,*G*,*r*,*s*],*simplified*])
apply(*simp_all add*:*arity_is_ifrangeF_body7_fm ord_simp_union is_ifrangeF_body7_fm_type*)
done

lemma (**in** *M_basic*) *ifrFb_body7_closed*: *M*(*B*) ⟹ *M*(*D*) ⟹ *M*(*G*) ⟹ *M*(*r*)
⟹ *M*(*s*) ⟹
ifrFb_body7(*B*,*D*,*G*, *r*, *s*, *x*, *i*) ⟷ *M*(*i*) ∧ *ifrFb_body7*(*B*,*D*,*G*, *r*, *s*, *x*, *i*)
using *If_abs*
unfolding *ifrangeF_body7_def is_ifrangeF_body7_def ifrFb_body7_def fun_apply_def*
by (*cases i*∈*range*(*s*); *cases r=0*; *auto dest*:*transM*)

lemma (**in** *M_basic*) *is_ifrFb_body7_closed*: *M*(*B*) ⟹ *M*(*D*) ⟹ *M*(*G*) ⟹
M(*r*) ⟹ *M*(*s*) ⟹
is_ifrFb_body7(*M*, *B*,*D*,*G*, *r*, *s*, *x*, *i*) ⟹ *M*(*i*)
using *If_abs*
unfolding *ifrangeF_body7_def is_ifrangeF_body7_def is_ifrFb_body7_def fun_apply_def*
by (*cases i*∈*range*(*s*); *cases r=0*; *auto dest*:*transM*)

lemma (**in** *M_ZF_trans*) *ifrangeF_body7_abs*:
assumes (##*M*)(*A*) (##*M*)(*B*) (##*M*)(*D*) (##*M*)(*G*) (##*M*)(*r*) (##*M*)(*s*)
(##*M*)(*x*)
shows *is_ifrangeF_body7*(##*M*,*A*,*B*,*D*,*G*,*r*,*s*,*x*) ⟷ *ifrangeF_body7*(##*M*,*A*,*B*,*D*,*G*,*r*,*s*,*x*)
proof -
from *assms*
have *sep_dr*: *y*∈*M* ⟹ *separation*(##*M*, λ*d* . ∃ *r*∈*M* . *r*∈*G*∧ *y* = *restrict*(*r*,
B) ∧ *d* = *domain*(*r*)) **for** *y*
by(*rule_tac separation_cong*[**where** *P'*=λ*d* . ∃ *r*∈ *M* . *r*∈*G* ∧ *y* = *restrict*(*r*,
B) ∧ *d* = *domain*(*r*),*THEN iffD1*,*OF* _
separation_restrict_eq_dom_eq[*rule_format*,*of G B y*]],*auto simp*:*transitivity*[*of*
_ *G*])

from *assms*
have *sep_dr''*: $y \in M \Longrightarrow$ *separation*(##*M*, λd . $\exists\, r \in M$. $r \in G \wedge d =$ *domain*(r) $\wedge$ *converse*(s) ‘ $y =$ *restrict*(r, B)) **for** y
apply(*rule_tac separation_cong*[**where** $P' = \lambda d$. $\exists\, r \in M$. $r \in G \wedge d =$ *domain*(r) $\wedge$ *converse*(s) ‘ $y =$ *restrict*(r, B),*THEN iffD1*,*OF _ separation_restrict_eq_dom_eq*[*rule_format*,*of G B converse*(s) ‘ y]])
by(*auto simp*:*transitivity*[*of _ G*] *apply_closed*[*simplified*] *converse_closed*[*simplified*])
from *assms*
have *sep_dr'*:*separation*(##*M*, λx. $\exists\, r \in M$. $r \in G \wedge x =$ *domain*(r) $\wedge$ $0 =$ *restrict*(r, B))
apply(*rule_tac separation_cong*[**where** $P' = \lambda d$. $\exists\, r \in M$. $r \in G \wedge d =$ *domain*(r) $\wedge$ $0 =$ *restrict*(r, B),*THEN iffD1*,*OF _ separation_restrict_eq_dom_eq*[*rule_format*,*of G B* 0]])
by(*auto simp*:*transitivity*[*of _ G*] *zero_in_M*)
{
fix a
assume $a \in M$
with *assms*
have (μ i. $i \in M \wedge$ *is_ifrFb_body7*(##*M*, B,D,G, r, s, z, i))= (μ i. *is_ifrFb_body7*(##*M*,B,D, G, r, s, z, i)) **for** z
using *is_ifrFb_body7_closed*[*of B D G r s z*]
by (*rule_tac Least_cong*[*of* λi. $i \in M \wedge$ *is_ifrFb_body7*(##*M*,B,D,G,r,s,z,i)]) *auto*
moreover from *this*
have (μ i. $i \in M \wedge$ *is_ifrFb_body7*(##*M*, B,D,G, r, s, z, i))= (μ i. $i \in M \wedge$ *ifrFb_body7*(B,D,G, r, s, z, i)) **if** $z \in M$ **for** z
proof (*rule_tac Least_cong*[*of* λi. $i \in M \wedge$ *is_ifrFb_body7*(##*M*,B,D,G,r,s,z,i) λi. $i \in M \wedge$ *ifrFb_body7*(B,D,G,r,s,z,i)])
from *assms* ‹$a \in M$› ‹$z \in M$›
have *is_ifrFb_body7*(##*M*, B,D,G, r, s, z, y) $\longleftrightarrow$ *ifrFb_body7*(B,D,G, r, s, z, y) **if** $y \in M$ **for** y
using *If_abs apply_0*
separation_closed converse_closed apply_closed range_closed zero_in_M
separation_restrict_eq_dom_eq[*of G*,*rule_format*]
transitivity[*of _ D*] *transitivity*[*of _ G*] *that sep_dr sep_dr' sep_dr''*
unfolding *ifrFb_body7_def is_ifrFb_body7_def*
by *auto*
then
show $y \in M \wedge$ *is_ifrFb_body7*(##*M*, B, D, G, r, s, z, y) $\longleftrightarrow$ $y \in M \wedge$ *ifrFb_body7*(B, D, G, r, s, z, y) **for** y
using *conj_cong*
by *simp*
qed
moreover from ‹$a \in M$›
have *least*(##*M*, λi. $i \in M \wedge$ *is_ifrFb_body7*(##*M*, B,D,G, r, s, z, i), a) $\longleftrightarrow$ a = (μ i. $i \in M \wedge$ *is_ifrFb_body7*(##*M*,B,D,G, r, s, z,i)) **for** z
using *If_abs least_abs'*[*of* λi. (##*M*)(i) $\wedge$ *is_ifrFb_body7*(##*M*,B,D,G,r,s,z,i) a]

```
      by simp
    ultimately
    have z∈M ⟹ least(##M, λi. i ∈ M ∧ is_ifrFb_body7(##M,B,D,G, r, s,
z, i), a)
      ⟷ a = (μ i. ifrFb_body7(B,D,G, r, s, z,i)) for z
      using Least_cong[OF ifrFb_body7_closed[of B D G r s]] assms
      by simp
  }
  with assms
  show ?thesis
    using pair_in_M_iff apply_closed zero_in_M transitivity[of _ A]
    unfolding ifrangeF_body7_def is_ifrangeF_body7_def
    by (auto dest:transM)
qed

lemma (in M_ZF_trans) separation_ifrangeF_body7:
  (##M)(A) ⟹ (##M)(B) ⟹ (##M)(D) ⟹ (##M)(G) ⟹ (##M)(b) ⟹
(##M)(f) ⟹
    separation(##M,
      λy. ∃x∈A. y = ⟨x, μ i. x ∈ if_range_F_else_F(drSR_Y(B, D, G), b, f, i)⟩)
  using separation_is_ifrangeF_body7 ifrangeF_body7_abs drSR_Y_equality
   separation_cong[where P=is_ifrangeF_body7(##M,A,B,D,G,b,f) and M=##M,THEN
iffD1]
  unfolding ifrangeF_body7_def if_range_F_def if_range_F_else_F_def ifrFb_body7_def
  by simp

end
theory Replacement_Instances
  imports
    Discipline_Function
    Forcing_Data
    Aleph_Relative
    FiniteFun_Relative
    Cardinal_Relative
    Separation_Instances
begin
```

28.2 More Instances of Replacement

This is the same way that we used for instances of separation.

```
lemma (in M_ZF_trans) replacement_is_range:
 strong_replacement(##M, λf y. is_range(##M,f,y))
  apply(rule_tac strong_replacement_cong[
        where P=λ x f. M,[x,f] ⊨ range_fm(0,1),THEN iffD1])
   apply(rule_tac range_iff_sats[where env=[_,_],symmetric])
  apply(simp_all)
  apply(rule_tac replacement_ax[where env=[],simplified])
    apply(simp_all add:arity_range_fm ord_simp_union range_type)
  done
```

lemma (**in** *M_ZF_trans*) *replacement_range*:
strong_replacement(##*M*, λ*f y*. *y* = *range*(*f*))
using *strong_replacement_cong*[*THEN iffD2*,*OF _ replacement_is_range*] *range_abs*
by *simp*

lemma (**in** *M_ZF_trans*) *replacement_is_domain*:
strong_replacement(##*M*, λ*f y*. *is_domain*(##*M*,*f*,*y*))
apply(*rule_tac strong_replacement_cong*[
where *P*=λ *x f*. *M*,[*x*,*f*] ⊨ *domain_fm*(*0*,*1*),*THEN iffD1*])
apply(*rule_tac domain_iff_sats*[**where** *env*=[_,_],*symmetric*])
apply(*simp_all*)
apply(*rule_tac replacement_ax*[**where** *env*=[],*simplified*])
apply(*simp_all add*:*arity_domain_fm ord_simp_union domain_type*)
done

lemma (**in** *M_ZF_trans*) *replacement_domain*:
strong_replacement(##*M*, λ*f y*. *y* = *domain*(*f*))
using *strong_replacement_cong*[*THEN iffD2*,*OF _ replacement_is_domain*]
by *simp*

Alternatively, we can use closure under lambda and get the stronger version.

lemma (**in** *M_ZF_trans*) *lam_replacement_domain* : *lam_replacement*(##*M*, *domain*)
using *lam_replacement_iff_lam_closed*[*THEN iffD2*,*of domain*]
Lambda_in_M[**where** φ=*domain_fm*(*0*,*1*) **and** *env*=[],
OF domain_type _ domain_iff_sats[*symmetric*] *domain_abs*,*simplified*]
arity_domain_fm[*of 0 1*] *ord_simp_union transitivity domain_closed*
by *simp*

Then we recover the original version. Notice that we need closure because we haven't yet interpreted *M_replacement*.

lemma (**in** *M_ZF_trans*) *replacement_domain'*:
strong_replacement(##*M*, λ*f y*. *y* = *domain*(*f*))
using *lam_replacement_imp_strong_replacement_aux lam_replacement_domain domain_closed*
by *simp*

lemma (**in** *M_ZF_trans*) *lam_replacement_fst* : *lam_replacement*(##*M*, *fst*)
using *lam_replacement_iff_lam_closed*[*THEN iffD2*,*of fst*]
Lambda_in_M[**where** φ=*fst_fm*(*0*,*1*) **and** *env*=[],*OF*
_ _ fst_iff_sats[*symmetric*] *fst_abs*] *fst_type*
arity_fst_fm[*of 0 1*] *ord_simp_union transitivity fst_closed*
by *simp*

lemma (**in** *M_ZF_trans*) *replacement_fst'*:
strong_replacement(##*M*, λ*f y*. *y* = *fst*(*f*))

using *lam_replacement_imp_strong_replacement_aux lam_replacement_fst fst_closed*
by *simp*

lemma (**in** *M_ZF_trans*) *lam_replacement_domain1* : *lam_replacement*(*##M*, *domain*)
using *lam_replacement_iff_lam_closed*[*THEN iffD2*,*of domain*]
Lambda_in_M[**where** *φ*=*domain_fm*(*0*,*1*) **and** *env*=[],*OF*
_ _ domain_iff_sats[*symmetric*] *domain_abs*] *domain_type*
arity_domain_fm[*of 0 1*] *ord_simp_union transitivity domain_closed*
by *simp*

lemma (**in** *M_ZF_trans*) *lam_replacement_snd* : *lam_replacement*(*##M*, *snd*)
using *lam_replacement_iff_lam_closed*[*THEN iffD2*,*of snd*]
Lambda_in_M[**where** *φ*=*snd_fm*(*0*,*1*) **and** *env*=[],*OF*
_ _ snd_iff_sats[*symmetric*] *snd_abs*] *snd_type*
arity_snd_fm[*of 0 1*] *ord_simp_union transitivity snd_closed*
by *simp*

lemma (**in** *M_ZF_trans*) *replacement_snd′*:
strong_replacement(*##M*, *λf y*. *y* = *snd*(*f*))
using *lam_replacement_imp_strong_replacement_aux lam_replacement_snd snd_closed*
by *simp*

lemma (**in** *M_ZF_trans*) *lam_replacement_Union* : *lam_replacement*(*##M*, *Union*)
using *lam_replacement_iff_lam_closed*[*THEN iffD2*,*of Union*]
Lambda_in_M[**where** *φ*=*big_union_fm*(*0*,*1*) **and** *env*=[],*OF*
_ _ _ Union_abs] *union_fm_def big_union_iff_sats*[*symmetric*]
arity_big_union_fm[*of 0 1*] *ord_simp_union transitivity Union_closed*
by *simp*

lemma (**in** *M_ZF_trans*) *replacement_Union′*:
strong_replacement(*##M*, *λf y*. *y* = *Union*(*f*))
using *lam_replacement_imp_strong_replacement_aux lam_replacement_Union Union_closed*
by *simp*

lemma (**in** *M_ZF_trans*) *lam_replacement_Un*:
lam_replacement(*##M*, *λp*. *fst*(*p*) ∪ *snd*(*p*))
using *lam_replacement_iff_lam_closed*[*THEN iffD2*,*of λp*. *fst*(*p*)∪*snd*(*p*)]
LambdaPair_in_M[**where** *φ*=*union_fm*(*0*,*1*,*2*) **and** *is_f*=*union*(*##M*) **and** *env*=[],*OF*
union_type _ union_iff_sats[*symmetric*] *union_abs*]
arity_union_fm[*of 0 1 2*] *ord_simp_union transitivity Un_closed fst_snd_closed*
by *simp*

lemma (**in** *M_ZF_trans*) *lam_replacement_image*:
lam_replacement(*##M*, *λp*. *fst*(*p*) `` *snd*(*p*))

using *lam_replacement_iff_lam_closed*[*THEN iffD2*,*of* $\lambda p.$ *fst*(p)‘‘*snd*(p)]
LambdaPair_in_M[**where** φ=*image_fm*(*0*,*1*,*2*) **and** *is_f*=*image*(*##M*) **and** *env*=[],*OF*
image_type _ *image_iff_sats*[*symmetric*] *image_abs*]
arity_image_fm[*of 0 1 2*] *ord_simp_union transitivity image_closed fst_snd_closed*
by *simp*

synthesize *setdiff* **from_definition** *setdiff* **assuming** *nonempty*
arity_theorem for *setdiff_fm*

lemma (**in** *M_ZF_trans*) *lam_replacement_Diff*:
lam_replacement(*##M*, $\lambda p.$ *fst*(p) - *snd*(p))
using *lam_replacement_iff_lam_closed*[*THEN iffD2*,*of* $\lambda p.$ *fst*(p) - *snd*(p)]
LambdaPair_in_M[**where** φ=*setdiff_fm*(*0*,*1*,*2*) **and** *is_f*=*setdiff*(*##M*) **and** *env*=[],
OF setdiff_fm_type _ *setdiff_iff_sats*[*symmetric*] *setdiff_abs*]
arity_setdiff_fm[*of 0 1 2*] *ord_simp_union transitivity Diff_closed fst_snd_closed*
nonempty
by *simp*

relationalize *first_rel is_first* **external**
synthesize *first_fm* **from_definition** *is_first* **assuming** *nonempty*

lemma (**in** *M_ZF_trans*) *minimum_closed*:
assumes $B{\in}M$
shows *minimum*(r,B) $\in M$
proof(*cases* $\exists!b.$ *first*(b,B,r))
case *True*
then
obtain b **where** b = *minimum*(r,B) *first*(b,B,r)
using *the_equality2*
unfolding *minimum_def*
by *auto*
then
show *?thesis*
using *first_is_elem transitivity*[*of b B*] *assms*
by *simp*
next
case *False*
then show *?thesis*
using *zero_in_M the_0*
unfolding *minimum_def*
by *auto*
qed

relationalize *minimum_rel is_minimum* **external**
definition *is_minimum′* **where**
is_minimum′(M,R,X,u) $\equiv$ ($M(u) \wedge u \in X \wedge (\forall\, v[M].\ \exists\, a[M].\ (v \in X \longrightarrow v \neq u \longrightarrow a \in R) \wedge$ *pair*(M, u, v, a))) $\wedge$

$(\exists\, x[M].$
$(M(x) \wedge x \in X \wedge (\forall\, v[M].\ \exists\, a[M].\ (v \in X \longrightarrow v \neq x \longrightarrow a \in R) \wedge pair(M,$
$x,\ v,\ a))) \wedge$
$(\forall\, y[M].\ M(y) \wedge y \in X \wedge (\forall\, v[M].\ \exists\, a[M].\ (v \in X \longrightarrow v \neq y \longrightarrow a \in R) \wedge$
$pair(M,\ y,\ v,\ a)) \longrightarrow y = x)) \vee$
$\neg\ (\exists\, x[M].\ (M(x) \wedge x \in X \wedge (\forall\, v[M].\ \exists\, a[M].\ (v \in X \longrightarrow v \neq x \longrightarrow a \in R) \wedge$
$pair(M,\ x,\ v,\ a))) \wedge$
$(\forall\, y[M].\ M(y) \wedge y \in X \wedge (\forall\, v[M].\ \exists\, a[M].\ (v \in X \longrightarrow v \neq y \longrightarrow a \in$
$R) \wedge pair(M,\ y,\ v,\ a)) \longrightarrow y = x)) \wedge$
$empty(M,\ u)$

synthesize *minimum* **from_definition** *is_minimum′* **assuming** *nonempty*
arity_theorem for *minimum_fm*

lemma *is_minimum_eq* :
$M(R) \Longrightarrow M(X) \Longrightarrow M(u) \Longrightarrow is\_minimum(M,R,X,u) \longleftrightarrow is\_minimum'(M,R,X,u)$
unfolding *is_minimum_def is_minimum′_def is_The_def is_first_def* **by** *simp*

context *M_trivial*
begin

lemma *first_closed*:
$M(B) \Longrightarrow M(r) \Longrightarrow first(u,r,B) \Longrightarrow M(u)$
using *transM*[*OF first_is_elem*] **by** *simp*

is_iff_rel for *first*
unfolding *is_first_def first_rel_def* **by** *auto*

is_iff_rel for *minimum*
unfolding *is_minimum_def minimum_rel_def*
using *is_first_iff The_abs nonempty*
by *force*

end

lemma (**in** *M_ZF_trans*) *lam_replacement_minimum*:
lam_replacement(*##M*, *λp. minimum*(*fst*(*p*), *snd*(*p*)))
apply(*rule_tac lam_replacement_iff_lam_closed*[*THEN iffD2,of λp. minimum*(*fst*(*p*),*snd*(*p*))])
apply (*auto*) **apply**(*rule minimum_closed*[*simplified*],*auto simp add*:*fst_snd_closed*[*simplified*])
apply (*rule_tac*
LambdaPair_in_M[**where** *φ*=*minimum_fm*(*0*,*1*,*2*) **and** *is_f*=*is_minimum′*(*##M*)
and *env*=[],*OF*
minimum_fm_type _ *minimum_iff_sats*[*symmetric*]])
apply (*auto simp*: *arity_minimum_fm*[*of 0 1 2*] *ord_simp_union transitivity*
fst_snd_closed zero_in_M)
using *Upair_closed*[*simplified*] *minimum_closed is_minimum_eq is_minimum_iff*
minimum_abs
by *simp_all*

lemma (**in** *M_ZF_trans*) *lam_replacement_Upair*:
lam_replacement(*##M*, *λp. Upair*(*fst*(*p*), *snd*(*p*)))
apply(*rule_tac lam_replacement_iff_lam_closed*[*THEN iffD2*,*of λp. Upair*(*fst*(*p*),*snd*(*p*))])
apply (*auto*) **apply**(*rule Upair_closed*[*simplified*],*auto simp add*:*fst_snd_closed*[*simplified*])
apply (*rule_tac*
LambdaPair_in_M[**where** *φ*=*upair_fm*(*0*,*1*,*2*) **and** *is_f*=*upair*(*##M*) **and** *env*=[],*OF*
upair_type _ *upair_iff_sats*[*symmetric*]])
apply (*auto simp*: *arity_upair_fm*[*of 0 1 2*] *ord_simp_union transitivity fst_snd_closed*)
using *Upair_closed*[*simplified*]
by *simp*

lemma (**in** *M_ZF_trans*) *lam_replacement_cartprod*:
lam_replacement(*##M*, *λp. fst*(*p*) × *snd*(*p*))
apply(*rule_tac lam_replacement_iff_lam_closed*[*THEN iffD2*,*of λp. fst*(*p*)×*snd*(*p*)])
apply (*auto*) **apply**(*rule cartprod_closed*[*simplified*],*auto simp add*:*fst_snd_closed*[*simplified*])
apply (*rule_tac*
LambdaPair_in_M[**where** *φ*=*cartprod_fm*(*0*,*1*,*2*) **and** *is_f*=*cartprod*(*##M*) **and** *env*=[],*OF*
cartprod_type _ *cartprod_iff_sats*[*symmetric*]])
apply (*auto simp*: *arity_cartprod_fm*[*of 0 1 2*] *ord_simp_union transitivity fst_snd_closed*)
using *cartprod_closed*[*simplified*]
by *simp*

synthesize *pre_image* **from_definition assuming** *nonempty*
arity_theorem for *pre_image_fm*

lemma (**in** *M_ZF_trans*) *lam_replacement_vimage*:
lam_replacement(*##M*, *λp. fst*(*p*) *-`` snd*(*p*))
apply(*rule_tac lam_replacement_iff_lam_closed*[*THEN iffD2*,*of λp. fst*(*p*)*-``snd*(*p*)])
apply (*auto*) **apply**(*rule vimage_closed*[*simplified*],*auto simp add*:*fst_snd_closed*[*simplified*])
apply (*rule_tac*
LambdaPair_in_M[**where** *φ*=*pre_image_fm*(*0*,*1*,*2*) **and** *is_f*=*pre_image*(*##M*) **and** *env*=[],*OF*
pre_image_fm_type _ *pre_image_iff_sats*[*symmetric*]])
apply (*auto simp*: *arity_pre_image_fm*[*of 0 1 2*] *ord_simp_union transitivity zero_in_M*)
using *vimage_closed*[*simplified*]
by *simp*

definition *is_omega_funspace* :: [*i⇒o*,*i*,*i*,*i*]⇒*o* **where**
is_omega_funspace(*N*,*B*,*n*,*z*) ≡ ∃ *o*[*N*]. *omega*(*N*,*o*) ∧ *n*∈*o* ∧*is_funspace*(*N*, *n*, *B*, *z*)

synthesize *omega_funspace* **from_definition** *is_omega_funspace* **assuming** *nonempty*

arity_theorem for *omega_funspace_fm*

lemma (**in** *M_ZF_trans*) *omega_funspace_abs*:
B∈M ⟹ n∈M ⟹ z∈M ⟹ is_omega_funspace(##M,B,n,z) ⟷ n∈ω ∧ is_funspace(##M,n,B,z)
unfolding *is_omega_funspace_def* **using** *nat_in_M* **by** *simp*

lemma (**in** *M_ZF_trans*) *replacement_is_omega_funspace*:
B∈M ⟹ strong_replacement(##M, is_omega_funspace(##M,B))
apply(*rule_tac strong_replacement_cong*[
where *P=λ x f. M,[x,f,B] ⊨ omega_funspace_fm(2,0,1),THEN iffD1*])
apply(*rule_tac omega_funspace_iff_sats*[**where** *env=*[_,_,*B*],*symmetric*])
apply(*simp_all add:zero_in_M*)
apply(*rule_tac replacement_ax*[**where** *env=*[*B*],*simplified*])
apply(*simp_all add:arity_omega_funspace_fm ord_simp_union*)
done

lemma (**in** *M_ZF_trans*) *replacement_omega_funspace*:
b∈M⟹strong_replacement(##M, λn z. n∈ω ∧ is_funspace(##M,n,b,z))
using *strong_replacement_cong*[*THEN iffD2*,*OF _ replacement_is_omega_funspace*[*of b*]]
omega_funspace_abs[*of b*] *setclass_iff*[*THEN iffD1*]
by (*simp del:setclass_iff*)

definition *HAleph_wfrec_repl_body* **where**
HAleph_wfrec_repl_body(N,mesa,x,z) ≡ ∃ y[N].
pair(N, x, y, z) ∧
(∃ f[N].
(∀ z[N].
z ∈ f ⟷
(∃ xa[N].
∃ y[N].
∃ xaa[N].
∃ sx[N].
∃ r_sx[N].
∃ f_r_sx[N].
pair(N, xa, y, z) ∧
pair(N, xa, x, xaa) ∧
upair(N, xa, xa, sx) ∧
pre_image(N, mesa, sx, r_sx) ∧ restriction(N, f, r_sx, f_r_sx) ∧ xaa ∈ mesa ∧ is_HAleph(N, xa, f_r_sx, y))) ∧
is_HAleph(N, x, f, y))

arity_theorem for *ordinal_fm*
arity_theorem for *is_Limit_fm*
arity_theorem for *empty_fm*
arity_theorem for *fun_apply_fm*

synthesize *HAleph_wfrec_repl_body* **from_definition assuming** *nonempty*
arity_theorem for *HAleph_wfrec_repl_body_fm*

— FIXME: Why $[\![?p \in formula;\ ?v \in \omega;\ ?n \in \omega;\ ?i \in \omega;\ arity(?p) = ?i]\!] \Longrightarrow arity(Replace\_fm(?v, ?p, ?n)) = succ(?n) \cup (succ(?v) \cup Arith.pred(Arith.pred(?i)))$ doesn't work here? Revise the method we're using.
lemma *arity_HAleph_wfrec_repl_body*: *arity*(*HAleph_wfrec_repl_body_fm*(*2*,*0*,*1*))
= *3*
by (*simp_all add*: *arity_HAleph_wfrec_repl_body_fm arity_is_If_fm ord_simp_union arity_fun_apply_fm*
arity_is_Limit_fm arity_empty_fm arity_Replace_fm[**where** *i=11*])

lemma (**in** *M_ZF_trans*) *replacement_HAleph_wfrec_repl_body*:
B∈*M* ⟹ *strong_replacement*(##*M*, *HAleph_wfrec_repl_body*(##*M*,*B*))
apply(*rule_tac strong_replacement_cong*[
where *P*=λ *x f*. *M*,[*x*,*f*,*B*] ⊨ *HAleph_wfrec_repl_body_fm*(*2*,*0*,*1*),*THEN iffD1*])
apply(*rule_tac HAleph_wfrec_repl_body_iff_sats*[**where** *env*=[_,_,*B*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac replacement_ax*[**where** *env*=[*B*],*simplified*])
apply(*simp_all add*: *arity_HAleph_wfrec_repl_body*)
done

lemma (**in** *M_ZF_trans*) *HAleph_wfrec_repl*:
(##*M*)(*sa*) ⟹
(##*M*)(*esa*) ⟹
(##*M*)(*mesa*) ⟹
strong_replacement
(##*M*,
λ*x z*. ∃ *y*[##*M*].
pair(##*M*, *x*, *y*, *z*) ∧
(∃ *f*[##*M*].
(∀ *z*[##*M*].
z ∈ *f* ⟷
(∃ *xa*[##*M*].
∃ *y*[##*M*].
∃ *xaa*[##*M*].
∃ *sx*[##*M*].
∃ *r_sx*[##*M*].
∃ *f_r_sx*[##*M*].
pair(##*M*, *xa*, *y*, *z*) ∧
pair(##*M*, *xa*, *x*, *xaa*) ∧
upair(##*M*, *xa*, *xa*, *sx*) ∧
pre_image(##*M*, *mesa*, *sx*, *r_sx*) ∧
restriction(##*M*, *f*, *r_sx*, *f_r_sx*) ∧ *xaa* ∈ *mesa* ∧ *is_HAleph*(##*M*, *xa*, *f_r_sx*, *y*))) ∧
is_HAleph(##*M*, *x*, *f*, *y*)))
using *replacement_HAleph_wfrec_repl_body* **unfolding** *HAleph_wfrec_repl_body_def*
by *simp*

definition *fst2_snd2*
  where *fst2_snd2*(x) ≡ ⟨*fst*(*fst*(x)), *snd*(*snd*(x))⟩

relativize functional *fst2_snd2 fst2_snd2_rel*
relationalize *fst2_snd2_rel is_fst2_snd2*

lemma (**in** *M_trivial*) *fst2_snd2_abs*:
  assumes *M*(x) *M*(*res*)
shows *is_fst2_snd2*(M, x, *res*) ⟷ *res* = *fst2_snd2*(x)
  unfolding *is_fst2_snd2_def fst2_snd2_def*
  using *fst_rel_abs*[*symmetric*] *snd_rel_abs*[*symmetric*] *fst_abs snd_abs assms*
  by *simp*

synthesize *is_fst2_snd2* **from_definition assuming** *nonempty*
arity_theorem for *is_fst2_snd2_fm*

lemma (**in** *M_ZF_trans*) *replacement_is_fst2_snd2*:
 strong_replacement(##*M*, *is_fst2_snd2*(##*M*))
  apply(*rule_tac strong_replacement_cong*[
        where *P*=λ x f. *M*,[x,f] ⊨ *is_fst2_snd2_fm*(*0*,*1*),*THEN iffD1*])
   apply(*rule_tac is_fst2_snd2_iff_sats*[**where** *env*=[_,_],*symmetric*])
  apply(*simp_all add*:*zero_in_M*)
  apply(*rule_tac replacement_ax*[**where** *env*=[],*simplified*])
    apply(*simp_all add*: *arity_is_fst2_snd2_fm ord_simp_union*)
  done

lemma (**in** *M_ZF_trans*) *replacement_fst2_snd2*: *strong_replacement*(##*M*, λx y. y = ⟨*fst*(*fst*(x)), *snd*(*snd*(x))⟩)
 using *strong_replacement_cong*[*THEN iffD1*,*OF fst2_snd2_abs replacement_is_fst2_snd2*,*simplified*]
  unfolding *fst2_snd2_def*
  by *simp*

definition *fst2_sndfst_snd2*
  where *fst2_sndfst_snd2*(x) ≡ ⟨*fst*(*fst*(x)), *snd*(*fst*(x)), *snd*(*snd*(x))⟩

relativize functional *fst2_sndfst_snd2 fst2_sndfst_snd2_rel*
relationalize *fst2_sndfst_snd2_rel is_fst2_sndfst_snd2*

lemma (**in** *M_trivial*) *fst2_sndfst_snd2_abs*:
  assumes *M*(x) *M*(*res*)
shows *is_fst2_sndfst_snd2*(M, x, *res*) ⟷ *res* = *fst2_sndfst_snd2*(x)
  unfolding *is_fst2_sndfst_snd2_def fst2_sndfst_snd2_def*
  using *fst_rel_abs*[*symmetric*] *snd_rel_abs*[*symmetric*] *fst_abs snd_abs assms*
  by *simp*

synthesize *is_fst2_sndfst_snd2* **from_definition assuming** *nonempty*
arity_theorem for *is_fst2_sndfst_snd2_fm*

lemma (**in** *M_ZF_trans*) *replacement_is_fst2_sndfst_snd2*:
strong_replacement(##*M*, *is_fst2_sndfst_snd2*(##*M*))
apply(*rule_tac strong_replacement_cong*[
where *P*=λ *x f*. *M*,[*x*,*f*] ⊨ *is_fst2_sndfst_snd2_fm*(*0*,*1*),*THEN iffD1*])
apply(*rule_tac is_fst2_sndfst_snd2_iff_sats*[**where** *env*=[_,_],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac replacement_ax*[**where** *env*=[],*simplified*])
apply(*simp_all add*: *arity_is_fst2_sndfst_snd2_fm ord_simp_union*)
done

lemma (**in** *M_ZF_trans*) *replacement_fst2_sndfst_snd2*:
strong_replacement(##*M*, λ*x y*. *y* = ⟨*fst*(*fst*(*x*)), *snd*(*fst*(*x*)), *snd*(*snd*(*x*))⟩)
using *strong_replacement_cong*[*THEN iffD1*,*OF fst2_sndfst_snd2_abs replacement_is_fst2_sndfst_snd2*,s
unfolding *fst2_sndfst_snd2_def*
by *simp*

lemmas (**in** *M_ZF_trans*) *M_replacement_ZF_instances* = *lam_replacement_domain*
lam_replacement_fst lam_replacement_snd lam_replacement_Union
lam_replacement_Upair lam_replacement_image
lam_replacement_Diff lam_replacement_vimage
separation_fst_equal separation_id_rel[*simplified*]
separation_equal_apply separation_sndfst_eq_fstsnd
separation_fstfst_eq_fstsnd separation_fstfst_eq
separation_restrict_elem
replacement_fst2_snd2 replacement_fst2_sndfst_snd2

sublocale *M_ZF_trans* ⊆ *M_replacement* ##*M*
by *unfold_locales* (*simp_all add*: *M_replacement_ZF_instances del*:*setclass_iff*)

definition *RepFun_body* :: *i* ⇒ *i* ⇒ *i***where**
RepFun_body(*u*,*v*) ≡ {{⟨*v*, *x*⟩} . *x* ∈ *u*}

relativize functional *RepFun_body RepFun_body_rel*
relationalize *RepFun_body_rel is_RepFun_body*

lemma (**in** *M_trivial*) *RepFun_body_abs*:
assumes *M*(*u*) *M*(*v*) *M*(*res*)
shows *is_RepFun_body*(*M*, *u*, *v*, *res*) ⟷ *res* = *RepFun_body*(*u*,*v*)
unfolding *is_RepFun_body_def RepFun_body_def*
using *fst_rel_abs*[*symmetric*] *snd_rel_abs*[*symmetric*] *fst_abs snd_abs assms*
Replace_abs[**where** *P*=λ*xa a*. *a* = {⟨*v*, *xa*⟩} **and** *A*=*u*]
univalent_triv transM[*of* _ *u*]
by *auto*

synthesize *is_RepFun_body* **from_definition assuming** *nonempty*
arity_theorem for *is_RepFun_body_fm*
lemma *arity_body_repfun*:

arity(·(·∃·*0* = *0*··) ∧ ·(·∃·*0* = *0*··) ∧ (·∃·*cons_fm*(*0*, *3*, *2*) ∧ *pair_fm*(*5*, *1*, *0*)
··)··) = *5*
using *arity_cons_fm arity_pair_fm pred_Un_distrib union_abs1*
by *auto*

lemma *arity_RepFun*: *arity*(*is_RepFun_body_fm*(*0*, *1*, *2*)) = *3*
unfolding *is_RepFun_body_fm_def*
using *arity_Replace_fm*[*OF* _ _ _ _ _ *arity_body_repfun*] *arity_fst_fm arity_snd_fm*
arity_empty_fm
pred_Un_distrib union_abs2 union_abs1
by *simp*

lemma (**in** *M_ZF_trans*) *RepFun_SigFun_closed*: *x* ∈ *M* ⟹ *z* ∈ *M* ⟹ {{⟨*z*,
x⟩} . *x* ∈ *x*} ∈ *M*
using *lam_replacement_sing_const_id lam_replacement_imp_strong_replacement*
RepFun_closed
transitivity singleton_in_M_iff pair_in_M_iff
by *simp*

lemma (**in** *M_ZF_trans*) *replacement_RepFun_body*:
lam_replacement(*##M*, λ*p* . {{⟨*snd*(*p*), *x*⟩} . *x* ∈ *fst*(*p*)})
apply(*rule_tac lam_replacement_iff_lam_closed*[*THEN iffD2*,*of* λ*p*. {{⟨*snd*(*p*),
x⟩} . *x* ∈ *fst*(*p*)}])
using *RepFun_SigFun_closed*[*OF fst_snd_closed*[*THEN conjunct1*,*simplified*]
fst_snd_closed[*THEN conjunct2*,*simplified*]] *transitivity*
apply *auto*
apply (*rule_tac*
LambdaPair_in_M[**where** *φ*=*is_RepFun_body_fm*(*0*,*1*,*2*) **and** *is_f*=*is_RepFun_body*(*##M*)
and *env*=[],*OF*
is_RepFun_body_fm_type _ *is_RepFun_body_iff_sats*[*symmetric*]])
apply (*auto simp*: *arity_RepFun ord_simp_union transitivity zero_in_M RepFun_body_def*
RepFun_body_abs RepFun_SigFun_closed)
done

sublocale *M_ZF_trans* ⊆ *M_replacement_extra ##M*
by *unfold_locales* (*simp_all add*: *replacement_RepFun_body*
lam_replacement_minimum del:*setclass_iff*)

sublocale *M_ZF_trans* ⊆ *M_Perm ##M*
using *separation_PiP_rel separation_injP_rel separation_surjP_rel*
lam_replacement_imp_strong_replacement[*OF*
lam_replacement_Sigfun[*OF lam_replacement_constant*]]
Pi_replacement1 **unfolding** *Sigfun_def*
by *unfold_locales simp_all*

definition *order_eq_map* **where**
order_eq_map(*M*,*A*,*r*,*a*,*z*) ≡ ∃ *x*[*M*]. ∃ *g*[*M*]. ∃ *mx*[*M*]. ∃ *par*[*M*].
ordinal(*M*,*x*) & *pair*(*M*,*a*,*x*,*z*) & *membership*(*M*,*x*,*mx*) &
pred_set(*M*,*A*,*a*,*r*,*par*) & *order_isomorphism*(*M*,*par*,*r*,*x*,*mx*,*g*)

synthesize *order_eq_map* **from_definition assuming** *nonempty*
arity_theorem for *is_ord_iso_fm*
arity_theorem for *order_eq_map_fm*

lemma (**in** *M_ZF_trans*) *replacement_is_order_eq_map*:
A∈M ⟹ *r∈M* ⟹ *strong_replacement*(##*M*, *order_eq_map*(##*M*,*A*,*r*))
apply(*rule_tac strong_replacement_cong*[
where *P*=λ *x f*. *M*,[*x*,*f*,*A*,*r*] ⊨ *order_eq_map_fm*(*2*,*3*,*0*,*1*),*THEN iffD1*])
apply(*rule_tac order_eq_map_iff_sats*[**where** *env*=[_,_,*A*,*r*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac replacement_ax*[**where** *env*=[*A*,*r*],*simplified*])
apply(*simp_all add*: *arity_order_eq_map_fm ord_simp_union*)
done

synthesize *is_banach_functor* **from_definition assuming** *nonempty*
arity_theorem for *is_banach_functor_fm*

definition *banach_body_iterates* **where**
banach_body_iterates(*M*,*X*,*Y*,*f*,*g*,*W*,*n*,*x*,*z*) ≡
∃ *y*[*M*].
pair(*M*, *x*, *y*, *z*) ∧
(∃ *fa*[*M*].
(∀ *z*[*M*].
z ∈ *fa* ⟷
(∃ *xa*[*M*].
∃ *y*[*M*].
∃ *xaa*[*M*].
∃ *sx*[*M*].
∃ *r_sx*[*M*].
∃ *f_r_sx*[*M*]. ∃ *sn*[*M*]. ∃ *msn*[*M*]. *successor*(*M*,*n*,*sn*)
∧
membership(*M*,*sn*,*msn*) ∧
pair(*M*, *xa*, *y*, *z*) ∧
pair(*M*, *xa*, *x*, *xaa*) ∧
upair(*M*, *xa*, *xa*, *sx*) ∧
pre_image(*M*, *msn*, *sx*, *r_sx*) ∧
restriction(*M*, *fa*, *r_sx*, *f_r_sx*) ∧
xaa ∈ *msn* ∧
(*empty*(*M*, *xa*) ⟶ *y* = *W*) ∧
(∀ *m*[*M*].
successor(*M*, *m*, *xa*) ⟶
(∃ *gm*[*M*].
is_apply(*M*, *f_r_sx*, *m*, *gm*) ∧
is_banach_functor(*M*, *X*, *Y*, *f*, *g*, *gm*, *y*))) ∧
(*is_quasinat*(*M*, *xa*) ∨ *empty*(*M*, *y*)))) ∧
(*empty*(*M*, *x*) ⟶ *y* = *W*) ∧

```
                    (∀ m[M].
                        successor(M, m, x) ⟶
                        (∃ gm[M]. is_apply(M, fa, m, gm) ∧ is_banach_functor(M,
X, Y, f, g, gm, y))) ∧
                    (is_quasinat(M, x) ∨ empty(M, y)))

synthesize is_quasinat from_definition assuming nonempty
arity_theorem for is_quasinat_fm

synthesize banach_body_iterates from_definition assuming nonempty
arity_theorem for banach_body_iterates_fm

lemma (in M_ZF_trans) banach_iterates:
  assumes X∈M Y∈M f∈M g∈M W∈M
  shows iterates_replacement(##M, is_banach_functor(##M,X,Y,f,g), W)
proof -
 have strong_replacement(##M, λ x z . banach_body_iterates(##M,X,Y,f,g,W,n,x,z))
if n∈ω for n
    apply(rule_tac strong_replacement_cong[
      where P=λ x z. M,[x,z,_,W,g,f,Y,X] ⊨ banach_body_iterates_fm(7,6,5,4,3,2,0,1),THEN
iffD1])
     prefer 2
     apply(rule_tac replacement_ax[where env=[n,W,g,f,Y,X],simplified])
    using assms that
   by(simp_all add:zero_in_M arity_banach_body_iterates_fm ord_simp_union
transitivity[OF _ nat_in_M])
  then
  show ?thesis
    using assms zero_in_M transitivity[OF _ nat_in_M] Memrel_closed
   unfolding iterates_replacement_def wfrec_replacement_def is_wfrec_def M_is_recfun_def
      is_nat_case_def iterates_MH_def banach_body_iterates_def
    by simp
qed

definition banach_is_iterates_body where
  banach_is_iterates_body(M,X,Y,f,g,W,n,y) ≡ ∃ om[M]. omega(M,om) ∧ n ∈
om ∧
            (∃ sn[M].
                ∃ msn[M].
                   successor(M, n, sn) ∧
                   membership(M, sn, msn) ∧
                   (∃ fa[M].
                       (∀ z[M].
                           z ∈ fa ⟷
                           (∃ x[M].
                               ∃ y[M].
                                  ∃ xa[M].
                                     ∃ sx[M].
                                        ∃ r_sx[M].
```

$$\exists f\_r\_sx[M].\ pair(M,\ x,\ y,\ z) \wedge\ pair(M,\ x,\ n,\ xa) \wedge\ upair(M,\ x,\ x,\ sx) \wedge\ pre\_image(M,\ msn,\ sx,\ r\_sx) \wedge\ restriction(M,\ fa,\ r\_sx,\ f\_r\_sx) \wedge\ xa \in msn \wedge\ (empty(M,\ x) \longrightarrow y = W) \wedge\ (\forall m[M].\ successor(M,\ m,\ x) \longrightarrow\ (\exists gm[M].\ fun\_apply(M,\ f\_r\_sx,\ m,\ gm) \wedge\ is\_banach\_functor(M,\ X,\ Y,\ f,\ g,\ gm,\ y))) \wedge\ (is\_quasinat(M,\ x) \vee empty(M,\ y)))) \wedge\ (empty(M,\ n) \longrightarrow y = W) \wedge\ (\forall m[M].\ successor(M,\ m,\ n) \longrightarrow\ (\exists gm[M].\ fun\_apply(M,\ fa,\ m,\ gm) \wedge\ is\_banach\_functor(M,\ X,\ Y,\ f,\ g,\ gm,\ y))) \wedge\ (is\_quasinat(M,\ n) \vee empty(M,\ y))))$$

synthesize *banach_is_iterates_body* **from_definition assuming** *nonempty*
arity_theorem for *banach_is_iterates_body_fm*

lemma (**in** *M_ZF_trans*) *banach_replacement_iterates*:
 assumes $X{\in}M$ $Y{\in}M$ $f{\in}M$ $g{\in}M$ $W{\in}M$
 shows *strong_replacement*(##*M*, $\lambda n\ y.\ n{\in}\omega \wedge$ *is_iterates*(##*M*,*is_banach_functor*(##*M*,*X*, *Y*, *f*, *g*),*W*,*n*,*y*))
proof -
 have *strong_replacement*(##*M*, $\lambda\ n\ z\ .$ *banach_is_iterates_body*(##*M*,*X*,*Y*,*f*,*g*,*W*,*n*,*z*))

  apply(*rule_tac strong_replacement_cong*[
    where $P{=}\lambda\ n\ z.\ M$,[*n*,*z*,*W*,*g*,*f*,*Y*,*X*] $\models$ *banach_is_iterates_body_fm*(6,5,4,3,2,0,1),*THEN iffD1*])
   prefer *2*
   apply(*rule_tac replacement_ax*[**where** *env*=[*W*,*g*,*f*,*Y*,*X*],*simplified*])
  using *assms*
 by(*simp_all add*:*zero_in_M arity_banach_is_iterates_body_fm ord_simp_union transitivity*[*OF* _ *nat_in_M*])
 then
 show *?thesis*
  using *assms nat_in_M*
 unfolding *is_iterates_def wfrec_replacement_def is_wfrec_def M_is_recfun_def is_nat_case_def iterates_MH_def banach_is_iterates_body_def*
  by *simp*
qed

lemma (**in** *M_ZF_trans*) *banach_replacement*:
 assumes (##*M*)(*X*) (##*M*)(*Y*) (##*M*)(*f*) (##*M*)(*g*)

shows *strong_replacement*(##*M*, λ*n y*. *n*∈*nat* ∧ *y* = *banach_functor*(*X*, *Y*, *f*, *g*)^*n* (*0*))
using *iterates_abs*[*OF banach_iterates banach_functor_abs*,*of X Y f g*]
assms banach_functor_closed zero_in_M
apply (*rule_tac strong_replacement_cong*[*THEN iffD1*, *OF _ banach_replacement_iterates*[*of X Y f g 0*]])
by(*rule_tac conj_cong*,*simp_all*)

lemma (**in** *M_ZF_trans*) *lam_replacement_cardinal* : *lam_replacement*(##*M*, *cardinal_rel*(##*M*))
using *lam_replacement_iff_lam_closed*[*THEN iffD2*,*of cardinal_rel*(##*M*)]
cardinal_rel_closed[*of _*]
Lambda_in_M[**where** φ=*is_cardinal_fm*(*0*,*1*) **and** *env*=[],*OF*
is_cardinal_fm_type[*of 0 1*] *_ is_cardinal_iff_sats*[*symmetric*] *is_cardinal_iff*]
arity_is_cardinal_fm[*of 0 1*] *ord_simp_union cardinal_rel_closed transitivity zero_in_M*
by *simp_all*

definition *trans_apply_image* **where**
trans_apply_image(*f*) ≡ λ*a g*. *f* ‘ (*g* ‘‘ *a*)

relativize functional *trans_apply_image trans_apply_image_rel*
relationalize *trans_apply_image is_trans_apply_image*

schematic_goal *arity_is_recfun_fm*[*arity*]:
p ∈ *formula* ⟹ *a* ∈ ω ⟹ *z* ∈ ω ⟹ *r* ∈ ω ⟹ *arity*(*is_recfun_fm*(*p*, *a*, *z* ,*r*)) = *?ar*
unfolding *is_recfun_fm_def*
by (*simp add*:*arity*)

schematic_goal *arity_is_wfrec_fm*[*arity*]:
p ∈ *formula* ⟹ *a* ∈ ω ⟹ *z* ∈ ω ⟹ *r* ∈ ω ⟹ *arity*(*is_wfrec_fm*(*p*, *a*, *z* ,*r*)) = *?ar*
unfolding *is_wfrec_fm_def*
by (*simp add*:*arity*)
schematic_goal *arity_is_transrec_fm*[*arity*]:
p ∈ *formula* ⟹ *a* ∈ ω ⟹ *z* ∈ ω ⟹ *arity*(*is_transrec_fm*(*p*, *a*, *z*)) = *?ar*
unfolding *is_transrec_fm_def*
by (*simp add*:*arity*)

synthesize *is_trans_apply_image* **from_definition assuming** *nonempty*
arity_theorem for *is_trans_apply_image_fm*

lemma (**in** *M_basic*) *rel2_trans_apply*:
M(*f*) ⟹ *relation2*(*M*,*is_trans_apply_image*(*M*,*f*),*trans_apply_image*(*f*))

unfolding *is_trans_apply_image_def trans_apply_image_def relation2_def*
by *auto*

lemma (**in** *M_basic*) *apply_image_closed*:
shows $M(f) \Longrightarrow \forall\, x[M].\ \forall\, g[M].\ function(g) \longrightarrow M(trans\_apply\_image(f,\ x,\ g))$
unfolding *trans_apply_image_def* **by** *simp*

lemma (**in** *M_basic*) *apply_image_closed'*:
shows $M(f) \Longrightarrow \forall\, x[M].\ \forall\, g[M].\ M(trans\_apply\_image(f,\ x,\ g))$
unfolding *trans_apply_image_def* **by** *simp*

definition *transrec_apply_image_body* **where**
$transrec\_apply\_image\_body(M,f,mesa,x,z) \equiv\ \exists\, y[M].\ pair(M,\ x,\ y,\ z) \wedge$
$(\exists\, fa[M].$
$(\forall\, z[M].$
$z \in fa \longleftrightarrow$
$(\exists\, xa[M].$
$\exists\, y[M].$
$\exists\, xaa[M].$
$\exists\, sx[M].$
$\exists\, r\_sx[M].$
$\exists\, f\_r\_sx[M].$
$pair(M,\ xa,\ y,\ z) \wedge$
$pair(M,\ xa,\ x,\ xaa) \wedge$
$upair(M,\ xa,\ xa,\ sx) \wedge$
$pre\_image(M,\ mesa,\ sx,\ r\_sx) \wedge$
$restriction(M,\ fa,\ r\_sx,\ f\_r\_sx) \wedge$
$xaa \in mesa \wedge is\_trans\_apply\_image(M,$
$f,\ xa,\ f\_r\_sx,\ y))) \wedge$
$is\_trans\_apply\_image(M,\ f,\ x,\ fa,\ y))$

synthesize *transrec_apply_image_body* **from_definition assuming** *nonempty*
arity_theorem for *transrec_apply_image_body_fm*

lemma (**in** *M_ZF_trans*) *replacement_transrec_apply_image_body* :
$(\#\#M)(f) \Longrightarrow (\#\#M)(mesa) \Longrightarrow strong\_replacement(\#\#M,transrec\_apply\_image\_body(\#\#M,f,mesa))$
apply(*rule_tac strong_replacement_cong*[
where $P{=}\lambda\, x\, z.\ M,[x,z,mesa,f] \models transrec\_apply\_image\_body\_fm(3,2,0,1)$, *THEN*
iffD1])
apply(*rule_tac transrec_apply_image_body_iff_sats*[**where** *env*=[_,_,*mesa*,*f*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac replacement_ax*[**where** *env*=[*mesa*,*f*],*simplified*])
apply(*simp_all add*: *arity_transrec_apply_image_body_fm ord_simp_union*)
done

lemma (**in** *M_ZF_trans*) *transrec_replacement_apply_image*:
assumes $(\#\#M)(f)$ $(\#\#M)(\alpha)$
shows $transrec\_replacement(\#\#M,\ is\_trans\_apply\_image(\#\#M,\ f),\ \alpha)$
unfolding *transrec_replacement_def wfrec_replacement_def is_wfrec_def M_is_recfun_def*

using *replacement_transrec_apply_image_body*[*unfolded transrec_apply_image_body_def*]
assms
Memrel_closed singleton_closed eclose_closed
by *simp*

lemma (**in** *M_ZF_trans*) *rec_trans_apply_image_abs*:
assumes (##*M*)(*f*) (##*M*)(*x*) (##*M*)(*y*) *Ord*(*x*)
shows *is_transrec*(##*M*,*is_trans_apply_image*(##*M*, *f*),*x*,*y*) ⟷ *y* = *transrec*(*x*,*trans_apply_image*(*f*))
using *transrec_abs*[*OF transrec_replacement_apply_image rel2_trans_apply*]
assms apply_image_closed
by *simp*

definition *is_trans_apply_image_body* **where**
is_trans_apply_image_body(*M*,*f*,*β*,*a*,*w*) ≡ ∃ *z*[*M*]. *pair*(*M*,*a*,*z*,*w*) ∧ *a*∈*β* ∧ (∃ *sa*[*M*].
∃ *esa*[*M*].
∃ *mesa*[*M*].
upair(*M*, *a*, *a*, *sa*) ∧
is_eclose(*M*, *sa*, *esa*) ∧
membership(*M*, *esa*, *mesa*) ∧
(∃ *fa*[*M*].
(∀ *z*[*M*].
z ∈ *fa* ⟷
(∃ *x*[*M*].
∃ *y*[*M*].
∃ *xa*[*M*].
∃ *sx*[*M*].
∃ *r_sx*[*M*].
∃ *f_r_sx*[*M*].
pair(*M*, *x*, *y*, *z*) ∧
pair(*M*, *x*, *a*, *xa*) ∧
upair(*M*, *x*, *x*, *sx*) ∧
pre_image(*M*, *mesa*, *sx*, *r_sx*) ∧
restriction(*M*, *fa*, *r_sx*, *f_r_sx*) ∧
xa ∈ *mesa* ∧ *is_trans_apply_image*(*M*, *f*,
x, *f_r_sx*, *y*))) ∧
is_trans_apply_image(*M*, *f*, *a*, *fa*, *z*)))

manual_schematic *is_trans_apply_image_body_schematic* **for** *is_trans_apply_image_body***assuming**
nonempty
unfolding *is_trans_apply_image_body_def*
by (*rule sep_rules is_eclose_iff_sats is_trans_apply_image_iff_sats* | *simp*)+

synthesize *is_trans_apply_image_body* **from_schematic** *is_trans_apply_image_body_schematic*
arity_theorem for *is_trans_apply_image_body_fm*

lemma (**in** *M_ZF_trans*) *replacement_is_trans_apply_image*:
(##*M*)(*f*) ⟹ (##*M*)(*β*) ⟹ *strong_replacement*(##*M*, λ *x z* .
∃ *y*[##*M*]. *pair*(##*M*,*x*,*y*,*z*) ∧ *x*∈*β* ∧ (*is_transrec*(##*M*,*is_trans_apply_image*(##*M*,
f),*x*,*y*)))

unfolding *is_transrec_def is_wfrec_def M_is_recfun_def*
apply(*rule_tac strong_replacement_cong*[
where $P=\lambda\, x\, z.\ M,[x,z,\beta,f] \models$ *is_trans_apply_image_body_fm*(3,2,0,1),*THEN iffD1*])
apply(*rule_tac is_trans_apply_image_body_iff_sats*[*symmetric*,*unfolded is_trans_apply_image_body_def env*=[_,_,β,*f*]])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac replacement_ax*[**where** *env*=[β,*f*],*simplified*])
apply(*simp_all add*: *arity_is_trans_apply_image_body_fm is_trans_apply_image_body_fm_type ord_simp_union*)
done

lemma (**in** *M_ZF_trans*) *trans_apply_abs*:
$(\#\#M)(f) \Longrightarrow (\#\#M)(\beta) \Longrightarrow Ord(\beta) \Longrightarrow (\#\#M)(x) \Longrightarrow (\#\#M)(z) \Longrightarrow$
$(x\in\beta \wedge z = \langle x,\ transrec(x,\ \lambda a\ g.\ f\ `\ (g\ ``\ a))\ \rangle) \longleftrightarrow$
$(\exists\, y[\#\#M].\ pair(\#\#M,x,y,z) \wedge x\in\beta \wedge (is\_transrec(\#\#M,is\_trans\_apply\_image(\#\#M,f),x,y)))$
using *rec_trans_apply_image_abs Ord_in_Ord*
transrec_closed[*OF transrec_replacement_apply_image rel2_trans_apply*,*of f*,*simplified*]
apply_image_closed'[*of f*]
unfolding *trans_apply_image_def*
by *auto*

lemma (**in** *M_ZF_trans*) *replacement_trans_apply_image*:
$(\#\#M)(f) \Longrightarrow (\#\#M)(\beta) \Longrightarrow Ord(\beta) \Longrightarrow$
strong_replacement$(\#\#M,\ \lambda x\ y.\ x\in\beta \wedge y = \langle x,\ transrec(x,\ \lambda a\ g.\ f\ `\ (g\ ``\ a))\rangle)$
using *strong_replacement_cong*[*THEN iffD1*,*OF _ replacement_is_trans_apply_image*,*simplified*]

trans_apply_abs Ord_in_Ord
by *simp*

definition *abs_apply_pair* **where**
$abs\_apply\_pair(A,f,x) \equiv \langle x,\ \lambda n\in A.\ f\ `\ \langle x,\ n\rangle\rangle$

relativize functional *abs_apply_pair abs_apply_pair_rel*
relationalize *abs_apply_pair_rel is_abs_apply_pair*

lemma (**in** *M_basic*) *abs_apply_pair_rel*:
assumes $M(A)$ $M(f)$ $M(x)$
shows *Relation1*$(M,A,\lambda n\ a.\ \exists\, b[M].\ is\_apply(M,\ f,\ b,\ a) \wedge pair(M,\ x,\ n,\ b),\ \lambda n.\ f\ `\ \langle x,\ n\rangle)$
using *fst_rel_abs*[*symmetric*] *snd_rel_abs*[*symmetric*] *fst_abs snd_abs assms*
unfolding *Relation1_def*
by *auto*

lemma (**in** *M_basic*) *abs_apply_pair_abs*:
assumes $M(A)$ $M(f)$ $M(x)$ $M(res)$
shows $is\_abs\_apply\_pair(M,A,f,\ x,\ res) \longleftrightarrow res = abs\_apply\_pair(A,f,x)$

unfolding *is_abs_apply_pair_def abs_apply_pair_def*
using *fst_rel_abs*[*symmetric*] *snd_rel_abs*[*symmetric*] *fst_abs snd_abs assms*
pair_abs lambda_abs2[*OF abs_apply_pair_rel*]
by *auto*

synthesize *is_abs_apply_pair* **from_definition assuming** *nonempty*

lemma *arity_is_abs_aux*: *arity*((·∃ ··7‘0 *is* 1· ∧ *pair_fm*(5, 2, 0) ··)) = 7
using *arity_fun_apply_fm arity_pair_fm pred_Un_distrib*
ord_simp_union **by** *simp*

lemma *arity_is_abs_apply_pair_fm* :
shows *arity*(*is_abs_apply_pair_fm*(3, 2, 0, 1)) = 4
unfolding *is_abs_apply_pair_fm_def*
using *arity_lambda_fm*[*OF _ _ _ _ arity_is_abs_aux*] *arity_pair_fm*
pred_Un_distrib ord_simp_union
by *simp*

lemma (**in** *M_ZF_trans*) *replacement_is_abs_apply_pair*:
assumes *A*∈*M f*∈*M*
shows *strong_replacement*(##*M*, *is_abs_apply_pair*(##*M*,*A*,*f*))
using *assms*
apply(*rule_tac strong_replacement_cong*[
where *P*=λ *x z*. *M*,[*x*,*z*,*f*,*A*] ⊨ *is_abs_apply_pair_fm*(3,2,0,1),*THEN*
iffD1])
apply(*rule_tac is_abs_apply_pair_iff_sats*[**where** *env*=[_,_,*f*,*A*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac replacement_ax*[**where** *env*=[*f*,*A*],*simplified*])
apply(*simp_all add*: *arity_is_abs_apply_pair_fm ord_simp_union*)
done

lemma (**in** *M_ZF_trans*) *replacement_abs_apply_pair*:
(##*M*)(*A*) ⟹ (##*M*)(*f*) ⟹ *strong_replacement*(##*M*, λ*x y*. *y* = ⟨*x*, λ*n*∈*A*.
f ‘ ⟨*x*, *n*⟩⟩)
using *strong_replacement_cong*[*THEN iffD1*,*OF abs_apply_pair_abs replacement_is_abs_apply_pair*,*simp*
unfolding *abs_apply_pair_def*
by *simp*

end

29 Separative notions and proper extensions

theory *Proper_Extension*
imports
Names

begin

The key ingredient to obtain a proper extension is to have a *separative*

preorder:

locale *separative_notion* = *forcing_notion* +
  assumes *separative*: $p{\in}P \Longrightarrow \exists\, q{\in}P.\ \exists\, r{\in}P.\ q \preceq p \wedge r \preceq p \wedge q \perp r$
begin

For separative preorders, the complement of every filter is dense. Hence an M-generic filter can't belong to the ground model.

lemma *filter_complement_dense*:
  assumes *filter*(G) **shows** *dense*(P - G)
proof
  fix p
  assume $p{\in}P$
  show $\exists\, d{\in}P$ - $G.\ d \preceq p$
  proof (*cases* $p{\in}G$)
    case *True*
    note ‹$p{\in}P$› *assms*
    moreover
    obtain $q\ r$ **where** $q \preceq p\ r \preceq p\ q \perp r\ q{\in}P\ r{\in}P$
      using *separative*[*OF* ‹$p{\in}P$›]
      by *force*
    with ‹*filter*(G)›
    obtain s **where** $s \preceq p\ s \notin G\ s \in P$
      using *filter_imp_compat*[*of* $G\ q\ r$]
      by *auto*
    then
    show *?thesis* **by** *blast*
  next
    case *False*
    with ‹$p{\in}P$›
    show *?thesis* **using** *refl_leq* **unfolding** *Diff_def* **by** *auto*
  qed
qed

end

locale *ctm_separative* = *forcing_data* + *separative_notion*
begin

lemma *generic_not_in_M*: **assumes** *M_generic*(G) **shows** $G \notin M$
proof
  assume $G{\in}M$
  then
  have P - $G \in M$
    using *P_in_M Diff_closed* **by** *simp*
  moreover
  have $\neg(\exists\, q{\in}G.\ q \in P$ - $G)\ (P$ - $G) \subseteq P$
    unfolding *Diff_def* **by** *auto*
  moreover
  note *assms*

ultimately
show *False*
using *filter_complement_dense*[*of G*] *M_generic_denseD*[*of G P-G*]
M_generic_def **by** *simp* — need to put generic ==> filter in claset
qed

theorem *proper_extension*: **assumes** *M_generic*(*G*) **shows** $M \neq M[G]$
using *assms G_in_Gen_Ext*[*of G*] *one_in_G*[*of G*] *generic_not_in_M*
by *force*

end

end

30 A poset of succession

theory *Succession_Poset*
imports
Replacement_Instances
Proper_Extension
FiniteFun_Relative

begin

sublocale *M_ZF_trans* ⊆ *M_seqspace* ##*M*
by (*unfold_locales*, *simp add*:*replacement_omega_funspace*)

definition *seq_upd* :: $i \Rightarrow i \Rightarrow i$ **where**
seq_upd(*f*,*a*) ≡ λ *j* ∈ *succ*(*domain*(*f*)) . *if j* < *domain*(*f*) *then f*‘*j else a*

lemma *seq_upd_succ_type* :
assumes *n*∈*nat f*∈*n*→*A a*∈*A*
shows *seq_upd*(*f*,*a*)∈ *succ*(*n*) → *A*
proof -
from *assms*
have *equ*: *domain*(*f*) = *n* **using** *domain_of_fun* **by** *simp*
{
fix *j*
assume *j*∈*succ*(*domain*(*f*))
with *equ* ‹*n*∈_›
have *j*≤*n* **using** *ltI* **by** *auto*
with ‹*n*∈_›
consider (*lt*) *j*<*n* | (*eq*) *j*=*n* **using** *leD* **by** *auto*
then
have (*if j* < *n then f*‘*j else a*) ∈ *A*
proof *cases*
case *lt*
with ‹*f*∈_›
show *?thesis* **using** *apply_type ltD*[*OF lt*] **by** *simp*

```
    next
      case eq
      with ‹a∈_›
      show ?thesis by auto
    qed
  }
  with equ
  show ?thesis
    unfolding seq_upd_def
    using lam_type[of succ(domain(f))]
    by auto
qed

lemma seq_upd_type :
  assumes f∈A^<ω a∈A
  shows seq_upd(f,a) ∈ A^<ω
proof -
  from ‹f∈_›
  obtain y where y∈nat f∈y→A
    unfolding seqspace_def by blast
  with ‹a∈A›
  have seq_upd(f,a)∈succ(y)→A
    using seq_upd_succ_type by simp
  with ‹y∈_›
  show ?thesis
    unfolding seqspace_def by auto
qed

lemma seq_upd_apply_domain [simp]:
  assumes f:n→A n∈nat
  shows seq_upd(f,a)`n = a
  unfolding seq_upd_def using assms domain_of_fun by auto

lemma zero_in_seqspace :
  shows 0 ∈ A^<ω
  unfolding seqspace_def
  by force

definition
  seqleR :: i ⇒ i ⇒ o where
  seqleR(f,g) ≡ g ⊆ f

definition
  seqlerel :: i ⇒ i where
  seqlerel(A) ≡ Rrel(λx y. y ⊆ x,A^<ω)

definition
  seqle :: i where
  seqle ≡ seqlerel(2)
```

lemma *seqleI*[*intro*!]:
  $\langle f,g\rangle \in 2^{<\omega}\times 2^{<\omega} \Longrightarrow g \subseteq f \Longrightarrow \langle f,g\rangle \in seqle$
  unfolding *seqspace_def seqle_def seqlerel_def Rrel_def*
  by *blast*

lemma *seqleD*[*dest*!]:
  $z \in seqle \Longrightarrow \exists\, x\ y.\ \langle x,y\rangle \in 2^{<\omega}\times 2^{<\omega} \wedge y \subseteq x \wedge z = \langle x,y\rangle$
  unfolding *seqle_def seqlerel_def Rrel_def*
  by *blast*

lemma *upd_leI* :
  assumes $f{\in}2^{<\omega}$ $a{\in}2$
  shows $\langle seq\_upd(f,a),f\rangle{\in}seqle$ (**is** $\langle ?f,\_\rangle{\in}\_$)
proof
  show $\langle ?f, f\rangle \in 2^{<\omega} \times 2^{<\omega}$
    using *assms seq_upd_type* **by** *auto*
next
  show $f \subseteq seq\_upd(f,a)$
  proof
    fix x
    assume $x \in f$
    moreover from $\langle f \in 2^{<\omega}\rangle$
    obtain n **where** $n{\in}nat\ f : n \to 2$
      by *blast*
    moreover from *calculation*
    obtain y **where** $y{\in}n$ $x{=}\langle y,f`y\rangle$ **using** *Pi_memberD*[*of f n* $\lambda\_ .\ 2$]
      by *blast*
    moreover from $\langle f{:}n{\to}2\rangle$
    have $domain(f) = n$ **using** *domain_of_fun* **by** *simp*
    ultimately
    show $x \in seq\_upd(f,a)$
      unfolding *seq_upd_def lam_def*
      by (*auto intro*:*ltI*)
  qed
qed

lemma *preorder_on_seqle*: *preorder_on*($2^{<\omega}$,*seqle*)
  unfolding *preorder_on_def refl_def trans_on_def* **by** *blast*

lemma *zero_seqle_max*: $x{\in}2^{<\omega} \Longrightarrow \langle x,0\rangle \in seqle$
  using *zero_in_seqspace*
  by *auto*

interpretation *sp*:*forcing_notion* $2^{<\omega}$ *seqle 0*
  using *preorder_on_seqle zero_seqle_max zero_in_seqspace*
  by *unfold_locales simp_all*

notation *sp.Leq* (**infixl** $\preceq s$ *50*)

notation *sp.Incompatible* (**infixl** $\perp s$ *50*)

lemma *seqspace_separative*:
assumes $f \in 2^{<\omega}$
shows *seq_upd*$(f,0)$ $\perp s$ *seq_upd*$(f,1)$ (**is** *?f* $\perp s$ *?g*)
proof
assume *sp.compat*(*?f*, *?g*)
then
obtain h **where** $h \in 2^{<\omega}$ *?f* $\subseteq h$ *?g* $\subseteq h$
by *blast*
moreover from ‹$f \in \_$›
obtain y **where** $y \in nat$ $f : y \to 2$ **by** *blast*
moreover from *this*
have *?f*: *succ*$(y) \to 2$ *?g*: *succ*$(y) \to 2$
using *seq_upd_succ_type* **by** *blast+*
moreover from *this*
have $\langle y, ?f`y\rangle \in ?f$ $\langle y, ?g`y\rangle \in ?g$ **using** *apply_Pair* **by** *auto*
ultimately
have $\langle y,0\rangle \in h$ $\langle y,1\rangle \in h$ **by** *auto*
moreover from ‹$h \in 2^{<\omega}$›
obtain n **where** $n \in nat$ $h : n \to 2$ **by** *blast*
ultimately
show *False*
using *fun_is_function*[*of h n* $\lambda\_.$ *2*]
unfolding *seqspace_def function_def* **by** *auto*
qed

definition *is_seqleR* :: $[i \Rightarrow o, i, i] \Rightarrow o$ **where**
is_seqleR$(Q,f,g) \equiv g \subseteq f$

definition *seqleR_fm* :: $i \Rightarrow i$ **where**
seqleR_fm$(fg) \equiv$ *Exists*(*Exists*(*And*(*pair_fm*(*0*,*1*,*fg*#+*2*),*subset_fm*(*1*,*0*))))

lemma *type_seqleR_fm* :
$fg \in nat \Longrightarrow$ *seqleR_fm*$(fg) \in$ *formula*
unfolding *seqleR_fm_def*
by *simp*

lemma *arity_seqleR_fm* :
$fg \in nat \Longrightarrow$ *arity*(*seqleR_fm*(fg)) = *succ*(fg)
unfolding *seqleR_fm_def*
using *arity_pair_fm arity_subset_fm ord_simp_union* **by** *simp*

lemma (**in** *M_basic*) *seqleR_abs*:
assumes $M(f)$ $M(g)$
shows *seqleR*$(f,g) \longleftrightarrow$ *is_seqleR*(M,f,g)
unfolding *seqleR_def is_seqleR_def*
using *assms apply_abs domain_abs domain_closed*[*OF* ‹$M(f)$›] *domain_closed*[*OF* ‹$M(g)$›]

by *auto*

definition
relP :: $[i\Rightarrow o,[i\Rightarrow o,i,i]\Rightarrow o,i] \Rightarrow o$ **where**
$relP(M,r,xy) \equiv (\exists\, x[M].\ \exists\, y[M].\ pair(M,x,y,xy) \wedge r(M,x,y))$

lemma (**in** *M_ctm*) *seqleR_fm_sats* :
assumes $fg\in nat$ $env\in list(M)$
shows $sats(M,seqleR\_fm(fg),env) \longleftrightarrow relP(\#\#M,is\_seqleR,nth(fg,\ env))$
unfolding *seqleR_fm_def is_seqleR_def relP_def*
using *assms trans_M sats_subset_fm pair_iff_sats*
by *auto*

lemma (**in** *M_basic*) *is_related_abs* :
assumes $\bigwedge f\ g\ .\ M(f) \Longrightarrow M(g) \Longrightarrow rel(f,g) \longleftrightarrow is\_rel(M,f,g)$
shows $\bigwedge z\ .\ M(z) \Longrightarrow relP(M,is\_rel,z) \longleftrightarrow (\exists\, x\ y.\ z = \langle x,y\rangle \wedge rel(x,y))$
unfolding *relP_def* **using** *pair_in_M_iff assms* **by** *auto*

definition
is_RRel :: $[i\Rightarrow o,[i\Rightarrow o,i,i]\Rightarrow o,i,i] \Rightarrow o$ **where**
$is\_RRel(M,is\_r,A,r) \equiv \exists\, A2[M].\ cartprod(M,A,A,A2) \wedge is\_Collect(M,A2,\ relP(M,is\_r),r)$

lemma (**in** *M_basic*) *is_Rrel_abs* :
assumes $M(A)$ $M(r)$
$\bigwedge f\ g\ .\ M(f) \Longrightarrow M(g) \Longrightarrow rel(f,g) \longleftrightarrow is\_rel(M,f,g)$
shows $is\_RRel(M,is\_rel,A,r) \longleftrightarrow\ r = Rrel(rel,A)$
proof -
from ‹$M(A)$›
have $M(z)$ **if** $z\in A\times A$ **for** z
using *cartprod_closed transM*[*of z A*×*A*] *that* **by** *simp*
then
have *A*:$relP(M,\ is\_rel,\ z) \longleftrightarrow (\exists\, x\ y.\ z = \langle x,\ y\rangle \wedge rel(x,\ y))\ M(z)$ **if** $z\in A\times A$
for z
using *that is_related_abs*[*of rel is_rel*,*OF assms*(*3*)] **by** *auto*
then
have $Collect(A\times A,relP(M,is\_rel)) = Collect(A\times A,\lambda z.\ (\exists\, x\ y.\ z = \langle x,y\rangle \wedge rel(x,y)))$
using *Collect_cong*[*of A*×*A A*×*A relP*(*M*,*is_rel*),*OF* _ *A*(*1*)] *assms*(*1*) *assms*(*2*)
by *auto*
with *assms*
show *?thesis* **unfolding** *is_RRel_def Rrel_def* **using** *cartprod_closed*
by *auto*
qed

definition
is_seqlerel :: $[i\Rightarrow o,i,i] \Rightarrow o$ **where**
$is\_seqlerel(M,A,r) \equiv is\_RRel(M,is\_seqleR,A,r)$

lemma (**in** *M_basic*) *seqlerel_abs* :
  assumes $M(A)$ $M(r)$
  shows *is_seqlerel*(M,A,r) $\longleftrightarrow$ r = *Rrel*(*seqleR*,A)
  unfolding *is_seqlerel_def*
  using *is_Rrel_abs*[*OF* ‹$M(A)$› ‹$M(r)$›,*of seqleR is_seqleR*] *seqleR_abs*
  by *auto*

definition *RrelP* :: [i⇒i⇒o,i] ⇒ i **where**
  RrelP(R,A) ≡ {z∈A×A. ∃x y. z = ⟨x, y⟩ ∧ R(x,y)}

lemma *Rrel_eq* : *RrelP*(R,A) = *Rrel*(R,A)
  unfolding *Rrel_def RrelP_def* **by** *auto*

context *M_ctm*
begin

lemma *Rrel_closed*:
  assumes A∈M
    ⋀ a. a ∈ *nat* ⟹ *rel_fm*(a)∈*formula*
    ⋀ f g . (##M)(f) ⟹ (##M)(g) ⟹ *rel*(f,g) ⟷ *is_rel*(##M,f,g)
    arity(*rel_fm*(0)) = 1
    ⋀ a . a ∈ M ⟹ *sats*(M,*rel_fm*(0),[a]) ⟷ *relP*(##M,*is_rel*,a)
  shows (##M)(*Rrel*(*rel*,A))
proof -
  have z∈ M ⟹ *relP*(##M, *is_rel*, z) ⟷ (∃x y. z = ⟨x, y⟩ ∧ *rel*(x, y)) **for** z
    using *assms*(3) *is_related_abs*[*of rel is_rel*]
    by *auto*
  with *assms*
  have *Collect*(A×A,λz. (∃x y. z = ⟨x,y⟩ ∧ *rel*(x,y))) ∈ M
    using *Collect_in_M*[*OF assms*(2),*of 0* []] *cartprod_closed*
    by *simp*
  then show *?thesis*
  unfolding *Rrel_def* **by** *simp*
qed

lemma *seqle_in_M*: *seqle* ∈ M
  using *Rrel_closed seqspace_closed*
    transitivity[*OF* _ *nat_in_M*] *type_seqleR_fm*[*of 0*] *arity_seqleR_fm*[*of 0*]
    seqleR_fm_sats[*of 0*] *seqleR_abs seqlerel_abs*
  unfolding *seqle_def seqlerel_def seqleR_def*
  by *auto*

30.1 Cohen extension is proper

interpretation *ctm_separative* $2^{<\omega}$ *seqle 0*
proof (*unfold_locales*)
  fix f
  let *?q*=*seq_upd*(f,0) **and** *?r*=*seq_upd*(f,1)
  assume f ∈ $2^{<\omega}$

then
have *?q* $\preceq s$ *f* $\wedge$ *?r* $\preceq s$ *f* $\wedge$ *?q* $\perp s$ *?r*
using *upd_leI seqspace_separative* **by** *auto*
moreover from *calculation*
have *?q* $\in 2^{<\omega}$ *?r* $\in 2^{<\omega}$
using *seq_upd_type*[*of f 2*] **by** *auto*
ultimately
show $\exists\, q{\in}2^{<\omega}.\ \exists\, r{\in}2^{<\omega}.\ q \preceq s\ f \wedge r \preceq s\ f \wedge q \perp s\ r$
by (*rule_tac bexI*)+ — why the heck auto-tools don't solve this?
next
show $2^{<\omega} \in M$ **using** *nat_into_M seqspace_closed* **by** *simp*
next
show *seqle* $\in M$ **using** *seqle_in_M* **.**
qed

lemma *cohen_extension_is_proper*: $\exists\, G.\ M\_generic(G) \wedge M \neq M^{2^{<\omega}}[G]$
using *proper_extension generic_filter_existence zero_in_seqspace*
by *force*

end

end

31 The ZFC axioms, internalized

theory *Internal_ZFC_Axioms*
imports
Forcing_Data

begin

schematic_goal *ZF_union_auto*:
$Union\_ax(\#\#A) \longleftrightarrow (A, [] \models ?zfunion)$
unfolding *Union_ax_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_union* **from_schematic** *ZF_union_auto*
notation *ZF_union_fm* ($\langle\cdot Union\ Ax\cdot\rangle$)

schematic_goal *ZF_power_auto*:
$power\_ax(\#\#A) \longleftrightarrow (A, [] \models ?zfpow)$
unfolding *power_ax_def powerset_def subset_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_power* **from_schematic** *ZF_power_auto*
notation *ZF_power_fm* ($\langle\cdot Powerset\ Ax\cdot\rangle$)

schematic_goal *ZF_pairing_auto*:
$upair\_ax(\#\#A) \longleftrightarrow (A, [] \models ?zfpair)$

unfolding *upair_ax_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_pairing* **from_schematic** *ZF_pairing_auto*
notation *ZF_pairing_fm* (‹·*Pairing*·›)

schematic_goal *ZF_foundation_auto*:
foundation_ax(##*A*) ⟷ (*A*, [] ⊨ *?zffound*)
unfolding *foundation_ax_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_foundation* **from_schematic** *ZF_foundation_auto*
notation *ZF_foundation_fm* (‹·*Foundation*·›)

schematic_goal *ZF_extensionality_auto*:
extensionality(##*A*) ⟷ (*A*, [] ⊨ *?zfext*)
unfolding *extensionality_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_extensionality* **from_schematic** *ZF_extensionality_auto*
notation *ZF_extensionality_fm* (‹·*Extensionality*·›)

schematic_goal *ZF_infinity_auto*:
infinity_ax(##*A*) ⟷ (*A*, [] ⊨ (*?φ*(*i*,*j*,*h*)))
unfolding *infinity_ax_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_infinity* **from_schematic** *ZF_infinity_auto*
notation *ZF_infinity_fm* (‹·*Infinity*·›)

schematic_goal *ZF_choice_auto*:
choice_ax(##*A*) ⟷ (*A*, [] ⊨ (*?φ*(*i*,*j*,*h*)))
unfolding *choice_ax_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_choice* **from_schematic** *ZF_choice_auto*
notation *ZF_choice_fm* (‹·*AC*·›)

lemmas *ZFC_fm_defs* = *ZF_extensionality_fm_def ZF_foundation_fm_def ZF_pairing_fm_def*
ZF_union_fm_def ZF_infinity_fm_def ZF_power_fm_def ZF_choice_fm_def

lemmas *ZFC_fm_sats* = *ZF_extensionality_auto ZF_foundation_auto ZF_pairing_auto*
ZF_union_auto ZF_infinity_auto ZF_power_auto ZF_choice_auto

definition
ZF_fin :: *i* **where**
ZF_fin ≡ {·*Extensionality*·, ·*Foundation*·, ·*Pairing*·,
·*Union Ax*·, ·*Infinity*·, ·*Powerset Ax*·}

definition
ZFC_fin :: *i* **where**
ZFC_fin ≡ *ZF_fin* ∪ {·*AC*·}

lemma *ZFC_fin_type* : *ZFC_fin* ⊆ *formula*
unfolding *ZFC_fin_def ZF_fin_def ZFC_fm_defs* **by** (*auto*)

31.1 The Axiom of Separation, internalized

lemma *iterates_Forall_type* [*TC*]:
⟦ *n* ∈ *nat*; *p* ∈ *formula* ⟧ ⟹ *Forall*^*n*(*p*) ∈ *formula*
by (*induct set*:*nat*, *auto*)

lemma *last_init_eq* :
assumes *l* ∈ *list*(*A*) *length*(*l*) = *succ*(*n*)
shows ∃ *a*∈*A*. ∃ *l*′∈*list*(*A*). *l* = *l*′@[*a*]
proof-
from ‹*l*∈_› ‹*length*(_) = _›
have *rev*(*l*) ∈ *list*(*A*) *length*(*rev*(*l*)) = *succ*(*n*)
by *simp_all*
then
obtain *a l*′ **where** *a*∈*A l*′∈*list*(*A*) *rev*(*l*) = *Cons*(*a*,*l*′)
by (*cases*;*simp*)
then
have *l* = *rev*(*l*′) @ [*a*] *rev*(*l*′) ∈ *list*(*A*)
using *rev_rev_ident*[*OF* ‹*l*∈_›] **by** *auto*
with ‹*a*∈_›
show *?thesis* **by** *blast*
qed

lemma *take_drop_eq* :
assumes *l*∈*list*(*M*)
shows ⋀ *n* . *n* < *succ*(*length*(*l*)) ⟹ *l* = *take*(*n*,*l*) @ *drop*(*n*,*l*)
using ‹*l*∈*list*(*M*)›
proof *induct*
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a l*)
then show *?case*
proof -
{
fix *i*
assume *i*<*succ*(*succ*(*length*(*l*)))
with ‹*l*∈*list*(*M*)›
consider (*lt*) *i* = *0* | (*eq*) ∃ *k*∈*nat*. *i* = *succ*(*k*) ∧ *k* < *succ*(*length*(*l*))
using ‹*l*∈*list*(*M*)› *le_natI nat_imp_quasinat*
by (*cases rule*:*nat_cases*[*of i*];*auto*)
then

```
    have take(i,Cons(a,l)) @ drop(i,Cons(a,l)) = Cons(a,l)
      using Cons
      by (cases;auto)
   }
   then show ?thesis using Cons by auto
  qed
qed

lemma list_split :
assumes n ≤ succ(length(rest)) rest ∈ list(M)
shows  ∃ re∈list(M). ∃ st∈list(M). rest = re @ st ∧ length(re) = pred(n)
proof -
  from assms
  have pred(n) ≤ length(rest)
    using pred_mono[OF _ ‹n≤_›] pred_succ_eq by auto
  with ‹rest∈_›
  have pred(n)∈nat rest = take(pred(n),rest) @ drop(pred(n),rest) (is _ = ?re @
?st)
    using take_drop_eq[OF ‹rest∈_›] le_natI by auto
  then
  have length(?re) = pred(n) ?re∈list(M) ?st∈list(M)
    using length_take[rule_format,OF _ ‹pred(n)∈_›] ‹pred(n) ≤ _› ‹rest∈_›
    unfolding min_def
    by auto
  then
  show ?thesis
   using rev_bexI[of _ _ λ re. ∃ st∈list(M). rest = re @ st ∧ length(re) = pred(n)]
     ‹length(?re) = _› ‹rest = _›
    by auto
qed

lemma sats_nForall:
  assumes
    φ ∈ formula
  shows
    n∈nat ⟹ ms ∈ list(M) ⟹
      (M, ms ⊨ (Forall^n(φ))) ⟷
      (∀ rest ∈ list(M). length(rest) = n ⟶ M, rest @ ms ⊨ φ)
proof (induct n arbitrary:ms set:nat)
  case 0
  with assms
  show ?case by simp
next
  case (succ n)
  have (∀ rest∈list(M). length(rest) = succ(n) ⟶ P(rest,n)) ⟷
       (∀ t∈M. ∀ res∈list(M). length(res) = n ⟶ P(res @ [t],n))
    if n∈nat for n P
    using that last_init_eq by force
  from this[of _ λrest _. (M, rest @ ms ⊨ φ)] ‹n∈nat›
```

have $(\forall\, rest{\in}list(M).\ length(rest) = succ(n) \longrightarrow M,\ rest\ @\ ms \models \varphi) \longleftrightarrow$
$(\forall\, t{\in}M.\ \forall\, res{\in}list(M).\ length(res) = n \longrightarrow\ M,\ (res\ @\ [t])\ @\ ms \models \varphi)$
by *simp*
with *assms succ(1,3) succ(2)*[*of Cons*(_,*ms*)]
show *?case*
using *arity_sats_iff*[*of* φ _ *M Cons*(_, *ms* @ _)] *app_assoc*
by (*simp*)
qed

definition
sep_body_fm :: $i \Rightarrow i$ **where**
$sep\_body\_fm(p) \equiv (\cdot\forall\,(\cdot\exists\,(\cdot\forall\cdot\cdot 0 \in 1\cdot \leftrightarrow \cdot\cdot 0 \in 2\cdot \wedge incr\_bv1\hat{}2\ (p)\ \cdots)\cdot)\cdot)$

lemma *sep_body_fm_type* [*TC*]: $p \in formula \Longrightarrow sep\_body\_fm(p) \in formula$
by (*simp add*: *sep_body_fm_def*)

lemma *sats_sep_body_fm*:
assumes
$\varphi \in formula$ $ms{\in}list(M)$ $rest{\in}list(M)$
shows
$(M,\ rest\ @\ ms \models sep\_body\_fm(\varphi)) \longleftrightarrow$
$separation(\#\#M, \lambda x.\ M,\ [x]\ @\ rest\ @\ ms \models \varphi)$
using *assms formula_add_params1*[*of* _ *2* _ _ [_,_]]
unfolding *sep_body_fm_def separation_def* **by** *simp*

definition
ZF_separation_fm :: $i \Rightarrow i$ ($\langle\cdot Separation'(\_')\cdot\rangle$) **where**
$ZF\_separation\_fm(p) \equiv Forall\hat{}(pred(arity(p)))(sep\_body\_fm(p))$

lemma *ZF_separation_fm_type* [*TC*]: $p \in formula \Longrightarrow ZF\_separation\_fm(p) \in formula$
by (*simp add*: *ZF_separation_fm_def*)

lemma *sats_ZF_separation_fm_iff*:
assumes
$\varphi{\in}formula$
shows
$(M,\ [] \models \cdot Separation(\varphi)\cdot)$
$\longleftrightarrow$
$(\forall\, env{\in}list(M).\ arity(\varphi) \leq 1\ \#{+}\ length(env) \longrightarrow$
$separation(\#\#M, \lambda x.\ M,\ [x]\ @\ env \models \varphi))$
proof (*intro iffI ballI impI*)
let *?n*=*Arith.pred*($arity(\varphi)$)
fix *env*
assume $M,\ [] \models ZF\_separation\_fm(\varphi)$
assume $arity(\varphi) \leq 1\ \#{+}\ length(env)$ $env{\in}list(M)$
moreover from *this*
have $arity(\varphi) \leq succ(length(env))$ **by** *simp*
then

```
  obtain some rest where some∈list(M) rest∈list(M)
    env = some @ rest length(some) = Arith.pred(arity(φ))
    using list_split[OF ‹arity(φ) ≤ succ(_)› ‹env∈_›] by force
  moreover from ‹φ∈_›
  have arity(φ) ≤ succ(Arith.pred(arity(φ)))
   using succpred_leI by simp
  moreover
  note assms
  moreover
  assume M, [] ⊨ ZF_separation_fm(φ)
  moreover from calculation
  have M, some ⊨ sep_body_fm(φ)
    using sats_nForall[of sep_body_fm(φ) ?n]
    unfolding ZF_separation_fm_def by simp
  ultimately
  show separation(##M, λx. M, [x] @ env ⊨ φ)
    unfolding ZF_separation_fm_def
    using sats_sep_body_fm[of φ [] M some]
      arity_sats_iff[of φ rest M [_] @ some]
      separation_cong[of ##M λx. M, Cons(x, some @ rest) ⊨ φ _ ]
    by simp
next — almost equal to the previous implication
  let ?n=Arith.pred(arity(φ))
  assume asm:∀ env∈list(M). arity(φ) ≤ 1 #+ length(env) ⟶
    separation(##M, λx. M, [x] @ env ⊨ φ)
  {
    fix some
    assume some∈list(M) length(some) = Arith.pred(arity(φ))
    moreover
    note ‹φ∈_›
    moreover from calculation
    have arity(φ) ≤ 1 #+ length(some)
      using le_trans[OF succpred_leI] succpred_leI by simp
    moreover from calculation and asm
    have separation(##M, λx. M, [x] @ some ⊨ φ) by blast
    ultimately
    have M, some ⊨ sep_body_fm(φ)
    using sats_sep_body_fm[of φ [] M some]
      arity_sats_iff[of φ _ M [_,_] @ some]
     strong_replacement_cong[of ##M λx y. M, Cons(x, Cons(y, some @ _)) ⊨
φ _ ]
    by simp
  }
  with ‹φ∈_›
  show M, [] ⊨ ZF_separation_fm(φ)
    using sats_nForall[of sep_body_fm(φ) ?n]
    unfolding ZF_separation_fm_def
    by simp
qed
```

31.2 The Axiom of Replacement, internalized

schematic_goal *sats_univalent_fm_auto*:
assumes

Q_iff_sats:$\bigwedge x\ y\ z.\ x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow$
$Q(x,z) \longleftrightarrow (A, Cons(z, Cons(y, Cons(x, env))) \models Q1\_fm)$
$\bigwedge x\ y\ z.\ x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow$
$Q(x,y) \longleftrightarrow (A, Cons(z, Cons(y, Cons(x, env))) \models Q2\_fm)$
and
asms: $nth(i, env) = B\ i \in nat\ env \in list(A)$
shows
$univalent(\#\#A, B, Q) \longleftrightarrow A, env \models ?ufm(i)$
unfolding *univalent_def*
by (*insert asms*; (*rule sep_rules Q_iff_sats* | *simp*)+)

synthesize_notc *univalent* **from_schematic** *sats_univalent_fm_auto*

lemma *univalent_fm_type* [*TC*]: $q1 \in formula \Longrightarrow q2 \in formula \Longrightarrow i \in nat \Longrightarrow$
$univalent\_fm(q2, q1, i) \in formula$
by (*simp add*:*univalent_fm_def*)

lemma *sats_univalent_fm* :
assumes
Q_iff_sats:$\bigwedge x\ y\ z.\ x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow$
$Q(x,z) \longleftrightarrow (A, Cons(z, Cons(y, Cons(x, env))) \models Q1\_fm)$
$\bigwedge x\ y\ z.\ x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow$
$Q(x,y) \longleftrightarrow (A, Cons(z, Cons(y, Cons(x, env))) \models Q2\_fm)$
and
asms: $nth(i, env) = B\ i \in nat\ env \in list(A)$
shows
$(A, env \models univalent\_fm(Q1\_fm, Q2\_fm, i)) \longleftrightarrow univalent(\#\#A, B, Q)$
unfolding *univalent_fm_def* **using** *asms sats_univalent_fm_auto*[*OF Q_iff_sats*]
by *simp*

definition
$swap\_vars :: i \Rightarrow i$ **where**
$swap\_vars(\varphi) \equiv$
$Exists(Exists(And(Equal(0,3), And(Equal(1,2), iterates(\lambda p.\ incr\_bv(p)\text{`}2\ ,\ 2, \varphi)))))$

lemma *swap_vars_type*[*TC*] :
$\varphi \in formula \Longrightarrow swap\_vars(\varphi) \in formula$
unfolding *swap_vars_def* **by** *simp*

lemma *sats_swap_vars* :
$[x,y]\ @\ env \in list(M) \Longrightarrow \varphi \in formula \Longrightarrow$
$(M,\ [x,y]\ @\ env \models swap\_vars(\varphi)) \longleftrightarrow M, [y,x]\ @\ env \models \varphi$
unfolding *swap_vars_def*
using *sats_incr_bv_iff* [*of* _ _ *M* _ [*y*,*x*]] **by** *simp*

definition
univalent_Q1 :: $i \Rightarrow i$ **where**
$univalent\_Q1(\varphi) \equiv incr\_bv1(swap\_vars(\varphi))$

definition
univalent_Q2 :: $i \Rightarrow i$ **where**
$univalent\_Q2(\varphi) \equiv incr\_bv(swap\_vars(\varphi))`0$

lemma *univalent_Qs_type* [*TC*]:
assumes φ∈*formula*
shows $univalent\_Q1(\varphi) \in formula\ univalent\_Q2(\varphi) \in formula$
unfolding *univalent_Q1_def univalent_Q2_def* **using** *assms* **by** *simp_all*

lemma *sats_univalent_fm_assm*:
assumes
$x \in A\ y \in A\ z \in A\ env \in list(A)\ \varphi \in formula$
shows
$(A, ([x,z]\ @\ env) \models \varphi) \longleftrightarrow (A, Cons(z,Cons(y,Cons(x,env))) \models (univalent\_Q1(\varphi)))$
$(A, ([x,y]\ @\ env) \models \varphi) \longleftrightarrow (A, Cons(z,Cons(y,Cons(x,env))) \models (univalent\_Q2(\varphi)))$
unfolding *univalent_Q1_def univalent_Q2_def*
using
sats_incr_bv_iff[*of* _ _ *A* _ []] — simplifies iterates of $\lambda x.\ incr\_bv(x)\ `\ 0$
sats_incr_bv1_iff[*of* _ *Cons*(*x*,*env*) *A z y*]
sats_swap_vars assms
by *simp_all*

definition
rep_body_fm :: $i \Rightarrow i$ **where**
$rep\_body\_fm(p) \equiv Forall(Implies($
$univalent\_fm(univalent\_Q1(incr\_bv(p)`2),univalent\_Q2(incr\_bv(p)`2),0),$
$Exists(Forall($
$Iff(Member(0,1),Exists(And(Member(0,3),incr\_bv(incr\_bv(p)`2)`2)))))))$

lemma *rep_body_fm_type* [*TC*]: $p \in formula \Longrightarrow rep\_body\_fm(p) \in formula$
by (*simp add*: *rep_body_fm_def*)

lemmas *ZF_replacement_simps* = *formula_add_params1*[*of* φ *2* _ *M* [_,_]]
sats_incr_bv_iff[*of* _ _ *M* _ []] — simplifies iterates of $\lambda x.\ incr\_bv(x)\ `\ 0$
sats_incr_bv_iff[*of* _ _ *M* _ [_,_]]— simplifies $\lambda x.\ incr\_bv(x)\ `\ 2$
sats_incr_bv1_iff[*of* _ _ *M*] *sats_swap_vars* **for** φ *M*

lemma *sats_rep_body_fm*:
assumes
$\varphi \in formula\ ms \in list(M)\ rest \in list(M)$
shows
$(M, rest\ @\ ms \models rep\_body\_fm(\varphi)) \longleftrightarrow$
$strong\_replacement(\#\#M,\lambda x\ y.\ M, [x,y]\ @\ rest\ @\ ms \models \varphi)$
using *assms ZF_replacement_simps*

unfolding *rep_body_fm_def strong_replacement_def univalent_def*
unfolding *univalent_fm_def univalent_Q1_def univalent_Q2_def*
by *simp*

definition
ZF_replacement_fm :: $i \Rightarrow i$ ($\langle\cdot Replacement'(\_')\cdot\rangle$) **where**
$ZF\_replacement\_fm(p) \equiv Forall^\frown(pred(pred(arity(p))))(rep\_body\_fm(p))$

lemma *ZF_replacement_fm_type* [*TC*]: $p \in formula \Longrightarrow ZF\_replacement\_fm(p) \in formula$
by (*simp add*: *ZF_replacement_fm_def*)

lemma *sats_ZF_replacement_fm_iff*:
assumes
$\varphi \in formula$
shows
$(M, [] \models \cdot Replacement(\varphi)\cdot)$
$\longleftrightarrow$
$(\forall\, env \in list(M).\ arity(\varphi) \le 2\ \#+\ length(env) \longrightarrow$
$strong\_replacement(\#\#M, \lambda x\ y.\ M, [x,y]\ @\ env \models \varphi))$
proof (*intro iffI ballI impI*)
let *?n*=$Arith.pred(Arith.pred(arity(\varphi)))$
fix *env*
assume $M, [] \models ZF\_replacement\_fm(\varphi)$ $arity(\varphi) \le 2\ \#+\ length(env)$ $env \in list(M)$
moreover from *this*
have $arity(\varphi) \le succ(succ(length(env)))$ **by** (*simp*)
moreover from *calculation*
have $pred(arity(\varphi)) \le succ(length(env))$
using *pred_mono*[*OF* _ $\langle arity(\varphi) \le succ(\_)\rangle$] *pred_succ_eq* **by** *simp*
moreover from *calculation*
obtain *some rest* **where** $some \in list(M)$ $rest \in list(M)$
$env = some\ @\ rest$ $length(some) = Arith.pred(Arith.pred(arity(\varphi)))$
using *list_split*[*OF* $\langle pred(\_) \le \_\rangle$ $\langle env \in \_\rangle$] **by** *auto*
moreover
note $\langle \varphi \in \_\rangle$
moreover from *this*
have $arity(\varphi) \le succ(succ(Arith.pred(Arith.pred(arity(\varphi)))))$
using *le_trans*[*OF succpred_leI*] *succpred_leI* **by** *simp*
moreover from *calculation*
have $M, some \models rep\_body\_fm(\varphi)$
using *sats_nForall*[*of rep_body_fm*(φ) *?n*]
unfolding *ZF_replacement_fm_def*
by *simp*
ultimately
show $strong\_replacement(\#\#M, \lambda x\ y.\ M, [x, y]\ @\ env \models \varphi)$
using *sats_rep_body_fm*[*of* φ [] *M some*]
arity_sats_iff[*of* φ *rest M* [_,_] @ *some*]
strong_replacement_cong[*of* $\#\#M\ \lambda x\ y.\ M$, $Cons(x, Cons(y, some\ @\ rest)) \models \varphi\ \_$]

by *simp*
next — almost equal to the previous implication
let *?n=Arith.pred(Arith.pred(arity(φ)))*
assume *asm:∀ env∈list(M). arity(φ) ≤ 2 #+ length(env) ⟶*
strong_replacement(##M, λx y. M, [x, y] @ env ⊨ φ)
{
fix *some*
assume *some∈list(M) length(some) = Arith.pred(Arith.pred(arity(φ)))*
moreover
note ‹*φ∈_*›
moreover from *calculation*
have *arity(φ) ≤ 2 #+ length(some)*
using *le_trans[OF succpred_leI] succpred_leI* **by** *simp*
moreover from *calculation* **and** *asm*
have *strong_replacement(##M, λx y. M, [x, y] @ some ⊨ φ)* **by** *blast*
ultimately
have *M, some ⊨ rep_body_fm(φ)*
using *sats_rep_body_fm[of φ [] M some]*
arity_sats_iff[of φ _ M [_,_] @ some]
strong_replacement_cong[of ##M λx y. M, Cons(x, Cons(y, some @ _)) ⊨ φ _]
by *simp*
}
with ‹*φ∈_*›
show *M, [] ⊨ ZF_replacement_fm(φ)*
using *sats_nForall[of rep_body_fm(φ) ?n]*
unfolding *ZF_replacement_fm_def*
by *simp*
qed

definition
ZF_inf :: *i* **where**
ZF_inf ≡ {·*Separation*(*p*)· . *p* ∈ *formula* } ∪ {·*Replacement*(*p*)· . *p* ∈ *formula* }

lemma *Un_subset_formula*: *A⊆formula ∧ B⊆formula ⟹ A∪B ⊆ formula*
by *auto*

lemma *ZF_inf_subset_formula* : *ZF_inf ⊆ formula*
unfolding *ZF_inf_def* **by** *auto*

definition
ZFC :: *i* **where**
ZFC ≡ *ZF_inf ∪ ZFC_fin*

definition
ZF :: *i* **where**
ZF ≡ *ZF_inf ∪ ZF_fin*

definition

ZF_minus_P :: *i* **where**
ZF_minus_P ≡ *ZF* - { ·*Powerset Ax*· }

lemma *ZFC_subset_formula*: *ZFC* ⊆ *formula*
by (*simp add*:*ZFC_def Un_subset_formula ZF_inf_subset_formula ZFC_fin_type*)

Satisfaction of a set of sentences

definition
satT :: [*i*,*i*] ⇒ *o* (_ ⊨ _ [*36*,*36*] *60*) **where**
A ⊨ Φ ≡ ∀φ∈Φ. (*A*,[] ⊨ φ)

lemma *satTI* [*intro*!]:
assumes ⋀φ. φ∈Φ ⟹ *A*,[] ⊨ φ
shows *A* ⊨ Φ
using *assms* **unfolding** *satT_def* **by** *simp*

lemma *satTD* [*dest*] :*A* ⊨ Φ ⟹ φ∈Φ ⟹ *A*,[] ⊨ φ
unfolding *satT_def* **by** *simp*

lemma *sats_ZFC_iff_sats_ZF_AC*:
(*N* ⊨ *ZFC*) ⟷ (*N* ⊨ *ZF*) ∧ (*N*, [] ⊨ ·*AC*·)
unfolding *ZFC_def ZFC_fin_def ZF_def* **by** *auto*

lemma *M_ZF_iff_M_satT*: *M_ZF*(*M*) ⟷ (*M* ⊨ *ZF*)
proof
assume *M* ⊨ *ZF*
then
have *fin*: *upair_ax*(##*M*) *Union_ax*(##*M*) *power_ax*(##*M*)
extensionality(##*M*) *foundation_ax*(##*M*) *infinity_ax*(##*M*)
unfolding *ZF_def ZF_fin_def ZFC_fm_defs satT_def*
using *ZFC_fm_sats*[*of M*] **by** *simp_all*
{
fix φ *env*
assume φ ∈ *formula env*∈*list*(*M*)
moreover from ‹*M* ⊨ *ZF*›
have ∀ *p*∈*formula*. (*M*, [] ⊨ (*ZF_separation_fm*(*p*)))
∀ *p*∈*formula*. (*M*, [] ⊨ (*ZF_replacement_fm*(*p*)))
unfolding *ZF_def ZF_inf_def* **by** *auto*
moreover from *calculation*
have *arity*(φ) ≤ *succ*(*length*(*env*)) ⟹ *separation*(##*M*, λ*x*. (*M*, *Cons*(*x*, *env*)
⊨ φ))
arity(φ) ≤ *succ*(*succ*(*length*(*env*))) ⟹ *strong_replacement*(##*M*,λ*x y*.
sats(*M*,φ,*Cons*(*x*,*Cons*(*y*, *env*))))
using *sats_ZF_separation_fm_iff sats_ZF_replacement_fm_iff* **by** *simp_all*

}
with *fin*
show *M_ZF*(*M*)
unfolding *M_ZF_def* **by** *simp*

next
assume ‹*M_ZF(M)*›
then
have *M* ⊨ *ZF_fin*
unfolding *M_ZF_def ZF_fin_def ZFC_fm_defs satT_def*
using *ZFC_fm_sats*[*of M*] **by** *blast*
moreover from ‹*M_ZF(M)*›
have ∀ *p*∈*formula*. (*M*, [] ⊨ (*ZF_separation_fm*(*p*)))
∀ *p*∈*formula*. (*M*, [] ⊨ (*ZF_replacement_fm*(*p*)))
unfolding *M_ZF_def* **using** *sats_ZF_separation_fm_iff*
sats_ZF_replacement_fm_iff **by** *simp_all*
ultimately
show *M* ⊨ *ZF*
unfolding *ZF_def ZF_inf_def* **by** *blast*
qed

lemma *M_ZFC_iff_M_satT*:
notes *iff_trans*[*trans*]
shows *M_ZFC*(*M*) ⟷ (*M* ⊨ *ZFC*)
proof -
have *M_ZFC*(*M*) ⟷ (*M* ⊨ *ZF*) ∧ *choice_ax*(##*M*)
using *M_ZF_iff_M_satT* **unfolding** *M_ZFC_def M_ZFC_axioms_def* **by** *simp*
also
have ... ⟷ *M* ⊨ *ZFC*
unfolding *ZFC_def ZFC_fin_def ZF_def*
by *auto*
ultimately
show *?thesis* **by** *simp*
qed

end

32 The definition of *forces*

theory *Forces_Definition* **imports** *Arities FrecR Synthetic_Definition FrecR_Arities*
begin

This is the core of our development.

32.1 The relation *frecrel*

definition
frecrelP :: [*i*⇒*o*,*i*] ⇒ *o* **where**
frecrelP(*M*,*xy*) ≡ (∃ *x*[*M*]. ∃ *y*[*M*]. *pair*(*M*,*x*,*y*,*xy*) ∧ *is_frecR*(*M*,*x*,*y*))

synthesize *frecrelP* **from_definition**

lemma *arity_frecrelP_fm* :

$a{\in}nat \Longrightarrow arity(frecrelP\_fm(a)) = succ(a)$
unfolding *frecrelP_fm_def*
using *arity_frecR_fm arity_pair_fm pred_Un_distrib*
by *simp*

definition
is_frecrel :: $[i{\Rightarrow}o,i,i] \Rightarrow o$ **where**
$is\_frecrel(M,A,r) \equiv \exists\, A2[M].\ cartprod(M,A,A,A2) \wedge is\_Collect(M,A2,frecrelP(M),r)$

declare *cartprod_iff_sats* [*iff_sats*]
declare *Collect_iff_sats* [*iff_sats*]
synthesize *frecrel* **from_definition** *is_frecrel*

lemma *arity_frecrel_fm* :
assumes $a{\in}nat$ $b{\in}nat$
shows $arity(frecrel\_fm(a,b)) = succ(a) \cup succ(b)$
unfolding *frecrel_fm_def*
using *assms arity_Collect_fm arity_cartprod_fm arity_frecrelP_fm pred_Un_distrib*
by *auto*

definition
names_below :: $i \Rightarrow i \Rightarrow i$ **where**
$names\_below(P,x) \equiv 2{\times}ecloseN(x){\times}ecloseN(x){\times}P$

lemma *names_belowsD*:
assumes $x \in names\_below(P,z)$
obtains *f n1 n2 p* **where**
$x = \langle f,n1,n2,p\rangle$ $f{\in}2$ $n1{\in}ecloseN(z)$ $n2{\in}ecloseN(z)$ $p{\in}P$
using *assms* **unfolding** *names_below_def* **by** *auto*

synthesize *number2* **from_definition**

lemma *number2_iff* :
$(A)(c) \Longrightarrow number2(A,c) \longleftrightarrow (\exists\, b[A].\ \exists\, a[A].\ successor(A,\ b,\ c) \wedge successor(A,\ a,\ b) \wedge empty(A,\ a))$
unfolding *number2_def number1_def* **by** *auto*

lemma *arity_number2_fm* :
$a{\in}nat \Longrightarrow arity(number2\_fm(a)) = succ(a)$
unfolding *number2_fm_def*
using *arity_number1_fm arity_succ_fm union_abs2 pred_Un_distrib*
by *simp*

reldb_add *ecloseN is_ecloseN*
relativize *names_below is_names_below*
synthesize *is_names_below* **from_definition**

lemma *arity_is_names_below_fm* :

$[\![P{\in}nat;x{\in}nat;nb{\in}nat]\!] \Longrightarrow arity(is\_names\_below\_fm(P,x,nb)) = succ(P) \cup succ(x) \cup succ(nb)$
unfolding *is_names_below_fm_def*
using *arity_cartprod_fm arity_succ_fm arity_empty_fm arity_ecloseN_fm union_abs2 pred_Un_distrib*
by *auto*

definition
is_tuple :: $[i{\Rightarrow}o,i,i,i,i,i] \Rightarrow o$ **where**
$is\_tuple(M,z,t1,t2,p,t) \equiv \exists\, t1t2p[M].\ \exists\, t2p[M].\ pair(M,t2,p,t2p) \wedge pair(M,t1,t2p,t1t2p) \wedge$
$pair(M,z,t1t2p,t)$

synthesize *is_tuple* **from_definition**

lemma *arity_is_tuple_fm* : $[\![\ z{\in}nat\ ;\ t1{\in}nat\ ;\ t2{\in}nat\ ;\ p{\in}nat\ ;\ tup{\in}nat\ ]\!] \Longrightarrow$
$arity(is\_tuple\_fm(z,t1,t2,p,tup)) = \bigcup\ \{succ(z),succ(t1),succ(t2),succ(p),succ(tup)\}$
unfolding *is_tuple_fm_def*
using *arity_pair_fm union_abs1 union_abs2 pred_Un_distrib*
by *auto*

32.2 Definition of *forces* for equality and membership

definition
eq_case :: $[i,i,i,i,i,i] \Rightarrow o$ **where**
$eq\_case(t1,t2,p,P,leq,f) \equiv \forall\, s.\ s{\in}domain(t1) \cup domain(t2) \longrightarrow$
$(\forall\, q.\ q{\in}P \wedge \langle q,p\rangle{\in}leq \longrightarrow (f`\langle 1,s,t1,q\rangle{=}1 \longleftrightarrow f`\langle 1,s,t2,q\rangle = 1))$

relativize *eq_case is_eq_case*
synthesize *eq_case* **from_definition** *is_eq_case*

lemma *arity_eq_case_fm* :
assumes
n1∈nat n2∈nat p∈nat P∈nat leq∈nat f∈nat
shows
$arity(eq\_case\_fm(n1,n2,p,P,leq,f)) =$
$succ(n1) \cup succ(n2) \cup succ(p) \cup succ(P) \cup succ(leq) \cup succ(f)$
unfolding *eq_case_fm_def*
using *assms arity_pair_fm arity_is_tuple_fm arity_succ_fm arity_fun_apply_fm arity_empty_fm*
arity_domain_fm pred_Un_distrib arity_union_fm
by *auto*

definition
mem_case :: $[i,i,i,i,i,i] \Rightarrow o$ **where**
$mem\_case(t1,t2,p,P,leq,f) \equiv \forall\, v{\in}P.\ \langle v,p\rangle{\in}leq \longrightarrow$
$(\exists\, q.\ \exists\, s.\ \exists\, r.\ r{\in}P \wedge q{\in}P \wedge \langle q,v\rangle{\in}leq \wedge \langle s,r\rangle \in t2 \wedge \langle q,r\rangle{\in}leq \wedge\ f`\langle 0,t1,s,q\rangle = 1)$

relativize *mem_case is_mem_case*
synthesize *mem_case* **from_definition** *is_mem_case*

lemma *arity_mem_case_fm* :
assumes
n1∈nat n2∈nat p∈nat P∈nat leq∈nat f∈nat
shows
arity(mem_case_fm(n1,n2,p,P,leq,f)) =
succ(n1) ∪ succ(n2) ∪ succ(p) ∪ succ(P) ∪ succ(leq) ∪ succ(f)
unfolding *mem_case_fm_def*
using *assms arity_pair_fm arity_is_tuple_fm arity_succ_fm arity_fun_apply_fm arity_empty_fm*
pred_Un_distrib
by *auto*

definition
Hfrc :: $[i,i,i,i] \Rightarrow o$ **where**
$Hfrc(P,leq,fnnc,f) \equiv \exists ft.\ \exists n1.\ \exists n2.\ \exists c.\ c{\in}P \wedge fnnc = \langle ft,n1,n2,c\rangle \wedge$
$(\ ft = 0 \wedge\ eq\_case(n1,n2,c,P,leq,f)$
$\vee\ ft = 1 \wedge mem\_case(n1,n2,c,P,leq,f))$

relativize *Hfrc is_Hfrc*

synthesize *Hfrc* **from_definition** *is_Hfrc*

lemma *arity_Hfrc_fm* :
assumes
P∈nat leq∈nat fnnc∈nat f∈nat
shows
$arity(Hfrc\_fm(P,leq,fnnc,f)) = succ(P) \cup succ(leq) \cup succ(fnnc) \cup succ(f)$
unfolding *Hfrc_fm_def*
using *assms arity_pair_fm arity_mem_case_fm arity_eq_case_fm*
arity_empty_fm arity_succ_fm pred_Un_distrib
by *auto*

definition
is_Hfrc_at :: $[i{\Rightarrow}o,i,i,i,i,i] \Rightarrow o$ **where**
$is\_Hfrc\_at(M,P,leq,fnnc,f,z) \equiv$
$(empty(M,z) \wedge \neg\ is\_Hfrc(M,P,leq,fnnc,f))$
$\vee\ (number1(M,z) \wedge is\_Hfrc(M,P,leq,fnnc,f))$

synthesize *Hfrc_at* **from_definition** *is_Hfrc_at*

lemma *arity_Hfrc_at_fm* :
assumes
P∈nat leq∈nat fnnc∈nat f∈nat z∈nat
shows
$arity(Hfrc\_at\_fm(P,leq,fnnc,f,z)) = succ(P) \cup succ(leq) \cup succ(fnnc) \cup succ(f)$

$\cup$ *succ*(z)
unfolding *Hfrc_at_fm_def*
using *assms arity_Hfrc_fm arity_empty_fm arity_number1_fm pred_Un_distrib*
by *auto*

32.3 The well-founded relation *forcerel*

definition
forcerel :: $i \Rightarrow i \Rightarrow i$ **where**
forcerel(P,x) $\equiv$ *frecrel*(*names_below*(P,x))$^+$

definition
is_forcerel :: [i⇒o,i,i,i] ⇒ o **where**
is_forcerel(M,P,x,z) $\equiv \exists\, r[M].\ \exists\, nb[M].$ *tran_closure*(M,r,z) $\wedge$
(*is_names_below*(M,P,x,nb) $\wedge$ *is_frecrel*(M,nb,r))

definition
forcerel_fm :: i⇒ $i \Rightarrow i \Rightarrow i$ **where**
forcerel_fm(p,x,z) $\equiv$ *Exists*(*Exists*(*And*(*trans_closure_fm*(*1*, *z*#+*2*),
And(*is_names_below_fm*(*p*#+*2*,*x*#+*2*,*0*),*frecrel_fm*(*0*,*1*)))))

lemma *arity_forcerel_fm*:
⟦p∈*nat*;x∈*nat*;z∈*nat*⟧ ⟹ *arity*(*forcerel_fm*(p,x,z)) = *succ*(p) $\cup$ *succ*(x) $\cup$ *succ*(z)
unfolding *forcerel_fm_def*
using *arity_frecrel_fm arity_tran_closure_fm arity_is_names_below_fm pred_Un_distrib*
by *auto*

lemma *forcerel_fm_type*[*TC*]:
⟦p∈*nat*;x∈*nat*;z∈*nat*⟧ ⟹ *forcerel_fm*(p,x,z)∈*formula*
unfolding *forcerel_fm_def* **by** *simp*

lemma *sats_forcerel_fm*:
assumes
p∈*nat* x∈*nat* z∈*nat* *env*∈*list*(A)
shows
sats(A,*forcerel_fm*(p,x,z),*env*) ⟷ *is_forcerel*(##A,*nth*(p,*env*),*nth*(x, *env*),*nth*(z, *env*))
proof -
have *sats*(A,*trans_closure_fm*(*1*,z #+ *2*),*Cons*(nb,*Cons*(r,*env*))) ⟷
tran_closure(##A, r, *nth*(z, *env*)) **if** r∈A nb∈A **for** r nb
using *that assms trans_closure_iff_sats*[*of 1* [nb,r]@*env* _ *z*#+*2*,*symmetric*]
by *simp*
moreover
have *sats*(A, *is_names_below_fm*(*succ*(*succ*(p)), *succ*(*succ*(x)), *0*), *Cons*(nb, *Cons*(r, *env*))) ⟷
is_names_below(##A, *nth*(p,*env*), *nth*(x, *env*), nb)
if r∈A nb∈A **for** nb r
using *assms that sats_is_names_below_fm*[*of p #+ 2 x #+ 2 0* [nb,r]@*env*]

by *simp*
moreover
have *sats*(*A*, *frecrel_fm*(*0*, *1*), *Cons*(*nb*, *Cons*(*r*, *env*))) ⟷
is_frecrel(##*A*, *nb*, *r*)
if *r*∈*A nb*∈*A* **for** *r nb*
using *assms that sats_frecrel_fm*[*of 0 1* [*nb*,*r*]@*env*] **by** *simp*
ultimately
show *?thesis* **using** *assms* **unfolding** *is_forcerel_def forcerel_fm_def* **by** *simp*
qed

32.4 *frc_at*, forcing for atomic formulas

definition
frc_at :: [*i*,*i*,*i*] ⇒ *i* **where**
frc_at(*P*,*leq*,*fnnc*) ≡ *wfrec*(*frecrel*(*names_below*(*P*,*fnnc*)),*fnnc*,
λ*x f*. *bool_of_o*(*Hfrc*(*P*,*leq*,*x*,*f*)))

definition
is_frc_at :: [*i*⇒*o*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_frc_at(*M*,*P*,*leq*,*x*,*z*) ≡ ∃ *r*[*M*]. *is_forcerel*(*M*,*P*,*x*,*r*) ∧
is_wfrec(*M*,*is_Hfrc_at*(*M*,*P*,*leq*),*r*,*x*,*z*)

definition
frc_at_fm :: [*i*,*i*,*i*,*i*] ⇒ *i* **where**
frc_at_fm(*p*,*l*,*x*,*z*) ≡ *Exists*(*And*(*forcerel_fm*(*succ*(*p*),*succ*(*x*),*0*),
is_wfrec_fm(*Hfrc_at_fm*(*6*#+*p*,*6*#+*l*,*2*,*1*,*0*),*0*,*succ*(*x*),*succ*(*z*))))

lemma *frc_at_fm_type* [*TC*] :
⟦*p*∈*nat*;*l*∈*nat*;*x*∈*nat*;*z*∈*nat*⟧ ⟹ *frc_at_fm*(*p*,*l*,*x*,*z*)∈*formula*
unfolding *frc_at_fm_def* **by** *simp*

lemma *arity_frc_at_fm* :
assumes *p*∈*nat l*∈*nat x*∈*nat z*∈*nat*
shows *arity*(*frc_at_fm*(*p*,*l*,*x*,*z*)) = *succ*(*p*) ∪ *succ*(*l*) ∪ *succ*(*x*) ∪ *succ*(*z*)
proof -
let *?φ* = *Hfrc_at_fm*(*6* #+ *p*, *6* #+ *l*, *2*, *1*, *0*)
from *assms*
have *arity*(*?φ*) = (*7*#+*p*) ∪ (*7*#+*l*) *?φ* ∈ *formula*
using *arity_Hfrc_at_fm ord_simp_union*
by *auto*
with *assms*
have *W*: *arity*(*is_wfrec_fm*(*?φ*, *0*, *succ*(*x*), *succ*(*z*))) = *2*#+*p* ∪ (*2*#+*l*) ∪
(*2*#+*x*) ∪ (*2*#+*z*)
using *arity_is_wfrec_fm*[*OF* ‹*?φ*∈_› _ _ _ _ ‹*arity*(*?φ*) = _›] *pred_Un_distrib*
pred_succ_eq
union_abs1
by *auto*
from *assms*
have *arity*(*forcerel_fm*(*succ*(*p*),*succ*(*x*),*0*)) = *succ*(*succ*(*p*)) ∪ *succ*(*succ*(*x*))

using *arity_forcerel_fm ord_simp_union*
by *auto*
with *W assms*
show *?thesis*
unfolding *frc_at_fm_def*
using *arity_forcerel_fm pred_Un_distrib*
by *auto*
qed

lemma *sats_frc_at_fm* :
assumes
p∈nat l∈nat i∈nat j∈nat env∈list(A) i < length(env) j < length(env)
shows
sats(A,frc_at_fm(p,l,i,j),env) ⟷
is_frc_at(##A,nth(p,env),nth(l,env),nth(i,env),nth(j,env))
proof -
{
fix *r pp ll*
assume *r∈A*
have *0:is_Hfrc_at(##A,nth(p,env),nth(l,env),a2, a1, a0) ⟷*
sats(A, Hfrc_at_fm(6#+p,6#+l,2,1,0), [a0,a1,a2,a3,a4,r]@env)
if *a0∈A a1∈A a2∈A a3∈A a4∈A* **for** *a0 a1 a2 a3 a4*
using *that assms ‹r∈A›*
Hfrc_at_iff_sats[of 6#+p 6#+l 2 1 0 [a0,a1,a2,a3,a4,r]@env A] **by** *simp*
have *sats(A,is_wfrec_fm(Hfrc_at_fm(6 #+ p, 6 #+ l, 2, 1, 0), 0, succ(i), succ(j)),[r]@env) ⟷*
is_wfrec(##A, is_Hfrc_at(##A, nth(p,env), nth(l,env)), r,nth(i, env), nth(j, env))
using *assms ‹r∈A›*
sats_is_wfrec_fm[OF 0[simplified]]
by *simp*
}
moreover
have *sats(A, forcerel_fm(succ(p), succ(i), 0), Cons(r, env)) ⟷*
is_forcerel(##A,nth(p,env),nth(i,env),r) **if** *r∈A* **for** *r*
using *assms sats_forcerel_fm that* **by** *simp*
ultimately
show *?thesis* **unfolding** *is_frc_at_def frc_at_fm_def*
using *assms* **by** *simp*
qed

definition
forces_eq′ :: *[i,i,i,i,i] ⇒ o* **where**
forces_eq′(P,l,p,t1,t2) ≡ frc_at(P,l,⟨0,t1,t2,p⟩) = 1

definition
forces_mem′ :: *[i,i,i,i,i] ⇒ o* **where**
forces_mem′(P,l,p,t1,t2) ≡ frc_at(P,l,⟨1,t1,t2,p⟩) = 1

definition
forces_neq′ :: [*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
forces_neq′(*P*,*l*,*p*,*t1*,*t2*) ≡ ¬ (∃ *q*∈*P*. ⟨*q*,*p*⟩∈*l* ∧ *forces_eq′*(*P*,*l*,*q*,*t1*,*t2*))

definition
forces_nmem′ :: [*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
forces_nmem′(*P*,*l*,*p*,*t1*,*t2*) ≡ ¬ (∃ *q*∈*P*. ⟨*q*,*p*⟩∈*l* ∧ *forces_mem′*(*P*,*l*,*q*,*t1*,*t2*))

definition
is_forces_eq′ :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_forces_eq′(*M*,*P*,*l*,*p*,*t1*,*t2*) ≡ ∃ *o*[*M*]. ∃ *z*[*M*]. ∃ *t*[*M*]. *number1*(*M*,*o*) ∧ *empty*(*M*,*z*)
∧
is_tuple(*M*,*z*,*t1*,*t2*,*p*,*t*) ∧ *is_frc_at*(*M*,*P*,*l*,*t*,*o*)

definition
is_forces_mem′ :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_forces_mem′(*M*,*P*,*l*,*p*,*t1*,*t2*) ≡ ∃ *o*[*M*]. ∃ *t*[*M*]. *number1*(*M*,*o*) ∧
is_tuple(*M*,*o*,*t1*,*t2*,*p*,*t*) ∧ *is_frc_at*(*M*,*P*,*l*,*t*,*o*)

definition
is_forces_neq′ :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_forces_neq′(*M*,*P*,*l*,*p*,*t1*,*t2*) ≡
¬ (∃ *q*[*M*]. *q*∈*P* ∧ (∃ *qp*[*M*]. *pair*(*M*,*q*,*p*,*qp*) ∧ *qp*∈*l* ∧ *is_forces_eq′*(*M*,*P*,*l*,*q*,*t1*,*t2*)))

definition
is_forces_nmem′ :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_forces_nmem′(*M*,*P*,*l*,*p*,*t1*,*t2*) ≡
¬ (∃ *q*[*M*]. ∃ *qp*[*M*]. *q*∈*P* ∧ *pair*(*M*,*q*,*p*,*qp*) ∧ *qp*∈*l* ∧ *is_forces_mem′*(*M*,*P*,*l*,*q*,*t1*,*t2*))

definition
forces_eq_fm :: [*i*,*i*,*i*,*i*,*i*] ⇒ *i* **where**
forces_eq_fm(*p*,*l*,*q*,*t1*,*t2*) ≡
Exists(*Exists*(*Exists*(*And*(*number1_fm*(*2*),*And*(*empty_fm*(*1*),
And(*is_tuple_fm*(*1*,*t1*#+*3*,*t2*#+*3*,*q*#+*3*,*0*),*frc_at_fm*(*p*#+*3*,*l*#+*3*,*0*,*2*)
))))))

definition
forces_mem_fm :: [*i*,*i*,*i*,*i*,*i*] ⇒ *i* **where**
forces_mem_fm(*p*,*l*,*q*,*t1*,*t2*) ≡ *Exists*(*Exists*(*And*(*number1_fm*(*1*),
And(*is_tuple_fm*(*1*,*t1*#+*2*,*t2*#+*2*,*q*#+*2*,*0*),*frc_at_fm*(*p*#+*2*,*l*#+*2*,*0*,*1*)))))

definition
forces_neq_fm :: [*i*,*i*,*i*,*i*,*i*] ⇒ *i* **where**
forces_neq_fm(*p*,*l*,*q*,*t1*,*t2*) ≡ *Neg*(*Exists*(*Exists*(*And*(*Member*(*1*,*p*#+*2*),
And(*pair_fm*(*1*,*q*#+*2*,*0*),*And*(*Member*(*0*,*l*#+*2*),*forces_eq_fm*(*p*#+*2*,*l*#+*2*,*1*,*t1*#+*2*,*t2*#+*2*)))))))

definition
forces_nmem_fm :: [*i*,*i*,*i*,*i*,*i*] ⇒ *i* **where**
forces_nmem_fm(*p*,*l*,*q*,*t1*,*t2*) ≡ *Neg*(*Exists*(*Exists*(*And*(*Member*(*1*,*p*#+*2*),

$And(pair\_fm(1,q\#+2,0),And(Member(0,l\#+2),forces\_mem\_fm(p\#+2,l\#+2,1,t1\#+2,t2\#+2)))))))$

lemma *forces_eq_fm_type* [*TC*]:
  ⟦ *p*∈*nat*;*l*∈*nat*;*q*∈*nat*;*t1*∈*nat*;*t2*∈*nat*⟧ ⟹ *forces_eq_fm*(*p*,*l*,*q*,*t1*,*t2*) ∈ *formula*
  unfolding *forces_eq_fm_def*
  by *simp*

lemma *forces_mem_fm_type* [*TC*]:
  ⟦ *p*∈*nat*;*l*∈*nat*;*q*∈*nat*;*t1*∈*nat*;*t2*∈*nat*⟧ ⟹ *forces_mem_fm*(*p*,*l*,*q*,*t1*,*t2*) ∈ *formula*
  unfolding *forces_mem_fm_def*
  by *simp*

lemma *forces_neq_fm_type* [*TC*]:
  ⟦ *p*∈*nat*;*l*∈*nat*;*q*∈*nat*;*t1*∈*nat*;*t2*∈*nat*⟧ ⟹ *forces_neq_fm*(*p*,*l*,*q*,*t1*,*t2*) ∈ *formula*
  unfolding *forces_neq_fm_def*
  by *simp*

lemma *forces_nmem_fm_type* [*TC*]:
  ⟦ *p*∈*nat*;*l*∈*nat*;*q*∈*nat*;*t1*∈*nat*;*t2*∈*nat*⟧ ⟹ *forces_nmem_fm*(*p*,*l*,*q*,*t1*,*t2*) ∈ *formula*
  unfolding *forces_nmem_fm_def*
  by *simp*

lemma *arity_forces_eq_fm* :
  p∈*nat* ⟹ *l*∈*nat* ⟹ *q*∈*nat* ⟹ *t1* ∈ *nat* ⟹ *t2* ∈ *nat* ⟹
    arity(*forces_eq_fm*(*p*,*l*,*q*,*t1*,*t2*)) = *succ*(*t1*) ∪ *succ*(*t2*) ∪ *succ*(*q*) ∪ *succ*(*p*) ∪ *succ*(*l*)
  unfolding *forces_eq_fm_def*
  using *arity_number1_fm arity_empty_fm arity_is_tuple_fm arity_frc_at_fm*
    pred_Un_distrib
  by *auto*

lemma *arity_forces_mem_fm* :
  p∈*nat* ⟹ *l*∈*nat* ⟹ *q*∈*nat* ⟹ *t1* ∈ *nat* ⟹ *t2* ∈ *nat* ⟹
   arity(*forces_mem_fm*(*p*,*l*,*q*,*t1*,*t2*)) = *succ*(*t1*) ∪ *succ*(*t2*) ∪ *succ*(*q*) ∪ *succ*(*p*) ∪ *succ*(*l*)
  unfolding *forces_mem_fm_def*
  using *arity_number1_fm arity_empty_fm arity_is_tuple_fm arity_frc_at_fm*
    pred_Un_distrib
  by *auto*

lemma *sats_forces_eq′_fm*:
  assumes *p*∈*nat l*∈*nat q*∈*nat t1*∈*nat t2*∈*nat env*∈*list*(*M*)
  shows *sats*(*M*,*forces_eq_fm*(*p*,*l*,*q*,*t1*,*t2*),*env*) ⟷
      is_forces_eq′(##*M*,*nth*(*p*,*env*),*nth*(*l*,*env*),*nth*(*q*,*env*),*nth*(*t1*,*env*),*nth*(*t2*,*env*))
  unfolding *forces_eq_fm_def is_forces_eq′_def* **using** *assms sats_is_tuple_fm sats_frc_at_fm*
  by *simp*

lemma *sats_forces_mem'_fm*:
assumes *p∈nat l∈nat q∈nat t1∈nat t2∈nat env∈list(M)*
shows *sats(M,forces_mem_fm(p,l,q,t1,t2),env) ⟷*
is_forces_mem'(##M,nth(p,env),nth(l,env),nth(q,env),nth(t1,env),nth(t2,env))
unfolding *forces_mem_fm_def is_forces_mem'_def* **using** *assms sats_is_tuple_fm sats_frc_at_fm*
by *simp*

lemma *sats_forces_neq'_fm*:
assumes *p∈nat l∈nat q∈nat t1∈nat t2∈nat env∈list(M)*
shows *sats(M,forces_neq_fm(p,l,q,t1,t2),env) ⟷*
is_forces_neq'(##M,nth(p,env),nth(l,env),nth(q,env),nth(t1,env),nth(t2,env))
unfolding *forces_neq_fm_def is_forces_neq'_def*
using *assms sats_forces_eq'_fm sats_is_tuple_fm sats_frc_at_fm*
by *simp*

lemma *sats_forces_nmem'_fm*:
assumes *p∈nat l∈nat q∈nat t1∈nat t2∈nat env∈list(M)*
shows *sats(M,forces_nmem_fm(p,l,q,t1,t2),env) ⟷*
is_forces_nmem'(##M,nth(p,env),nth(l,env),nth(q,env),nth(t1,env),nth(t2,env))
unfolding *forces_nmem_fm_def is_forces_nmem'_def*
using *assms sats_forces_mem'_fm sats_is_tuple_fm sats_frc_at_fm*
by *simp*

context *forcing_data*
begin

lemma *ftype_abs*:
⟦*x∈M*; *y∈M* ⟧ ⟹ *is_ftype(##M,x,y) ⟷ y = ftype(x)*
unfolding *ftype_def is_ftype_def* **by** (*simp add*:*absolut*)

lemma *name1_abs*:
⟦*x∈M*; *y∈M* ⟧ ⟹ *is_name1(##M,x,y) ⟷ y = name1(x)*
unfolding *name1_def is_name1_def*
by (*rule is_hcomp_abs*[*OF fst_abs*],*simp_all add*: *fst_snd_closed*[*simplified*] *absolut*)

lemma *snd_snd_abs*:
⟦*x∈M*; *y∈M* ⟧ ⟹ *is_snd_snd(##M,x,y) ⟷ y = snd(snd(x))*
unfolding *is_snd_snd_def*
by (*rule is_hcomp_abs*[*OF snd_abs*],
simp_all add: *conjunct2*[*OF fst_snd_closed*,*simplified*] *absolut*)

lemma *name2_abs*:
⟦*x∈M*; *y∈M* ⟧ ⟹ *is_name2(##M,x,y) ⟷ y = name2(x)*
unfolding *name2_def is_name2_def*
by (*rule is_hcomp_abs*[*OF fst_abs snd_snd_abs*],*simp_all add*:*absolut conjunct2*[*OF fst_snd_closed*,*simplified*])

lemma *cond_of_abs*:
 $[\![x{\in}M;\ y{\in}M\ ]\!] \Longrightarrow is\_cond\_of(\#\#M,x,y) \longleftrightarrow y = cond\_of(x)$
 unfolding *cond_of_def is_cond_of_def*
 by (*rule is_hcomp_abs*[*OF snd_abs snd_snd_abs*];*simp_all add*:*fst_snd_closed*[*simplified*])

lemma *tuple_abs*:
 $[\![z{\in}M;t1{\in}M;t2{\in}M;p{\in}M;t{\in}M]\!] \Longrightarrow$
  $is\_tuple(\#\#M,z,t1,t2,p,t) \longleftrightarrow t = \langle z,t1,t2,p\rangle$
 unfolding *is_tuple_def* **using** *pair_in_M_iff* **by** *simp*

lemmas *components_abs* = *ftype_abs name1_abs name2_abs cond_of_abs*
 tuple_abs

lemma *oneN_in_M*[*simp*]: $1{\in}M$
 by (*simp flip*: *setclass_iff*)

lemma *twoN_in_M* : $2{\in}M$
 by (*simp flip*: *setclass_iff*)

lemma *comp_in_M*:
 $p \preceq q \Longrightarrow p{\in}M$
 $p \preceq q \Longrightarrow q{\in}M$
 using *leq_in_M transitivity*[*of _ leq*] *pair_in_M_iff* **by** *auto*

lemma *eq_case_abs* [*simp*]:
 assumes
  $t1{\in}M\ t2{\in}M\ p{\in}M\ f{\in}M$
 shows
  $is\_eq\_case(\#\#M,t1,t2,p,P,leq,f) \longleftrightarrow eq\_case(t1,t2,p,P,leq,f)$
proof -
 have $q \preceq p \Longrightarrow q{\in}M$ **for** q
  using *comp_in_M* **by** *simp*
 moreover
 have $\langle s,y\rangle{\in}t \Longrightarrow s{\in}domain(t)$ **if** $t{\in}M$ **for** $s\ y\ t$
  using *that* **unfolding** *domain_def* **by** *auto*
 ultimately
 have
  $(\forall\, s{\in}M.\ s \in domain(t1) \vee s \in domain(t2) \longrightarrow (\forall\, q{\in}M.\ q{\in}P \wedge q \preceq p \longrightarrow$
  $(f\ `\ \langle 1,\ s,\ t1,\ q\rangle = 1 \longleftrightarrow f\ `\ \langle 1,\ s,\ t2,\ q\rangle = 1))) \longleftrightarrow$
  $(\forall\, s.\ s \in domain(t1) \vee s \in domain(t2) \longrightarrow (\forall\, q.\ q{\in}P \wedge q \preceq p \longrightarrow$
  $(f\ `\ \langle 1,\ s,\ t1,\ q\rangle = 1 \longleftrightarrow f\ `\ \langle 1,\ s,\ t2,\ q\rangle = 1)))$
  using *assms domain_trans*[*OF trans_M*,*of t1*]
   domain_trans[*OF trans_M*,*of t2*] **by** *auto*
 then show *?thesis*
  unfolding *eq_case_def is_eq_case_def*
  using *assms pair_in_M_iff nat_into_M*[*of 1*] *domain_closed*

```
    apply_closed leq_in_M nonempty Un_closed
  by (simp add:components_abs)
qed

lemma mem_case_abs [simp]:
 assumes
  t1∈M t2∈M p∈M f∈M
 shows
  is_mem_case(##M,t1,t2,p,P,leq,f) ⟷ mem_case(t1,t2,p,P,leq,f)
proof
 {
  fix v
  assume v∈P v ≼ p is_mem_case(##M,t1,t2,p,P,leq,f)
  moreover
  from this
  have v∈M ⟨v,p⟩ ∈ M (##M)(v)
   using transitivity[OF _ P_in_M,of v] transitivity[OF _ leq_in_M]
   by simp_all
  moreover
  from calculation assms
  obtain q r s where
    r ∈ P ∧ q ∈ P ∧ ⟨q, v⟩ ∈ M ∧ ⟨s, r⟩ ∈ M ∧ ⟨q, r⟩ ∈ M ∧ 0 ∈ M ∧
     ⟨0, t1, s, q⟩ ∈ M ∧ q ≼ v ∧ ⟨s, r⟩ ∈ t2 ∧ q ≼ r ∧ f ‘ ⟨0, t1, s, q⟩ = 1
   unfolding is_mem_case_def by (auto simp add:components_abs)
  then
  have ∃ q s r. r ∈ P ∧ q ∈ P ∧ q ≼ v ∧ ⟨s, r⟩ ∈ t2 ∧ q ≼ r ∧ f ‘ ⟨0, t1, s, q⟩
= 1
   by auto
 }
 then
 show mem_case(t1, t2, p, P, leq, f) if is_mem_case(##M, t1, t2, p, P, leq, f)
  unfolding mem_case_def using that assms by auto
next
 { fix v
  assume v ∈ M v ∈ P ⟨v, p⟩ ∈ M v ≼ p mem_case(t1, t2, p, P, leq, f)
  moreover
  from this
  obtain q s r where r ∈ P ∧ q ∈ P ∧ q ≼ v ∧ ⟨s, r⟩ ∈ t2 ∧ q ≼ r ∧ f ‘ ⟨0,
t1, s, q⟩ = 1
   unfolding mem_case_def by auto
  moreover
  from this ‹t2∈M›
  have r∈M q∈M s∈M r ∈ P ∧ q ∈ P ∧ q ≼ v ∧ ⟨s, r⟩ ∈ t2 ∧ q ≼ r ∧ f ‘ ⟨0,
t1, s, q⟩ = 1
   using transitivity P_in_M domain_closed[of t2] by auto
  moreover
  note ‹t1∈M›
  ultimately
  have ∃ q∈M . ∃ s∈M. ∃ r∈M.
```

$r \in P \wedge q \in P \wedge \langle q, v\rangle \in M \wedge \langle s, r\rangle \in M \wedge \langle q, r\rangle \in M \wedge 0 \in M \wedge$
$\langle 0, t1, s, q\rangle \in M \wedge q \preceq v \wedge \langle s, r\rangle \in t2 \wedge q \preceq r \wedge f\ `\ \langle 0, t1, s, q\rangle = 1$
using *pair_in_M_iff zero_in_M* **by** *auto*
}
then
show *is_mem_case*(*##M*, *t1*, *t2*, *p*, *P*, *leq*, *f*) **if** *mem_case*(*t1*, *t2*, *p*, *P*, *leq*, *f*)
unfolding *is_mem_case_def*
using *assms that nonempty pair_in_M_iff apply_closed*
by (*auto simp add*:*components_abs*)
qed

lemma *Hfrc_abs*:
⟦*fnnc*∈*M*; *f*∈*M*⟧ ⟹
is_Hfrc(*##M*,*P*,*leq*,*fnnc*,*f*) ⟷ *Hfrc*(*P*,*leq*,*fnnc*,*f*)
unfolding *is_Hfrc_def Hfrc_def* **using** *pair_in_M_iff nonempty*
by (*auto simp add*:*components_abs*)

lemma *Hfrc_at_abs*:
⟦*fnnc*∈*M*; *f*∈*M* ; *z*∈*M*⟧ ⟹
is_Hfrc_at(*##M*,*P*,*leq*,*fnnc*,*f*,*z*) ⟷ *z* = *bool_of_o*(*Hfrc*(*P*,*leq*,*fnnc*,*f*))
unfolding *is_Hfrc_at_def* **using** *Hfrc_abs*
by *auto*

lemma *components_closed* :
x∈*M* ⟹ (*##M*)(*ftype*(*x*))
x∈*M* ⟹ (*##M*)(*name1*(*x*))
x∈*M* ⟹ (*##M*)(*name2*(*x*))
x∈*M* ⟹ (*##M*)(*cond_of*(*x*))
unfolding *ftype_def name1_def name2_def cond_of_def* **using** *fst_snd_closed*
by *simp_all*

lemma *ecloseN_closed*:
(*##M*)(*A*) ⟹ (*##M*)(*ecloseN*(*A*))
(*##M*)(*A*) ⟹ (*##M*)(*eclose_n*(*name1*,*A*))
(*##M*)(*A*) ⟹ (*##M*)(*eclose_n*(*name2*,*A*))
unfolding *ecloseN_def eclose_n_def*
using *components_closed eclose_closed singleton_closed Un_closed* **by** *auto*

lemma *eclose_n_abs* :
assumes *x*∈*M ec*∈*M*
shows *is_eclose_n*(*##M*,*is_name1*,*ec*,*x*) ⟷ *ec* = *eclose_n*(*name1*,*x*)
is_eclose_n(*##M*,*is_name2*,*ec*,*x*) ⟷ *ec* = *eclose_n*(*name2*,*x*)
unfolding *is_eclose_n_def eclose_n_def*
using *assms name1_abs name2_abs eclose_abs singleton_closed components_closed*
by *auto*

lemma *ecloseN_abs* :

$[\![x{\in}M; ec{\in}M]\!] \Longrightarrow is\_ecloseN(\#\#M,x,ec) \longleftrightarrow ec = ecloseN(x)$
unfolding *is_ecloseN_def ecloseN_def*
using *eclose_n_abs Un_closed union_abs ecloseN_closed*
by *auto*

lemma *frecR_abs* :
$x{\in}M \Longrightarrow y{\in}M \Longrightarrow frecR(x,y) \longleftrightarrow is\_frecR(\#\#M,x,y)$
unfolding *frecR_def is_frecR_def*
using *nonempty domain_closed Un_closed components_closed*
by (*auto simp add*: *components_abs*)

lemma *frecrelP_abs* :
$z{\in}M \Longrightarrow frecrelP(\#\#M,z) \longleftrightarrow (\exists\, x\ y.\ z = \langle x,y\rangle \wedge frecR(x,y))$
using *pair_in_M_iff frecR_abs* **unfolding** *frecrelP_def* **by** *auto*

lemma *frecrel_abs*:
assumes
$A{\in}M\ r{\in}M$
shows
$is\_frecrel(\#\#M,A,r) \longleftrightarrow\ r = frecrel(A)$
proof -
from ‹$A{\in}M$›
have $z{\in}M$ **if** $z{\in}A{\times}A$ **for** z
using *cartprod_closed transitivity that* **by** *simp*
then
have $Collect(A{\times}A,frecrelP(\#\#M)) = Collect(A{\times}A,\lambda z.\ (\exists\, x\ y.\ z = \langle x,y\rangle \wedge frecR(x,y)))$
using *Collect_cong*[*of* $A{\times}A$ $A{\times}A$ $frecrelP(\#\#M)$] *assms frecrelP_abs* **by** *simp*
with *assms*
show *?thesis* **unfolding** *is_frecrel_def def_frecrel* **using** *cartprod_closed*
by *simp*
qed

lemma *frecrel_closed*:
assumes
$x{\in}M$
shows
$frecrel(x){\in}M$
proof -
have $Collect(x{\times}x,\lambda z.\ (\exists\, x\ y.\ z = \langle x,y\rangle \wedge frecR(x,y))){\in}M$
using *Collect_in_M*[*of frecrelP_fm*(*0*) []] *arity_frecrelP_fm sats_frecrelP_fm*
frecrelP_abs ‹$x{\in}M$› *cartprod_closed* **by** *simp*
then show *?thesis*
unfolding *frecrel_def Rrel_def frecrelP_def* **by** *simp*
qed

lemma *field_frecrel* : $field(frecrel(names\_below(P,x))) \subseteq names\_below(P,x)$
unfolding *frecrel_def*
using *field_Rrel* **by** *simp*

lemma *forcerelD* : *uv* ∈ *forcerel*(*P*,*x*) ⟹ *uv*∈ *names_below*(*P*,*x*) × *names_below*(*P*,*x*)
unfolding *forcerel_def*
using *trancl_type field_frecrel* **by** *blast*

lemma *wf_forcerel* :
wf(*forcerel*(*P*,*x*))
unfolding *forcerel_def* **using** *wf_trancl wf_frecrel* **.**

lemma *restrict_trancl_forcerel*:
assumes *frecR*(*w*,*y*)
shows *restrict*(*f*,*frecrel*(*names_below*(*P*,*x*))-‘‘{*y*})‘*w*
= *restrict*(*f*,*forcerel*(*P*,*x*)-‘‘{*y*})‘*w*
unfolding *forcerel_def frecrel_def* **using** *assms restrict_trancl_Rrel*[*of frecR*]
by *simp*

lemma *names_belowI* :
assumes *frecR*(⟨*ft*,*n1*,*n2*,*p*⟩,⟨*a*,*b*,*c*,*d*⟩) *p*∈*P*
shows ⟨*ft*,*n1*,*n2*,*p*⟩ ∈ *names_below*(*P*,⟨*a*,*b*,*c*,*d*⟩) (**is** *?x* ∈ *names_below*(_,*?y*))
proof -
from *assms*
have *ft* ∈ *2 a* ∈ *2*
unfolding *frecR_def* **by** (*auto simp add*:*components_simp*)
from *assms*
consider (*e*) *n1* ∈ *domain*(*b*) ∪ *domain*(*c*) ∧ (*n2* = *b* ∨ *n2* =*c*)
| (*m*) *n1* = *b* ∧ *n2* ∈ *domain*(*c*)
unfolding *frecR_def* **by** (*auto simp add*:*components_simp*)
then show *?thesis*
proof *cases*
case *e*
then
have *n1* ∈ *eclose*(*b*) ∨ *n1* ∈ *eclose*(*c*)
using *Un_iff in_dom_in_eclose* **by** *auto*
with *e*
have *n1* ∈ *ecloseN*(*?y*) *n2* ∈ *ecloseN*(*?y*)
using *ecloseNI components_in_eclose* **by** *auto*
with ‹*ft*∈*2*› ‹*p*∈*P*›
show *?thesis* **unfolding** *names_below_def* **by** *auto*
next
case *m*
then
have *n1* ∈ *ecloseN*(*?y*) *n2* ∈ *ecloseN*(*?y*)
using *mem_eclose_trans ecloseNI*
in_dom_in_eclose components_in_eclose **by** *auto*
with ‹*ft*∈*2*› ‹*p*∈*P*›
show *?thesis* **unfolding** *names_below_def*
by *auto*
qed
qed

```
lemma names_below_tr :
  assumes x∈ names_below(P,y)
    y∈ names_below(P,z)
  shows x∈ names_below(P,z)
proof -
  let ?A=λy . names_below(P,y)
  from assms
  obtain fx x1 x2 px where
    x = ⟨fx,x1,x2,px⟩ fx∈2 x1∈ecloseN(y) x2∈ecloseN(y) px∈P
    unfolding names_below_def by auto
  from assms
  obtain fy y1 y2 py where
    y = ⟨fy,y1,y2,py⟩ fy∈2 y1∈ecloseN(z) y2∈ecloseN(z) py∈P
    unfolding names_below_def by auto
  from ‹x1∈_› ‹x2∈_› ‹y1∈_› ‹y2∈_› ‹x=_› ‹y=_›
  have x1∈ecloseN(z) x2∈ecloseN(z)
    using ecloseN_mono names_simp by auto
  with ‹fx∈2› ‹px∈P› ‹x=_›
  have x∈?A(z)
    unfolding names_below_def by simp
  then show ?thesis using subsetI by simp
qed

lemma arg_into_names_below2 :
  assumes ⟨x,y⟩ ∈ frecrel(names_below(P,z))
  shows  x ∈ names_below(P,y)
proof -
  {
    from assms
    have x∈names_below(P,z) y∈names_below(P,z) frecR(x,y)
      unfolding frecrel_def Rrel_def
      by auto
    obtain f n1 n2 p where
      x = ⟨f,n1,n2,p⟩ f∈2 n1∈ecloseN(z) n2∈ecloseN(z) p∈P
      using ‹x∈names_below(P,z)›
      unfolding names_below_def by auto
    moreover
    obtain fy m1 m2 q where
      q∈P y = ⟨fy,m1,m2,q⟩
      using ‹y∈names_below(P,z)›
      unfolding names_below_def by auto
    moreover
    note ‹frecR(x,y)›
    ultimately
    have x∈names_below(P,y) using names_belowI by simp
  }
  then show ?thesis .
qed
```

lemma *arg_into_names_below* :
assumes $\langle x,y\rangle \in$ *frecrel*(*names_below*(*P*,*z*))
shows $x \in$ *names_below*(*P*,*x*)
proof -
{
from *assms*
have *x*∈*names_below*(*P*,*z*)
unfolding *frecrel_def Rrel_def*
by *auto*
from ‹*x*∈*names_below*(*P*,*z*)›
obtain *f n1 n2 p* **where**
$x = \langle f,n1,n2,p\rangle$ *f*∈*2 n1*∈*ecloseN*(*z*) *n2*∈*ecloseN*(*z*) *p*∈*P*
unfolding *names_below_def* **by** *auto*
then
have *n1*∈*ecloseN*(*x*) *n2*∈*ecloseN*(*x*)
using *components_in_eclose* **by** *simp_all*
with ‹*f*∈*2*› ‹*p*∈*P*› ‹$x = \langle f,n1,n2,p\rangle$›
have *x*∈*names_below*(*P*,*x*)
unfolding *names_below_def* **by** *simp*
}
then show *?thesis* **.**
qed

lemma *forcerel_arg_into_names_below* :
assumes $\langle x,y\rangle \in$ *forcerel*(*P*,*z*)
shows $x \in$ *names_below*(*P*,*x*)
using *assms*
unfolding *forcerel_def*
by(*rule trancl_induct*;*auto simp add*: *arg_into_names_below*)

lemma *names_below_mono* :
assumes $\langle x,y\rangle \in$ *frecrel*(*names_below*(*P*,*z*))
shows *names_below*(*P*,*x*) ⊆ *names_below*(*P*,*y*)
proof -
from *assms*
have *x*∈*names_below*(*P*,*y*)
using *arg_into_names_below2* **by** *simp*
then
show *?thesis*
using *names_below_tr subsetI* **by** *simp*
qed

lemma *frecrel_mono* :
assumes $\langle x,y\rangle \in$ *frecrel*(*names_below*(*P*,*z*))
shows *frecrel*(*names_below*(*P*,*x*)) ⊆ *frecrel*(*names_below*(*P*,*y*))
unfolding *frecrel_def*
using *Rrel_mono names_below_mono assms* **by** *simp*

lemma *forcerel_mono2* :
assumes $\langle x,y\rangle \in$ *frecrel*(*names_below*(*P*,*z*))
shows *forcerel*(*P*,*x*) $\subseteq$ *forcerel*(*P*,*y*)
unfolding *forcerel_def*
using *trancl_mono frecrel_mono assms* **by** *simp*

lemma *forcerel_mono_aux* :
assumes $\langle x,y\rangle \in$ *frecrel*(*names_below*(*P*, *w*))$^+$
shows *forcerel*(*P*,*x*) $\subseteq$ *forcerel*(*P*,*y*)
using *assms*
by (*rule trancl_induct*,*simp_all add*: *subset_trans forcerel_mono2*)

lemma *forcerel_mono* :
assumes $\langle x,y\rangle \in$ *forcerel*(*P*,*z*)
shows *forcerel*(*P*,*x*) $\subseteq$ *forcerel*(*P*,*y*)
using *forcerel_mono_aux assms* **unfolding** *forcerel_def* **by** *simp*

lemma *forcerel_eq_aux*: $x \in$ *names_below*(*P*, *w*) $\Longrightarrow$ $\langle x,y\rangle \in$ *forcerel*(*P*,*z*) $\Longrightarrow$
($y \in$ *names_below*(*P*, *w*) $\longrightarrow$ $\langle x,y\rangle \in$ *forcerel*(*P*,*w*))
unfolding *forcerel_def*
proof(*rule_tac* $a{=}x$ **and** $b{=}y$ **and** $P{=}\lambda\ y\ .\ y \in$ *names_below*(*P*, *w*) $\longrightarrow$ $\langle x,y\rangle \in$ *frecrel*(*names_below*(*P*,*w*))$^+$ **in** *trancl_induct*,*simp*)
let *?A*$=\lambda\ a\ .$ *names_below*(*P*, *a*)
let *?R*$=\lambda\ a\ .$ *frecrel*(*?A*(*a*))
let *?fR*$=\lambda\ a\ .$*forcerel*(*a*)
show $u \in$*?A*(*w*) $\longrightarrow$ $\langle x,u\rangle \in$*?R*(*w*)$^+$ **if** $x \in$*?A*(*w*) $\langle x,y\rangle \in$*?R*(*z*)$^+$ $\langle x,u\rangle \in$*?R*(*z*)
for *u*
using *that frecrelD frecrelI r_into_trancl* **unfolding** *names_below_def* **by** *simp*
{
fix *u v*
assume $x \in$ *?A*(*w*)
$\langle x,\ y\rangle \in$ *?R*(*z*)$^+$
$\langle x,\ u\rangle \in$ *?R*(*z*)$^+$
$\langle u,\ v\rangle \in$ *?R*(*z*)
$u \in$ *?A*(*w*) $\Longrightarrow$ $\langle x,\ u\rangle \in$ *?R*(*w*)$^+$
then
have $v \in$ *?A*(*w*) $\Longrightarrow$ $\langle x,\ v\rangle \in$ *?R*(*w*)$^+$
proof -
assume $v \in$*?A*(*w*)
from ‹$\langle u,v\rangle \in$_› **have** $u \in$*?A*(*v*)
using *arg_into_names_below2* **by** *simp*
with ‹$v \in$*?A*(*w*)› **have** $u \in$*?A*(*w*)
using *names_below_tr* **by** *simp*
with ‹$v \in$_› ‹$\langle u,v\rangle \in$_› **have** $\langle u,v\rangle \in$ *?R*(*w*)
using *frecrelD frecrelI r_into_trancl* **unfolding** *names_below_def* **by** *simp*

```
      with ‹u ∈ ?A(w) ⟹ ⟨x, u⟩ ∈ ?R(w)^+› ‹u∈?A(w)›
      have ⟨x, u⟩ ∈ ?R(w)^+ by simp
      with ‹⟨u,v⟩∈ ?R(w)›
      show ⟨x,v⟩∈ ?R(w)^+ using trancl_trans r_into_trancl
        by simp
    qed
  }
  then show v ∈ ?A(w) ⟶ ⟨x, v⟩ ∈ ?R(w)^+
    if x ∈ ?A(w)
      ⟨x, y⟩ ∈ ?R(z)^+
      ⟨x, u⟩ ∈ ?R(z)^+
      ⟨u, v⟩ ∈ ?R(z)
      u ∈ ?A(w) ⟶ ⟨x, u⟩ ∈ ?R(w)^+ for u v
    using that by simp
qed

lemma forcerel_eq :
  assumes ⟨z,x⟩ ∈ forcerel(P,x)
  shows forcerel(P,z) = forcerel(P,x) ∩ names_below(P,z)×names_below(P,z)
  using assms forcerel_eq_aux forcerelD forcerel_mono[of z x x] subsetI
  by auto

lemma forcerel_below_aux :
  assumes ⟨z,x⟩ ∈ forcerel(P,x) ⟨u,z⟩ ∈ forcerel(P,x)
  shows u ∈ names_below(P,z)
  using assms(2)
  unfolding forcerel_def
proof(rule trancl_induct)
  show  u ∈ names_below(P,y) if ⟨u, y⟩ ∈ frecrel(names_below(P, x)) for y
    using that vimage_singleton_iff arg_into_names_below2 by simp
next
  show u ∈ names_below(P,z)
    if ⟨u, y⟩ ∈ frecrel(names_below(P, x))^+
      ⟨y, z⟩ ∈ frecrel(names_below(P, x))
      u ∈ names_below(P, y)
    for y z
    using that arg_into_names_below2[of y z x] names_below_tr by simp
qed

lemma forcerel_below :
  assumes ⟨z,x⟩ ∈ forcerel(P,x)
  shows forcerel(P,x) -`` {z} ⊆ names_below(P,z)
  using vimage_singleton_iff assms forcerel_below_aux by auto

lemma relation_forcerel :
  shows relation(forcerel(P,z)) trans(forcerel(P,z))
  unfolding forcerel_def using relation_trancl trans_trancl by simp_all

lemma Hfrc_restrict_trancl: bool_of_o(Hfrc(P, leq, y, restrict(f,frecrel(names_below(P,x))-``{y})))
```

$= bool\_of\_o(Hfrc(P,\ leq,\ y,\ restrict(f,(frecrel(names\_below(P,x))\hat{}+)\text{-}``\{y\})))$
unfolding *Hfrc_def bool_of_o_def eq_case_def mem_case_def*
using *restrict_trancl_forcerel frecRI1 frecRI2 frecRI3*
unfolding *forcerel_def*
by *simp*

lemma *frc_at_trancl*: $frc\_at(P,leq,z) = wfrec(forcerel(P,z),z,\lambda x\, f.\ bool\_of\_o(Hfrc(P,leq,x,f)))$
unfolding *frc_at_def forcerel_def* **using** *wf_eq_trancl Hfrc_restrict_trancl* **by** *simp*

lemma *forcerelI1* :
assumes $n1 \in domain(b) \vee n1 \in domain(c)$ $p{\in}P$ $d{\in}P$
shows $\langle\langle 1,\ n1,\ b,\ p\rangle,\ \langle 0,b,c,d\rangle\rangle{\in}\ forcerel(P,\langle 0,b,c,d\rangle)$
proof -
let $?x{=}\langle 1,\ n1,\ b,\ p\rangle$
let $?y{=}\langle 0,b,c,d\rangle$
from *assms*
have $frecR(?x,?y)$
using *frecRI1* **by** *simp*
then
have $?x{\in}names\_below(P,?y)$ $?y \in names\_below(P,?y)$
using *names_belowI assms components_in_eclose*
unfolding *names_below_def* **by** *auto*
with ‹$frecR(?x,?y)$›
show *?thesis*
unfolding *forcerel_def frecrel_def*
using *subsetD*[*OF r_subset_trancl*[*OF relation_Rrel*]] *RrelI*
by *auto*
qed

lemma *forcerelI2* :
assumes $n1 \in domain(b) \vee n1 \in domain(c)$ $p{\in}P$ $d{\in}P$
shows $\langle\langle 1,\ n1,\ c,\ p\rangle,\ \langle 0,b,c,d\rangle\rangle{\in}\ forcerel(P,\langle 0,b,c,d\rangle)$
proof -
let $?x{=}\langle 1,\ n1,\ c,\ p\rangle$
let $?y{=}\langle 0,b,c,d\rangle$
from *assms*
have $frecR(?x,?y)$
using *frecRI2* **by** *simp*
then
have $?x{\in}names\_below(P,?y)$ $?y \in names\_below(P,?y)$
using *names_belowI assms components_in_eclose*
unfolding *names_below_def* **by** *auto*
with ‹$frecR(?x,?y)$›
show *?thesis*
unfolding *forcerel_def frecrel_def*
using *subsetD*[*OF r_subset_trancl*[*OF relation_Rrel*]] *RrelI*

```
    by auto
qed

lemma forcerelI3 :
  assumes ⟨n2, r⟩ ∈ c p∈P d∈P r ∈ P
  shows ⟨⟨0, b, n2, p⟩,⟨1, b, c, d⟩⟩ ∈ forcerel(P,⟨1,b,c,d⟩)
proof -
  let ?x=⟨0, b, n2, p⟩
  let ?y=⟨1, b, c, d⟩
  from assms
  have frecR(?x,?y)
    using assms frecRI3 by simp
  then
  have ?x∈names_below(P,?y)  ?y ∈ names_below(P,?y)
    using names_belowI  assms components_in_eclose
    unfolding names_below_def by auto
  with ‹frecR(?x,?y)›
  show ?thesis
    unfolding forcerel_def frecrel_def
    using subsetD[OF r_subset_trancl[OF relation_Rrel]] RrelI
    by auto
qed

lemmas forcerelI = forcerelI1[THEN vimage_singleton_iff[THEN iffD2]]
  forcerelI2[THEN vimage_singleton_iff[THEN iffD2]]
  forcerelI3[THEN vimage_singleton_iff[THEN iffD2]]

lemma  aux_def_frc_at:
  assumes z ∈ forcerel(P,x) -`` {x}
  shows wfrec(forcerel(P,x), z, H) =  wfrec(forcerel(P,z), z, H)
proof -
  let ?A=names_below(P,z)
  from assms
  have ⟨z,x⟩ ∈ forcerel(P,x)
    using vimage_singleton_iff by simp
  then
  have z ∈ ?A
    using forcerel_arg_into_names_below by simp
  from ‹⟨z,x⟩ ∈ forcerel(P,x)›
  have E:forcerel(P,z) = forcerel(P,x) ∩ (?A×?A)
    forcerel(P,x) -`` {z} ⊆ ?A
    using forcerel_eq forcerel_below
    by auto
  with ‹z∈?A›
  have wfrec(forcerel(P,x), z, H) = wfrec[?A](forcerel(P,x), z, H)
   using wfrec_trans_restr[OF relation_forcerel(1) wf_forcerel relation_forcerel(2),
of x z ?A]
    by simp
  then show ?thesis
```

using *E wfrec_restr_eq* **by** *simp*
qed

32.5 Recursive expression of *frc_at*

lemma *def_frc_at* :
assumes $p{\in}P$
shows
$frc\_at(P,leq,\langle ft,n1,n2,p\rangle) =$
$bool\_of\_o(\ p \in P \wedge$
$(\ \ ft = 0 \wedge\ (\forall\, s.\ s{\in}domain(n1) \cup domain(n2) \longrightarrow$
$(\forall\, q.\ q{\in}P \wedge q \preceq p \longrightarrow (frc\_at(P,leq,\langle 1,s,n1,q\rangle) = 1 \longleftrightarrow frc\_at(P,leq,\langle 1,s,n2,q\rangle) = 1)))$
$\vee\ ft = 1 \wedge (\ \forall\, v{\in}P.\ v \preceq p \longrightarrow$
$(\exists\, q.\ \exists\, s.\ \exists\, r.\ r{\in}P \wedge q{\in}P \wedge q \preceq v \wedge \langle s,r\rangle \in n2 \wedge q \preceq r \wedge\ frc\_at(P,leq,\langle 0,n1,s,q\rangle) = 1))))$
proof -
let $?r{=}\lambda y.\ forcerel(P,y)$ **and** $?Hf{=}\lambda x\ f.\ bool\_of\_o(Hfrc(P,leq,x,f))$
let $?t{=}\lambda y.\ ?r(y)$ -`` $\{y\}$
let $?arg{=}\langle ft,n1,n2,p\rangle$
from *wf_forcerel*
have *wfr*: $\forall\, w\ .\ wf(?r(w))$ **..**
with *wfrec* [*of* $?r(?arg)$ *?arg ?Hf*]
have $frc\_at(P,leq,?arg) = ?Hf(\ ?arg,\ \lambda x{\in}?r(?arg)$ -`` $\{?arg\}.\ wfrec(?r(?arg),\ x,\ ?Hf))$
using *frc_at_trancl* **by** *simp*
also
have $\ldots = ?Hf(\ ?arg,\ \lambda x{\in}?r(?arg)$ -`` $\{?arg\}.\ frc\_at(P,leq,x))$
using *aux_def_frc_at frc_at_trancl* **by** *simp*
finally
show *?thesis*
unfolding *Hfrc_def mem_case_def eq_case_def*
using *forcerelI assms*
by *auto*
qed

32.6 Absoluteness of *frc_at*

lemma *forcerel_in_M* :
assumes
$x{\in}M$
shows
$forcerel(P,x){\in}M$
unfolding *forcerel_def def_frecrel names_below_def*
proof -
let $?Q = 2 \times ecloseN(x) \times ecloseN(x) \times P$
have $?Q \times ?Q \in M$
using ‹$x{\in}M$› *P_in_M twoN_in_M ecloseN_closed cartprod_closed* **by** *simp*
moreover
have $separation(\#\#M,\lambda z.\ \exists\, x\ y.\ z = \langle x,\ y\rangle \wedge frecR(x,\ y))$

```
  proof -
    have arity(frecrelP_fm(0)) = 1
      unfolding number1_fm_def frecrelP_fm_def
      by (simp del:FOL_sats_iff pair_abs empty_abs
          add: fm_definitions components_defs ord_simp_union)
    then
    have separation(##M, λz. sats(M,frecrelP_fm(0) , [z]))
      using separation_ax by simp
    moreover
    have frecrelP(##M,z) ⟷ sats(M,frecrelP_fm(0),[z])
      if z∈M for z
      using that sats_frecrelP_fm[of 0 [z]] by simp
    ultimately
    have separation(##M,frecrelP(##M))
      unfolding separation_def by (simp del:sats_frecrelP_fm)
    then
    show ?thesis using frecrelP_abs
        separation_cong[of ##M frecrelP(##M) λz. ∃x y. z = ⟨x, y⟩ ∧ frecR(x,
y)]
      by simp
  qed
  ultimately
  show {z ∈ ?Q × ?Q . ∃x y. z = ⟨x, y⟩ ∧ frecR(x, y)}^+ ∈ M
    using separation_closed frecrelP_abs trancl_closed by simp
qed

lemma relation2_Hfrc_at_abs:
  relation2(##M,is_Hfrc_at(##M,P,leq),λx f. bool_of_o(Hfrc(P,leq,x,f)))
  unfolding relation2_def using Hfrc_at_abs
  by simp

lemma Hfrc_at_closed :
  ∀x∈M. ∀g∈M. function(g) ⟶ bool_of_o(Hfrc(P,leq,x,g))∈M
  unfolding bool_of_o_def using zero_in_M nat_into_M[of 1] by simp

lemma wfrec_Hfrc_at :
  assumes
    X∈M
  shows
    wfrec_replacement(##M,is_Hfrc_at(##M,P,leq),forcerel(P,X))
proof -
  have 0:is_Hfrc_at(##M,P,leq,a,b,c) ⟷
        sats(M,Hfrc_at_fm(8,9,2,1,0),[c,b,a,d,e,y,x,z,P,leq,forcerel(P,X)])
    if a∈M b∈M c∈M d∈M e∈M y∈M x∈M z∈M
    for a b c d e y x z
    using that P_in_M leq_in_M ‹X∈M› forcerel_in_M
      Hfrc_at_iff_sats[of concl:M P leq a b c 8 9 2 1 0
        [c,b,a,d,e,y,x,z,P,leq,forcerel(P,X)]] by simp
 have 1:sats(M,is_wfrec_fm(Hfrc_at_fm(8,9,2,1,0),5,1,0),[y,x,z,P,leq,forcerel(P,X)])
```

$\longleftrightarrow$
is_wfrec(*##M*, *is_Hfrc_at*(*##M*,*P*,*leq*),*forcerel*(*P*,*X*), *x*, *y*)
if *x*∈*M* *y*∈*M* *z*∈*M* **for** *x y z*
using *that* ‹*X*∈*M*› *forcerel_in_M P_in_M leq_in_M*
sats_is_wfrec_fm[*OF 0*]
by *simp*
let
?f=*Exists*(*And*(*pair_fm*(1,0,2),*is_wfrec_fm*(*Hfrc_at_fm*(8,9,2,1,0),5,1,0)))
have *satsf*:*sats*(*M*, *?f*, [*x*,*z*,*P*,*leq*,*forcerel*(*P*,*X*)]) $\longleftrightarrow$
(∃ *y*∈*M*. *pair*(*##M*,*x*,*y*,*z*) & *is_wfrec*(*##M*, *is_Hfrc_at*(*##M*,*P*,*leq*),*forcerel*(*P*,*X*),
x, *y*))
if *x*∈*M* *z*∈*M* **for** *x z*
using *that* 1 ‹*X*∈*M*› *forcerel_in_M P_in_M leq_in_M* **by** (*simp del*:*pair_abs*)
have *artyf*:*arity*(*?f*) = 5
unfolding *fm_definitions PHcheck_fm_def is_tuple_fm_def*
by (*simp add*:*ord_simp_union*)
moreover
have *?f*∈*formula*
unfolding *fm_definitions* **by** *simp*
ultimately
have *strong_replacement*(*##M*,λ*x z*. *sats*(*M*,*?f*,[*x*,*z*,*P*,*leq*,*forcerel*(*P*,*X*)]))
using *replacement_ax*[*of ?f*] 1 *artyf* ‹*X*∈*M*› *forcerel_in_M P_in_M leq_in_M*
by *simp*
then
have *strong_replacement*(*##M*,λ*x z*.
∃ *y*∈*M*. *pair*(*##M*,*x*,*y*,*z*) & *is_wfrec*(*##M*, *is_Hfrc_at*(*##M*,*P*,*leq*),*forcerel*(*P*,*X*),
x, *y*))
using *repl_sats*[*of M ?f* [*P*,*leq*,*forcerel*(*P*,*X*)]] *satsf* **by** (*simp del*:*pair_abs*)
then
show *?thesis* **unfolding** *wfrec_replacement_def* **by** *simp*
qed

lemma *names_below_abs* :
⟦*Q*∈*M*;*x*∈*M*;*nb*∈*M*⟧ ⟹ *is_names_below*(*##M*,*Q*,*x*,*nb*) $\longleftrightarrow$ *nb* = *names_below*(*Q*,*x*)
unfolding *is_names_below_def names_below_def*
using *succ_in_M_iff zero_in_M cartprod_closed ecloseN_abs ecloseN_closed*
by *auto*

lemma *names_below_closed*:
⟦*Q*∈*M*;*x*∈*M*⟧ ⟹ *names_below*(*Q*,*x*) ∈ *M*
unfolding *names_below_def*
using *zero_in_M cartprod_closed ecloseN_closed succ_in_M_iff*
by *simp*

lemma *names_below_productE* :
assumes *Q* ∈ *M x* ∈ *M*
⋀*A1 A2 A3 A4*. *A1* ∈ *M* ⟹ *A2* ∈ *M* ⟹ *A3* ∈ *M* ⟹ *A4* ∈ *M* ⟹ *R*(*A1*
× *A2* × *A3* × *A4*)
shows *R*(*names_below*(*Q*,*x*))

unfolding *names_below_def* **using** *assms zero_in_M ecloseN_closed*[*of x*] *twoN_in_M* **by** *auto*

lemma *forcerel_abs* :
 ⟦*x*∈*M*;*z*∈*M*⟧ ⟹ *is_forcerel*(##*M*,*P*,*x*,*z*) ⟷ *z* = *forcerel*(*P*,*x*)
 unfolding *is_forcerel_def forcerel_def*
 using *frecrel_abs names_below_abs trancl_abs P_in_M twoN_in_M ecloseN_closed names_below_closed*
 names_below_productE[*of concl*:λ*p*. *is_frecrel*(##*M*,*p*,_) ⟷ _ = *frecrel*(*p*)] *frecrel_closed*
 by *simp*

lemma *frc_at_abs*:
 assumes *fnnc*∈*M z*∈*M*
 shows *is_frc_at*(##*M*,*P*,*leq*,*fnnc*,*z*) ⟷ *z* = *frc_at*(*P*,*leq*,*fnnc*)
proof -
 from *assms*
 have (∃ *r*∈*M*. *is_forcerel*(##*M*,*P*,*fnnc*, *r*) ∧ *is_wfrec*(##*M*, *is_Hfrc_at*(##*M*, *P*, *leq*), *r*, *fnnc*, *z*))
 ⟷ *is_wfrec*(##*M*, *is_Hfrc_at*(##*M*, *P*, *leq*), *forcerel*(*P*,*fnnc*), *fnnc*, *z*)
 using *forcerel_abs forcerel_in_M* **by** *simp*
 then
 show *?thesis*
 unfolding *frc_at_trancl is_frc_at_def*
 using *assms wfrec_Hfrc_at*[*of fnnc*] *wf_forcerel relation_forcerel forcerel_in_M*
 Hfrc_at_closed relation2_Hfrc_at_abs
 trans_wfrec_abs[*of forcerel*(*P*,*fnnc*) *fnnc z is_Hfrc_at*(##*M*,*P*,*leq*) λ*x f*. *bool_of_o*(*Hfrc*(*P*,*leq*,*x*,*f*))]
 by (*simp flip*:*setclass_iff*)
qed

lemma *forces_eq'_abs* :
 ⟦*p*∈*M* ; *t1*∈*M* ; *t2*∈*M*⟧ ⟹ *is_forces_eq'*(##*M*,*P*,*leq*,*p*,*t1*,*t2*) ⟷ *forces_eq'*(*P*,*leq*,*p*,*t1*,*t2*)
 unfolding *is_forces_eq'_def forces_eq'_def*
 using *frc_at_abs zero_in_M pair_in_M_iff* **by** (*auto simp add*:*components_abs*)

lemma *forces_mem'_abs* :
 ⟦*p*∈*M* ; *t1*∈*M* ; *t2*∈*M*⟧ ⟹ *is_forces_mem'*(##*M*,*P*,*leq*,*p*,*t1*,*t2*) ⟷ *forces_mem'*(*P*,*leq*,*p*,*t1*,*t2*)
 unfolding *is_forces_mem'_def forces_mem'_def*
 using *frc_at_abs zero_in_M pair_in_M_iff* **by** (*auto simp add*:*components_abs*)

lemma *forces_neq'_abs* :
 assumes
 p∈*M t1*∈*M t2*∈*M*
 shows
 is_forces_neq'(##*M*,*P*,*leq*,*p*,*t1*,*t2*) ⟷ *forces_neq'*(*P*,*leq*,*p*,*t1*,*t2*)
proof -
 have *q*∈*M* **if** *q*∈*P* **for** *q*
 using *that transitivity P_in_M* **by** *simp*

then show *?thesis*
unfolding *is_forces_neq'_def forces_neq'_def*
using *assms forces_eq'_abs pair_in_M_iff*
by (*auto simp add:components_abs,blast*)
qed

lemma *forces_nmem'_abs* :
assumes
p∈M t1∈M t2∈M
shows
is_forces_nmem'(##M,P,leq,p,t1,t2) ⟷ forces_nmem'(P,leq,p,t1,t2)
proof -
have *q∈M* **if** *q∈P* **for** *q*
using *that transitivity P_in_M* **by** *simp*
then show *?thesis*
unfolding *is_forces_nmem'_def forces_nmem'_def*
using *assms forces_mem'_abs pair_in_M_iff*
by (*auto simp add:components_abs,blast*)
qed

end

32.7 Forcing for general formulas

definition
ren_forces_nand :: *i⇒i* **where**
ren_forces_nand(φ) ≡ Exists(And(Equal(0,1),iterates(λp. incr_bv(p)`1 , 2, φ)))

lemma *ren_forces_nand_type*[*TC*] :
φ∈formula ⟹ ren_forces_nand(φ) ∈formula
unfolding *ren_forces_nand_def*
by *simp*

lemma *arity_ren_forces_nand* :
assumes *φ∈formula*
shows *arity(ren_forces_nand(φ)) ≤ succ(arity(φ))*
proof -
consider (*lt*) *1<arity(φ)* | (*ge*) ¬ *1 < arity(φ)*
by *auto*
then
show *?thesis*
proof *cases*
case *lt*
with ‹*φ∈_*›
have *2 < succ(arity(φ))* *2<arity(φ)#+2*
using *succ_ltI* **by** *auto*
with ‹*φ∈_*›
have *arity(iterates(λp. incr_bv(p)`1,2,φ)) = 2#+arity(φ)*

```
    using arity_incr_bv_lemma lt
    by auto
  with ‹φ∈_›
  show ?thesis
    unfolding ren_forces_nand_def
    using lt pred_Un_distrib union_abs1 Un_assoc[symmetric] Un_le_compat
    by simp
next
  case ge
  with ‹φ∈_›
  have arity(φ) ≤ 1 pred(arity(φ)) ≤ 1
    using not_lt_iff_le le_trans[OF le_pred]
    by simp_all
  with ‹φ∈_›
  have arity(iterates(λp. incr_bv(p)‘1,2,φ)) = (arity(φ))
    using arity_incr_bv_lemma ge
    by simp
  with ‹arity(φ) ≤ 1› ‹φ∈_› ‹pred(_) ≤ 1›
  show ?thesis
    unfolding ren_forces_nand_def
    using  pred_Un_distrib union_abs1 Un_assoc[symmetric] union_abs2
    by simp
qed
qed

lemma sats_ren_forces_nand:
  [q,P,leq,o,p] @ env ∈ list(M) ⟹ φ∈formula ⟹
   sats(M, ren_forces_nand(φ),[q,p,P,leq,o] @ env) ⟷ sats(M, φ,[q,P,leq,o] @
env)
  unfolding ren_forces_nand_def
  using sats_incr_bv_iff [of _ _ M _ [q]]
  by simp


definition
  ren_forces_forall :: i⇒i where
  ren_forces_forall(φ) ≡
      Exists(Exists(Exists(Exists(Exists(
        And(Equal(0,6),And(Equal(1,7),And(Equal(2,8),And(Equal(3,9),
        And(Equal(4,5),iterates(λp. incr_bv(p)‘5 , 5, φ)))))))))))

lemma arity_ren_forces_all :
  assumes φ∈formula
  shows arity(ren_forces_forall(φ)) = 5 ∪ arity(φ)
proof -
  consider (lt) 5<arity(φ) | (ge) ¬ 5 < arity(φ)
    by auto
  then
  show ?thesis
```

```
  proof cases
    case lt
    with ‹φ∈_›
    have 5 < succ(arity(φ)) 5<arity(φ)#+2  5<arity(φ)#+3  5<arity(φ)#+4
      using succ_ltI by auto
    with ‹φ∈_›
    have arity(iterates(λp. incr_bv(p)‘5,5,φ)) = 5#+arity(φ)
      using arity_incr_bv_lemma lt
      by simp
    with ‹φ∈_›
    show ?thesis
      unfolding ren_forces_forall_def
      using pred_Un_distrib union_abs1 Un_assoc[symmetric] union_abs2
      by simp
  next
    case ge
    with ‹φ∈_›
    have arity(φ) ≤ 5 pred^5(arity(φ)) ≤ 5
      using not_lt_iff_le le_trans[OF le_pred]
      by simp_all
    with ‹φ∈_›
    have arity(iterates(λp. incr_bv(p)‘5,5,φ)) = arity(φ)
      using arity_incr_bv_lemma ge
      by simp
    with ‹arity(φ) ≤ 5› ‹φ∈_› ‹pred^5(_) ≤ 5›
    show ?thesis
      unfolding ren_forces_forall_def
      using  pred_Un_distrib union_abs1 Un_assoc[symmetric] union_abs2
      by simp
  qed
qed

lemma ren_forces_forall_type[TC] :
  φ∈formula ⟹ ren_forces_forall(φ) ∈formula
  unfolding ren_forces_forall_def by simp

lemma sats_ren_forces_forall :
  [x,P,leq,o,p] @ env ∈ list(M) ⟹ φ∈formula ⟹
    sats(M, ren_forces_forall(φ),[x,p,P,leq,o] @ env) ⟷ sats(M, φ,[p,P,leq,o,x]
@ env)
  unfolding ren_forces_forall_def
  using sats_incr_bv_iff [of _ _ M _ [p,P,leq,o,x]]
  by simp


definition
  is_leq :: [i⇒o,i,i,i] ⇒ o where
  is_leq(A,l,q,p) ≡ ∃ qp[A]. (pair(A,q,p,qp) ∧ qp∈l)
```

lemma (**in** *forcing_data*) *leq_abs*:
  ⟦ *l∈M* ; *q∈M* ; *p∈M* ⟧ ⟹ *is_leq*(*##M*,*l*,*q*,*p*) ⟷ ⟨*q*,*p*⟩∈*l*
  unfolding *is_leq_def* **using** *pair_in_M_iff* **by** *simp*

definition
  leq_fm :: [*i*,*i*,*i*] ⇒ *i* **where**
  leq_fm(*leq*,*q*,*p*) ≡ *Exists*(*And*(*pair_fm*(*q#+1*,*p#+1*,*0*),*Member*(*0*,*leq#+1*)))

lemma *arity_leq_fm* :
 ⟦*leq∈nat*;*q∈nat*;*p∈nat*⟧ ⟹ *arity*(*leq_fm*(*leq*,*q*,*p*)) = *succ*(*q*) ∪ *succ*(*p*) ∪ *succ*(*leq*)
  unfolding *leq_fm_def*
  using *arity_pair_fm pred_Un_distrib ord_simp_union*
  by *auto*

lemma *leq_fm_type*[*TC*] :
  ⟦*leq∈nat*;*q∈nat*;*p∈nat*⟧ ⟹ *leq_fm*(*leq*,*q*,*p*)∈*formula*
  unfolding *leq_fm_def* **by** *simp*

lemma *sats_leq_fm* :
  ⟦ *leq∈nat*;*q∈nat*;*p∈nat*;*env∈list*(*A*) ⟧ ⟹
   sats(*A*,*leq_fm*(*leq*,*q*,*p*),*env*) ⟷ *is_leq*(*##A*,*nth*(*leq*,*env*),*nth*(*q*,*env*),*nth*(*p*,*env*))
  unfolding *leq_fm_def is_leq_def* **by** *simp*

32.7.1 The primitive recursion

consts *forces′* :: *i*⇒*i*
primrec
  forces′(*Member*(*x*,*y*)) = *forces_mem_fm*(*1*,*2*,*0*,*x#+4*,*y#+4*)
  forces′(*Equal*(*x*,*y*)) = *forces_eq_fm*(*1*,*2*,*0*,*x#+4*,*y#+4*)
  forces′(*Nand*(*p*,*q*)) =
     Neg(*Exists*(*And*(*Member*(*0*,*2*),*And*(*leq_fm*(*3*,*0*,*1*),*And*(*ren_forces_nand*(*forces′*(*p*)),
                                  ren_forces_nand(*forces′*(*q*)))))))
  forces′(*Forall*(*p*)) = *Forall*(*ren_forces_forall*(*forces′*(*p*)))

definition
  forces :: *i*⇒*i* **where**
  forces(*φ*) ≡ *And*(*Member*(*0*,*1*),*forces′*(*φ*))

lemma *forces′_type* [*TC*]: *φ∈formula* ⟹ *forces′*(*φ*) ∈ *formula*
  by (*induct φ set*:*formula*; *simp*)

lemma *forces_type*[*TC*] : *φ∈formula* ⟹ *forces*(*φ*) ∈ *formula*
  unfolding *forces_def* **by** *simp*

context *forcing_data*
begin

32.8 Forcing for atomic formulas in context

definition
forces_eq :: $[i,i,i] \Rightarrow o$ ($\langle$_ *forces*$_a$ '(_ = _')$\rangle$ [*36,1,1*] *60*) **where**
forces_eq $\equiv$ *forces_eq'*(*P*,*leq*)

definition
forces_mem :: $[i,i,i] \Rightarrow o$ ($\langle$_ *forces*$_a$ '(_ $\in$ _')$\rangle$ [*36,1,1*] *60*) **where**
forces_mem $\equiv$ *forces_mem'*(*P*,*leq*)

definition
is_forces_eq :: $[i,i,i] \Rightarrow o$ **where**
is_forces_eq $\equiv$ *is_forces_eq'*(*##M*,*P*,*leq*)

definition
is_forces_mem :: $[i,i,i] \Rightarrow o$ **where**
is_forces_mem $\equiv$ *is_forces_mem'*(*##M*,*P*,*leq*)

lemma *def_forces_eq*: $p{\in}P \Longrightarrow p$ *forces*$_a$ $(t1 = t2) \longleftrightarrow$
$(\forall\, s{\in}domain(t1) \cup domain(t2).\ \forall\, q.\ q{\in}P \wedge q \preceq p \longrightarrow$
$(q$ *forces*$_a$ $(s \in t1) \longleftrightarrow q$ *forces*$_a$ $(s \in t2)))$
unfolding *forces_eq_def forces_mem_def forces_eq'_def forces_mem'_def*
using *def_frc_at*[*of p 0 t1 t2*] **unfolding** *bool_of_o_def*
by *auto*

lemma *def_forces_mem*: $p{\in}P \Longrightarrow p$ *forces*$_a$ $(t1 \in t2) \longleftrightarrow$
$(\forall\, v{\in}P.\ v \preceq p \longrightarrow$
$(\exists\, q.\ \exists\, s.\ \exists\, r.\ r{\in}P \wedge q{\in}P \wedge q \preceq v \wedge \langle s,r\rangle \in t2 \wedge q \preceq r \wedge q$ *forces*$_a$ $(t1 = s)))$
unfolding *forces_eq'_def forces_mem'_def forces_eq_def forces_mem_def*
using *def_frc_at*[*of p 1 t1 t2*] **unfolding** *bool_of_o_def*
by *auto*

lemma *forces_eq_abs* :
$[\![p{\in}M$; $t1{\in}M$; $t2{\in}M]\!] \Longrightarrow$ *is_forces_eq*$(p,t1,t2) \longleftrightarrow p$ *forces*$_a$ $(t1 = t2)$
unfolding *is_forces_eq_def forces_eq_def*
using *forces_eq'_abs* **by** *simp*

lemma *forces_mem_abs* :
$[\![p{\in}M$; $t1{\in}M$; $t2{\in}M]\!] \Longrightarrow$ *is_forces_mem*$(p,t1,t2) \longleftrightarrow p$ *forces*$_a$ $(t1 \in t2)$
unfolding *is_forces_mem_def forces_mem_def*
using *forces_mem'_abs* **by** *simp*

lemma *sats_forces_eq_fm*:
assumes $p{\in}nat$ $l{\in}nat$ $q{\in}nat$ $t1{\in}nat$ $t2{\in}nat$ $env{\in}list(M)$
$nth(p,env){=}P$ $nth(l,env){=}leq$
shows *sats*$(M,forces\_eq\_fm(p,l,q,t1,t2),env) \longleftrightarrow$
is_forces_eq$(nth(q,env),nth(t1,env),nth(t2,env))$

unfolding *forces_eq_fm_def is_forces_eq_def is_forces_eq'_def*
using *assms sats_is_tuple_fm sats_frc_at_fm*
by *simp*

lemma *sats_forces_mem_fm*:
assumes *p∈nat l∈nat q∈nat t1∈nat t2∈nat env∈list(M)*
nth(p,env)=P nth(l,env)=leq
shows *sats(M,forces_mem_fm(p,l,q,t1,t2),env) ⟷*
is_forces_mem(nth(q,env),nth(t1,env),nth(t2,env))
unfolding *forces_mem_fm_def is_forces_mem_def is_forces_mem'_def*
using *assms sats_is_tuple_fm sats_frc_at_fm*
by *simp*

definition
forces_neq :: *[i,i,i] ⇒ o* (‹_ *forces*$_a$ '(_ ≠ _')› *[36,1,1] 60*) **where**
p forces$_a$ *(t1 ≠ t2) ≡ ¬ (∃ q∈P. q≼p ∧ q forces*$_a$ *(t1 = t2))*

definition
forces_nmem :: *[i,i,i] ⇒ o* (‹_ *forces*$_a$ '(_ ∉ _')› *[36,1,1] 60*) **where**
p forces$_a$ *(t1 ∉ t2) ≡ ¬ (∃ q∈P. q≼p ∧ q forces*$_a$ *(t1 ∈ t2))*

lemma *forces_neq* :
p forces$_a$ *(t1 ≠ t2) ⟷ forces_neq'(P,leq,p,t1,t2)*
unfolding *forces_neq_def forces_neq'_def forces_eq_def* **by** *simp*

lemma *forces_nmem* :
p forces$_a$ *(t1 ∉ t2) ⟷ forces_nmem'(P,leq,p,t1,t2)*
unfolding *forces_nmem_def forces_nmem'_def forces_mem_def* **by** *simp*

abbreviation *Forces* :: *[i, i, i] ⇒ o* (_ ⊩ _ _ *[36,36,36] 60*) **where**
p ⊩ φ env ≡ M, ([p,P,leq,one] @ env) ⊨ forces(φ)

lemma *sats_forces_Member* :
assumes *x∈nat y∈nat env∈list(M)*
nth(x,env)=xx nth(y,env)=yy q∈M
shows *q ⊩ ·x ∈ y· env ⟷ q ∈ P ∧ is_forces_mem(q, xx, yy)*
unfolding *forces_def*
using *assms sats_forces_mem_fm P_in_M leq_in_M one_in_M*
by *simp*

lemma *sats_forces_Equal* :
assumes *x∈nat y∈nat env∈list(M)*
nth(x,env)=xx nth(y,env)=yy q∈M
shows *q ⊩ ·x = y· env ⟷ q ∈ P ∧ is_forces_eq(q, xx, yy)*
unfolding *forces_def*
using *assms sats_forces_eq_fm P_in_M leq_in_M one_in_M*
by *simp*

lemma *sats_forces_Nand* :
assumes φ∈*formula* ψ∈*formula* *env*∈*list*(M) p∈M
shows $p \Vdash \cdot\neg(\varphi \wedge \psi)\cdot$ *env* $\longleftrightarrow$
p∈$P \wedge \neg(\exists\, q$∈$M.\ q$∈$P \wedge$ *is_leq*(##M,*leq*,q,p) $\wedge$
(M,[q,P,*leq*,*one*]@*env* $\models$ *forces*′(φ)) $\wedge$ (M,[q,P,*leq*,*one*]@*env* $\models$ *forces*′(ψ)))
unfolding *forces_def* **using** *sats_leq_fm assms sats_ren_forces_nand P_in_M leq_in_M one_in_M*
by *simp*

lemma *sats_forces_Neg* :
assumes φ∈*formula* *env*∈*list*(M) p∈M
shows $p \Vdash \cdot\neg\varphi\cdot$ *env* $\longleftrightarrow$
(p∈$P \wedge \neg(\exists\, q$∈$M.\ q$∈$P \wedge$ *is_leq*(##M,*leq*,q,p) $\wedge$
(M, [q, P, *leq*, *one*] @ *env* $\models$ *forces*′(φ))))
unfolding *Neg_def* **using** *assms sats_forces_Nand*
by *simp*

lemma *sats_forces_Forall* :
assumes φ∈*formula* *env*∈*list*(M) p∈M
shows $p \Vdash (\cdot\forall\,\varphi\cdot)$ *env* $\longleftrightarrow p \in P \wedge (\forall\, x$∈$M.\ M$,[$p$,$P$,*leq*,*one*,$x$] @ *env* $\models$ *forces*′(φ))
unfolding *forces_def* **using** *assms sats_ren_forces_forall P_in_M leq_in_M one_in_M*
by *simp*

end

32.9 The arity of *forces*

lemma *arity_forces_at*:
assumes $x \in nat$ $y \in nat$
shows *arity*(*forces*(*Member*(x, y))) = (*succ*(x) $\cup$ *succ*(y)) #+ 4
arity(*forces*(*Equal*(x, y))) = (*succ*(x) $\cup$ *succ*(y)) #+ 4
unfolding *forces_def*
using *assms arity_forces_mem_fm arity_forces_eq_fm succ_Un_distrib ord_simp_union*
by *auto*

lemma *arity_forces*′:
assumes φ∈*formula*
shows *arity*(*forces*′(φ)) $\leq$ *arity*(φ) #+ 4
using *assms*
proof (*induct set*:*formula*)
case (*Member x y*)
then
show *?case*
using *arity_forces_mem_fm succ_Un_distrib ord_simp_union*
by *simp*
next
case (*Equal x y*)
then

```
  show ?case
    using arity_forces_eq_fm succ_Un_distrib ord_simp_union
    by simp
next
  case (Nand φ ψ)
  let ?φ' = ren_forces_nand(forces'(φ))
  let ?ψ' = ren_forces_nand(forces'(ψ))
  have arity(leq_fm(3, 0, 1)) = 4
    using arity_leq_fm succ_Un_distrib ord_simp_union
    by simp
  have 3 ≤ (4#+arity(φ)) ∪ (4#+arity(ψ)) (is _ ≤ ?rhs)
    using ord_simp_union by simp
  from ‹φ∈_› Nand
  have pred(arity(?φ')) ≤ ?rhs  pred(arity(?ψ')) ≤ ?rhs
  proof -
    from ‹φ∈_› ‹ψ∈_›
    have A:pred(arity(?φ')) ≤ arity(forces'(φ))
      pred(arity(?ψ')) ≤ arity(forces'(ψ))
      using pred_mono[OF _  arity_ren_forces_nand] pred_succ_eq
      by simp_all
    from Nand
    have 3 ∪ arity(forces'(φ)) ≤ arity(φ) #+ 4
      3 ∪ arity(forces'(ψ)) ≤ arity(ψ) #+ 4
      using Un_le by simp_all
    with Nand
    show pred(arity(?φ')) ≤ ?rhs
      pred(arity(?ψ')) ≤ ?rhs
      using le_trans[OF A(1)] le_trans[OF A(2)] le_Un_iff
      by simp_all
  qed
  with Nand ‹_=4›
  show ?case
     using pred_Un_distrib Un_assoc[symmetric] succ_Un_distrib union_abs1
Un_leI3[OF ‹3 ≤ ?rhs›]
    by simp
next
  case (Forall φ)
  let ?φ' = ren_forces_forall(forces'(φ))
  show ?case
  proof (cases arity(φ) = 0)
    case True
    with Forall
    show ?thesis
    proof -
      from Forall True
      have arity(forces'(φ)) ≤ 5
        using le_trans[of _ 4 5] by auto
      with ‹φ∈_›
      have arity(?φ') ≤ 5
```

```
      using arity_ren_forces_all[OF forces'_type[OF ‹φ∈_›]] union_abs2
      by auto
    with Forall True
    show ?thesis
      using pred_mono[OF _ ‹arity(?φ') ≤ 5›]
      by simp
  qed
 next
  case False
  with Forall
  show ?thesis
  proof -
    from Forall False
    have arity(?φ') = 5 ∪ arity(forces'(φ))
      arity(forces'(φ)) ≤ 5 #+ arity(φ)
      4 ≤ succ(succ(succ(arity(φ))))
      using Ord_0_lt arity_ren_forces_all
        le_trans[OF _ add_le_mono[of 4 5, OF _ le_refl]]
      by auto
    with ‹φ∈_›
    have 5 ∪ arity(forces'(φ)) ≤ 5#+arity(φ)
      using ord_simp_union by auto
    with ‹φ∈_› ‹arity(?φ') = 5 ∪ _›
    show ?thesis
      using pred_Un_distrib succ_pred_eq[OF _ ‹arity(φ)≠0›]
        pred_mono[OF _ Forall(2)] Un_le[OF ‹4≤succ(_)›]
      by simp
  qed
 qed
qed

lemma arity_forces :
 assumes φ∈formula
 shows arity(forces(φ)) ≤ 4#+arity(φ)
 unfolding forces_def
 using assms arity_forces' le_trans ord_simp_union by auto

lemma arity_forces_le :
 assumes φ∈formula n∈nat arity(φ) ≤ n
 shows arity(forces(φ)) ≤ 4#+n
using assms le_trans[OF _ add_le_mono[OF le_refl[of 5] ‹arity(φ)≤_›]] arity_forces
 by auto

end
```

33 The Forcing Theorems

```
theory Forcing_Theorems
 imports
```

Forces_Definition

begin

context *forcing_data*
begin

33.1 The forcing relation in context

lemma *Collect_forces* :
  assumes
    fty: $\varphi \in formula$ **and**
    far: $arity(\varphi) \leq length(env)$ **and**
    envty: $env \in list(M)$
  shows
    $\{p \in P . p \Vdash \varphi\ env\} \in M$
proof -
  have $z \in P \Longrightarrow z \in M$ **for** z
    using *P_in_M transitivity*[*of z P*] **by** *simp*
  moreover
  have *separation*($\#\#M$,$\lambda p.\ (p \Vdash \varphi\ env)$)
        using *separation_ax arity_forces far fty P_in_M leq_in_M one_in_M envty arity_forces_le*
    by *simp*
  then
  have *Collect*(P,$\lambda p.\ (p \Vdash \varphi\ env)$)$\in M$
    using *separation_closed P_in_M* **by** *simp*
  then show *?thesis* **by** *simp*
qed

lemma *forces_mem_iff_dense_below*: $p \in P \Longrightarrow p\ forces_a\ (t1 \in t2) \longleftrightarrow dense\_below($
    $\{q \in P.\ \exists s.\ \exists r.\ r \in P \wedge \langle s,r \rangle \in t2 \wedge q \preceq r \wedge q\ forces_a\ (t1 = s)\}$
    $,p)$
  using *def_forces_mem*[*of p t1 t2*] **by** *blast*

33.2 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_eq*:
  assumes $p \in P\ r \in P\ r \preceq p\ p\ forces_a\ (t1 = t2)$
  shows $r\ forces_a\ (t1 = t2)$
  using *assms def_forces_eq*[*of _ t1 t2*] *leq_transD* **by** *blast*

33.3 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_mem*:
  assumes $p \in P\ r \in P\ r \preceq p\ p\ forces_a\ (t1 \in t2)$
  shows $r\ forces_a\ (t1 \in t2)$
  using *assms forces_mem_iff_dense_below dense_below_under* **by** *auto*

33.4 Kunen 2013, Lemma IV.2.37(b)

lemma *density_mem*:
assumes $p{\in}P$
shows p *forces*$_a$ $(t1 \in t2)$ $\longleftrightarrow$ *dense_below*$(\{q{\in}P.\ q$ *forces*$_a$ $(t1 \in t2)\},p)$
proof
assume p *forces*$_a$ $(t1 \in t2)$
with *assms*
show *dense_below*$(\{q{\in}P.\ q$ *forces*$_a$ $(t1 \in t2)\},p)$
using *forces_mem_iff_dense_below strengthening_mem*[*of p*] *ideal_dense_below*
by *auto*
next
assume *dense_below*$(\{q \in P\ .\ q$ *forces*$_a$ $(\ t1 \in\ \ t2)\},\ p)$
with *assms*
have *dense_below*$(\{q{\in}P.$
dense_below$(\{q'{\in}P.\ \exists\, s\ r.\ r \in P \wedge \langle s,r\rangle{\in}t2 \wedge q'{\preceq}r \wedge q'$ *forces*$_a$ $(t1 = s)\},q)$
$\},p)$
using *forces_mem_iff_dense_below* **by** *simp*
with *assms*
show p *forces*$_a$ $(t1 \in t2)$
using *dense_below_dense_below forces_mem_iff_dense_below*[*of p t1 t2*] **by** *blast*
qed

lemma *aux_density_eq*:
assumes
dense_below(
$\{q'{\in}P.\ \forall\, q.\ q{\in}P \wedge q{\preceq}q' \longrightarrow q$ *forces*$_a$ $(s \in t1) \longleftrightarrow q$ *forces*$_a$ $(s \in t2)\}$
$,p)$
q *forces*$_a$ $(s \in t1)$ $q{\in}P$ $p{\in}P$ $q{\preceq}p$
shows
dense_below$(\{r{\in}P.\ r$ *forces*$_a$ $(s \in t2)\},q)$
proof
fix r
assume $r{\in}P$ $r{\preceq}q$
moreover from *this* **and** ‹$p{\in}P$› ‹$q{\preceq}p$› ‹$q{\in}P$›
have $r{\preceq}p$
using *leq_transD* **by** *simp*
moreover
note ‹q *forces*$_a$ $(s \in t1)$› ‹*dense_below*$(\_,p)$› ‹$q{\in}P$›
ultimately
obtain $q1$ **where** $q1{\preceq}r$ $q1{\in}P$ $q1$ *forces*$_a$ $(s \in t2)$
using *strengthening_mem*[*of q _ s t1*] *refl_leq leq_transD*[*of _ r q*] **by** *blast*
then
show $\exists\, d{\in}\{r \in P\ .\ r$ *forces*$_a$ $(\ s \in\ \ t2)\}.\ d \in P \wedge d{\preceq}\ r$
by *blast*
qed

lemma *density_eq*:

assumes $p{\in}P$
shows $p\ forces_a\ (t1 = t2) \longleftrightarrow dense\_below(\{q{\in}P.\ q\ forces_a\ (t1 = t2)\},p)$
proof
assume $p\ forces_a\ (t1 = t2)$
with ‹$p{\in}P$›
show $dense\_below(\{q{\in}P.\ q\ forces_a\ (t1 = t2)\},p)$
using *strengthening_eq ideal_dense_below* **by** *auto*
next
assume $dense\_below(\{q{\in}P.\ q\ forces_a\ (t1 = t2)\},p)$
{
fix $s\ q$
let $?D1{=}\{q'{\in}P.\ \forall\, s{\in}domain(t1) \cup domain(t2).\ \forall\, q.\ q \in P \wedge q{\preceq}q' \longrightarrow$
$q\ forces_a\ (s \in t1){\longleftrightarrow}q\ forces_a\ (s \in t2)\}$
let $?D2{=}\{q'{\in}P.\ \forall\, q.\ q{\in}P \wedge q{\preceq}q' \longrightarrow q\ forces_a\ (s \in t1) \longleftrightarrow q\ forces_a\ (s \in t2)\}$
assume $s{\in}domain(t1) \cup domain(t2)$
then
have $?D1{\subseteq}?D2$ **by** *blast*
with ‹$dense\_below(\_,p)$›
have $dense\_below(\{q'{\in}P.\ \forall\, s{\in}domain(t1) \cup domain(t2).\ \forall\, q.\ q \in P \wedge q{\preceq}q'$
$\longrightarrow$
$q\ forces_a\ (s \in t1){\longleftrightarrow}q\ forces_a\ (s \in t2)\},p)$
using $dense\_below\_cong'[OF$ ‹$p{\in}P$› $def\_forces\_eq[of\ \_\ t1\ t2]]$ **by** *simp*
with ‹$p{\in}P$› ‹$?D1{\subseteq}?D2$›
have $dense\_below(\{q'{\in}P.\ \forall\, q.\ q{\in}P \wedge q{\preceq}q' \longrightarrow$
$q\ forces_a\ (s \in t1) \longleftrightarrow q\ forces_a\ (s \in t2)\},p)$
using *dense_below_mono* **by** *simp*
moreover from *this*
have $dense\_below(\{q'{\in}P.\ \forall\, q.\ q{\in}P \wedge q{\preceq}q' \longrightarrow$
$q\ forces_a\ (s \in t2) \longleftrightarrow q\ forces_a\ (s \in t1)\},p)$
by *blast*
moreover
assume $q \in P\ q{\preceq}p$
moreover
note ‹$p{\in}P$›
ultimately
have $q\ forces_a\ (s \in t1) \Longrightarrow dense\_below(\{r{\in}P.\ r\ forces_a\ (s \in t2)\},q)$
$q\ forces_a\ (s \in t2) \Longrightarrow dense\_below(\{r{\in}P.\ r\ forces_a\ (s \in t1)\},q)$
using *aux_density_eq* **by** *simp_all*
then
have $q\ forces_a\ (\ s \in\ \ t1) \longleftrightarrow q\ forces_a\ (\ s \in\ \ t2)$
using $density\_mem[OF$ ‹$q{\in}P$›$]$ **by** *blast*
}
with ‹$p{\in}P$›
show $p\ forces_a\ (t1 = t2)$ **using** *def_forces_eq* **by** *blast*
qed

33.5 Kunen 2013, Lemma IV.2.38

lemma *not_forces_neq*:

assumes $p{\in}P$
shows p *forces*$_a$ $(t1 = t2) \longleftrightarrow \neg\ (\exists\, q{\in}P.\ q{\preceq}p \wedge q$ *forces*$_a$ $(t1 \neq t2))$
using *assms density_eq* **unfolding** *forces_neq_def* **by** *blast*

lemma *not_forces_nmem*:
assumes $p{\in}P$
shows p *forces*$_a$ $(t1 \in t2) \longleftrightarrow \neg\ (\exists\, q{\in}P.\ q{\preceq}p \wedge q$ *forces*$_a$ $(t1 \notin t2))$
using *assms density_mem* **unfolding** *forces_nmem_def* **by** *blast*

lemma *sats_forces_Nand'*:
assumes
$p{\in}P$ $\varphi{\in}$*formula* $\psi{\in}$*formula* *env* $\in$ *list*(M)
shows
$(M,\ [p,P,leq,one]\ @\ env \models forces(Nand(\varphi,\psi))) \longleftrightarrow$
$\neg(\exists\, q{\in}M.\ q{\in}P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
$(M,\ [q,P,leq,one]\ @\ env \models forces(\varphi)) \wedge$
$(M,\ [q,P,leq,one]\ @\ env \models forces(\psi)))$
using *assms sats_forces_Nand*[*OF assms*(*2-4*) *transitivity*[*OF* ‹$p{\in}P$›]]
P_in_M leq_in_M one_in_M **unfolding** *forces_def*
by *simp*

lemma *sats_forces_Neg'*:
assumes
$p{\in}P$ *env* $\in$ *list*(M) $\varphi{\in}$*formula*
shows
$(M,\ [p,P,leq,one]\ @\ env \models forces(Neg(\varphi))) \longleftrightarrow$
$\neg(\exists\, q{\in}M.\ q{\in}P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
$(M,\ [q,P,leq,one]@env \models forces(\varphi)))$
using *assms sats_forces_Neg transitivity*
P_in_M leq_in_M one_in_M **unfolding** *forces_def*
by (*simp*, *blast*)

lemma *sats_forces_Forall'*:
assumes
$p{\in}P$ *env* $\in$ *list*(M) $\varphi{\in}$*formula*
shows
$(M,[p,P,leq,one]\ @\ env \models forces(Forall(\varphi))) \longleftrightarrow$
$(\forall\, x{\in}M.\quad M,\ [p,P,leq,one,x]\ @\ env \models forces(\varphi))$
using *assms sats_forces_Forall transitivity*
P_in_M leq_in_M one_in_M sats_ren_forces_forall **unfolding** *forces_def*
by *simp*

33.6 The relation of forcing and atomic formulas

lemma *Forces_Equal*:
assumes
$p{\in}P$ $t1{\in}M$ $t2{\in}M$ $env{\in}list(M)$ $nth(n,env) = t1$ $nth(m,env) = t2$ $n{\in}nat$ $m{\in}nat$
shows
$(p \Vdash Equal(n,m)\ env) \longleftrightarrow p\ forces_a\ (t1 = t2)$
using *assms sats_forces_Equal forces_eq_abs transitivity P_in_M*
by *simp*

lemma *Forces_Member*:
assumes
$p{\in}P$ $t1{\in}M$ $t2{\in}M$ $env{\in}list(M)$ $nth(n,env) = t1$ $nth(m,env) = t2$ $n{\in}nat$ $m{\in}nat$
shows
$(p \Vdash Member(n,m)\ env) \longleftrightarrow p\ forces_a\ (t1 \in t2)$
using *assms sats_forces_Member forces_mem_abs transitivity P_in_M*
by *simp*

lemma *Forces_Neg*:
assumes
$p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$
shows
$(p \Vdash Neg(\varphi)\ env) \longleftrightarrow \neg(\exists\, q{\in}M.\ q{\in}P \wedge q \preceq p \wedge (q \Vdash \varphi\ env))$
using *assms sats_forces_Neg' transitivity*
P_in_M pair_in_M_iff leq_in_M leq_abs **by** *simp*

33.7 The relation of forcing and connectives

lemma *Forces_Nand*:
assumes
$p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$ $\psi{\in}formula$
shows
$(p \Vdash Nand(\varphi,\psi)\ env) \longleftrightarrow \neg(\exists\, q{\in}M.\ q{\in}P \wedge q \preceq p \wedge (q \Vdash \varphi\ env) \wedge (q \Vdash \psi\ env))$
using *assms sats_forces_Nand' transitivity*
P_in_M pair_in_M_iff leq_in_M leq_abs **by** *simp*

lemma *Forces_And_aux*:
assumes
$p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$ $\psi{\in}formula$
shows
$p \Vdash And(\varphi,\psi)\ env \longleftrightarrow$
$(\forall\, q{\in}M.\ q{\in}P \wedge q \preceq p \longrightarrow (\exists\, r{\in}M.\ r{\in}P \wedge r \preceq q \wedge (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)))$
unfolding *And_def* **using** *assms Forces_Neg Forces_Nand* **by** (*auto simp only*:)

lemma *Forces_And_iff_dense_below*:
assumes
$p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$ $\psi{\in}formula$
shows
$(p \Vdash And(\varphi,\psi)\ env) \longleftrightarrow dense\_below(\{r{\in}P.\ (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)\},p)$
unfolding *dense_below_def* **using** *Forces_And_aux assms*

by (*auto dest*:*transitivity*[*OF* _ *P_in_M*]; *rename_tac q*; *drule_tac x*=*q* **in** *bspec*)+

lemma *Forces_Forall*:
assumes
p∈*P env* ∈ *list*(*M*) *φ*∈*formula*
shows
(*p* ⊩ *Forall*(*φ*) *env*) ⟷ (∀ *x*∈*M*. (*p* ⊩ *φ* ([*x*] @ *env*)))
using *sats_forces_Forall′ assms* **by** *simp*

bundle *some_rules* = *elem_of_val_pair* [*dest*] *SepReplace_iff* [*simp del*] *SepReplace_iff*[*iff*]

context
includes *some_rules*
begin

lemma *elem_of_valI*: ∃ *ϑ*. ∃ *p*∈*P*. *p*∈*G* ∧ ⟨*ϑ*,*p*⟩∈*π* ∧ *val*(*P*,*G*,*ϑ*) = *x* ⟹ *x*∈*val*(*P*,*G*,*π*)
by (*subst def_val*, *auto*)

lemma *GenExtD*: *x*∈*M*[*G*] ⟷ (∃ *τ*∈*M*. *x* = *val*(*P*,*G*,*τ*))
unfolding *GenExt_def* **by** *simp*

lemma *left_in_M* : *tau*∈*M* ⟹ ⟨*a*,*b*⟩∈*tau* ⟹ *a*∈*M*
using *fst_snd_closed*[*of* ⟨*a*,*b*⟩] *transitivity* **by** *auto*

33.8 Kunen 2013, Lemma IV.2.29

lemma *generic_inter_dense_below*:
assumes *D*∈*M M_generic*(*G*) *dense_below*(*D*,*p*) *p*∈*G*
shows *D* ∩ *G* ≠ *0*
proof -
let *?D*={*q*∈*P*. *p*⊥*q* ∨ *q*∈*D*}
have *dense*(*?D*)
proof
fix *r*
assume *r*∈*P*
show ∃ *d*∈{*q* ∈ *P* . *p* ⊥ *q* ∨ *q* ∈ *D*}. *d* ≼ *r*
proof (*cases p* ⊥ *r*)
case *True*
with ‹*r*∈*P*›

show *?thesis* **using** *refl_leq*[*of r*] **by** (*intro bexI*) (*blast*+)
next
case *False*
then
obtain *s* **where** *s*∈*P s*≼*p s*≼*r* **by** *blast*
with *assms* ‹*r*∈*P*›
show *?thesis*

```
      using dense_belowD[OF assms(3), of s] leq_transD[of _ s r]
      by blast
    qed
  qed
  have ?D⊆P by auto

  let ?d_fm=Or(Neg(compat_in_fm(1,2,3,0)),Member(0,4))
  have 1:p∈M
    using ‹M_generic(G)› M_genericD transitivity[OF _ P_in_M]
        ‹p∈G› by simp
  moreover
  have ?d_fm∈formula by simp
  moreover
 have arity(?d_fm) = 5 unfolding compat_in_fm_def pair_fm_def upair_fm_def
    by (simp add: union_abs1 Un_commute)
  moreover
   have (M, [q,P,leq,p,D] ⊨ ?d_fm) ⟷ (¬ is_compat_in(##M,P,leq,p,q) ∨
q∈D)
    if q∈M for q
    using that sats_compat_in_fm P_in_M leq_in_M 1 ‹D∈M› by simp
  moreover
  have (¬ is_compat_in(##M,P,leq,p,q) ∨ q∈D) ⟷ p⊥q ∨ q∈D if q∈M for q
    unfolding compat_def using that compat_in_abs P_in_M leq_in_M 1 by
simp
  ultimately
  have ?D∈M using Collect_in_M[of ?d_fm [P,leq,p,D]]
                  P_in_M leq_in_M ‹D∈M› by simp
  note asm = ‹M_generic(G)› ‹dense(?D)› ‹?D⊆P› ‹?D∈M›
  obtain x where x∈G x∈?D using M_generic_denseD[OF asm]
    by force
  moreover from this and ‹M_generic(G)›
  have x∈D
    using M_generic_compatD[OF _ ‹p∈G›, of x]
      refl_leq compatI[of _ p x] by force
  ultimately
  show ?thesis by auto
qed
```

33.9 Auxiliary results for Lemma IV.2.40(a)

```
lemma IV240a_mem_Collect:
  assumes
    π∈M τ∈M
  shows
    {q∈P. ∃σ. ∃r. r∈P ∧ ⟨σ,r⟩ ∈ τ ∧ q≼r ∧ q forces_a (π = σ)}∈M
proof -
  let ?rel_pred= λM x a1 a2 a3 a4. ∃σ[M]. ∃r[M]. ∃σr[M].
        r∈a1 ∧ pair(M,σ,r,σr) ∧ σr∈a4 ∧ is_leq(M,a2,x,r) ∧ is_forces_eq'(M,a1,a2,x,a3,σ)
  let ?φ=Exists(Exists(Exists(And(Member(1,4),And(pair_fm(2,1,0),
```

And(Member(0,7),And(leq_fm(5,3,1),forces_eq_fm(4,5,3,6,2)))))))
have $\sigma{\in}M \wedge r{\in}M$ **if** $\langle\sigma,\ r\rangle \in \tau$ **for** $\sigma\ r$
using *that* ‹$\tau{\in}M$› *pair_in_M_iff transitivity*[*of* $\langle\sigma,r\rangle\ \tau$] **by** *simp*
then
have *?rel_pred*(*##M*,*q*,*P*,*leq*,π,τ) $\longleftrightarrow$ ($\exists\,\sigma.\ \exists\,r.\ r{\in}P \wedge \langle\sigma,r\rangle \in \tau \wedge q{\preceq}r \wedge q$ *forces*$_a$ $(\pi = \sigma)$)
if $q{\in}M$ **for** q
unfolding *forces_eq_def* **using** *assms that P_in_M leq_in_M leq_abs forces_eq′_abs pair_in_M_iff*
by *auto*
moreover
have (M, [*q*,*P*,*leq*,π,τ] $\models$ *?φ*) $\longleftrightarrow$ *?rel_pred*(*##M*,*q*,*P*,*leq*,π,τ) **if** $q{\in}M$ **for** q
using *assms that sats_forces_eq′_fm sats_leq_fm P_in_M leq_in_M* **by** *simp*
moreover
have *?φ*∈*formula* **by** *simp*
moreover
have *arity*(*?φ*)=*5*
unfolding *leq_fm_def pair_fm_def upair_fm_def*
using *arity_forces_eq_fm* **by** (*simp add*:*ord_simp_union Un_commute*)
ultimately
show *?thesis*
unfolding *forces_eq_def* **using** *P_in_M leq_in_M assms*
Collect_in_M[*of ?φ* [*P*,*leq*,π,τ]] **by** *simp*
qed

lemma *IV240a_mem*:
assumes
M_generic(G) $p{\in}G$ $\pi{\in}M$ $\tau{\in}M$ p *forces*$_a$ $(\pi \in \tau)$
$\bigwedge q\ \sigma.\ q{\in}P \Longrightarrow q{\in}G \Longrightarrow \sigma{\in}domain(\tau) \Longrightarrow q$ *forces*$_a$ $(\pi = \sigma) \Longrightarrow$
$val(P,G,\pi) = val(P,G,\sigma)$
shows
$val(P,G,\pi){\in}val(P,G,\tau)$
proof (*intro elem_of_valI*)
let *?D*={$q{\in}P.\ \exists\,\sigma.\ \exists\,r.\ r{\in}P \wedge \langle\sigma,r\rangle \in \tau \wedge q{\preceq}r \wedge q$ *forces*$_a$ $(\pi = \sigma)$}
from ‹*M_generic*(G)› ‹$p{\in}G$›
have $p{\in}P$ **by** *blast*
moreover
note ‹$\pi{\in}M$› ‹$\tau{\in}M$›
ultimately
have *?D* $\in M$ **using** *IV240a_mem_Collect* **by** *simp*
moreover from *assms* ‹$p{\in}P$›
have *dense_below*(*?D*,p)
using *forces_mem_iff_dense_below* **by** *simp*
moreover
note ‹*M_generic*(G)› ‹$p{\in}G$›
ultimately
obtain q **where** $q{\in}G$ $q{\in}$*?D* **using** *generic_inter_dense_below* **by** *blast*
then

obtain σ r **where** $r{\in}P$ $\langle\sigma,r\rangle \in \tau$ $q{\preceq}r$ q $forces_a$ $(\pi = \sigma)$ **by** *blast*
moreover from *this* **and** ‹$q{\in}G$› *assms*
have $r \in G$ $val(P,G,\pi) = val(P,G,\sigma)$ **by** *blast+*
ultimately
show $\exists\ \sigma.\ \exists\, p{\in}P.\ p \in G \wedge \langle\sigma,\ p\rangle \in \tau \wedge val(P,G,\ \sigma) = val(P,G,\ \pi)$ **by** *auto*
qed

lemma *refl_forces_eq*:$p{\in}P \Longrightarrow p$ $forces_a$ $(x = x)$
using *def_forces_eq* **by** *simp*

lemma *forces_memI*: $\langle\sigma,r\rangle{\in}\tau \Longrightarrow p{\in}P \Longrightarrow r{\in}P \Longrightarrow p{\preceq}r \Longrightarrow p$ $forces_a$ $(\sigma \in \tau)$
using *refl_forces_eq*[*of* _ σ] *leq_transD refl_leq*
by (*blast intro*:*forces_mem_iff_dense_below*[*THEN iffD2*])

lemma *IV240a_eq_1st_incl*:
assumes
$M\_generic(G)$ $p{\in}G$ p $forces_a$ $(\tau = \vartheta)$
and
IH:$\bigwedge q\ \sigma.\ q{\in}P \Longrightarrow q{\in}G \Longrightarrow \sigma{\in}domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(q$ $forces_a$ $(\sigma \in \tau) \longrightarrow val(P,G,\sigma) \in val(P,G,\tau)) \wedge$
$(q$ $forces_a$ $(\sigma \in \vartheta) \longrightarrow val(P,G,\sigma) \in val(P,G,\vartheta))$

shows
$val(P,G,\tau) \subseteq val(P,G,\vartheta)$
proof
fix x
assume $x{\in}val(P,G,\tau)$
then
obtain σ r **where** $\langle\sigma,r\rangle{\in}\tau$ $r{\in}G$ $val(P,G,\sigma){=}x$ **by** *blast*
moreover from *this* **and** ‹$p{\in}G$› ‹$M\_generic(G)$›
obtain q **where** $q{\in}G$ $q{\preceq}p$ $q{\preceq}r$ **by** *force*
moreover from *this* **and** ‹$p{\in}G$› ‹$M\_generic(G)$›
have $q{\in}P$ $p{\in}P$ **by** *blast+*
moreover from *calculation* **and** ‹$M\_generic(G)$›
have q $forces_a$ $(\sigma \in \tau)$
using *forces_memI* **by** *blast*
moreover
note ‹p $forces_a$ $(\tau = \vartheta)$›
ultimately
have q $forces_a$ $(\sigma \in \vartheta)$
using *def_forces_eq* **by** *blast*
with ‹$q{\in}P$› ‹$q{\in}G$› *IH*[*of* q σ] ‹$\langle\sigma,r\rangle{\in}\tau$› ‹$val(P,G,\sigma) = x$›
show $x{\in}val(P,G,\vartheta)$ **by** (*blast*)
qed

lemma *IV240a_eq_2nd_incl*:
assumes
M_generic(*G*) *p*∈*G* *p forces*$_a$ (*τ* = *ϑ*)
and
IH:⋀*q σ*. *q*∈*P* ⟹ *q*∈*G* ⟹ *σ*∈*domain*(*τ*) ∪ *domain*(*ϑ*) ⟹
(*q forces*$_a$ (*σ* ∈ *τ*) ⟶ *val*(*P*,*G*,*σ*) ∈ *val*(*P*,*G*,*τ*)) ∧
(*q forces*$_a$ (*σ* ∈ *ϑ*) ⟶ *val*(*P*,*G*,*σ*) ∈ *val*(*P*,*G*,*ϑ*))
shows
val(*P*,*G*,*ϑ*) ⊆ *val*(*P*,*G*,*τ*)
proof
fix *x*
assume *x*∈*val*(*P*,*G*,*ϑ*)
then
obtain *σ r* **where** ⟨*σ*,*r*⟩∈*ϑ* *r*∈*G* *val*(*P*,*G*,*σ*)=*x* **by** *blast*
moreover from *this* **and** ‹*p*∈*G*› ‹*M_generic*(*G*)›
obtain *q* **where** *q*∈*G* *q*⪯*p* *q*⪯*r* **by** *force*
moreover from *this* **and** ‹*p*∈*G*› ‹*M_generic*(*G*)›
have *q*∈*P* *p*∈*P* **by** *blast*+
moreover from *calculation* **and** ‹*M_generic*(*G*)›
have *q forces*$_a$ (*σ* ∈ *ϑ*)
using *forces_memI* **by** *blast*
moreover
note ‹*p forces*$_a$ (*τ* = *ϑ*)›
ultimately
have *q forces*$_a$ (*σ* ∈ *τ*)
using *def_forces_eq* **by** *blast*
with ‹*q*∈*P*› ‹*q*∈*G*› *IH*[*of q σ*] ‹⟨*σ*,*r*⟩∈*ϑ*› ‹*val*(*P*,*G*,*σ*) = *x*›
show *x*∈*val*(*P*,*G*,*τ*) **by** (*blast*)
qed

lemma *IV240a_eq*:
assumes
M_generic(*G*) *p*∈*G* *p forces*$_a$ (*τ* = *ϑ*)
and
IH:⋀*q σ*. *q*∈*P* ⟹ *q*∈*G* ⟹ *σ*∈*domain*(*τ*) ∪ *domain*(*ϑ*) ⟹
(*q forces*$_a$ (*σ* ∈ *τ*) ⟶ *val*(*P*,*G*,*σ*) ∈ *val*(*P*,*G*,*τ*)) ∧
(*q forces*$_a$ (*σ* ∈ *ϑ*) ⟶ *val*(*P*,*G*,*σ*) ∈ *val*(*P*,*G*,*ϑ*))
shows
val(*P*,*G*,*τ*) = *val*(*P*,*G*,*ϑ*)
using *IV240a_eq_1st_incl*[*OF assms*] *IV240a_eq_2nd_incl*[*OF assms*] *IH* **by** *blast*

33.10 Induction on names

lemma *core_induction*:
assumes
⋀*τ ϑ p*. *p* ∈ *P* ⟹ ⟦⋀*q σ*. ⟦*q*∈*P* ; *σ*∈*domain*(*ϑ*)⟧ ⟹ *Q*(*0*,*τ*,*σ*,*q*)⟧ ⟹ *Q*(*1*,*τ*,*ϑ*,*p*)
⋀*τ ϑ p*. *p* ∈ *P* ⟹ ⟦⋀*q σ*. ⟦*q*∈*P* ; *σ*∈*domain*(*τ*) ∪ *domain*(*ϑ*)⟧ ⟹ *Q*(*1*,*σ*,*τ*,*q*)

$\wedge\ Q(1,\sigma,\vartheta,q)]\!] \Longrightarrow Q(0,\tau,\vartheta,p)$
ft ∈ *2 p* ∈ *P*
shows
Q(*ft*,τ,ϑ,*p*)
proof -
{
fix *ft p* τ ϑ
have *Transset*(*eclose*({τ,ϑ})) (**is** *Transset*(*?e*))
using *Transset_eclose* **by** *simp*
have τ ∈ *?e* ϑ ∈ *?e*
using *arg_into_eclose* **by** *simp_all*
moreover
assume *ft* ∈ *2 p* ∈ *P*
ultimately
have ⟨*ft*,τ,ϑ,*p*⟩∈ *2*×*?e*×*?e*×*P* (**is** *?a*∈*2*×*?e*×*?e*×*P*) **by** *simp*
then
have *Q*(*ftype*(*?a*), *name1*(*?a*), *name2*(*?a*), *cond_of*(*?a*))
using *core_induction_aux*[*of ?e P Q ?a*,*OF* ‹*Transset*(*?e*)› *assms*(*1*,*2*) ‹*?a*∈_›]

by (*clarify*) (*blast*)
then have *Q*(*ft*,τ,ϑ,*p*) **by** (*simp add*:*components_simp*)
}
then show *?thesis* **using** *assms* **by** *simp*
qed

lemma *forces_induction_with_conds*:
assumes
⋀τ ϑ *p*. *p* ∈ *P* ⟹ ⟦⋀*q* σ. ⟦*q*∈*P* ; σ∈*domain*(ϑ)⟧ ⟹ *Q*(*q*,τ,σ)⟧ ⟹ *R*(*p*,τ,ϑ)
⋀τ ϑ *p*. *p* ∈ *P* ⟹ ⟦⋀*q* σ. ⟦*q*∈*P* ; σ∈*domain*(τ) ∪ *domain*(ϑ)⟧ ⟹ *R*(*q*,σ,τ) ∧ *R*(*q*,σ,ϑ)⟧ ⟹ *Q*(*p*,τ,ϑ)
p ∈ *P*
shows
Q(*p*,τ,ϑ) ∧ *R*(*p*,τ,ϑ)
proof -
let *?Q*=λ*ft* τ ϑ *p*. (*ft* = *0* ⟶ *Q*(*p*,τ,ϑ)) ∧ (*ft* = *1* ⟶ *R*(*p*,τ,ϑ))
from *assms*(*1*)
have ⋀τ ϑ *p*. *p* ∈ *P* ⟹ ⟦⋀*q* σ. ⟦*q*∈*P* ; σ∈*domain*(ϑ)⟧ ⟹ *?Q*(*0*,τ,σ,*q*)⟧ ⟹ *?Q*(*1*,τ,ϑ,*p*)
by *simp*
moreover from *assms*(*2*)
have ⋀τ ϑ *p*. *p* ∈ *P* ⟹ ⟦⋀*q* σ. ⟦*q*∈*P* ; σ∈*domain*(τ) ∪ *domain*(ϑ)⟧ ⟹ *?Q*(*1*,σ,τ,*q*) ∧ *?Q*(*1*,σ,ϑ,*q*)⟧ ⟹ *?Q*(*0*,τ,ϑ,*p*)
by *simp*
moreover
note ‹*p*∈*P*›
ultimately
have *?Q*(*ft*,τ,ϑ,*p*) **if** *ft*∈*2* **for** *ft*
by (*rule core_induction*[*OF* _ _ *that*, *of ?Q*])
then

show *?thesis* **by** *auto*
qed

lemma *forces_induction*:
assumes
$\bigwedge\tau\ \vartheta.\ [\![\bigwedge\sigma.\ \sigma{\in}domain(\vartheta) \Longrightarrow Q(\tau,\sigma)]\!] \Longrightarrow R(\tau,\vartheta)$
$\bigwedge\tau\ \vartheta.\ [\![\bigwedge\sigma.\ \sigma{\in}domain(\tau) \cup domain(\vartheta) \Longrightarrow R(\sigma,\tau) \wedge R(\sigma,\vartheta)]\!] \Longrightarrow Q(\tau,\vartheta)$
shows
$Q(\tau,\vartheta) \wedge R(\tau,\vartheta)$
proof (*intro forces_induction_with_conds*[*OF* _ _ *one_in_P*])
fix $\tau\ \vartheta\ p$
assume $q \in P \Longrightarrow \sigma \in domain(\vartheta) \Longrightarrow Q(\tau,\ \sigma)$ **for** $q\ \sigma$
with *assms*(*1*)
show $R(\tau,\vartheta)$
using *one_in_P* **by** *simp*
next
fix $\tau\ \vartheta\ p$
assume $q \in P \Longrightarrow \sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow R(\sigma,\tau) \wedge R(\sigma,\vartheta)$ **for** $q\ \sigma$
with *assms*(*2*)
show $Q(\tau,\vartheta)$
using *one_in_P* **by** *simp*
qed

33.11 Lemma IV.2.40(a), in full

lemma *IV240a*:
assumes
M_generic(*G*)
shows
$(\tau{\in}M \longrightarrow \vartheta{\in}M \longrightarrow (\forall\, p{\in}G.\ p\ forces_a\ (\tau = \vartheta) \longrightarrow val(P,G,\tau) = val(P,G,\vartheta)))$
$\wedge$
$(\tau{\in}M \longrightarrow \vartheta{\in}M \longrightarrow (\forall\, p{\in}G.\ p\ forces_a\ (\tau \in \vartheta) \longrightarrow val(P,G,\tau) \in val(P,G,\vartheta)))$
(**is** *?Q*(τ,ϑ) $\wedge$ *?R*(τ,ϑ))
proof (*intro forces_induction*[*of ?Q ?R*] *impI*)
fix $\tau\ \vartheta$
assume $\tau{\in}M\ \vartheta{\in}M\ \ \sigma{\in}domain(\vartheta) \Longrightarrow ?Q(\tau,\sigma)$ **for** σ
moreover from *this*
have $\sigma{\in}domain(\vartheta) \Longrightarrow q\ forces_a\ (\tau = \sigma) \Longrightarrow val(P,G,\ \tau) = val(P,G,\ \sigma)$
if $q{\in}P\ q{\in}G$ **for** $q\ \sigma$
using *that domain_closed*[*of* ϑ] *transitivity* **by** *auto*
moreover
note *assms*
ultimately
show $\forall\, p{\in}G.\ p\ forces_a\ (\tau \in \vartheta) \longrightarrow val(P,G,\tau) \in val(P,G,\vartheta)$
using *IV240a_mem domain_closed transitivity* **by** (*simp*)
next
fix $\tau\ \vartheta$
assume $\tau{\in}M\ \vartheta{\in}M\ \sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow ?R(\sigma,\tau) \wedge ?R(\sigma,\vartheta)$ **for** σ
moreover from *this*

have IH':$\sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow q{\in}G \Longrightarrow$
$(q\ forces_a\ (\sigma \in \tau) \longrightarrow val(P,G,\sigma) \in val(P,G,\tau)) \wedge$
$(q\ forces_a\ (\sigma \in \vartheta) \longrightarrow val(P,G,\sigma) \in val(P,G,\vartheta))$ **for** $q\ \sigma$
by (*auto intro*: *transitivity*[*OF* _ *domain_closed*[*simplified*]])
ultimately
show $\forall\, p{\in}G.\ p\ forces_a\ (\tau = \vartheta) \longrightarrow val(P,G,\tau) = val(P,G,\vartheta)$
using *IV240a_eq*[*OF assms*(*1*) _ _ IH'] **by** (*simp*)
qed

33.12 Lemma IV.2.40(b)

lemma *IV240b_mem*:
assumes
$M\_generic(G)$ $val(P,G,\pi){\in}val(P,G,\tau)$ $\pi{\in}M$ $\tau{\in}M$
and
IH:$\bigwedge\sigma.\ \sigma{\in}domain(\tau) \Longrightarrow val(P,G,\pi) = val(P,G,\sigma) \Longrightarrow$
$\exists\, p{\in}G.\ p\ forces_a\ (\pi = \sigma)$
shows
$\exists\, p{\in}G.\ p\ forces_a\ (\pi \in \tau)$
proof -
from ‹$val(P,G,\pi){\in}val(P,G,\tau)$›
obtain $\sigma\ r$ **where** $r{\in}G$ $\langle\sigma,r\rangle{\in}\tau$ $val(P,G,\pi) = val(P,G,\sigma)$ **by** *auto*
moreover from *this* **and** *IH*
obtain p' **where** $p'{\in}G$ $p'\ forces_a\ (\pi = \sigma)$ **by** *blast*
moreover
note ‹$M\_generic(G)$›
ultimately
obtain p **where** $p{\preceq}r$ $p{\in}G$ $p\ forces_a\ (\pi = \sigma)$
using *M_generic_compatD strengthening_eq*[*of* p'] **by** *blast*
moreover
note ‹$M\_generic(G)$›
moreover from *calculation*
have $q\ forces_a\ (\pi = \sigma)$ **if** $q{\in}P$ $q{\preceq}p$ **for** q
using *that strengthening_eq* **by** *blast*
moreover
note ‹$\langle\sigma,r\rangle{\in}\tau$› ‹$r{\in}G$›
ultimately
have $r{\in}P \wedge \langle\sigma,r\rangle \in \tau \wedge q{\preceq}r \wedge q\ forces_a\ (\pi = \sigma)$ **if** $q{\in}P$ $q{\preceq}p$ **for** q
using *that leq_transD*[*of* _ p r] **by** *blast*
then
have *dense_below*($\{q{\in}P.\ \exists\, s\ r.\ r{\in}P \wedge \langle s,r\rangle \in \tau \wedge q{\preceq}r \wedge q\ forces_a\ (\pi = s)\}$,$p$)
using *refl_leq* **by** *blast*
moreover
note ‹$M\_generic(G)$› ‹$p{\in}G$›
moreover from *calculation*
have $p\ forces_a\ (\pi \in \tau)$
using *forces_mem_iff_dense_below* **by** *blast*
ultimately
show *?thesis* **by** *blast*

qed

end

lemma *Collect_forces_eq_in_M*:
assumes $\tau \in M$ $\vartheta \in M$
shows $\{p{\in}P.\ p\ forces_a\ (\tau = \vartheta)\} \in M$
using *assms Collect_in_M*[*of forces_eq_fm*(1,2,0,3,4) [*P*,*leq*,τ,ϑ]]
arity_forces_eq_fm P_in_M leq_in_M sats_forces_eq_fm forces_eq_abs forces_eq_fm_type
by (*simp add*: *union_abs1 Un_commute*)

lemma *IV240b_eq_Collects*:
assumes $\tau \in M$ $\vartheta \in M$
shows $\{p{\in}P.\ \exists \sigma{\in}domain(\tau) \cup domain(\vartheta).\ p\ forces_a\ (\sigma \in \tau) \wedge p\ forces_a\ (\sigma \notin \vartheta)\}{\in}M$ **and**
$\{p{\in}P.\ \exists \sigma{\in}domain(\tau) \cup domain(\vartheta).\ p\ forces_a\ (\sigma \notin \tau) \wedge p\ forces_a\ (\sigma \in \vartheta)\}{\in}M$
proof -
let *?rel_pred*=$\lambda M\ x\ a1\ a2\ a3\ a4.$
$\exists \sigma[M].\ \exists u[M].\ \exists da3[M].\ \exists da4[M].\ is\_domain(M,a3,da3) \wedge is\_domain(M,a4,da4) \wedge$
$union(M,da3,da4,u) \wedge \sigma{\in}u \wedge is\_forces\_mem'(M,a1,a2,x,\sigma,a3) \wedge$
$is\_forces\_nmem'(M,a1,a2,x,\sigma,a4)$
let *?*φ=*Exists*(*Exists*(*Exists*(*Exists*(*And*(*domain_fm*(7,1),*And*(*domain_fm*(8,0),
And(*union_fm*(1,0,2),*And*(*Member*(3,2),*And*(*forces_mem_fm*(5,6,4,3,7),
forces_nmem_fm(5,6,4,3,8)))))))))))
have *1*:$\sigma{\in}M$ **if** $\langle\sigma,y\rangle{\in}\delta$ $\delta{\in}M$ **for** σ δ y
using *that pair_in_M_iff transitivity*[*of* $\langle\sigma,y\rangle$ δ] **by** *simp*
have *abs1*:*?rel_pred*(##*M*,*p*,*P*,*leq*,τ,ϑ) $\longleftrightarrow$
$(\exists \sigma{\in}domain(\tau) \cup domain(\vartheta).\ forces\_mem'(P,leq,p,\sigma,\tau) \wedge forces\_nmem'(P,leq,p,\sigma,\vartheta))$
if $p{\in}M$ **for** p
unfolding *forces_mem_def forces_nmem_def*
using *assms that forces_mem'_abs forces_nmem'_abs P_in_M leq_in_M domain_closed Un_closed*
by (*auto simp add*:*1*[*of* _ _ τ] *1*[*of* _ _ ϑ])
have *abs2*:*?rel_pred*(##*M*,*p*,*P*,*leq*,ϑ,τ) $\longleftrightarrow$ $(\exists \sigma{\in}domain(\tau) \cup domain(\vartheta).$
$forces\_nmem'(P,leq,p,\sigma,\tau) \wedge forces\_mem'(P,leq,p,\sigma,\vartheta))$ **if** $p{\in}M$ **for** p
unfolding *forces_mem_def forces_nmem_def*
using *assms that forces_mem'_abs forces_nmem'_abs P_in_M leq_in_M domain_closed Un_closed*
by (*auto simp add*:*1*[*of* _ _ τ] *1*[*of* _ _ ϑ])
have *fsats1*:$(M,[p,P,leq,\tau,\vartheta] \models ?\varphi) \longleftrightarrow$ *?rel_pred*(##*M*,*p*,*P*,*leq*,τ,ϑ) **if** $p{\in}M$ **for** p
using *that assms sats_forces_mem'_fm sats_forces_nmem'_fm P_in_M leq_in_M domain_closed Un_closed* **by** *simp*
have *fsats2*:$(M,[p,P,leq,\vartheta,\tau] \models ?\varphi) \longleftrightarrow$ *?rel_pred*(##*M*,*p*,*P*,*leq*,ϑ,τ) **if** $p{\in}M$

for p
using *that assms sats_forces_mem'_fm sats_forces_nmem'_fm P_in_M leq_in_M*
domain_closed Un_closed **by** *simp*
have *fty*:$?\varphi \in formula$ **by** *simp*
have *farit*:$arity(?\varphi)=5$
unfolding *forces_nmem_fm_def domain_fm_def pair_fm_def upair_fm_def union_fm_def*
using *arity_forces_mem_fm* **by** (*simp add*:*ord_simp_union Un_commute*)
show
$\{p \in P \,.\, \exists \sigma \in domain(\tau) \cup domain(\vartheta).\ p\ forces_a\ (\sigma \in \tau) \wedge p\ forces_a\ (\sigma \notin \vartheta)\} \in M$
and $\{p \in P \,.\, \exists \sigma \in domain(\tau) \cup domain(\vartheta).\ p\ forces_a\ (\sigma \notin \tau) \wedge p\ forces_a\ (\sigma \in \vartheta)\} \in M$
unfolding *forces_mem_def*
using *abs1 fty fsats1 farit P_in_M leq_in_M assms forces_nmem*
Collect_in_M[*of* $?\varphi$ [P,leq,τ,ϑ]]
using *abs2 fty fsats2 farit P_in_M leq_in_M assms forces_nmem domain_closed Un_closed*
Collect_in_M[*of* $?\varphi$ [P,leq,ϑ,τ]]
by *simp_all*
qed

lemma *IV240b_eq*:
assumes
$M\_generic(G)$ $val(P,G,\tau) = val(P,G,\vartheta)$ $\tau \in M$ $\vartheta \in M$
and
IH:$\bigwedge \sigma.\ \sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(val(P,G,\sigma) \in val(P,G,\tau) \longrightarrow (\exists\, q \in G.\ q\ forces_a\ (\sigma \in \tau))) \wedge$
$(val(P,G,\sigma) \in val(P,G,\vartheta) \longrightarrow (\exists\, q \in G.\ q\ forces_a\ (\sigma \in \vartheta)))$

shows
$\exists\, p \in G.\ p\ forces_a\ (\tau = \vartheta)$
proof -
let $?D1 = \{p \in P.\ p\ forces_a\ (\tau = \vartheta)\}$
let $?D2 = \{p \in P.\ \exists \sigma \in domain(\tau) \cup domain(\vartheta).\ p\ forces_a\ (\sigma \in \tau) \wedge p\ forces_a\ (\sigma \notin \vartheta)\}$
let $?D3 = \{p \in P.\ \exists \sigma \in domain(\tau) \cup domain(\vartheta).\ p\ forces_a\ (\sigma \notin \tau) \wedge p\ forces_a\ (\sigma \in \vartheta)\}$
let $?D = ?D1 \cup ?D2 \cup ?D3$
note *assms*
moreover from *this*
have $domain(\tau) \cup domain(\vartheta) \in M$ (**is** $?B \in M$) **using** *domain_closed Un_closed* **by** *auto*
moreover from *calculation*
have $?D2 \in M$ **and** $?D3 \in M$ **using** *IV240b_eq_Collects* **by** *simp_all*
ultimately
have $?D \in M$ **using** *Collect_forces_eq_in_M Un_closed* **by** *auto*

moreover
have *dense(?D)*
proof
  fix *p*
  assume *p∈P*
  have ∃ *d∈P*. (*d forces$_a$* (*τ* = *ϑ*) ∨
        (∃ *σ∈domain(τ)* ∪ *domain(ϑ)*. *d forces$_a$* (*σ* ∈ *τ*) ∧ *d forces$_a$* (*σ* ∉ *ϑ*)) ∨
        (∃ *σ∈domain(τ)* ∪ *domain(ϑ)*. *d forces$_a$* (*σ* ∉ *τ*) ∧ *d forces$_a$* (*σ* ∈ *ϑ*))) ∧
       d ≼ *p*
  proof (*cases p forces$_a$* (*τ* = *ϑ*))
    case *True*
    with ‹*p∈P*›
    show *?thesis* **using** *refl_leq* **by** *blast*
  next
    case *False*
    moreover note ‹*p∈P*›
    moreover from *calculation*
    obtain *σ q* **where** *σ∈domain(τ)∪domain(ϑ) q∈P q≼p*
      (*q forces$_a$* (*σ* ∈ *τ*) ∧ ¬ *q forces$_a$* (*σ* ∈ *ϑ*)) ∨
       (¬ *q forces$_a$* (*σ* ∈ *τ*) ∧ *q forces$_a$* (*σ* ∈ *ϑ*))
      using *def_forces_eq* **by** *blast*
    moreover from *this*
    obtain *r* **where** *r≼q r∈P*
      (*r forces$_a$* (*σ* ∈ *τ*) ∧ *r forces$_a$* (*σ* ∉ *ϑ*)) ∨
       (*r forces$_a$* (*σ* ∉ *τ*) ∧ *r forces$_a$* (*σ* ∈ *ϑ*))
      using *not_forces_nmem strengthening_mem* **by** *blast*
    ultimately
    show *?thesis* **using** *leq_transD* **by** *blast*
  qed
  then
  show ∃ *d∈?D1* ∪ *?D2* ∪ *?D3*. *d* ≼ *p* **by** *blast*
qed
moreover
have *?D* ⊆ *P*
  by *auto*
moreover
note ‹*M_generic(G)*›
ultimately
obtain *p* **where** *p∈G p∈?D*
  unfolding *M_generic_def* **by** *blast*
then
consider
  (*1*) *p forces$_a$* (*τ* = *ϑ*) |
  (*2*) ∃ *σ∈domain(τ)* ∪ *domain(ϑ)*. *p forces$_a$* (*σ* ∈ *τ*) ∧ *p forces$_a$* (*σ* ∉ *ϑ*) |
  (*3*) ∃ *σ∈domain(τ)* ∪ *domain(ϑ)*. *p forces$_a$* (*σ* ∉ *τ*) ∧ *p forces$_a$* (*σ* ∈ *ϑ*)
  by *blast*
then
show *?thesis*
proof (*cases*)

```
    case 1
    with ‹p∈G›
    show ?thesis by blast
  next
    case 2
    then
    obtain σ where σ∈domain(τ) ∪ domain(ϑ) p forces_a (σ ∈ τ) p forces_a (σ ∉
ϑ)
      by blast
    moreover from this and ‹p∈G› and assms
    have val(P,G,σ)∈val(P,G,τ)
     using IV240a[of G σ τ] transitivity[OF _ domain_closed[simplified]] by blast
    moreover note IH ‹val(P,G,τ) = _›
    ultimately
    obtain q where q∈G q forces_a (σ ∈ ϑ) by auto
    moreover from this and ‹p∈G› ‹M_generic(G)›
    obtain r where r∈P r≼p r≼q
      by blast
    moreover
    note ‹M_generic(G)›
    ultimately
    have r forces_a (σ ∈ ϑ)
      using strengthening_mem by blast
    with ‹r≼p› ‹p forces_a (σ ∉ ϑ)› ‹r∈P›
    have False
      unfolding forces_nmem_def by blast
    then
    show ?thesis by simp
  next
    case 3
    then
    obtain σ where σ∈domain(τ) ∪ domain(ϑ) p forces_a (σ ∈ ϑ) p forces_a (σ ∉
τ)
      by blast
    moreover from this and ‹p∈G› and assms
    have val(P,G,σ)∈val(P,G,ϑ)
     using IV240a[of G σ ϑ] transitivity[OF _ domain_closed[simplified]] by blast
    moreover note IH ‹val(P,G,τ) = _›
    ultimately
    obtain q where q∈G q forces_a (σ ∈ τ) by auto
    moreover from this and ‹p∈G› ‹M_generic(G)›
    obtain r where r∈P r≼p r≼q
      by blast
    moreover
    note ‹M_generic(G)›
    ultimately
    have r forces_a (σ ∈ τ)
      using strengthening_mem by blast
    with ‹r≼p› ‹p forces_a (σ ∉ τ)› ‹r∈P›
```

have *False*
unfolding *forces_nmem_def* **by** *blast*
then
show *?thesis* **by** *simp*
qed
qed

lemma *IV240b*:
assumes
M_generic(G)
shows
(τ∈M⟶ϑ∈M⟶*val*(P,G,τ) = *val*(P,G,ϑ) ⟶ (∃ p∈G. p *forces*$_a$ ($\tau = \vartheta$))) ∧
(τ∈M⟶ϑ∈M⟶*val*(P,G,τ) ∈ *val*(P,G,ϑ) ⟶ (∃ p∈G. p *forces*$_a$ ($\tau \in \vartheta$)))
(**is** *?Q*(τ,ϑ) ∧ *?R*(τ,ϑ))
proof (*intro forces_induction*)
fix τ ϑ p
assume σ∈*domain*(ϑ) ⟹ *?Q*(τ, σ) **for** σ
with *assms*
show *?R*(τ, ϑ)
using *IV240b_mem domain_closed transitivity* **by** (*simp*)
next
fix τ ϑ p
assume σ ∈ *domain*(τ) ∪ *domain*(ϑ) ⟹ *?R*(σ,τ) ∧ *?R*(σ,ϑ) **for** σ
moreover from *this*
have *IH'*:τ∈M ⟹ ϑ∈M ⟹ σ ∈ *domain*(τ) ∪ *domain*(ϑ) ⟹
(*val*(P,G, σ) ∈ *val*(P,G, τ) ⟶ (∃ q∈G. q *forces*$_a$ ($\sigma \in \tau$))) ∧
(*val*(P,G, σ) ∈ *val*(P,G, ϑ) ⟶ (∃ q∈G. q *forces*$_a$ ($\sigma \in \vartheta$))) **for** σ
by (*blast intro*:*left_in_M*)
ultimately
show *?Q*(τ,ϑ)
using *IV240b_eq*[*OF assms*(*1*)] **by** (*auto*)
qed

lemma *map_val_in_MG*:
assumes
env∈*list*(M)
shows
map(*val*(P,G),*env*)∈*list*(M[G])
unfolding *GenExt_def* **using** *assms map_type2* **by** *simp*

lemma *truth_lemma_mem*:
assumes
env∈*list*(M) *M_generic*(G)
n∈*nat* m∈*nat* n<*length*(*env*) m<*length*(*env*)
shows
(∃ p∈G. p ⊩ *Member*(n,m) *env*) ⟷ M[G], *map*(*val*(P,G),*env*) ⊨ *Member*(n,m)
using *assms IV240a*[*OF assms*(*2*), *of nth*(n,*env*) *nth*(m,*env*)]
IV240b[*OF assms*(*2*), *of nth*(n,*env*) *nth*(m,*env*)]

P_in_M leq_in_M one_in_M
Forces_Member[of _ nth(n,env) nth(m,env) env n m] map_val_in_MG
by (*auto*)

lemma *truth_lemma_eq*:
assumes
env∈list(M) M_generic(G)
n∈nat m∈nat n<length(env) m<length(env)
shows
(∃ p∈G. p ⊩ Equal(n,m) env) ⟷ M[G], map(val(P,G),env) ⊨ Equal(n,m)
using *assms IV240a(1)[OF assms(2), of nth(n,env) nth(m,env)]*
IV240b(1)[OF assms(2), of nth(n,env) nth(m,env)]
P_in_M leq_in_M one_in_M
Forces_Equal[of _ nth(n,env) nth(m,env) env n m] map_val_in_MG
by (*auto*)

lemma *arities_at_aux*:
assumes
n ∈ nat m ∈ nat env ∈ list(M) succ(n) ∪ succ(m) ≤ length(env)
shows
n < length(env) m < length(env)
using *assms succ_leE[OF Un_leD1, of n succ(m) length(env)]*
succ_leE[OF Un_leD2, of succ(n) m length(env)] **by** *auto*

33.13 The Strenghtening Lemma

lemma *strengthening_lemma*:
assumes
p∈P φ∈formula r∈P r≼p
env∈list(M) arity(φ)≤length(env)
shows
p ⊩ φ env ⟹ r ⊩ φ env
using *assms(2-)*
proof (*induct arbitrary*:*env*)
case (*Member n m*)
then
have *n<length(env) m<length(env)*
using *arities_at_aux* **by** *simp_all*
moreover
assume *env∈list(M)*
moreover
note *assms Member*
ultimately
show *?case*
using *Forces_Member[of _ nth(n,env) nth(m,env) env n m]*
strengthening_mem[of p r nth(n,env) nth(m,env)] **by** *simp*
next
case (*Equal n m*)
then

```
  have n<length(env) m<length(env)
    using arities_at_aux by simp_all
  moreover
  assume env∈list(M)
  moreover
  note assms Equal
  ultimately
  show ?case
    using Forces_Equal[of _ nth(n,env) nth(m,env) env n m]
      strengthening_eq[of p r nth(n,env) nth(m,env)] by simp
next
  case (Nand φ ψ)
  with assms
  show ?case
    using Forces_Nand transitivity[OF _ P_in_M] pair_in_M_iff
      transitivity[OF _ leq_in_M] leq_transD by auto
next
  case (Forall φ)
  with assms
  have p ⊩ φ ([x] @ env) if x∈M for x
    using that Forces_Forall by simp
  with Forall
  have r ⊩ φ ([x] @ env) if x∈M for x
    using that pred_le2 by (simp)
  with assms Forall
  show ?case
    using Forces_Forall by simp
qed
```

33.14 The Density Lemma

```
lemma arity_Nand_le:
  assumes φ ∈ formula ψ ∈ formula arity(Nand(φ, ψ)) ≤ length(env) env∈list(A)
  shows arity(φ) ≤ length(env) arity(ψ) ≤ length(env)
  using assms
  by (rule_tac Un_leD1, rule_tac [5] Un_leD2, auto)

lemma dense_below_imp_forces:
  assumes
    p∈P φ∈formula
    env∈list(M) arity(φ)≤length(env)
  shows
    dense_below({q∈P. (q ⊩ φ env)},p) ⟹ (p ⊩ φ env)
  using assms(2-)
proof (induct arbitrary:env)
  case (Member n m)
  then
  have n<length(env) m<length(env)
    using arities_at_aux by simp_all
```

```
  moreover
  assume env∈list(M)
  moreover
  note assms Member
  ultimately
  show ?case
    using Forces_Member[of _ nth(n,env) nth(m,env) env n m]
      density_mem[of p nth(n,env) nth(m,env)] by simp
next
  case (Equal n m)
  then
  have n<length(env) m<length(env)
    using arities_at_aux by simp_all
  moreover
  assume env∈list(M)
  moreover
  note assms Equal
  ultimately
  show ?case
    using Forces_Equal[of _ nth(n,env) nth(m,env) env n m]
      density_eq[of p nth(n,env) nth(m,env)] by simp
next
case (Nand φ ψ)
  {
    fix q
    assume q∈M q∈P q≼ p q ⊩ φ env
    moreover
    note Nand
    moreover from calculation
    obtain d where d∈P d ⊩ Nand(φ, ψ) env d≼ q
      using dense_belowI by auto
    moreover from calculation
    have ¬(d⊩ ψ env) if d ⊩ φ env
      using that Forces_Nand refl_leq transitivity[OF _ P_in_M, of d] by auto
    moreover
    note arity_Nand_le[of φ ψ]
    moreover from calculation
    have d ⊩ φ env
       using strengthening_lemma[of q φ d env] Un_leD1 by auto
    ultimately
    have ¬ (q ⊩ ψ env)
      using strengthening_lemma[of q ψ d env] by auto
  }
  with ‹p∈P›
  show ?case
    using Forces_Nand[symmetric, OF _ Nand(6,1,3)] by blast
next
  case (Forall φ)
  have dense_below({q∈P. q ⊩ φ ([a]@env)},p) if a∈M for a
```

proof
fix r
assume $r{\in}P$ $r{\preceq}p$
with ‹*dense_below*(_,p)›
obtain q **where** $q{\in}P$ $q{\preceq}r$ $q \Vdash \mathit{Forall}(\varphi)$ *env*
by *blast*
moreover
note *Forall* ‹$a{\in}M$›
moreover from *calculation*
have $q \Vdash \varphi$ ([a]@*env*)
using *Forces_Forall* **by** *simp*
ultimately
show $\exists\, d \in \{q{\in}P.\ q \Vdash \varphi\ ([a]@\mathit{env})\}.\ d \in P \wedge d{\preceq}r$
by *auto*
qed
moreover
note *Forall*(*2*)[*of Cons*(_,*env*)] *Forall*(*1*,*3-5*)
ultimately
have $p \Vdash \varphi$ ([a]@*env*) **if** $a{\in}M$ **for** a
using *that pred_le2* **by** *simp*
with *assms Forall*
show *?case* **using** *Forces_Forall* **by** *simp*
qed

lemma *density_lemma*:
assumes
$p{\in}P$ $\varphi{\in}\mathit{formula}$ $\mathit{env}{\in}\mathit{list}(M)$ $\mathit{arity}(\varphi){\le}\mathit{length}(\mathit{env})$
shows
$p \Vdash \varphi\ \mathit{env} \longleftrightarrow \mathit{dense\_below}(\{q{\in}P.\ (q \Vdash \varphi\ \mathit{env})\},p)$
proof
assume *dense_below*($\{q{\in}P.\ (q \Vdash \varphi\ \mathit{env})\}$,$p$)
with *assms*
show $(p \Vdash \varphi\ \mathit{env})$
using *dense_below_imp_forces* **by** *simp*
next
assume $p \Vdash \varphi\ \mathit{env}$
with *assms*
show *dense_below*($\{q{\in}P.\ q \Vdash \varphi\ \mathit{env}\}$,$p$)
using *strengthening_lemma refl_leq* **by** *auto*
qed

33.15 The Truth Lemma

lemma *Forces_And*:
assumes
$p{\in}P$ $\mathit{env} \in \mathit{list}(M)$ $\varphi{\in}\mathit{formula}$ $\psi{\in}\mathit{formula}$
$\mathit{arity}(\varphi) \le \mathit{length}(\mathit{env})$ $\mathit{arity}(\psi) \le \mathit{length}(\mathit{env})$
shows
$p \Vdash \mathit{And}(\varphi,\psi)\ \mathit{env} \longleftrightarrow (p \Vdash \varphi\ \mathit{env}) \wedge (p \Vdash \psi\ \mathit{env})$

proof
assume $p \Vdash And(\varphi, \psi)\ env$
with *assms*
have $dense\_below(\{r \in P\ .\ (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)\},\ p)$
using *Forces_And_iff_dense_below* **by** *simp*
then
have $dense\_below(\{r \in P\ .\ (r \Vdash \varphi\ env)\},\ p)\ dense\_below(\{r \in P\ .\ (r \Vdash \psi\ env)\},\ p)$
by *blast+*
with *assms*
show $(p \Vdash \varphi\ env) \wedge (p \Vdash \psi\ env)$
using *density_lemma*[*symmetric*] **by** *simp*
next
assume $(p \Vdash \varphi\ env) \wedge (p \Vdash \psi\ env)$
have $dense\_below(\{r \in P\ .\ (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)\},\ p)$
proof (*intro dense_belowI bexI conjI*, *assumption*)
fix q
assume $q{\in}P\ q{\preceq}\ p$
with *assms* ‹$(p \Vdash \varphi\ env) \wedge (p \Vdash \psi\ env)$›
show $q{\in}\{r \in P\ .\ (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)\}\ q{\preceq}\ q$
using *strengthening_lemma refl_leq* **by** *auto*
qed
with *assms*
show $p \Vdash And(\varphi,\psi)\ env$
using *Forces_And_iff_dense_below* **by** *simp*
qed

lemma *Forces_Nand_alt*:
assumes
$p{\in}P\ env \in list(M)\ \varphi{\in}formula\ \psi{\in}formula$
$arity(\varphi) \leq length(env)\ arity(\psi) \leq length(env)$
shows
$(p \Vdash Nand(\varphi,\psi)\ env) \longleftrightarrow (p \Vdash Neg(And(\varphi,\psi))\ env)$
using *assms Forces_Nand Forces_And Forces_Neg* **by** *auto*

lemma *truth_lemma_Neg*:
assumes
$\varphi{\in}formula\ M\_generic(G)\ env{\in}list(M)\ arity(\varphi){\leq}length(env)$ **and**
IH: $(\exists\, p{\in}G.\ p \Vdash \varphi\ env) \longleftrightarrow M[G],\ map(val(P,G),env) \models \varphi$
shows
$(\exists\, p{\in}G.\ p \Vdash Neg(\varphi)\ env) \longleftrightarrow M[G],\ map(val(P,G),env) \models Neg(\varphi)$
proof (*intro iffI*, *elim bexE*, *rule ccontr*)

fix p
assume $p{\in}G\ p \Vdash Neg(\varphi)\ env\ \neg(M[G],map(val(P,G),env) \models Neg(\varphi))$
moreover
note *assms*
moreover from *calculation*
have $M[G],\ map(val(P,G),env) \models \varphi$

```
    using map_val_in_MG by simp
  with IH
  obtain r where r ⊩ φ env r∈G by blast
  moreover from this and ‹M_generic(G)› ‹p∈G›
  obtain q where q≼p q≼r q∈G
    by blast
  moreover from calculation
  have q ⊩ φ env
    using strengthening_lemma[where φ=φ] by blast
  ultimately
  show False
    using Forces_Neg[where φ=φ] transitivity[OF _ P_in_M] by blast
next
  assume M[G], map(val(P,G),env) ⊨ Neg(φ)
  with assms
  have ¬ (M[G], map(val(P,G),env) ⊨ φ)
    using map_val_in_MG by simp
  let ?D={p∈P. (p ⊩ φ env) ∨ (p ⊩ Neg(φ) env)}
  have separation(##M,λp. (p ⊩ φ env))
       using separation_ax[of forces(φ)] arity_forces assms P_in_M leq_in_M
one_in_M arity_forces_le
    by simp
  moreover
  have separation(##M,λp. (p ⊩ ·¬φ· env))
     using separation_ax[of forces( ·¬φ· )] arity_forces assms P_in_M leq_in_M
one_in_M arity_forces_le
    by simp
  ultimately
  have separation(##M,λp. (p ⊩ φ env) ∨ (p ⊩ Neg(φ) env))
    using separation_disj by simp
  then
  have ?D ∈ M
    using separation_closed P_in_M by simp
  moreover
  have ?D ⊆ P by auto
  moreover
  have dense(?D)
  proof
    fix q
    assume q∈P
    show ∃ d∈{p ∈ P . (p ⊩ φ env) ∨ (p ⊩ Neg(φ) env)}. d≼ q
    proof (cases q ⊩ Neg(φ) env)
      case True
      with ‹q∈P›
      show ?thesis using refl_leq by blast
    next
      case False
      with ‹q∈P› and assms
      show ?thesis using Forces_Neg by auto
```

```
    qed
  qed
  moreover
  note ‹M_generic(G)›
  ultimately
  obtain p where p∈G (p ⊩ φ env) ∨ (p ⊩ Neg(φ) env)
    by blast
  then
  consider (1) p ⊩ φ env | (2) p ⊩ Neg(φ) env by blast
  then
  show ∃ p∈G. (p ⊩ Neg(φ) env)
  proof (cases)
    case 1
    with ‹¬ (M[G],map(val(P,G),env) ⊨ φ)› ‹p∈G› IH
    show ?thesis
      by blast
  next
    case 2
    with ‹p∈G›
    show ?thesis by blast
  qed
qed

lemma truth_lemma_And:
  assumes
    env∈list(M) φ∈formula ψ∈formula
    arity(φ)≤length(env) arity(ψ) ≤ length(env) M_generic(G)
    and
    IH: (∃ p∈G. p ⊩ φ env)  ⟷   M[G], map(val(P,G),env) ⊨ φ
        (∃ p∈G. p ⊩ ψ env)  ⟷   M[G], map(val(P,G),env) ⊨ ψ
  shows
    (∃ p∈G. (p ⊩ And(φ,ψ) env)) ⟷ M[G] , map(val(P,G),env) ⊨ And(φ,ψ)
  using assms map_val_in_MG Forces_And[OF M_genericD assms(1-5)]
proof (intro iffI, elim bexE)
  fix p
  assume p∈G p ⊩ And(φ,ψ) env
  with assms
  show M[G], map(val(P,G),env) ⊨ And(φ,ψ)
    using Forces_And[OF M_genericD, of _ _ _ φ ψ] map_val_in_MG by auto
next
  assume M[G], map(val(P,G),env) ⊨ And(φ,ψ)
  moreover
  note assms
  moreover from calculation
  obtain q r where q ⊩ φ env r ⊩ ψ env q∈G r∈G
    using map_val_in_MG Forces_And[OF M_genericD assms(1-5)] by auto
  moreover from calculation
  obtain p where p≼q p≼r p∈G
    by blast
```

```
  moreover from calculation
  have (p ⊩ φ env) ∧ (p ⊩ ψ env)
    using strengthening_lemma by (blast)
  ultimately
  show ∃p∈G. (p ⊩ And(φ,ψ) env)
    using Forces_And[OF M_genericD assms(1-5)] by auto
qed

definition
  ren_truth_lemma :: i⇒i where
  ren_truth_lemma(φ) ≡
    Exists(Exists(Exists(Exists(Exists(
   And(Equal(0,5),And(Equal(1,8),And(Equal(2,9),And(Equal(3,10),And(Equal(4,6),
    iterates(λp. incr_bv(p)`5 , 6, φ))))))))))

lemma ren_truth_lemma_type[TC] :
  φ∈formula ⟹ ren_truth_lemma(φ) ∈formula
  unfolding ren_truth_lemma_def
  by simp

lemma arity_ren_truth :
  assumes φ∈formula
  shows arity(ren_truth_lemma(φ)) ≤ 6 ∪ succ(arity(φ))
proof -
  consider (lt) 5 <arity(φ) | (ge) ¬ 5 < arity(φ)
    by auto
  then
  show ?thesis
  proof cases
    case lt
    consider (a) 5<arity(φ)#+5 | (b) arity(φ)#+5 ≤ 5
      using not_lt_iff_le ‹φ∈_› by force
    then
    show ?thesis
    proof cases
      case a
      with ‹φ∈_› lt
      have 5 < succ(arity(φ)) 5<arity(φ)#+2  5<arity(φ)#+3  5<arity(φ)#+4
        using succ_ltI by auto
       with ‹φ∈_›
      have c:arity(iterates(λp. incr_bv(p)`5,5,φ)) = 5#+arity(φ) (is arity(?φ') =
_)
        using arity_incr_bv_lemma lt a
        by simp
      with ‹φ∈_›
      have arity(incr_bv(?φ')`5) = 6#+arity(φ)
        using arity_incr_bv_lemma[of ?φ' 5] a by auto
      with ‹φ∈_›
      show ?thesis
```

```
      unfolding ren_truth_lemma_def
      using pred_Un_distrib union_abs1 Un_assoc[symmetric] a c union_abs2
      by simp
    next
     case b
     with ‹φ∈_› lt
     have 5 < succ(arity(φ)) 5<arity(φ)#+2  5<arity(φ)#+3  5<arity(φ)#+4
5<arity(φ)#+5
      using succ_ltI by auto
     with ‹φ∈_›
      have arity(iterates(λp. incr_bv(p)‘5,6,φ)) = 6#+arity(φ) (is arity(?φ') =
_)
      using arity_incr_bv_lemma lt
      by simp
     with ‹φ∈_›
     show ?thesis
      unfolding ren_truth_lemma_def
      using pred_Un_distrib union_abs1 Un_assoc[symmetric]  union_abs2
      by simp
    qed
  next
    case ge
    with ‹φ∈_›
    have arity(φ) ≤ 5 pred^5(arity(φ)) ≤ 5
      using not_lt_iff_le le_trans[OF le_pred]
      by auto
    with ‹φ∈_›
   have arity(iterates(λp. incr_bv(p)‘5,6,φ)) = arity(φ) arity(φ)≤6 pred^5(arity(φ))
≤ 6
        using arity_incr_bv_lemma ge le_trans[OF ‹arity(φ)≤5›] le_trans[OF
‹pred^5(arity(φ))≤5›]
      by auto
    with ‹arity(φ) ≤ 5› ‹φ∈_› ‹pred^5(_) ≤ 5›
    show ?thesis
      unfolding ren_truth_lemma_def
      using  pred_Un_distrib union_abs1 Un_assoc[symmetric] union_abs2
      by simp
  qed
qed

lemma sats_ren_truth_lemma:
  [q,b,d,a1,a2,a3] @ env ∈ list(M) ⟹ φ ∈ formula ⟹
   (M, [q,b,d,a1,a2,a3] @ env ⊨ ren_truth_lemma(φ) ) ⟷
   (M, [q,a1,a2,a3,b] @ env ⊨ φ)
  unfolding ren_truth_lemma_def
  by (insert sats_incr_bv_iff [of _ _ M _ [q,a1,a2,a3,b]], simp)

lemma truth_lemma' :
  assumes
```

```
    φ∈formula env∈list(M) arity(φ) ≤ succ(length(env))
  shows
    separation(##M,λd. ∃ b∈M. ∀ q∈P. q⪯d ⟶ ¬(q ⊩ φ ([b]@env)))
proof -
  let ?rel_pred=λM x a1 a2 a3. ∃ b∈M. ∀ q∈M. q∈a1 ∧ is_leq(##M,a2,q,x) ⟶
                ¬(M, [q,a1,a2,a3,b] @ env ⊨ forces(φ))
  let ?ψ=Exists(Forall(Implies(And(Member(0,3),leq_fm(4,0,2)),
        Neg(ren_truth_lemma(forces(φ))))))
  have q∈M if q∈P for q using that transitivity[OF _ P_in_M] by simp
  then
  have 1:∀ q∈M. q∈P ∧ R(q) ⟶ Q(q) ⟹ (∀ q∈P. R(q) ⟶ Q(q)) for R Q
    by auto
  then
  have ⟦b ∈ M; ∀ q∈M. q ∈ P ∧ q ⪯ d ⟶ ¬(q ⊩ φ ([b]@env))⟧ ⟹
       ∃ c∈M. ∀ q∈P. q ⪯ d ⟶ ¬(q ⊩ φ ([c]@env)) for b d
    by (rule bexI,simp_all)
  then
 have ?rel_pred(M,d,P,leq,one) ⟷ (∃ b∈M. ∀ q∈P. q⪯d ⟶ ¬(q ⊩ φ ([b]@env)))
if d∈M for d
    using that leq_abs leq_in_M P_in_M one_in_M assms
    by auto
  moreover
  have ?ψ∈formula using assms by simp
  moreover
  have (M, [d,P,leq,one]@env ⊨ ?ψ) ⟷ ?rel_pred(M,d,P,leq,one) if d∈M for d
   using assms that P_in_M leq_in_M one_in_M sats_leq_fm sats_ren_truth_lemma
    by simp
  moreover
  have arity(?ψ) ≤ 4##+length(env)
  proof -
    have eq:arity(leq_fm(4, 0, 2)) = 5
      using arity_leq_fm succ_Un_distrib ord_simp_union
      by simp
    with ‹φ∈_›
    have arity(?ψ) = 3 ∪ (pred^2(arity(ren_truth_lemma(forces(φ)))))
      using union_abs1 pred_Un_distrib by simp
    moreover
    have ... ≤ 3 ∪ (pred(pred(6 ∪ succ(arity(forces(φ)))))) (is _ ≤ ?r)
      using ‹φ∈_› Un_le_compat[OF le_refl[of 3]]
                  le_imp_subset arity_ren_truth[of forces(φ)]
                  pred_mono
      by auto
    finally
    have arity(?ψ) ≤ ?r by simp
    have i:?r ≤ 4 ∪ pred(arity(forces(φ)))
    using pred_Un_distrib pred_succ_eq ‹φ∈_› Un_assoc[symmetric] union_abs1
by simp
    have h:4 ∪ pred(arity(forces(φ))) ≤ 4 ∪ (4##+length(env))
      using ‹env∈_› add_commute ‹φ∈_›
```

```
        Un_le_compat[of 4 4,OF _ pred_mono[OF _ arity_forces_le[OF _ _
‹arity(φ)≤_›]] ]
        ‹env∈_› by auto
    with ‹φ∈_› ‹env∈_›
    show ?thesis
      using le_trans[OF ‹arity(?ψ) ≤ ?r› le_trans[OF i h]] ord_simp_union by
simp
  qed
  ultimately
  show ?thesis using assms P_in_M leq_in_M one_in_M
      separation_ax[of ?ψ [P,leq,one]@env]
      separation_cong[of ##M λy. (M, [y,P,leq,one]@env ⊨ ?ψ)]
    by simp
qed


lemma truth_lemma:
  assumes
    φ∈formula M_generic(G)
    env∈list(M) arity(φ)≤length(env)
  shows
    (∃ p∈G. p ⊩ φ env)  ⟷  M[G], map(val(P,G),env) ⊨ φ
  using assms
proof (induct arbitrary:env)
  case (Member x y)
  then
  show ?case
    using assms truth_lemma_mem[OF ‹env∈list(M)› assms(2) ‹x∈nat› ‹y∈nat›]
      arities_at_aux by simp
next
  case (Equal x y)
  then
  show ?case
    using assms truth_lemma_eq[OF ‹env∈list(M)› assms(2) ‹x∈nat› ‹y∈nat›]
      arities_at_aux by simp
next
  case (Nand φ ψ)
  moreover
  note ‹M_generic(G)›
  ultimately
  show ?case
    using truth_lemma_And truth_lemma_Neg[of ·φ ∧ ψ·] Forces_Nand_alt
      M_genericD map_val_in_MG arity_Nand_le[of φ ψ] by auto
next
  case (Forall φ)
  with ‹M_generic(G)›
  show ?case
  proof (intro iffI)
    assume ∃ p∈G. (p ⊩ Forall(φ) env)
```

with ‹*M_generic*(*G*)›
obtain *p* **where** *p*∈*G* *p*∈*M* *p*∈*P* *p* ⊩ *Forall*(*φ*) *env*
using *transitivity*[*OF* _ *P_in_M*] **by** *auto*
with ‹*env*∈*list*(*M*)› ‹*φ*∈*formula*›
have *p* ⊩ *φ* ([*x*]@*env*) **if** *x*∈*M* **for** *x*
using *that Forces_Forall* **by** *simp*
with ‹*p*∈*G*› ‹*φ*∈*formula*› ‹*env*∈_› ‹*arity*(*Forall*(*φ*)) ≤ *length*(*env*)›
Forall(*2*)[*of Cons*(_,*env*)] ‹*M_generic*(*G*)›
show *M*[*G*], *map*(*val*(*P*,*G*),*env*) ⊨ *Forall*(*φ*)
using *pred_le2 map_val_in_MG*
by (*auto iff*:*GenExtD*)
next
assume *M*[*G*], *map*(*val*(*P*,*G*),*env*) ⊨ *Forall*(*φ*)
let *?D1*={*d*∈*P*. (*d* ⊩ *Forall*(*φ*) *env*)}
let *?D2*={*d*∈*P*. ∃ *b*∈*M*. ∀ *q*∈*P*. *q*≼*d* ⟶ ¬(*q* ⊩ *φ* ([*b*]@*env*))}
define *D* **where** *D* ≡ *?D1* ∪ *?D2*
have *arφ*:*arity*(*φ*)≤*succ*(*length*(*env*))
using *assms* ‹*arity*(*Forall*(*φ*)) ≤ *length*(*env*)› ‹*φ*∈*formula*› ‹*env*∈*list*(*M*)›
pred_le2
by *simp*
then
have *arity*(*Forall*(*φ*)) ≤ *length*(*env*)
using *pred_le* ‹*φ*∈*formula*› ‹*env*∈*list*(*M*)› **by** *simp*
then
have *?D1*∈*M* **using** *Collect_forces arφ* ‹*φ*∈*formula*› ‹*env*∈*list*(*M*)› **by** *simp*
moreover from ‹*env*∈*list*(*M*)› ‹*φ*∈*formula*›
have *?D2*∈*M*
using *truth_lemma'*[*of φ*] *separation_closed arφ P_in_M*
by *simp*
ultimately
have *D*∈*M* **unfolding** *D_def* **using** *Un_closed* **by** *simp*
moreover
have *D* ⊆ *P* **unfolding** *D_def* **by** *auto*
moreover
have *dense*(*D*)
proof
fix *p*
assume *p*∈*P*
show ∃ *d*∈*D*. *d*≼ *p*
proof (*cases p* ⊩ *Forall*(*φ*) *env*)
case *True*
with ‹*p*∈*P*›
show *?thesis* **unfolding** *D_def* **using** *refl_leq* **by** *blast*
next
case *False*
with *Forall* ‹*p*∈*P*›
obtain *b* **where** *b*∈*M* ¬(*p* ⊩ *φ* ([*b*]@*env*))
using *Forces_Forall* **by** *blast*
moreover from *this* ‹*p*∈*P*› *Forall*

have $\neg dense\_below(\{q{\in}P.\ q \Vdash \varphi\ ([b]@env)\},p)$
using *density_lemma pred_le2* **by** *auto*
moreover from *this*
obtain d **where** $d{\preceq}p\ \forall\, q{\in}P.\ q{\preceq}d \longrightarrow \neg(q \Vdash \varphi\ ([b]\ @\ env))$
$d{\in}P$ **by** *blast*
ultimately
show *?thesis* **unfolding** *D_def* **by** *auto*
qed
qed
moreover
note ‹*M_generic*(G)›
ultimately
obtain d **where** $d \in D\ \ d \in G$ **by** *blast*
then
consider (*1*) $d{\in}?D1$ | (*2*) $d{\in}?D2$ **unfolding** *D_def* **by** *blast*
then
show $\exists\, p{\in}G.\ (p \Vdash Forall(\varphi)\ env)$
proof (*cases*)
case *1*
with ‹$d{\in}G$›
show *?thesis* **by** *blast*
next
case *2*
then
obtain b **where** $b{\in}M\ \forall\, q{\in}P.\ q{\preceq}d \longrightarrow \neg(q \Vdash \varphi\ ([b]\ @\ env))$
by *blast*
moreover from *this*(*1*) **and** ‹$M[G]$, _ $\models$ *Forall*(φ)› **and**
Forall(*2*)[*of Cons*(b,*env*)] *Forall*(*1,3-5*) ‹*M_generic*(G)›
obtain p **where** $p{\in}G\ p{\in}P\ p \Vdash \varphi\ ([b]\ @\ env)$
using *pred_le2* **using** *map_val_in_MG* **by** (*auto iff*:*GenExtD*)
moreover
note ‹$d{\in}G$› ‹*M_generic*(G)›
ultimately
obtain q **where** $q{\in}G\ q{\in}P\ q{\preceq}d\ q{\preceq}p$ **by** *blast*
moreover from *this* **and** ‹$p \Vdash \varphi\ ([b]\ @\ env)$›
Forall ‹$b{\in}M$› ‹$p{\in}P$›
have $q \Vdash \varphi\ ([b]\ @\ env)$
using *pred_le2 strengthening_lemma* **by** *simp*
moreover
note ‹$\forall\, q{\in}P.\ q{\preceq}d \longrightarrow \neg(q \Vdash \varphi\ ([b]\ @\ env))$›
ultimately
show *?thesis* **by** *simp*
qed
qed
qed

33.16 The "Definition of forcing"

lemma *definition_of_forcing*:

assumes
$p{\in}P$ $\varphi{\in}formula$ $env{\in}list(M)$ $arity(\varphi){\leq}length(env)$
shows
$(p \Vdash \varphi\ env) \longleftrightarrow$
$(\forall\, G.\ M\_generic(G) \wedge p{\in}G \longrightarrow M[G],\ map(val(P,G),env) \models \varphi)$
proof (*intro iffI allI impI*, *elim conjE*)
fix G
assume $(p \Vdash \varphi\ env)$ $M\_generic(G)$ $p \in G$
with *assms*
show $M[G],\ map(val(P,G),env) \models \varphi$
using *truth_lemma*[*of* φ] **by** *blast*
next
assume *1*: $\forall\, G.(M\_generic(G) \wedge p{\in}G) \longrightarrow M[G]\ ,\ map(val(P,G),env) \models \varphi$
{
fix r
assume *2*: $r{\in}P$ $r{\preceq}p$
then
obtain G **where** $r{\in}G$ $M\_generic(G)$
using *generic_filter_existence* **by** *auto*
moreover from *calculation 2* ‹$p{\in}P$›
have $p{\in}G$
unfolding *M_generic_def* **using** *filter_leqD* **by** *simp*
moreover note *1*
ultimately
have $M[G],\ map(val(P,G),env) \models \varphi$
by *simp*
with *assms* ‹$M\_generic(G)$›
obtain s **where** $s{\in}G$ $(s \Vdash \varphi\ env)$
using *truth_lemma*[*of* φ] **by** *blast*
moreover from *this* **and** ‹$M\_generic(G)$› ‹$r{\in}G$›
obtain q **where** $q{\in}G$ $q{\preceq}s$ $q{\preceq}r$
by *blast*
moreover from *calculation* ‹$s{\in}G$› ‹$M\_generic(G)$›
have $s{\in}P$ $q{\in}P$
unfolding *M_generic_def filter_def* **by** *auto*
moreover
note *assms*
ultimately
have $\exists\, q{\in}P.\ q{\preceq}r \wedge (q \Vdash \varphi\ env)$
using *strengthening_lemma* **by** *blast*
}
then
have *dense_below*($\{q{\in}P.\ (q \Vdash \varphi\ env)\}$,$p$)
unfolding *dense_below_def* **by** *blast*
with *assms*
show $(p \Vdash \varphi\ env)$
using *density_lemma* **by** *blast*
qed

lemmas *definability = forces_type*

end

end

34 Auxiliary renamings for Separation

theory *Separation_Rename*
  imports *Interface Renaming*
begin

lemmas *apply_fun = apply_iff[THEN iffD1]*

lemma *nth_concat* : *[p,t] ∈ list(A) ⟹ env∈ list(A) ⟹ nth(1 #+ length(env),[p]@ env @ [t]) = t*
  by(*auto simp add:nth_append*)

lemma *nth_concat2* : *env∈ list(A) ⟹ nth(length(env),env @ [p,t]) = p*
  by(*auto simp add:nth_append*)

lemma *nth_concat3* : *env∈ list(A) ⟹ u = nth(succ(length(env)), env @ [pi, u])*
  by(*auto simp add:nth_append*)

definition
  sep_var :: *i ⇒ i* **where**
  sep_var(n) ≡ {⟨0,1⟩,⟨1,3⟩,⟨2,4⟩,⟨3,5⟩,⟨4,0⟩,⟨5#+n,6⟩,⟨6#+n,2⟩}

definition
  sep_env :: *i ⇒ i* **where**
  sep_env(n) ≡ λ i ∈ (5#+n)-5 . i#+2

definition *weak* :: *[i, i] ⇒ i* **where**
  weak(n,m) ≡ {i#+m . i ∈ n}

lemma *weakD* :
  assumes *n ∈ nat k∈nat x ∈ weak(n,k)*
  shows *∃ i ∈ n . x = i#+k*
  using *assms* **unfolding** *weak_def* **by** *blast*

lemma *weak_equal* :
  assumes *n∈nat m∈nat*
  shows *weak(n,m) = (m#+n) - m*
proof -
  have *weak(n,m)⊆(m#+n)-m*
  proof(*intro subsetI*)
    fix *x*
    assume *x∈weak(n,m)*
    with *assms*

obtain i **where**
$i{\in}n$ $x{=}i\#{+}m$
using *weakD* **by** *blast*
then
have $m{\leq}i\#{+}m$ $i{<}n$
using *add_le_self2*[*of* m i] ‹$m{\in}nat$› ‹$n{\in}nat$› *ltI*[*OF* ‹$i{\in}n$›] **by** *simp_all*
then
have $\neg i\#{+}m{<}m$
using *not_lt_iff_le in_n_in_nat*[*OF* ‹$n{\in}nat$› ‹$i{\in}n$›] ‹$m{\in}nat$› **by** *simp*
with ‹$x{=}i\#{+}m$›
have $x{\notin}m$
using *ltI* ‹$m{\in}nat$› **by** *auto*
moreover
from *assms* ‹$x{=}i\#{+}m$› ‹$i{<}n$›
have $x{<}m\#{+}n$
using *add_lt_mono1*[*OF* ‹$i{<}n$› ‹$n{\in}nat$›] **by** *simp*
ultimately
show $x{\in}(m\#{+}n)\text{-}m$
using *ltD DiffI* **by** *simp*
qed
moreover
have $(m\#{+}n)\text{-}m{\subseteq}weak(n,m)$
proof (*intro subsetI*)
fix x
assume $x{\in}(m\#{+}n)\text{-}m$
then
have $x{\in}m\#{+}n$ $x{\notin}m$
using *DiffD1*[*of* x $n\#{+}m$ m] *DiffD2*[*of* x $n\#{+}m$ m] **by** *simp_all*
then
have $x{<}m\#{+}n$ $x{\in}nat$
using *ltI in_n_in_nat*[*OF add_type*[*of* m n]] **by** *simp_all*
then
obtain i **where**
$m\#{+}n = succ(x\#{+}i)$
using *less_iff_succ_add*[*OF* ‹$x{\in}nat$›,*of* $m\#{+}n$] *add_type* **by** *auto*
then
have $x\#{+}i{<}m\#{+}n$ **using** *succ_le_iff* **by** *simp*
with ‹$x{\notin}m$›
have $\neg x{<}m$ **using** *ltD* **by** *blast*
with ‹$m{\in}nat$› ‹$x{\in}nat$›
have $m{\leq}x$ **using** *not_lt_iff_le* **by** *simp*
with ‹$x{<}m\#{+}n$› ‹$n{\in}nat$›
have $x\#\text{-}m{<}m\#{+}n\#\text{-}m$
using *diff_mono*[*OF* ‹$x{\in}nat$› _ ‹$m{\in}nat$›] **by** *simp*
have $m\#{+}n\#\text{-}m = n$ **using** *diff_cancel2* ‹$m{\in}nat$› ‹$n{\in}nat$› **by** *simp*
with ‹$x\#\text{-}m{<}m\#{+}n\#\text{-}m$› ‹$x{\in}nat$›
have $x\#\text{-}m \in n$ $x{=}x\#\text{-}m\#{+}m$
using *ltD add_diff_inverse2*[*OF* ‹$m{\leq}x$›] **by** *simp_all*
then

```
    show x∈weak(n,m)
      unfolding weak_def by auto
  qed
  ultimately
  show ?thesis by auto
qed

lemma weak_zero:
  shows weak(0,n) = 0
  unfolding weak_def by simp

lemma weakening_diff :
  assumes n ∈ nat
  shows weak(n,7) - weak(n,5) ⊆ {5#+n, 6#+n}
  unfolding weak_def using assms
proof(auto)
  {
    fix i
    assume i∈n succ(succ(natify(i)))≠n ∀ w∈n. succ(succ(natify(i))) ≠ natify(w)
    then
    have i<n
      using ltI ‹n∈nat› by simp
    from ‹n∈nat› ‹i∈n› ‹succ(succ(natify(i)))≠n›
    have i∈nat succ(succ(i))≠n using in_n_in_nat by simp_all
    from ‹i<n›
    have succ(i)≤n using succ_leI by simp
    with ‹n∈nat›
    consider (a) succ(i) = n | (b) succ(i) < n
      using leD by auto
    then have succ(i) = n
    proof cases
      case a
      then show ?thesis .
    next
      case b
      then
      have succ(succ(i))≤n using succ_leI by simp
      with ‹n∈nat›
      consider (a) succ(succ(i)) = n | (b) succ(succ(i)) < n
        using leD by auto
      then have succ(i) = n
      proof cases
        case a
        with ‹succ(succ(i))≠n› show ?thesis by blast
      next
        case b
        then
        have succ(succ(i))∈n using ltD by simp
        with ‹i∈nat›
```

```
      have succ(succ(natify(i))) ≠ natify(succ(succ(i)))
        using ‹∀w∈n. succ(succ(natify(i))) ≠ natify(w)› by auto
      then
      have False using ‹i∈nat› by auto
      then show ?thesis by blast
    qed
    then show ?thesis .
  qed
  with ‹i∈nat› have succ(natify(i)) = n by simp
 }
 then
 show n ∈ nat ⟹
  succ(succ(natify(y))) ≠ n ⟹
  ∀x∈n. succ(succ(natify(y))) ≠ natify(x) ⟹
  y ∈ n ⟹ succ(natify(y)) = n for y
  by blast
qed

lemma in_add_del :
 assumes x∈j#+n n∈nat j∈nat
 shows x < j ∨ x ∈ weak(n,j)
proof (cases x<j)
 case True
 then show ?thesis ..
next
 case False
 have x∈nat j#+n∈nat
  using in_n_in_nat[OF _ ‹x∈j#+n›] assms by simp_all
 then
 have j ≤ x x < j#+n
  using not_lt_iff_le False ‹j∈nat› ‹n∈nat› ltI[OF ‹x∈j#+n›] by auto
 then
 have x#-j < (j #+ n) #- j x = j #+ (x #-j)
  using diff_mono ‹x∈nat› ‹j#+n∈nat› ‹j∈nat› ‹n∈nat›
    add_diff_inverse[OF ‹j≤x›] by simp_all
 then
 have x#-j < n x = (x #-j ) #+ j
  using diff_add_inverse ‹n∈nat› add_commute by simp_all
 then
 have x#-j ∈n using ltD by simp
 then
 have x ∈ weak(n,j)
  unfolding weak_def
  using ‹x= (x#-j) #+j› RepFunI[OF ‹x#-j∈n›] add_commute by force
 then show ?thesis ..
qed


lemma sep_env_action:
```

```
  assumes
    [t,p,u,P,leq,o,pi] ∈ list(M)
    env ∈ list(M)
  shows ∀ i . i ∈ weak(length(env),5) ⟶
      nth(sep_env(length(env))`i,[t,p,u,P,leq,o,pi]@env) = nth(i,[p,P,leq,o,t] @ env
@ [pi,u])
proof -
  from assms
  have A: 5#+length(env)∈nat [p, P, leq, o, t] ∈list(M)
    by simp_all
  let ?f=sep_env(length(env))
  have EQ: weak(length(env),5) = 5#+length(env) - 5
    using weak_equal length_type[OF ‹env∈list(M)›] by simp
  let ?tgt=[t,p,u,P,leq,o,pi]@env
  let ?src=[p,P,leq,o,t] @ env @ [pi,u]
  have nth(?f`i,[t,p,u,P,leq,o,pi]@env) = nth(i,[p,P,leq,o,t] @ env @ [pi,u])
    if i ∈ (5#+length(env)-5) for i
  proof -
    from that
    have 2: i ∈ 5#+length(env)  i ∉ 5 i ∈ nat i#-5∈nat i#+2∈nat
      using in_n_in_nat[OF ‹5#+length(env)∈nat›] by simp_all
    then
    have 3: ¬ i < 5 using ltD by force
    then
    have 5 ≤ i 2 ≤ 5
      using  not_lt_iff_le ‹i∈nat› by simp_all
    then have 2 ≤ i using le_trans[OF ‹2≤5›] by simp
    from A ‹i ∈ 5#+length(env)›
    have i < 5#+length(env) using ltI by simp
    with ‹i∈nat› ‹2≤i› A
    have C:i#+2 < 7#+length(env)  by simp
    with that
    have B: ?f`i = i#+2 unfolding sep_env_def by simp
    from 3 assms(1) ‹i∈nat›
    have ¬ i#+2 < 7 using not_lt_iff_le add_le_mono by simp
    from ‹i < 5#+length(env)› 3 ‹i∈nat›
    have i#-5 < 5#+length(env) #- 5
      using diff_mono[of i 5#+length(env) 5,OF _ _ _ ‹i < 5#+length(env)›]
        not_lt_iff_le[THEN iffD1] by force
    with assms(2)
    have i#-5 < length(env) using diff_add_inverse length_type by simp
    have nth(i,?src) =nth(i#-5,env@[pi,u])
      using nth_append[OF A(2) ‹i∈nat›] 3 by simp
    also
    have ... = nth(i#-5, env)
      using nth_append[OF ‹env ∈list(M)› ‹i#-5∈nat›] ‹i#-5 < length(env)› by
simp
    also
    have ... = nth(i#+2, ?tgt)
```

```
    using nth_append[OF assms(1) ‹i#+2∈nat›] ‹¬ i#+2 <7› by simp
   ultimately
   have nth(i,?src) = nth(?f‘i,?tgt)
    using B by simp
   then show ?thesis using that by simp
  qed
  then show ?thesis using EQ by force
qed

lemma sep_env_type :
  assumes n ∈ nat
  shows sep_env(n) : (5#+n)-5 → (7#+n)-7
proof -
  let ?h=sep_env(n)
  from ‹n∈nat›
  have (5#+n)#+2 = 7#+n 7#+n∈nat 5#+n∈nat by simp_all
  have
    D: sep_env(n)‘x ∈ (7#+n)-7 if x ∈ (5#+n)-5 for x
  proof -
    from ‹x∈5#+n-5›
    have ?h‘x = x#+2 x<5#+n x∈nat
    unfolding sep_env_def using ltI in_n_in_nat[OF ‹5#+n∈nat›] by simp_all
    then
    have x#+2 < 7#+n by simp
    then
    have x#+2 ∈ 7#+n using ltD by simp
    from ‹x∈5#+n-5›
    have x∉5 by simp
    then have ¬x<5 using ltD by blast
    then have 5≤x using not_lt_iff_le ‹x∈nat› by simp
    then have 7≤x#+2 using add_le_mono ‹x∈nat› by simp
    then have ¬x#+2<7 using not_lt_iff_le ‹x∈nat› by simp
    then have x#+2 ∉ 7 using ltI ‹x∈nat› by force
    with ‹x#+2 ∈ 7#+n› show ?thesis using ‹?h‘x = x#+2› DiffI by simp
  qed
  then show ?thesis unfolding sep_env_def using lam_type by simp
qed

lemma sep_var_fin_type :
  assumes n ∈ nat
  shows sep_var(n) : 7#+n -||> 7#+n
  unfolding sep_var_def
  using consI ltD emptyI by force

lemma sep_var_domain :
  assumes n ∈ nat
  shows domain(sep_var(n)) = 7#+n - weak(n,5)
proof -
  let ?A=weak(n,5)
```

```
have A:domain(sep_var(n)) ⊆ (7#+n)
  unfolding sep_var_def
  by(auto simp add: le_natE)
have C: x=5#+n ∨ x=6#+n ∨ x ≤ 4 if x∈domain(sep_var(n)) for x
  using that unfolding sep_var_def by auto
have D : x<n#+7 if x∈7#+n for x
  using that ‹n∈nat› ltI by simp
have ¬ 5#+n < 5#+n using ‹n∈nat›  lt_irrefl[of _ False] by force
have ¬ 6#+n < 5#+n using ‹n∈nat› by force
have R: x < 5#+n if x∈?A for x
proof -
  from that
  obtain i where
    i<n x=5#+i
    unfolding weak_def
    using ltI ‹n∈nat› RepFun_iff by force
  with ‹n∈nat›
  have 5#+i < 5#+n using add_lt_mono2 by simp
  with ‹x=5#+i›
  show x < 5#+n by simp
qed
then
have 1:x∉?A if ¬x <5#+n for x using that by blast
have 5#+n ∉ ?A 6#+n∉?A
proof -
  show 5#+n ∉ ?A using 1 ‹¬5#+n<5#+n› by blast
  with 1 show 6#+n ∉ ?A using  ‹¬6#+n<5#+n› by blast
qed
then
have E:x∉?A if x∈domain(sep_var(n)) for x
  unfolding weak_def
  using C that by force
then
have F: domain(sep_var(n)) ⊆ 7#+n - ?A using A by auto
from assms
have x<7 ∨ x∈weak(n,7) if x∈7#+n for x
  using in_add_del[OF ‹x∈7#+n›] by simp
moreover
{
  fix x
  assume asm:x∈7#+n  x∉?A  x∈weak(n,7)
  then
  have x∈domain(sep_var(n))
  proof -
    from ‹n∈nat›
    have weak(n,7)-weak(n,5)⊆{n#+5,n#+6}
      using weakening_diff by simp
    with  ‹x∉?A› asm
    have x∈{n#+5,n#+6} using  subsetD DiffI by blast
```

```
    then
    show ?thesis unfolding sep_var_def by simp
  qed
}
moreover
{
  fix x
  assume asm:x∈7#+n  x∉?A x<7
  then have x∈domain(sep_var(n))
  proof (cases 2 ≤ n)
    case True
    moreover
    have 0<n using  leD[OF ‹n∈nat› ‹2≤n›] lt_imp_0_lt by auto
    ultimately
    have x<5
      using ‹x<7› ‹x∉?A› ‹n∈nat› in_n_in_nat
      unfolding weak_def
      by (clarsimp simp add:not_lt_iff_le, auto simp add:lt_def)
    then
    show ?thesis unfolding sep_var_def
      by (clarsimp simp add:not_lt_iff_le, auto simp add:lt_def)
  next
    case False
    then
    show ?thesis
    proof (cases n=0)
      case True
      then show ?thesis
        unfolding sep_var_def using ltD asm ‹n∈nat› by auto
    next
      case False
      then
      have n < 2 using  ‹n∈nat› not_lt_iff_le ‹¬ 2 ≤ n›  by force
      then
      have ¬ n <1 using ‹n≠0› by simp
      then
      have n=1 using not_lt_iff_le ‹n<2› le_iff by auto
      then show ?thesis
        using ‹x∉?A›
        unfolding weak_def sep_var_def
        using ltD asm ‹n∈nat› by force
    qed
  qed
}
ultimately
have w∈domain(sep_var(n)) if w∈ 7#+n - ?A for w
  using that by blast
then
have 7#+n - ?A ⊆ domain(sep_var(n)) by blast
```

```
  with F
  show ?thesis by auto
qed

lemma sep_var_type :
  assumes n ∈ nat
  shows sep_var(n) : (7#+n)-weak(n,5) → 7#+n
  using FiniteFun_is_fun[OF sep_var_fin_type[OF ‹n∈nat›]]
    sep_var_domain[OF ‹n∈nat›] by simp

lemma sep_var_action :
  assumes
    [t,p,u,P,leq,o,pi] ∈ list(M)
    env ∈ list(M)
  shows ∀ i . i ∈ (7#+length(env)) - weak(length(env),5) ⟶
     nth(sep_var(length(env)) ‘i,[t,p,u,P,leq,o,pi]@env) = nth(i,[p,P,leq,o,t] @ env
@ [pi,u])
  using assms
proof (subst sep_var_domain[OF length_type[OF ‹env∈list(M)›],symmetric],auto)
  fix i y
  assume ⟨i, y⟩ ∈ sep_var(length(env))
  with assms
  show nth(sep_var(length(env)) ‘ i,
           Cons(t, Cons(p, Cons(u, Cons(P, Cons(leq, Cons(o, Cons(pi, env)))))))
=
          nth(i, Cons(p, Cons(P, Cons(leq, Cons(o, Cons(t, env @ [pi, u]))))))
    using apply_fun[OF sep_var_type] assms
      unfolding sep_var_def
    using nth_concat2[OF ‹env∈list(M)›] nth_concat3[OF ‹env∈list(M)›,symmetric]
      by force
  qed

definition
  rensep :: i ⇒ i where
  rensep(n) ≡ union_fun(sep_var(n),sep_env(n),7#+n-weak(n,5),weak(n,5))

lemma rensep_aux :
  assumes n∈nat
   shows (7#+n-weak(n,5)) ∪ weak(n,5) = 7#+n 7#+n ∪ ( 7 #+ n - 7) =
7#+n
proof -
  from ‹n∈nat›
  have weak(n,5) = n#+5-5
    using weak_equal by simp
  with ‹n∈nat›
 show (7#+n-weak(n,5)) ∪ weak(n,5) = 7#+n 7#+n ∪ ( 7 #+ n - 7) = 7#+n
    using Diff_partition le_imp_subset by auto
qed
```

lemma *rensep_type* :
  assumes *n∈nat*
  shows *rensep(n) ∈ 7#+n → 7#+n*
proof -
  from ‹*n∈nat*›
  have *rensep(n) ∈ (7#+n-weak(n,5)) ∪ weak(n,5) → 7#+n ∪ (7#+n - 7)*
    unfolding *rensep_def*
    using *union_fun_type sep_var_type* ‹*n∈nat*› *sep_env_type weak_equal*
    by *force*
  then
  show *?thesis* **using** *rensep_aux* ‹*n∈nat*› **by** *auto*
qed

lemma *rensep_action* :
  assumes *[t,p,u,P,leq,o,pi] @ env ∈ list(M)*
 shows *∀ i . i < 7#+length(env) ⟶ nth(rensep(length(env)) ‘i,[t,p,u,P,leq,o,pi]@env)*
= nth(i,[p,P,leq,o,t] @ env @ [pi,u])
proof -
  let *?tgt=[t,p,u,P,leq,o,pi]@env*
  let *?src=[p,P,leq,o,t] @ env @ [pi,u]*
  let *?m=7 #+ length(env) - weak(length(env),5)*
  let *?p=weak(length(env),5)*
  let *?f=sep_var(length(env))*
  let *?g=sep_env(length(env))*
  let *?n=length(env)*
  from *assms*
  have *1* : *[t,p,u,P,leq,o,pi] ∈ list(M) env ∈ list(M)*
    ?src ∈ list(M) ?tgt ∈ list(M)
    7#+?n = (7#+?n-weak(?n,5)) ∪ weak(?n,5)
     length(?src) = (7#+?n-weak(?n,5)) ∪ weak(?n,5)
    using *Diff_partition le_imp_subset rensep_aux* **by** *auto*
  then
 have *nth(i, ?src) = nth(union_fun(?f, ?g, ?m, ?p) ‘ i, ?tgt)* **if** *i < 7#+length(env)*
for *i*
  proof -
    from ‹*i<7#+?n*›
    have *i ∈ (7#+?n-weak(?n,5)) ∪ weak(?n,5)*
      using *ltD* **by** *simp*
    then show *?thesis*
      unfolding *rensep_def* **using**
     union_fun_action[OF ‹*?src∈list(M)*› ‹*?tgt∈list(M)*› ‹*length(?src) = (7#+?n-weak(?n,5))*
∪ weak(?n,5)›
         sep_var_action[OF ‹*[t,p,u,P,leq,o,pi] ∈ list(M)*› ‹*env∈list(M)*›]
         sep_env_action[OF ‹*[t,p,u,P,leq,o,pi] ∈ list(M)*› ‹*env∈list(M)*›]
         ] that
      by *simp*
  qed
  then show *?thesis* **unfolding** *rensep_def* **by** *simp*
qed

definition *sep_ren* :: $[i,i] \Rightarrow i$ **where**
  $sep\_ren(n,\varphi) \equiv ren(\varphi)\text{`}(7\#+n)\text{`}(7\#+n)\text{`}rensep(n)$

lemma *arity_rensep*: **assumes** $\varphi\in formula$ $env \in list(M)$
  $arity(\varphi) \leq 7\#+length(env)$
shows $arity(sep\_ren(length(env),\varphi)) \leq 7\#+length(env)$
  unfolding *sep_ren_def*
  using *arity_ren rensep_type assms*
  by *simp*

lemma *type_rensep* [*TC*]:
  assumes $\varphi\in formula$ $env\in list(M)$
  shows $sep\_ren(length(env),\varphi) \in formula$
  unfolding *sep_ren_def*
  using *ren_tc rensep_type assms*
  by *simp*

lemma *sepren_action*:
  assumes $arity(\varphi) \leq 7 \#+ length(env)$
    $[t,p,u,P,leq,o,pi] \in list(M)$
    $env\in list(M)$
    $\varphi\in formula$
   shows $sats(M,\ sep\_ren(length(env),\varphi),[t,p,u,P,leq,o,pi]\ @\ env) \longleftrightarrow sats(M,$
$\varphi,[p,P,leq,o,t]\ @\ env\ @\ [pi,u])$
proof -
  from *assms*
  have *1*: $[t,\ p,\ u,\ P,\ leq,\ o,\ pi]\ @\ env \in list(M)$
    $[P,leq,o,p,t] \in list(M)$
    $[pi,u] \in list(M)$
    by *simp_all*
  then
  have *2*: $[p,P,leq,o,t]\ @\ env\ @\ [pi,u] \in list(M)$ **using** *app_type* **by** *simp*
  show *?thesis*
    unfolding *sep_ren_def*
    using *sats_iff_sats_ren*[*OF* ‹$\varphi\in formula$›
        add_type[*of* 7 *length*(*env*)]
        add_type[*of* 7 *length*(*env*)]
        2 1(*1*)
        rensep_type[*OF length_type*[*OF* ‹$env\in list(M)$›]]
        ‹$arity(\varphi) \leq 7 \#+ length(env)$›]
      rensep_action[*OF 1*(*1*),*rule_format*,*symmetric*]
    by *simp*
qed

end

35 The Axiom of Separation in $M[G]$

theory *Separation_Axiom*
  imports *Forcing_Theorems Separation_Rename*
begin

context *G_generic*
begin

lemma *map_val* :
  assumes *env∈list(M[G])*
  shows ∃ *nenv∈list(M). env = map(val(P,G),nenv)*
  using *assms*
  proof(*induct env*)
    case *Nil*
    have *map(val(P,G),Nil) = Nil* **by** *simp*
    then show *?case* **by** *force*
  next
    case (*Cons a l*)
    then obtain *a′ l′* **where**
      l′ ∈ list(M) l=map(val(P,G),l′) a = val(P,G,a′)
      Cons(a,l) = map(val(P,G),Cons(a′,l′)) Cons(a′,l′) ∈ list(M)
      using ‹*a∈M[G]*› *GenExtD*
      by *force*
    then show *?case* **by** *force*
qed

lemma *Collect_sats_in_MG* :
  assumes
    c∈M[G]
    φ ∈ formula env∈list(M[G]) arity(φ) ≤ 1 #+ length(env)
  shows
    {*x∈c.* (*M[G],* [*x*] @ *env* ⊨ *φ*)}∈ *M[G]*
proof -
  from ‹*c∈M[G]*›
  obtain *π* **where** *π ∈ M val(P,G, π) = c*
    using *GenExt_def* **by** *auto*
  let *?χ*=·· *0 ∈* (*1 #+ length(env)*) · ∧ *φ* · **and** *?Pl1*=[*P,leq,one*]
  let *?new_form*=*sep_ren*(*length*(*env*),*forces*(*?χ*))
  let *?ψ*=*Exists*(*Exists*(*And*(*pair_fm*(*0,1,2*),*?new_form*)))
  note *phi* = ‹*φ∈formula*› ‹*arity*(*φ*) ≤ *1 #+ length*(*env*)›
  then
  have *?χ∈formula* **by** *simp*
  with ‹*env∈_*› *phi*
  have *arity*(*?χ*) ≤ *2#+length*(*env*)
    using *ord_simp_union leI* **by** *simp*
  with ‹*env∈list*(*_*)› *phi*
  have *arity*(*forces*(*?χ*)) ≤ *6 #+ length*(*env*)
    using *arity_forces_le* **by** *simp*

then
have *arity(forces(?χ)) ≤ 7 #+ length(env)*
using *ord_simp_union arity_forces leI* **by** *simp*
with ‹*arity(forces(?χ)) ≤7 #+ _*› ‹*env ∈ _*› ‹*φ ∈ formula*›
have *arity(?new_form) ≤ 7 #+ length(env) ?new_form ∈ formula*
using *arity_rensep*[*OF definability*[*of ?χ*]] *definability*[*of ?χ*] *type_rensep*
by *auto*
then
have *pred(pred(arity(?new_form))) ≤ 5 #+ length(env) ?ψ∈formula*
unfolding *pair_fm_def upair_fm_def*
using *ord_simp_union length_type*[*OF* ‹*env∈list(M*[*G*])›]
pred_mono[*OF _ pred_mono*[*OF _* ‹*arity(?new_form) ≤ _*›]]
by *auto*
with ‹*arity(?new_form) ≤ _*› ‹*?new_form ∈ formula*›
have *arity(?ψ) ≤ 5 #+ length(env)*
unfolding *pair_fm_def upair_fm_def*
using *ord_simp_union arity_forces*
by *auto*
from ‹*φ∈formula*›
have *forces(?χ) ∈ formula*
using *definability* **by** *simp*
from ‹*π∈M*› *P_in_M*
have *domain(π)∈M domain(π) × P ∈ M*
by (*simp_all flip*:*setclass_iff*)
from ‹*env ∈ _*›
obtain *nenv* **where** *nenv∈list(M) env = map(val(P,G),nenv) length(nenv) = length(env)*
using *map_val* **by** *auto*
from ‹*arity(φ) ≤ _*› ‹*env∈_*› ‹*φ∈_*›
have *arity(φ) ≤ 2#+ length(env)*
using *le_trans*[*OF* ‹*arity(φ)≤_*›] *add_le_mono*[*of 1 2*,*OF _ le_refl*]
by *auto*
with ‹*nenv∈_*› ‹*env∈_*› ‹*π∈M*› ‹*φ∈_*› ‹*length(nenv) = length(env)*›
have *arity(?χ) ≤ length*([*ϑ*] @ *nenv* @ [*π*]) **for** *ϑ*
using *union_abs2*[*OF* ‹*arity(φ) ≤ 2#+ _*›] *ord_simp_union*
by *simp*
note *in_M* = ‹*π∈M*› ‹*domain(π) × P ∈ M*› *P_in_M one_in_M leq_in_M*
{
fix *u*
assume *u ∈ domain(π) × P u ∈ M*
with *in_M* ‹*?new_form ∈ formula*› ‹*?ψ∈formula*› ‹*nenv ∈ _*›
have *Eq1*: (*M*, [*u*] @ *?Pl1* @ [*π*] @ *nenv* ⊨ *?ψ*) ⟷
(∃ *ϑ*∈*M*. ∃ *p*∈*P*. *u* =⟨*ϑ*,*p*⟩ ∧
(*M*, [*ϑ*,*p*,*u*]@*?Pl1*@[*π*] @ *nenv* ⊨ *?new_form*))
by (*auto simp add*: *transitivity*)
have *Eq3*: *ϑ∈M* ⟹ *p∈P* ⟹
(*M*, [*ϑ*,*p*,*u*]@*?Pl1*@[*π*]@*nenv* ⊨ *?new_form*) ⟷
(∀ *F*. *M_generic*(*F*) ∧ *p* ∈ *F* ⟶ (*M*[*F*], *map*(*val*(*P*,*F*), [*ϑ*] @ *nenv*@[*π*]) ⊨ *?χ*))

```
    for ϑ p
  proof -
    fix p ϑ
    assume ϑ ∈ M p∈P
    then
    have p∈M using P_in_M by (simp add: transitivity)
   note in_M' = in_M ‹ϑ ∈ M› ‹p∈M› ‹u ∈ domain(π) × P› ‹u ∈ M› ‹nenv∈_›
    then
    have [ϑ,u] ∈ list(M) by simp
    let ?env=[p]@?Pl1@[ϑ] @ nenv @ [π,u]
    let ?new_env= [ϑ,p,u,P,leq,one,π] @ nenv
    let ?ψ=Exists(Exists(And(pair_fm(0,1,2),?new_form)))
    have [ϑ, p, u, π, leq, one, π] ∈ list(M)
      using in_M' by simp
    have ?χ ∈ formula forces(?χ)∈ formula
      using phi by simp_all
    from in_M'
    have ?Pl1 ∈ list(M) by simp
    from in_M' have ?env ∈ list(M) by simp
    have Eq1': ?new_env ∈ list(M) using in_M' by simp
    then
     have (M, [ϑ,p,u]@?Pl1@[π] @ nenv ⊨ ?new_form) ⟷ (M, ?new_env ⊨
?new_form)
      by simp
    from in_M' ‹env ∈ _› Eq1' ‹length(nenv) = length(env)›
      ‹arity(forces(?χ)) ≤ 7 #+ length(env)› ‹forces(?χ)∈ formula›
      ‹[ϑ, p, u, π, leq, one, π] ∈ list(M)›
    have ... ⟷ M, ?env ⊨ forces(?χ)
      using sepren_action[of forces(?χ)  nenv,OF _ _ ‹nenv∈list(M)›]
      by simp
    also from in_M'
    have ... ⟷ M,  ([p,P, leq, one,ϑ]@nenv@ [π])@[u] ⊨ forces(?χ)
      using app_assoc by simp
    also
    from in_M' ‹env∈_› phi ‹length(nenv) = length(env)›
      ‹arity(forces(?χ)) ≤ 6 #+ length(env)› ‹forces(?χ)∈formula›
    have ... ⟷ M,  [p,P, leq, one,ϑ]@ nenv @ [π] ⊨ forces(?χ)
      by (rule_tac arity_sats_iff,auto)
    also
   from ‹arity(forces(?χ)) ≤ 6 #+ length(env)› ‹forces(?χ)∈formula› in_M' phi

    have  ... ⟷ (∀ F. M_generic(F) ∧ p ∈ F ⟶
                        M[F],  map(val(P,F), [ϑ] @ nenv @ [π]) ⊨  ?χ)
    proof (intro iffI)
      assume a1: M,  [p,P, leq, one,ϑ] @ nenv @ [π] ⊨  forces(?χ)
      note ‹arity(φ)≤ 1#+_›
      with ‹nenv∈_› ‹arity(?χ) ≤ length([ϑ] @ nenv @ [π])› ‹env∈_›
      have p ∈ P ⟹ ?χ∈formula ⟹ [ϑ,π] ∈ list(M) ⟹
              M, [p,P, leq, one] @ [ϑ]@ nenv@[π] ⊨ forces(?χ) ⟹
```

```
            ∀ G. M_generic(G) ∧ p ∈ G ⟶ M[G],  map(val(P,G), [ϑ] @ nenv
@[π]) ⊨ ?χ
        using definition_of_forcing[where φ=·· 0 ∈ (1 #+ length(env)) · ∧ φ ·]
        by auto
      then
      show ∀ F. M_generic(F) ∧ p ∈ F ⟶
            M[F],  map(val(P,F), [ϑ] @ nenv @ [π]) ⊨  ?χ
        using ‹?χ∈formula› ‹p∈P› a1 ‹ϑ∈M› ‹π∈M› by simp
    next
      assume ∀ F. M_generic(F) ∧ p ∈ F ⟶
            M[F],  map(val(P,F), [ϑ] @ nenv @[π]) ⊨  ?χ
      with ‹?χ∈formula› ‹p∈P› in_M'
        ‹arity(?χ) ≤ length([ϑ] @ nenv @ [π])›
      show M,  [p, P, leq, one,ϑ] @ nenv @ [π] ⊨  forces(?χ)
        using definition_of_forcing[where φ=·· 0 ∈ (1 #+ length(env)) · ∧ φ ·,
          THEN iffD2] by auto
    qed
    finally
    show (M, [ϑ,p,u]@?Pl1@[π]@nenv ⊨ ?new_form) ⟷ (∀ F. M_generic(F)
∧ p ∈ F ⟶
                  M[F],  map(val(P,F), [ϑ] @ nenv @ [π]) ⊨  ?χ)
      by simp
  qed
  with Eq1
  have (M, [u] @ ?Pl1 @ [π] @ nenv ⊨ ?ψ) ⟷
      (∃ ϑ∈M. ∃ p∈P. u =⟨ϑ,p⟩ ∧
      (∀ F. M_generic(F) ∧ p ∈ F ⟶ M[F],  map(val(P,F), [ϑ] @ nenv @ [π])
⊨  ?χ))
    by auto
}
then
have Equivalence: u∈ domain(π) × P ⟹ u ∈ M ⟹
    (M, [u] @ ?Pl1 @ [π] @ nenv ⊨ ?ψ) ⟷
      (∃ ϑ∈M. ∃ p∈P. u =⟨ϑ,p⟩ ∧
      (∀ F. M_generic(F) ∧ p ∈ F ⟶ M[F],  map(val(P,F), [ϑ] @ nenv @[π])
⊨  ?χ))
  for u
  by simp
moreover from ‹env = _› ‹π∈M› ‹nenv∈list(M)›
have map_nenv:map(val(P,G), nenv@[π]) = env @ [val(P,G,π)]
  using map_app_distrib append1_eq_iff by auto
ultimately
have aux:(∃ ϑ∈M. ∃ p∈P. u =⟨ϑ,p⟩ ∧ (p∈G ⟶ M[G], [val(P,G,ϑ)] @ env @
[val(P,G,π)] ⊨ ?χ))
  (is (∃ ϑ∈M. ∃ p∈P. _ ( _ ⟶ _, ?vals(ϑ) ⊨ _)))
  if u ∈ domain(π) × P u ∈ M  M, [u]@ ?Pl1 @[π] @ nenv ⊨ ?ψ for u
  using Equivalence[THEN iffD1, OF that] generic by force
moreover
have ϑ∈M ⟹ val(P,G,ϑ)∈M[G] for ϑ
```

using *GenExt_def* **by** *auto*
moreover
have $\vartheta\in M \Longrightarrow [val(P,G,\vartheta)]$ @ *env* @ $[val(P,G,\pi)] \in list(M[G])$ **for** ϑ
proof -
from ‹$\pi\in M$›
have $val(P,G,\pi)\in M[G]$ **using** *GenExtI* **by** *simp*
moreover
assume $\vartheta \in M$
moreover
note ‹$env \in list(M[G])$›
ultimately
show *?thesis*
using *GenExtI* **by** *simp*
qed
ultimately
have $(\exists\,\vartheta\in M.\ \exists\,p\in P.\ u=\langle\vartheta,p\rangle \wedge (p\in G \longrightarrow val(P,G,\vartheta)\in nth(1\ \#+\ length(env),[val(P,G,\vartheta)]$ @ *env* @ $[val(P,G,\pi)])$
$\wedge\ (M[G],\ ?vals(\vartheta) \models\ \varphi)))$
if $u \in domain(\pi) \times P\ u \in M\ \ M,\ [u]$ @ *?Pl1* @$[\pi]$ @ *nenv* $\models$ *?psi* **for** u
using *aux*[*OF that*] **by** *simp*
moreover from ‹$env \in \_$› ‹$\pi\in M$›
have *nth*:$nth(1\ \#+\ length(env),[val(P,G,\vartheta)]$ @ *env* @ $[val(P,G,\pi)]) = val(P,G,\pi)$

if $\vartheta\in M$ **for** ϑ
using *nth_concat*[*of* $val(P,G,\vartheta)$ $val(P,G,\pi)$ $M[G]$] **using** *that GenExtI* **by** *simp*
ultimately
have $(\exists\,\vartheta\in M.\ \exists\,p\in P.\ u=\langle\vartheta,p\rangle \wedge (p\in G \longrightarrow val(P,G,\vartheta)\in val(P,G,\pi) \wedge (M[G],?vals(\vartheta)$ $\models\ \varphi)))$
if $u \in domain(\pi) \times P\ u \in M\ \ M,\ [u]$ @ *?Pl1* @$[\pi]$ @ *nenv* $\models$ *?psi* **for** u
using *that* ‹$\pi\in M$› ‹$env \in \_$› **by** *simp*
with ‹$domain(\pi)\times P\in M$›
have $\forall\, u\in domain(\pi)\times P\ .\ (M,\ [u]$ @ *?Pl1* @$[\pi]$ @ *nenv* $\models$ *?psi*$) \longrightarrow (\exists\,\vartheta\in M.$ $\exists\,p\in P.\ u =\langle\vartheta,p\rangle \wedge$
$(p \in G \longrightarrow val(P,G,\vartheta)\in val(P,G,\pi) \wedge (M[G],?vals(\vartheta) \models\ \varphi)))$
by (*simp add*:*transitivity*)
then
have $\{u\in domain(\pi)\times P\ .\ (M,[u]$ @ *?Pl1* @$[\pi]$ @ *nenv* $\models$ *?psi*$)\ \} \subseteq$
$\{u\in domain(\pi)\times P\ .\ \exists\,\vartheta\in M.\ \exists\,p\in P.\ u =\langle\vartheta,p\rangle \wedge$
$(p \in G \longrightarrow val(P,G,\vartheta)\in val(P,G,\pi) \wedge (M[G],\ ?vals(\vartheta) \models \varphi))\}$
(**is** *?n*$\subseteq$*?m*)
by *auto*
with *val_mono*
have *first_incl*: $val(P,G,?n) \subseteq val(P,G,?m)$
by *simp*
note ‹$val(P,G,\pi) = c$›
with ‹*?psi*$\in$*formula*› ‹$arity(?psi) \le \_$› *in_M* ‹*nenv* $\in \_$› ‹*env* $\in \_$› ‹$length(nenv)$ $= \_$›
have *?n*$\in M$
using *separation_ax leI separation_iff* **by** *auto*

from *generic*
have *filter*(G) $G{\subseteq}P$
unfolding *M_generic_def filter_def* **by** *simp_all*
from ‹*val*$(P,G,\pi) = c$›
have *val*$(P,G,?m) =$
$\{val(P,G,t)$ .. $t{\in}domain(\pi)$, $\exists\, q{\in}P$.
$(\exists\, \vartheta{\in}M.\ \exists\, p{\in}P.\ \langle t,q\rangle = \langle \vartheta,\ p\rangle \wedge$
$(p \in G \longrightarrow val(P,G,\ \vartheta) \in c \wedge (M[G],\ [val(P,G,\ \vartheta)]\ @\ env\ @\ [c] \models\ \varphi))$
$\wedge\ q \in G)\}$
using *val_of_name* **by** *auto*
also
have ... = $\{val(P,G,t)$ .. $t{\in}domain(\pi)$, $\exists\, q{\in}P.$
$val(P,G,\ t) \in c \wedge (M[G],\ [val(P,G,\ t)]\ @\ env\ @\ [c] \models\ \varphi) \wedge q \in G\}$
proof -

have $t{\in}M \Longrightarrow$
$(\exists\, q{\in}P.\ (\exists\, \vartheta{\in}M.\ \exists\, p{\in}P.\ \langle t,q\rangle = \langle \vartheta,\ p\rangle \wedge$
$(p \in G \longrightarrow val(P,G,\ \vartheta) \in c \wedge (M[G],\ [val(P,G,\ \vartheta)]\ @\ env\ @\ [c] \models \varphi)) \wedge q \in G))$
$\longleftrightarrow$
$(\exists\, q{\in}P.\ val(P,G,\ t) \in c \wedge (\ M[G],\ [val(P,G,\ t)]@env@[c]{\models}\ \varphi\ ) \wedge q \in G)$ **for** t
by *auto*
then show *?thesis* **using** ‹*domain*$(\pi){\in}M$› **by** (*auto simp add:transitivity*)
qed
also
have ... = $\{x$ .. $x{\in}c$, $\exists\, q{\in}P.\ x \in c \wedge (M[G],\ [x]\ @\ env\ @\ [c] \models\ \varphi) \wedge q \in G\}$
proof

show ... $\subseteq \{x$ .. $x{\in}c$, $\exists\, q{\in}P.\ x \in c \wedge (M[G],\ [x]\ @\ env\ @\ [c] \models\ \varphi) \wedge q \in G\}$
by *auto*
next

{
fix x
assume $x{\in}\{x$ .. $x{\in}c$, $\exists\, q{\in}P.\ x \in c \wedge (M[G],\ [x]\ @\ env\ @\ [c] \models\ \varphi) \wedge q \in G\}$
then
have $\exists\, q{\in}P.\ x \in c \wedge (M[G],\ [x]\ @\ env\ @\ [c] \models\ \varphi) \wedge q \in G$
by *simp*
with ‹*val*$(P,G,\pi) = c$›
have $\exists\, q{\in}P.\ \exists\, t{\in}domain(\pi).\ val(P,G,t) = x \wedge (M[G],\ [val(P,G,t)]\ @\ env\ @\ [c] \models\ \varphi) \wedge q \in G$
using *Sep_and_Replace elem_of_val* **by** *auto*
}
then
show $\{x$ .. $x{\in}c$, $\exists\, q{\in}P.\ x \in c \wedge (M[G],\ [x]\ @\ env\ @\ [c] \models\ \varphi) \wedge q \in G\} \subseteq$ ...
using *SepReplace_iff* **by** *force*

```
  qed
  also
  have  ... = {x∈c. (M[G], [x] @ env @ [c] ⊨ φ)}
    using ‹G⊆P› G_nonempty by force
  finally
  have val_m: val(P,G,?m) = {x∈c. (M[G], [x] @ env @ [c] ⊨ φ)} by simp
  have val(P,G,?m) ⊆ val(P,G,?n)
  proof
    fix x
    assume x ∈ val(P,G,?m)
    with val_m
    have Eq4: x ∈ {x∈c. (M[G], [x] @ env @ [c] ⊨ φ)} by simp
    with ‹val(P,G,π) = c›
    have x ∈ val(P,G,π) by simp
    then
    have ∃ϑ. ∃q∈G. ⟨ϑ,q⟩∈π ∧ val(P,G,ϑ) =x
      using elem_of_val_pair by auto
    then obtain ϑ q where
      ⟨ϑ,q⟩∈π  q∈G val(P,G,ϑ)=x by auto
    from ‹⟨ϑ,q⟩∈π›
    have ϑ∈M
      using domain_trans[OF trans_M ‹π∈_›] by auto
    with ‹π∈M› ‹nenv ∈ _› ‹env = _›
    have [val(P,G,ϑ), val(P,G,π)] @ env ∈list(M[G])
      using GenExt_def by auto
    with  Eq4 ‹val(P,G,ϑ)=x› ‹val(P,G,π) = c› ‹x ∈ val(P,G,π)› nth ‹ϑ∈M›
    have Eq5: M[G],  [val(P,G,ϑ)] @ env @[val(P,G,π)] ⊨ And(Member(0,1 #+
length(env)),φ)
      by auto

    with ‹ϑ∈M› ‹π∈M›  Eq5 ‹M_generic(G)› ‹φ∈formula› ‹nenv ∈ _ › ‹env = _ ›
map_nenv
      ‹arity(?χ) ≤ length([ϑ] @ nenv @ [π])›
    have (∃r∈G. M,  [r,P,leq,one,ϑ] @ nenv @[π] ⊨ forces(?χ))
      using truth_lemma[of ·· 0 ∈ (1 #+ length(env)) · ∧ φ ·]
      by auto
    then obtain r where
      r∈G M,  [r,P,leq,one,ϑ] @ nenv @ [π] ⊨ forces(?χ) by auto
    with ‹filter(G)› and ‹q∈G› obtain p where
      p∈G p⪯q p⪯r
      unfolding filter_def compat_in_def by force
    with ‹r∈G›  ‹q∈G› ‹G⊆P›
    have p∈P r∈P q∈P p∈M
      using  P_in_M  by (auto simp add:transitivity)
    with ‹φ∈formula› ‹ϑ∈M› ‹π∈M›  ‹p⪯r› ‹nenv ∈ _› ‹arity(?χ) ≤ length([ϑ] @
nenv @ [π])›
      ‹M, [r,P,leq,one,ϑ] @ nenv @ [π] ⊨ forces(?χ)› ‹env∈_›
    have M,  [p,P,leq,one,ϑ] @ nenv @ [π] ⊨ forces(?χ)
      using strengthening_lemma
```

```
    by simp
  with ‹p∈P› ‹φ∈formula› ‹ϑ∈M› ‹π∈M› ‹nenv ∈ _› ‹arity(?χ) ≤ length([ϑ] @
nenv @ [π])›
  have ∀F. M_generic(F) ∧ p ∈ F ⟶
            M[F],  map(val(P,F), [ϑ] @ nenv @[π]) ⊨ ?χ
    using definition_of_forcing[where φ=·· 0 ∈ (1 #+ length(env)) · ∧ φ ·]
    by simp
  with ‹p∈P› ‹ϑ∈M›
  have Eq6: ∃ϑ'∈M. ∃p'∈P. ⟨ϑ,p⟩ = <ϑ',p'> ∧ (∀F. M_generic(F) ∧ p' ∈ F
⟶
            M[F],  map(val(P,F), [ϑ'] @ nenv @ [π]) ⊨ ?χ) by auto
  from ‹π∈M› ‹⟨ϑ,q⟩∈π›
  have ⟨ϑ,q⟩ ∈ M by (simp add:transitivity)
  from ‹⟨ϑ,q⟩∈π› ‹ϑ∈M› ‹p∈P› ‹p∈M›
  have ⟨ϑ,p⟩∈M ⟨ϑ,p⟩∈domain(π)×P
    using pair_in_M_iff by auto
  with ‹ϑ∈M› Eq6 ‹p∈P›
  have M, [⟨ϑ,p⟩] @ ?Pl1 @ [π] @ nenv ⊨ ?ψ
    using Equivalence by auto
  with ‹⟨ϑ,p⟩∈domain(π)×P›
  have ⟨ϑ,p⟩∈?n by simp
  with ‹p∈G› ‹p∈P›
  have val(P,G,ϑ)∈val(P,G,?n)
    using val_of_elem[of ϑ p] by simp
  with ‹val(P,G,ϑ)=x›
  show x∈val(P,G,?n) by simp
 qed
 with val_m first_incl
 have val(P,G,?n) = {x∈c. (M[G], [x] @ env @ [c] ⊨ φ)} by auto
 also
 have ... = {x∈c. (M[G], [x] @ env ⊨ φ)}
 proof -
  {
   fix x
   assume x∈c
   moreover from assms
   have c∈M[G]
    unfolding GenExt_def by auto
   moreover from this and ‹x∈c›
   have x∈M[G]
    using transitivity_MG
    by simp
   ultimately
   have (M[G], ([x] @ env) @[c] ⊨ φ) ⟷ (M[G], [x] @ env ⊨ φ)
    using phi ‹env ∈ _› by (rule_tac arity_sats_iff, simp_all)
  }
  then show ?thesis by auto
 qed
 finally
```

show $\{x{\in}c.\ (M[G],\ [x]\ @\ env \models \varphi)\}{\in}\ M[G]$
using ‹*?n∈M*› *GenExt_def* **by** *force*
qed

theorem *separation_in_MG*:
assumes
$\varphi{\in}formula$ **and** $arity(\varphi) \le 1\ \#{+}\ length(env)$ **and** $env{\in}list(M[G])$
shows
$separation(\#\#M[G],\lambda x.\ (M[G],\ [x]\ @\ env \models \varphi))$
proof -
{
fix *c*
assume $c{\in}M[G]$
moreover from ‹*env* ∈ _›
obtain *nenv* **where** $nenv{\in}list(M)$
$env = map(val(P,G),nenv)\ length(env) = length(nenv)$
using *GenExt_def map_val*[*of env*] **by** *auto*
moreover note ‹φ ∈ _› ‹$arity(\varphi) \le$ _› ‹*env* ∈ _›
ultimately
have *Eq1*: $\{x{\in}c.\ (M[G],\ [x]\ @\ env \models \varphi)\} \in M[G]$
using *Collect_sats_in_MG* **by** *auto*
}
then
show *?thesis*
using *separation_iff rev_bexI* **unfolding** *is_Collect_def* **by** *force*
qed

end

end

36 The Axiom of Pairing in $M[G]$

theory *Pairing_Axiom* **imports** *Names* **begin**

context *forcing_data*
begin

lemma *val_Upair* :
$one \in G \Longrightarrow val(P,G,\{\langle\tau,one\rangle,\langle\varrho,one\rangle\}) = \{val(P,G,\tau),val(P,G,\varrho)\}$
by (*insert one_in_P*, *rule trans*, *subst def_val*,*auto simp add*: *Sep_and_Replace*)

lemma *pairing_in_MG* :
assumes $M\_generic(G)$
shows $upair\_ax(\#\#M[G])$
proof -
{
fix *x y*
have $one{\in}G$ **using** *assms one_in_G* **by** *simp*

from *assms*
have $G{\subseteq}P$ **unfolding** *M_generic_def* **and** *filter_def* **by** *simp*
with ‹*one*∈*G*›
have *one*∈*P* **using** *subsetD* **by** *simp*
then
have *one*∈*M* **using** *transitivity*[*OF* _ *P_in_M*] **by** *simp*
assume $x \in M[G]$ $y \in M[G]$
then
obtain τ ϱ **where**
0 : $val(P,G,\tau) = x$ $val(P,G,\varrho) = y$ $\varrho \in M$ $\tau \in M$
using *GenExtD* **by** *blast*
with ‹*one*∈*M*›
have $\langle\tau,one\rangle \in M$ $\langle\varrho,one\rangle{\in}M$ **using** *pair_in_M_iff* **by** *auto*
then
have *1*: $\{\langle\tau,one\rangle,\langle\varrho,one\rangle\} \in M$ (**is** *?σ* ∈ _) **using** *upair_in_M_iff* **by** *simp*
then
have $val(P,G,?\sigma) \in M[G]$ **using** *GenExtI* **by** *simp*
with *1*
have $\{val(P,G,\tau),val(P,G,\varrho)\} \in M[G]$ **using** *val_Upair assms one_in_G* **by** *simp*
with *0*
have $\{x,y\} \in M[G]$ **by** *simp*
}
then show *?thesis* **unfolding** *upair_ax_def upair_def* **by** *auto*
qed

end
end

37 The Axiom of Unions in $M[G]$

theory *Union_Axiom*
imports *Names*
begin

context *forcing_data*
begin

definition *Union_name_body* :: $[i,i,i,i] \Rightarrow o$ **where**
Union_name_body($P',leq',\tau,\vartheta p$) ≡ ($\exists$ σ[##*M*].
$\exists$ q[##*M*]. ($q{\in}$ $P' \wedge$ ($\langle\sigma,q\rangle \in \tau \wedge$
($\exists$ r[##*M*].$r{\in}P' \wedge$ ($\langle fst(\vartheta p),r\rangle \in \sigma \wedge \langle snd(\vartheta p),r\rangle \in leq' \wedge \langle snd(\vartheta p),q\rangle \in leq'$)))))

definition *Union_name_fm* :: *i* **where**
Union_name_fm ≡
Exists(
Exists(*And*(*pair_fm*(*1*,*0*,*2*),

```
    Exists (
    Exists (And(Member(0,7),
      Exists (And(And(pair_fm(2,1,0),Member(0,6)),
        Exists (And(Member(0,9),
         Exists (And(And(pair_fm(6,1,0),Member(0,4)),
          Exists (And(And(pair_fm(6,2,0),Member(0,10)),
          Exists (And(pair_fm(7,5,0),Member(0,11))))))))))))))))

lemma Union_name_fm_type [TC]:
  Union_name_fm ∈formula
  unfolding Union_name_fm_def by simp


lemma arity_Union_name_fm :
  arity(Union_name_fm) = 4
  unfolding Union_name_fm_def upair_fm_def pair_fm_def
  by(auto simp add: ord_simp_union)

lemma sats_Union_name_fm :
  ⟦ env ∈ list(M); P' ∈ M ; p ∈ M ; ϑ ∈ M ; τ ∈ M ; leq' ∈ M ⟧ ⟹
     sats(M,Union_name_fm,[⟨ϑ,p⟩,τ,leq',P']@env) ⟷
     Union_name_body(P',leq',τ,⟨ϑ,p⟩)
  unfolding Union_name_fm_def Union_name_body_def
  by (simp_all add:pair_in_M_iff[simplified])

definition Union_name :: i ⇒ i where
  Union_name(τ) ≡
    {u ∈ domain(⋃(domain(τ))) × P . Union_name_body(P,leq,τ,u)}

lemma Union_name_M : assumes τ ∈ M
  shows Union_name(τ) ∈ M
proof -
  let ?P=λ x . sats(M,Union_name_fm,[x,τ,leq,P])
  let ?Q=λ x . Union_name_body(P,leq,τ,x)
  from ‹τ∈M›
 have domain(⋃(domain(τ)))∈M (is ?d ∈ _) using domain_closed Union_closed
by simp
  then
  have ?d × P ∈ M using cartprod_closed P_in_M by simp
  have arity(Union_name_fm)≤4 using arity_Union_name_fm by simp
  with ‹τ∈M› P_in_M leq_in_M
  have separation(##M,?P)
    using separation_ax by simp
  with ‹?d × P ∈ M›
  have A:{ u ∈ ?d × P . ?P(u) } ∈ M
    using separation_iff by force
  have ?P(x)⟷ ?Q(x) if x∈?d×P for x
  proof -
    from ‹x∈?d×P›
```

```
    have x = ⟨fst(x),snd(x)⟩ using Pair_fst_snd_eq by simp
    with ‹x∈?d×P› ‹?d∈M›
    have fst(x) ∈ M snd(x) ∈ M
      using transitivity fst_type snd_type P_in_M by auto
    then
    have ?P(⟨fst(x),snd(x)⟩) ⟷ ?Q(⟨fst(x),snd(x)⟩)
      using P_in_M sats_Union_name_fm P_in_M ‹τ∈M› leq_in_M by simp
    with ‹x = ⟨fst(x),snd(x)⟩›
    show ?P(x) ⟷ ?Q(x) using ‹x∈_› by simp
  qed
  then show ?thesis using Collect_cong A unfolding Union_name_def by simp
qed

lemma Union_MG_Eq :
  assumes a ∈ M[G] and a = val(P,G,τ) and filter(G) and τ ∈ M
  shows ⋃ a = val(P,G,Union_name(τ))
proof -
  {
    fix x
    assume x ∈ ⋃ (val(P,G,τ))
    then obtain i where i ∈ val(P,G,τ) x ∈ i by blast
    with ‹τ ∈ M› obtain σ q where
      q ∈ G ⟨σ,q⟩ ∈ τ val(P,G,σ) = i σ ∈ M
      using elem_of_val_pair domain_trans[OF trans_M] by blast
    with ‹x ∈ i› obtain ϑ r where
      r ∈ G ⟨ϑ,r⟩ ∈ σ val(P,G,ϑ) = x ϑ ∈ M
      using elem_of_val_pair domain_trans[OF trans_M] by blast
    with ‹⟨σ,q⟩∈τ› have ϑ ∈ domain(⋃(domain(τ))) by auto
    with ‹filter(G)› ‹q∈G› ‹r∈G› obtain p where
      A: p ∈ G ⟨p,r⟩ ∈ leq ⟨p,q⟩ ∈ leq p ∈ P r ∈ P q ∈ P
      using low_bound_filter filterD by blast
    then
    have p ∈ M q∈M r∈M
      using P_in_M by (auto dest:transM)
    with A ‹⟨ϑ,r⟩ ∈ σ› ‹⟨σ,q⟩ ∈ τ› ‹ϑ ∈ M› ‹ϑ ∈ domain(⋃(domain(τ)))› ‹σ∈M›
    have ⟨ϑ,p⟩ ∈ Union_name(τ)
      unfolding Union_name_def Union_name_body_def
      by auto
    with ‹p∈P› ‹p∈G›
    have val(P,G,ϑ) ∈ val(P,G,Union_name(τ))
      using val_of_elem by simp
    with ‹val(P,G,ϑ)=x›
    have x ∈ val(P,G,Union_name(τ)) by simp
  }
  with ‹a=val(P,G,τ)›
  have 1: x ∈ ⋃ a ⟹ x ∈ val(P,G,Union_name(τ)) for x by simp
  {
    fix x
    assume x ∈ (val(P,G,Union_name(τ)))
```

```
    then obtain ϑ p where
      p ∈ G ⟨ϑ,p⟩ ∈ Union_name(τ) val(P,G,ϑ) = x
      using elem_of_val_pair by blast
    with ‹filter(G)› have p∈P using filterD by simp
    from ‹⟨ϑ,p⟩ ∈ Union_name(τ)› obtain σ q r where
      σ ∈ domain(τ)  ⟨σ,q⟩ ∈ τ  ⟨ϑ,r⟩ ∈ σ r∈P q∈P ⟨p,r⟩ ∈ leq ⟨p,q⟩ ∈ leq
      unfolding Union_name_def Union_name_body_def by force
    with ‹p∈G› ‹filter(G)› have r ∈ G q ∈ G
      using filter_leqD by auto
    with ‹⟨ϑ,r⟩ ∈ σ› ‹⟨σ,q⟩∈τ› ‹q∈P› ‹r∈P› have
      val(P,G,σ) ∈ val(P,G,τ) val(P,G,ϑ) ∈ val(P,G,σ)
      using val_of_elem by simp+
    then have val(P,G,ϑ) ∈ ⋃ val(P,G,τ) by blast
    with ‹val(P,G,ϑ)=x› ‹a=val(P,G,τ)› have
      x ∈ ⋃ a by simp
  }
  with ‹a=val(P,G,τ)›
  have x ∈ val(P,G,Union_name(τ)) ⟹ x ∈ ⋃ a for x by blast
  then
  show ?thesis using 1 by blast
qed

lemma union_in_MG : assumes filter(G)
  shows Union_ax(##M[G])
proof -
  { fix a
    assume a ∈ M[G]
    then
    interpret mgtrans : M_trans ##M[G]
      using transitivity_MG by (unfold_locales; auto)
    from ‹a∈_› obtain τ where τ ∈ M a=val(P,G,τ) using GenExtD by blast
    then
     have Union_name(τ) ∈ M (is ?π ∈ _) using Union_name_M unfolding
Union_name_def by simp
    then
    have val(P,G,?π) ∈ M[G] (is ?U ∈ _) using GenExtI by simp
    with ‹a∈_›
    have (##M[G])(a) (##M[G])(?U) by auto
    with ‹τ ∈ M› ‹filter(G)› ‹?U ∈ M[G]› ‹a=val(P,G,τ)›
    have big_union(##M[G],a,?U)
      using Union_MG_Eq Union_abs by simp
    with ‹?U ∈ M[G]›
    have ∃ z[##M[G]]. big_union(##M[G],a,z) by auto
  }
  then
  show ?thesis unfolding Union_ax_def by simp
qed

theorem Union_MG : M_generic(G) ⟹ Union_ax(##M[G])
```

by (*simp add*:*M_generic_def union_in_MG*)

end
end

38 The Powerset Axiom in $M[G]$

theory *Powerset_Axiom*
imports *Renaming_Auto Separation_Axiom Pairing_Axiom Union_Axiom*
begin

simple_rename *perm_pow* **src** [*ss*,*p*,*l*,*o*,*fs*,χ] **tgt** [*fs*,*ss*,*sp*,*p*,*l*,*o*,χ]

lemma *Collect_inter_Transset*:
assumes
Transset(M) $b \in M$
shows
$\{x{\in}b\ .\ P(x)\} = \{x{\in}b\ .\ P(x)\} \cap M$
using *assms* **unfolding** *Transset_def*
by (*auto*)

context *G_generic* **begin**

lemma *name_components_in_M*:
assumes $\langle\sigma,p\rangle{\in}\vartheta$ $\vartheta \in M$
shows $\sigma{\in}M$ $p{\in}M$
proof -
from *assms* **obtain** a **where**
$\sigma \in a$ $p \in a$ $a{\in}\langle\sigma,p\rangle$
unfolding *Pair_def* **by** *auto*
moreover from *assms*
have $\langle\sigma,p\rangle{\in}M$
using *transitivity* **by** *simp*
moreover from *calculation*
have $a{\in}M$
using *transitivity* **by** *simp*
ultimately
show $\sigma{\in}M$ $p{\in}M$
using *transitivity* **by** *simp_all*
qed

lemma *sats_fst_snd_in_M*:
assumes
$A{\in}M$ $B{\in}M$ $\varphi \in$ *formula* $p{\in}M$ $l{\in}M$ $o{\in}M$ $\chi{\in}M$
arity(φ) $\leq$ 6
shows
$\{\langle s,q\rangle{\in}A{\times}B\ .\ M,\ [q,p,l,o,s,\chi] \models \varphi\} \in M$
(**is** *?*$\vartheta \in M$)
proof -

have $6{\in}nat\ 7{\in}nat$ **by** *simp_all*
let $?\varphi' = ren(\varphi)\text{`}6\text{`}7\text{`}perm\_pow\_fn$
from ‹$A{\in}M$› ‹$B{\in}M$› **have**
  $A{\times}B \in M$
  using *cartprod_closed* **by** *simp*
from ‹$arity(\varphi) \le 6$› ‹$\varphi{\in}\ formula$› ‹$6{\in}\_$› ‹$7{\in}\_$›
have $?\varphi' \in formula\ arity(?\varphi'){\le}7$
  unfolding *perm_pow_fn_def*
  using *perm_pow_thm arity_ren ren_tc Nil_type*
  by *auto*
with ‹$?\varphi' \in formula$›
have *1*: $arity(Exists(Exists(And(pair\_fm(0,1,2),?\varphi')))){\le}5$ (**is** $arity(?\psi){\le}5$)
  unfolding *pair_fm_def upair_fm_def*
  using *ord_simp_union pred_le arity_type* **by** *auto*
{
  fix *sp*
  note ‹$A{\times}B \in M$›
  moreover
  assume $sp \in A{\times}B$
  moreover from *calculation*
  have $fst(sp) \in A\ snd(sp) \in B$
    using *fst_type snd_type* **by** *simp_all*
  ultimately
  have $sp \in M\ fst(sp) \in M\ snd(sp) \in M$
    using ‹$A{\in}M$› ‹$B{\in}M$› *transitivity*
    by *simp_all*
  note $inM =$ ‹$A{\in}M$› ‹$B{\in}M$› ‹$p{\in}M$› ‹$l{\in}M$› ‹$o{\in}M$› ‹$\chi{\in}M$›
    ‹$sp{\in}M$› ‹$fst(sp){\in}M$› ‹$snd(sp){\in}M$›
  with *1* ‹$sp \in M$› ‹$?\varphi' \in formula$›
 have $(M, [sp,p,l,o,\chi]@[p] \models ?\psi) \longleftrightarrow M,[sp,p,l,o,\chi] \models ?\psi$ (**is** $(M, ?env0@\_{\models}\_)$
$\longleftrightarrow \_$)
    using *arity_sats_iff*[*of* $?\psi$ $[p]$ M *?env0*] **by** *auto*
  also from *inM* ‹$sp \in A{\times}B$›
  have $... \longleftrightarrow sats(M,?\varphi',[fst(sp),snd(sp),sp,p,l,o,\chi])$
    by *auto*
  also from *inM* ‹$\varphi \in formula$› ‹$arity(\varphi) \le 6$›
  have $... \longleftrightarrow M, [snd(sp),p,l,o,fst(sp),\chi] \models \varphi$
    (**is** $sats(\_,\_,?env1) \longleftrightarrow sats(\_,\_,?env2)$)
  using *sats_iff_sats_ren*[*of* φ *6 7 ?env2 M ?env1 perm_pow_fn*] *perm_pow_thm*
    unfolding *perm_pow_fn_def* **by** *simp*
  finally
  have $(M,[sp,p,l,o,\chi,p] \models ?\psi) \longleftrightarrow M, [snd(sp),p,l,o,fst(sp),\chi] \models \varphi$
    by *simp*
}
then have
  $?\vartheta = \{sp{\in}A{\times}B\ .\ sats(M,?\psi,[sp,p,l,o,\chi,p])\}$
  by *auto*
also from *assms* ‹$A{\times}B{\in}M$› **have**
  $... \in M$

```
  proof -
    from 1
    have arity(?ψ) ≤ 6
      using leI by simp
    moreover from ‹?φ' ∈ formula›
    have ?ψ ∈ formula
      by simp
    moreover note assms ‹A×B∈M›
    ultimately
    show {x ∈ A×B . M, [x, p, l, o, χ, p] ⊨ ?ψ} ∈ M
      using separation_ax separation_iff
      by simp
  qed
  finally show ?thesis .
qed

lemma Pow_inter_MG:
  assumes
    a∈M[G]
  shows
    Pow(a) ∩ M[G] ∈ M[G]
proof -
  from assms obtain τ where τ ∈ M val(P,G, τ) = a
    using GenExtD by auto
  let ?Q=Pow(domain(τ)×P) ∩ M
  from ‹τ∈M›
  have domain(τ)×P ∈ M domain(τ) ∈ M
    using domain_closed cartprod_closed P_in_M
    by simp_all
  then
  have ?Q ∈ M
  proof -
    from power_ax ‹domain(τ)×P ∈ M› obtain Q where
      powerset(##M,domain(τ)×P,Q) Q ∈ M
      unfolding power_ax_def by auto
    moreover from calculation
    have z∈Q ⟹ z∈M for z
      using transitivity by blast
    ultimately
    have Q = {a∈Pow(domain(τ)×P) . a∈M}
      using ‹domain(τ)×P ∈ M› powerset_abs[of domain(τ)×P Q]
      by (simp flip: setclass_iff)
    also
    have ... = ?Q
      by auto
    finally
    show ?thesis using ‹Q∈M› by simp
  qed
  let ?π=?Q×{one}
```

let *?b*=*val*(*P*,*G*,*?π*)
from ‹*?Q*∈*M*›
have *?π*∈*M*
  using *one_in_P P_in_M transitivity*
  by (*simp flip*: *setclass_iff*)
then
have *?b* ∈ *M*[*G*]
  using *GenExtI* **by** *simp*
have *Pow*(*a*) ∩ *M*[*G*] ⊆ *?b*
proof
  fix *c*
  assume *c* ∈ *Pow*(*a*) ∩ *M*[*G*]
  then obtain *χ* **where** *c*∈*M*[*G*] *χ* ∈ *M val*(*P*,*G*,*χ*) = *c*
    using *GenExtD* **by** *auto*
  let *?ϑ*={⟨*σ*,*p*⟩ ∈*domain*(*τ*)×*P* . *p* ⊩ ·*0* ∈ *1*· [*σ*,*χ*] }
  have *arity*(*forces*(*Member*(*0*,*1*))) = *6*
    using *arity_forces_at* **by** *auto*
  with ‹*domain*(*τ*) ∈ *M*› ‹*χ* ∈ *M*›
  have *?ϑ* ∈ *M*
    using *P_in_M one_in_M leq_in_M sats_fst_snd_in_M*
    by *simp*
  then
  have *?ϑ* ∈ *?Q* **by** *auto*
  then
  have *val*(*P*,*G*,*?ϑ*) ∈ *?b*
    using *one_in_G one_in_P generic val_of_elem* [*of ?ϑ one ?π G*]
    by *auto*
  have *val*(*P*,*G*,*?ϑ*) = *c*
  proof(*intro equalityI subsetI*)
    fix *x*
    assume *x* ∈ *val*(*P*,*G*,*?ϑ*)
    then obtain *σ p* **where**
      1: ⟨*σ*,*p*⟩∈*?ϑ p*∈*G val*(*P*,*G*,*σ*) = *x*
      using *elem_of_val_pair*
      by *blast*
    moreover from ‹⟨*σ*,*p*⟩∈*?ϑ*› ‹*?ϑ* ∈ *M*›
    have *σ*∈*M*
      using *name_components_in_M*[*of _ _ ?ϑ*] **by** *auto*
    moreover from *1*
    have *p* ⊩ ·*0* ∈ *1*· [*σ*,*χ*] *p*∈*P*
      by *simp_all*
    moreover
    note ‹*val*(*P*,*G*,*χ*) = *c*›
    ultimately
    have *M*[*G*], [*x*, *c*] ⊨ ·*0* ∈ *1*·
      using ‹*χ* ∈ *M*› *generic definition_of_forcing*[**where** *φ*=·*0* ∈ *1*·]
        ord_simp_union **by** *auto*
    moreover
    have *x*∈*M*[*G*]

```
    using ‹val(P,G,σ) =  x› ‹σ∈M›  ‹χ∈M› GenExtI by blast
  ultimately
  show x∈c
    using ‹c∈M[G]› by simp
next
  fix x
  assume x ∈ c
  with ‹c ∈ Pow(a) ∩ M[G]›
  have x ∈ a c∈M[G] x∈M[G]
    using transitivity_MG by auto
  with ‹val(P,G, τ) = a›
  obtain σ where σ∈domain(τ) val(P,G,σ) = x
    using elem_of_val by blast
  moreover note ‹x∈c› ‹val(P,G,χ) = c›
  moreover from calculation
  have val(P,G,σ) ∈ val(P,G,χ)
    by simp
  moreover note ‹c∈M[G]› ‹x∈M[G]›
  moreover from calculation
  have M[G], [x, c] ⊨ ·0 ∈ 1·
    by simp
  moreover
  have σ∈M
  proof -
    from ‹σ∈domain(τ)›
    obtain p where ⟨σ,p⟩ ∈ τ
      by auto
    with ‹τ∈M›
    show ?thesis
      using name_components_in_M by blast
  qed
  moreover
  note ‹χ ∈ M›
  ultimately
  obtain p where p∈G p ⊩ ·0 ∈ 1· [σ,χ]
    using generic truth_lemma[of ·0 ∈ 1· G [σ,χ] ] ord_simp_union
    by auto
  moreover from ‹p∈G›
  have p∈P
    using generic by blast
  ultimately
  have ⟨σ,p⟩∈?ϑ
    using ‹σ∈domain(τ)› by simp
  with ‹val(P,G,σ) =  x› ‹p∈G›
  show x∈val(P,G,?ϑ)
    using val_of_elem [of _ _ ?ϑ] by auto
qed
with ‹val(P,G,?ϑ) ∈ ?b›
show c∈?b by simp
```

```
  qed
  then
  have Pow(a) ∩ M[G] = {x∈?b . x⊆a ∧ x∈M[G]}
    by auto
  also from ‹a∈M[G]›
  have  ... = {x∈?b . ( M[G], [x,a] ⊨ ·0 ⊆ 1· ) ∧ x∈M[G]}
    using Transset_MG by force
  also
  have  ... = {x∈?b . ( M[G], [x,a] ⊨ ·0 ⊆ 1· )} ∩ M[G]
    by auto
  also from ‹?b∈M[G]›
  have  ... = {x∈?b . ( M[G], [x,a] ⊨ ·0 ⊆ 1· )}
    using Collect_inter_Transset Transset_MG
    by simp
  also from ‹?b∈M[G]› ‹a∈M[G]›
  have  ... ∈ M[G]
    using Collect_sats_in_MG GenExtI ord_simp_union by simp
  finally show ?thesis .
qed
end


context G_generic begin

interpretation mgtriv: M_trivial ##M[G]
  using generic Union_MG pairing_in_MG zero_in_MG transitivity_MG
  unfolding M_trivial_def M_trans_def M_trivial_axioms_def by (simp; blast)


theorem power_in_MG : power_ax(##(M[G]))
  unfolding power_ax_def
proof (intro rallI, simp only:setclass_iff rex_setclass_is_bex)

  fix a
  assume a ∈ M[G]
  then
  have (##M[G])(a) by simp
  have {x∈Pow(a) . x ∈ M[G]} = Pow(a) ∩ M[G]
    by auto
  also from ‹a∈M[G]›
  have  ... ∈ M[G]
    using Pow_inter_MG by simp
  finally
  have {x∈Pow(a) . x ∈ M[G]} ∈ M[G] .
  moreover from ‹a∈M[G]› ‹{x∈Pow(a) . x ∈ M[G]} ∈ _›
  have powerset(##M[G], a, {x∈Pow(a) . x ∈ M[G]})
    using mgtriv.powerset_abs[OF ‹(##M[G])(a)›]
    by simp
  ultimately
```

show $\exists\, x{\in}M[G]\ .\ powerset(\#\#M[G],\ a,\ x)$
by *auto*
qed

end

end

39 The Axiom of Extensionality in $M[G]$

theory *Extensionality_Axiom*
imports
Names
begin

context *forcing_data*
begin

lemma *extensionality_in_MG* : *extensionality*$(\#\#(M[G]))$
proof -
{
fix $x\ y\ z$
assume
asms: $x{\in}M[G]\ y{\in}M[G]\ (\forall\, w{\in}M[G]\ .\ w \in x \longleftrightarrow w \in y)$
from ‹$x{\in}M[G]$› **have**
$z{\in}x \longleftrightarrow z{\in}M[G] \wedge z{\in}x$
using *transitivity_MG* **by** *auto*
also have
$\ldots \longleftrightarrow z{\in}y$
using *asms transitivity_MG* **by** *auto*
finally have
$z{\in}x \longleftrightarrow z{\in}y$ **.**
}
then have
$\forall\, x{\in}M[G]\ .\ \forall\, y{\in}M[G]\ .\ (\forall\, z{\in}M[G]\ .\ z \in x \longleftrightarrow z \in y) \longrightarrow x = y$
by *blast*
then show *?thesis* **unfolding** *extensionality_def* **by** *simp*
qed

end
end

40 The Axiom of Foundation in $M[G]$

theory *Foundation_Axiom*
imports
Names
begin

context *forcing_data*
begin

lemma *foundation_in_MG* : *foundation_ax*(##(*M*[*G*]))
unfolding *foundation_ax_def*
by (*rule rallI*, *cut_tac A*=*x* **in** *foundation*, *auto intro*: *transitivity_MG*)

lemma *foundation_ax*(##(*M*[*G*]))
proof -
{
fix *x*
assume *x*∈*M*[*G*] ∃ *y*∈*M*[*G*] . *y*∈*x*
then
have ∃ *y*∈*M*[*G*] . *y*∈*x*∩*M*[*G*] **by** *simp*
then
obtain *y* **where** *y*∈*x*∩*M*[*G*] ∀ *z*∈*y*. *z* ∉ *x*∩*M*[*G*]
using *foundation*[*of x*∩*M*[*G*]] **by** *blast*
then
have ∃ *y*∈*M*[*G*] . *y* ∈ *x* ∧ (∀ *z*∈*M*[*G*] . *z* ∉ *x* ∨ *z* ∉ *y*)**by** *auto*
}
then show *?thesis*
unfolding *foundation_ax_def* **by** *auto*
qed

end
end

41 The Axiom of Replacement in $M[G]$

theory *Replacement_Axiom*
imports
Least Relative_Univ Separation_Axiom Renaming_Auto
begin

rename *renrep1* **src** [*p*,*P*,*leq*,*o*,*ϱ*,*τ*] **tgt** [*V*,*τ*,*ϱ*,*p*,*α*,*P*,*leq*,*o*]

definition *renrep_fn* :: *i* ⇒ *i* **where**
renrep_fn(*env*) ≡ *rsum*(*renrep1_fn*,*id*(*length*(*env*)),6,8,*length*(*env*))

definition
renrep :: [*i*,*i*] ⇒ *i* **where**
renrep(*φ*,*env*) = *ren*(*φ*)`(6#+*length*(*env*))`(8#+*length*(*env*))`*renrep_fn*(*env*)

lemma *renrep_type* [*TC*]:
assumes *φ*∈*formula env* ∈ *list*(*M*)
shows *renrep*(*φ*,*env*) ∈ *formula*

unfolding *renrep_def renrep_fn_def renrep1_fn_def*
using *assms renrep1_thm(1) ren_tc*
by *simp*

lemma *arity_renrep*:
assumes φ∈*formula arity*(φ)≤ *6*#+*length*(*env*) *env* ∈ *list*(*M*)
shows *arity*(*renrep*(φ,*env*)) ≤ *8*#+*length*(*env*)
unfolding *renrep_def renrep_fn_def renrep1_fn_def*
using *assms renrep1_thm(1) arity_ren*
by *simp*

lemma *renrep_sats* :
assumes *arity*(φ) ≤ *6* #+ *length*(*env*)
[*P*,*leq*,*o*,*p*,ϱ,τ] @ *env* ∈ *list*(*M*)
V ∈ *M* α ∈ *M*
φ∈*formula*
shows *sats*(*M*, φ, [*p*,*P*,*leq*,*o*,ϱ,τ] @ *env*) ⟷ *sats*(*M*, *renrep*(φ,*env*), [*V*,τ,ϱ,*p*,α,*P*,*leq*,*o*] @ *env*)
unfolding *renrep_def renrep_fn_def renrep1_fn_def*
by (*rule sats_iff_sats_ren*,*insert assms*, *auto simp add*:*renrep1_thm*(*1*)[*of* _ *M*,*simplified*]
renrep1_thm(*2*)[*simplified*,**where** *p*=*p* **and** α=α])

rename *renpbdy1* **src** [ϱ,*p*,α,*P*,*leq*,*o*] **tgt** [ϱ,*p*,*x*,α,*P*,*leq*,*o*]

definition *renpbdy_fn* :: *i* ⇒ *i* **where**
renpbdy_fn(*env*) ≡ *rsum*(*renpbdy1_fn*,*id*(*length*(*env*)),*6*,*7*,*length*(*env*))

definition
renpbdy :: [*i*,*i*] ⇒ *i* **where**
renpbdy(φ,*env*) = *ren*(φ)‘(*6*#+*length*(*env*))‘(*7*#+*length*(*env*))‘*renpbdy_fn*(*env*)

lemma
renpbdy_type [*TC*]: φ∈*formula* ⟹ *env*∈*list*(*M*) ⟹ *renpbdy*(φ,*env*) ∈ *formula*
unfolding *renpbdy_def renpbdy_fn_def renpbdy1_fn_def*
using *renpbdy1_thm*(*1*) *ren_tc*
by *simp*

lemma *arity_renpbdy*: φ∈*formula* ⟹ *arity*(φ) ≤ *6* #+ *length*(*env*) ⟹ *env*∈*list*(*M*) ⟹ *arity*(*renpbdy*(φ,*env*)) ≤ *7* #+ *length*(*env*)
unfolding *renpbdy_def renpbdy_fn_def renpbdy1_fn_def*
using *renpbdy1_thm*(*1*) *arity_ren*
by *simp*

lemma
sats_renpbdy: *arity*(φ) ≤ *6* #+ *length*(*nenv*) ⟹ [ϱ,*p*,*x*,α,*P*,*leq*,*o*,π] @ *nenv* ∈ *list*(*M*) ⟹ φ∈*formula* ⟹
sats(*M*, φ, [ϱ,*p*,α,*P*,*leq*,*o*] @ *nenv*) ⟷ *sats*(*M*, *renpbdy*(φ,*nenv*), [ϱ,*p*,*x*,α,*P*,*leq*,*o*]

@ *nenv*)
unfolding *renpbdy_def renpbdy_fn_def renpbdy1_fn_def*
by (*rule sats_iff_sats_ren,auto simp add*: *renpbdy1_thm(1)*[*of* _ *M*,*simplified*]
renpbdy1_thm(2)[*simplified*,**where** α=α **and** *x*=*x*])

rename *renbody1* **src** [x,α,P,*leq*,o] **tgt** [α,x,m,P,*leq*,o]

definition *renbody_fn* :: $i \Rightarrow i$ **where**
renbody_fn(*env*) $\equiv$ *rsum*(*renbody1_fn*,*id*(*length*(*env*)),5,6,*length*(*env*))

definition
renbody :: [i,i] $\Rightarrow$ i **where**
renbody(φ,*env*) = *ren*(φ) ‘(5#+*length*(*env*)) ‘(6#+*length*(*env*)) ‘*renbody_fn*(*env*)

lemma
renbody_type [*TC*]: φ∈*formula* $\Longrightarrow$ *env*∈*list*(M) $\Longrightarrow$ *renbody*(φ,*env*) ∈ *formula*
unfolding *renbody_def renbody_fn_def renbody1_fn_def*
using *renbody1_thm(1) ren_tc*
by *simp*

lemma *arity_renbody*: φ∈*formula* $\Longrightarrow$ *arity*(φ) $\leq$ 5 #+ *length*(*env*) $\Longrightarrow$ *env*∈*list*(M)
$\Longrightarrow$
arity(*renbody*(φ,*env*)) $\leq$ 6 #+ *length*(*env*)
unfolding *renbody_def renbody_fn_def renbody1_fn_def*
using *renbody1_thm(1) arity_ren*
by *simp*

lemma
sats_renbody: *arity*(φ) $\leq$ 5 #+ *length*(*nenv*) $\Longrightarrow$ [α,x,m,P,*leq*,o] @ *nenv* ∈ *list*(M)
$\Longrightarrow$ φ∈*formula* $\Longrightarrow$
sats(M, φ, [x,α,P,*leq*,o] @ *nenv*) $\longleftrightarrow$ *sats*(M, *renbody*(φ,*nenv*), [α,x,m,P,*leq*,o]
@ *nenv*)
unfolding *renbody_def renbody_fn_def renbody1_fn_def*
by (*rule sats_iff_sats_ren*, *auto simp add*:*renbody1_thm(1)*[*of* _ *M*,*simplified*]
renbody1_thm(2)[**where** α=α **and** m=m,*simplified*])

context *G_generic*
begin

lemma *pow_inter_M*:
assumes
x∈M y∈M
shows
powerset(##M,x,y) $\longleftrightarrow$ y = *Pow*(x) ∩ M
using *assms* **by** *auto*

schematic_goal *sats_prebody_fm_auto*:
assumes
$\varphi\in formula$ $[P,leq,one,p,\varrho,\pi]$ @ $nenv\in list(M)$ $\alpha\in M$ $arity(\varphi) \le 2 \#+ length(nenv)$
shows
$(\exists\tau\in M.\ \exists\ V\in M.\ is\_Vset(\#\#M,\alpha,V) \wedge \tau\in V \wedge sats(M,forces(\varphi),[p,P,leq,one,\varrho,\tau]$
@ $nenv))$
$\longleftrightarrow sats(M,?prebody\_fm,[\varrho,p,\alpha,P,leq,one]$ @ $nenv)$
apply (*insert assms*; (*rule sep_rules is_Vset_iff_sats*[*OF* _ _ _ _ _ _ *nonempty*[*simplified*]]
| *simp*))
apply (*rule sep_rules is_Vset_iff_sats is_Vset_iff_sats*[*OF* _ _ _ _ _ _
nonempty[*simplified*]] | *simp*)+
apply (*rule nonempty*[*simplified*])
apply (*simp_all*)
apply (*rule length_type*[*THEN nat_into_Ord*], *blast*)+
apply ((*rule sep_rules* | *simp*))
apply ((*rule sep_rules* | *simp*))
apply ((*rule sep_rules* | *simp*))
apply ((*rule sep_rules* | *simp*))
apply ((*rule sep_rules* | *simp*))
apply ((*rule sep_rules* | *simp*))
apply ((*rule sep_rules* | *simp*))
apply (*rule renrep_sats*[*simplified*])
apply (*insert assms*)
apply(*auto simp add*: *renrep_type definability*)
proof -
from *assms*
have $nenv\in list(M)$ **by** *simp*
with ‹$arity(\varphi)\le\_$› ‹$\varphi\in\_$›
show $arity(forces(\varphi)) \le succ(succ(succ(succ(succ(succ(length(nenv)))))))$
using *arity_forces_le* **by** *simp*
qed

synthesize_notc *prebody* **from_schematic** *sats_prebody_fm_auto*

lemma *prebody_fm_type* [*TC*]:
assumes $\varphi\in formula$
$env \in list(M)$
shows $prebody\_fm(\varphi,env)\in formula$
proof -
from ‹$\varphi\in formula$›
have $forces(\varphi)\in formula$ **by** *simp*
then
have $renrep(forces(\varphi),env)\in formula$
using ‹$env\in list(M)$› **by** *simp*
then show *?thesis* **unfolding** *prebody_fm_def* **by** *simp*
qed

declare *is_eclose_fm_def*[*fm_definitions*]

is_eclose_fm_def[*fm_definitions*]
mem_eclose_fm_def[*fm_definitions*]
eclose_n_fm_def[*fm_definitions*]

lemma *sats_prebody_fm*:
assumes
[*P*,*leq*,*one*,*p*,*ϱ*] @ *nenv* ∈*list*(*M*) *φ*∈*formula α*∈*M arity*(*φ*) ≤ 2 #+ *length*(*nenv*)
shows
sats(*M*,*prebody_fm*(*φ*,*nenv*),[*ϱ*,*p*,*α*,*P*,*leq*,*one*] @ *nenv*) ⟷
(∃*τ*∈*M*. ∃ *V*∈*M*. *is_Vset*(##*M*,*α*,*V*) ∧ *τ*∈*V* ∧ *sats*(*M*,*forces*(*φ*),[*p*,*P*,*leq*,*one*,*ϱ*,*τ*] @ *nenv*))
unfolding *prebody_fm_def* **using** *assms sats_prebody_fm_auto* **by** *force*

lemma *arity_prebody_fm*:
assumes
φ∈*formula α*∈*M env* ∈ *list*(*M*) *arity*(*φ*) ≤ 2 #+ *length*(*env*)
shows
arity(*prebody_fm*(*φ*,*env*))≤6 #+ *length*(*env*)
unfolding *prebody_fm_def is_HVfrom_fm_def is_powapply_fm_def*
using *assms fm_definitions ord_simp_union*
arity_renrep[*of forces*(*φ*)] *arity_forces_le*[*simplified*] *pred_le* **by** *auto*

definition
body_fm′ :: [*i*,*i*]⇒*i* **where**
body_fm′(*φ*,*env*) ≡ *Exists*(*Exists*(*And*(*pair_fm*(*0*,*1*,*2*),*renpbdy*(*prebody_fm*(*φ*,*env*),*env*))))

lemma *body_fm′_type*[*TC*]: *φ*∈*formula* ⟹ *env*∈*list*(*M*) ⟹ *body_fm′*(*φ*,*env*)∈*formula*
unfolding *body_fm′_def* **using** *prebody_fm_type*
by *simp*

lemma *arity_body_fm′*:
assumes
φ∈*formula α*∈*M env*∈*list*(*M*) *arity*(*φ*) ≤ 2 #+ *length*(*env*)
shows
arity(*body_fm′*(*φ*,*env*))≤5 #+ *length*(*env*)
unfolding *body_fm′_def*
using *assms fm_definitions ord_simp_union arity_prebody_fm pred_le arity_renpbdy*[*of prebody_fm*(*φ*,*env*)]
by *auto*

lemma *sats_body_fm′*:
assumes
∃ *t p*. *x*=⟨*t*,*p*⟩ *x*∈*M* [*α*,*P*,*leq*,*one*,*p*,*ϱ*] @ *nenv* ∈*list*(*M*) *φ*∈*formula arity*(*φ*) ≤ 2 #+ *length*(*nenv*)
shows
sats(*M*,*body_fm′*(*φ*,*nenv*),[*x*,*α*,*P*,*leq*,*one*] @ *nenv*) ⟷
sats(*M*,*renpbdy*(*prebody_fm*(*φ*,*nenv*),*nenv*),[*fst*(*x*),*snd*(*x*),*x*,*α*,*P*,*leq*,*one*] @ *nenv*)

using *assms fst_snd_closed*[*simplified*,*OF* ‹*x*∈*M*›] **unfolding** *body_fm'_def*
by (*auto*)

definition
body_fm :: [*i*,*i*]⇒*i* **where**
body_fm(φ,*env*) ≡ *renbody*(*body_fm'*(φ,*env*),*env*)

lemma *body_fm_type* [*TC*]: *env*∈*list*(*M*) ⟹ φ∈*formula* ⟹ *body_fm*(φ,*env*)∈*formula*
unfolding *body_fm_def* **by** *simp*

lemma *sats_body_fm*:
assumes
∃ *t p*. *x*=⟨*t*,*p*⟩ [α,*x*,*m*,*P*,*leq*,*one*] @ *nenv* ∈*list*(*M*)
φ∈*formula* *arity*(φ) ≤ *2* #+ *length*(*nenv*)
shows
sats(*M*,*body_fm*(φ,*nenv*),[α,*x*,*m*,*P*,*leq*,*one*] @ *nenv*) ⟷
sats(*M*,*renpbdy*(*prebody_fm*(φ,*nenv*),*nenv*),[*fst*(*x*),*snd*(*x*),*x*,α,*P*,*leq*,*one*] @ *nenv*)
using *assms sats_body_fm' sats_renbody*[*OF _ assms*(*2*), *symmetric*] *arity_body_fm'*
unfolding *body_fm_def*
by *auto*

lemma *sats_renpbdy_prebody_fm*:
assumes
∃ *t p*. *x*=⟨*t*,*p*⟩ *x*∈*M* [α,*m*,*P*,*leq*,*one*] @ *nenv* ∈*list*(*M*)
φ∈*formula* *arity*(φ) ≤ *2* #+ *length*(*nenv*)
shows
sats(*M*,*renpbdy*(*prebody_fm*(φ,*nenv*),*nenv*),[*fst*(*x*),*snd*(*x*),*x*,α,*P*,*leq*,*one*] @ *nenv*)
⟷
sats(*M*,*prebody_fm*(φ,*nenv*),[*fst*(*x*),*snd*(*x*),α,*P*,*leq*,*one*] @ *nenv*)
using *assms fst_snd_closed*[*simplified*,*OF* ‹*x*∈*M*›]
sats_renpbdy[*OF arity_prebody_fm _ prebody_fm_type*, *of concl*:*M*, *symmetric*]
by *force*

lemma *body_lemma*:
assumes
∃ *t p*. *x*=⟨*t*,*p*⟩ *x*∈*M* [*x*,α,*m*,*P*,*leq*,*one*] @ *nenv* ∈*list*(*M*)
φ∈*formula* *arity*(φ) ≤ *2* #+ *length*(*nenv*)
shows
sats(*M*,*body_fm*(φ,*nenv*),[α,*x*,*m*,*P*,*leq*,*one*] @ *nenv*) ⟷
(∃τ∈*M*. ∃ *V*∈*M*. *is_Vset*(λ*a*. (##*M*)(*a*),α,*V*) ∧ τ ∈ *V* ∧ (*snd*(*x*) ⊩ φ ([*fst*(*x*),τ]@*nenv*)))
using *assms sats_body_fm*[*of x α m nenv*] *sats_renpbdy_prebody_fm*[*of x α*]
sats_prebody_fm[*of snd*(*x*) *fst*(*x*)] *fst_snd_closed*[*simplified*,*OF* ‹*x*∈*M*›]
by (*simp*, *simp flip*: *setclass_iff*,*simp*)

lemma *Replace_sats_in_MG*:
assumes
c∈*M*[*G*] *env* ∈ *list*(*M*[*G*])
φ ∈ *formula* *arity*(φ) ≤ *2* #+ *length*(*env*)
univalent(##*M*[*G*], *c*, λ*x v*. (*M*[*G*] , [*x*,*v*]@*env* ⊨ φ))

shows
$\{v.\ x{\in}c,\ v{\in}M[G] \wedge (M[G]\ ,\ [x,v]@env \models \varphi)\} \in M[G]$
proof -
let *?R* = $\lambda\ x\ v\ .\ v{\in}M[G] \wedge (M[G]\ ,\ [x,v]@env \models \varphi)$
from ‹$c{\in}M[G]$›
obtain π' **where** $val(P,G,\ \pi') = c\ \pi' \in M$
using *GenExt_def* **by** *auto*
then
have $domain(\pi'){\times}P{\in}M$ (**is** *?π*$\in M$)
using *cartprod_closed P_in_M domain_closed* **by** *simp*
from ‹$val(P,G,\ \pi') = c$›
have $c \subseteq val(P,G,?\pi)$
using *def_val*[*of G ?π*] *one_in_P one_in_G*[*OF generic*] *elem_of_val*
domain_of_prod[*OF one_in_P, of domain*(π')] **by** *force*
from ‹*env* ∈ _›
obtain *nenv* **where** *nenv*∈*list*(*M*) *env* = *map*(*val*(*P*,*G*),*nenv*)
using *map_val* **by** *auto*
then
have *length*(*nenv*) = *length*(*env*) **by** *simp*
define *f* **where** $f(\varrho p) \equiv \mu\ \alpha.\ \alpha{\in}M \wedge (\exists \tau{\in}M.\ \tau \in Vset(\alpha) \wedge$
$(snd(\varrho p) \Vdash \varphi\ ([fst(\varrho p),\tau]\ @\ nenv)))$ (**is** _ ≡ $\mu\ \alpha.$ *?P*(ϱp,α)) **for** ϱp
have $f(\varrho p) = (\mu\ \alpha.\ \alpha{\in}M \wedge (\exists \tau{\in}M.\ \exists\ V{\in}M.\ is\_Vset(\#\#M,\alpha,V) \wedge \tau{\in}V \wedge$
$(snd(\varrho p) \Vdash \varphi\ ([fst(\varrho p),\tau]\ @\ nenv))))$ (**is** _ = ($\mu\ \alpha.\ \alpha{\in}M \wedge$ *?Q*(ϱp,α))) **for** ϱp
unfolding *f_def* **using** *Vset_abs Vset_closed Ord_Least_cong*[*of ?P*(ϱp) $\lambda\ \alpha.\ \alpha{\in}M \wedge$ *?Q*(ϱp,α)]
by (*simp*, *simp del*:*setclass_iff*)
moreover
have $f(\varrho p) \in M$ **for** ϱp
unfolding *f_def* **using** *Least_closed'*[*of ?P*(ϱp)] **by** *simp*
ultimately
have *1*:*least*(*##M*,$\lambda\alpha.$ *?Q*(ϱp,α),$f(\varrho p)$) **for** ϱp
using *least_abs'*[*of* $\lambda\alpha.\ \alpha{\in}M \wedge$ *?Q*(ϱp,α) $f(\varrho p)$] *least_conj*
by (*simp flip*: *setclass_iff*)
have *Ord*($f(\varrho p)$) **for** ϱp **unfolding** *f_def* **by** *simp*
define *QQ* **where** *QQ*≡*?Q*
from *1*
have *least*(*##M*,$\lambda\alpha.$ *QQ*(ϱp,α),$f(\varrho p)$) **for** ϱp
unfolding *QQ_def* **.**
from ‹*arity*(φ) ≤ _› ‹*length*(*nenv*) = _›
have *arity*(φ) ≤ *2* #+ *length*(*nenv*)
by *simp*
moreover
note *assms* ‹*nenv*∈*list*(*M*)› ‹*?π*∈*M*›
moreover
have $\varrho p{\in}?\pi \Longrightarrow \exists\ t\ p.\ \varrho p{=}\langle t,p\rangle$ **for** ϱp
by *auto*
ultimately
have *body*:(*M* , [α,ϱp,*m*,*P*,*leq*,*one*] @ *nenv* ⊨ *body_fm*(φ,*nenv*)) ⟷ *?Q*(ϱp,α)

```
    if ϱp∈?π ϱp∈M m∈M α∈M for α ϱp m
    using that P_in_M leq_in_M one_in_M body_lemma[of ϱp α m nenv φ] by
simp
  let ?f_fm=least_fm(body_fm(φ,nenv),1)
  {
    fix ϱp m
    assume asm: ϱp∈M ϱp∈?π m∈M
    note inM = this P_in_M leq_in_M one_in_M ‹nenv∈list(M)›
    with body
    have body':⋀α. α ∈ M ⟹ (∃τ∈M. ∃V∈M. is_Vset(λa. (##M)(a), α, V)
∧ τ ∈ V ∧
          (snd(ϱp) ⊩ φ ([fst(ϱp),τ] @ nenv))) ⟷
          M, Cons(α, [ϱp, m, P, leq, one] @ nenv) ⊨ body_fm(φ,nenv) by simp
    from inM
    have (M , [ϱp,m,P,leq,one] @ nenv ⊨ ?f_fm) ⟷ least(##M, QQ(ϱp), m)
      using sats_least_fm[OF body', of 1] unfolding QQ_def
      by (simp, simp flip: setclass_iff)
  }
  then
  have (M, [ϱp,m,P,leq,one] @ nenv ⊨ ?f_fm) ⟷ least(##M, QQ(ϱp), m)
    if ϱp∈M ϱp∈?π m∈M for ϱp m using that by simp
  then
  have univalent(##M, ?π, λϱp m. M , [ϱp,m] @ ([P,leq,one] @ nenv) ⊨ ?f_fm)
    unfolding univalent_def by (auto intro:unique_least)
  moreover from ‹length(_) = _› ‹env ∈ _›
  have length([P,leq,one] @ nenv) = 3 #+ length(env) by simp
  moreover from ‹arity(_) ≤ 2 #+ length(nenv)›
    ‹length(_) = length(_)›[symmetric] ‹nenv∈_› ‹φ∈_›
  have arity(?f_fm) ≤ 5 #+ length(env)
    unfolding body_fm_def fm_definitions least_fm_def
    using arity_forces arity_renrep arity_renbody arity_body_fm' nonempty
   by (simp add: pred_Un Un_assoc, simp add: Un_assoc[symmetric] union_abs1
pred_Un)
      (auto simp add: ord_simp_union, rule pred_le, auto intro:leI)
  moreover from ‹φ∈formula› ‹nenv∈list(M)›
  have ?f_fm∈formula by simp
  moreover
  note inM = P_in_M leq_in_M one_in_M ‹nenv∈list(M)› ‹?π∈M›
  ultimately
  obtain Y where Y∈M
    ∀m∈M. m ∈ Y ⟷ (∃ϱp∈M. ϱp ∈ ?π ∧ (M, [ϱp,m] @ ([P,leq,one] @ nenv)
⊨ ?f_fm))
    using replacement_ax[of ?f_fm [P,leq,one] @ nenv]
    unfolding strong_replacement_def by auto
  with ‹least(_,QQ(_),f(_))› ‹f(_) ∈ M› ‹?π∈M›
    ‹_ ⟹ _ ⟹ _ ⟹ (M,_ ⊨ ?f_fm) ⟷ least(_,_,_)›
  have f(ϱp)∈Y if ϱp∈?π for ϱp
    using that transitivity[OF _ ‹?π∈M›]
    by (clarsimp, rule_tac x=⟨x,y⟩ in bexI, auto)
```

moreover
have $\{y{\in}Y.\ Ord(y)\} \in M$
  using ‹$Y{\in}M$› *separation_ax sats_ordinal_fm trans_M*
    separation_cong[*of* ##M $\lambda y.\ sats(M,ordinal\_fm(0),[y])$ *Ord*]
    separation_closed **by** *simp*
then
have $\bigcup\ \{y{\in}Y.\ Ord(y)\} \in M$ (**is** *?sup* $\in M$)
  using *Union_closed* **by** *simp*
then
have $\{x{\in}Vset(?sup).\ x \in M\} \in M$
  using *Vset_closed* **by** *simp*
moreover
have $\{one\} \in M$
  using *one_in_M singleton_closed* **by** *simp*
ultimately
have $\{x{\in}Vset(?sup).\ x \in M\} \times \{one\} \in M$ (**is** *?big_name* $\in M$)
  using *cartprod_closed* **by** *simp*
then
have $val(P,G,?big\_name) \in M[G]$
  by (*blast intro*:*GenExtI*)
{
  fix $v\ x$
  assume $x{\in}c$
  moreover
  note ‹$val(P,G,\pi')=c$› ‹$\pi'{\in}M$›
  moreover
  from *calculation*
  obtain $\varrho\ p$ **where** $\langle\varrho,p\rangle{\in}\pi'\ \ val(P,G,\varrho) = x\ p{\in}G\ \varrho{\in}M$
    using *elem_of_val_pair'*[*of* $\pi'\ x\ G$] **by** *blast*
  moreover
  assume $v{\in}M[G]$
  then
  obtain σ **where** $val(P,G,\sigma) = v\ \sigma{\in}M$
    using *GenExtD* **by** *auto*
  moreover
  assume $sats(M[G],\ \varphi,\ [x,v]\ @\ env)$
  moreover
  note ‹$\varphi{\in}\_$› ‹$nenv{\in}\_$› ‹$env = \_$› ‹$arity(\varphi){\le}\ 2\ \#{+}\ length(env)$›
  ultimately
  obtain q **where** $q{\in}G\ q \Vdash \varphi\ ([\varrho,\sigma]@nenv)$
    using *truth_lemma*[*OF* ‹$\varphi{\in}\_$› *generic, symmetric, of* $[\varrho,\sigma]\ @\ nenv$]
    by *auto*
  with ‹$\langle\varrho,p\rangle{\in}\pi'$› ‹$\langle\varrho,q\rangle{\in}?\pi \Longrightarrow f(\langle\varrho,q\rangle){\in}Y$›
  have $f(\langle\varrho,q\rangle){\in}Y$
    using *generic* **unfolding** *M_generic_def filter_def* **by** *blast*
  let $?\alpha{=}succ(rank(\sigma))$
  note ‹$\sigma{\in}M$›
  moreover from *this*
  have $?\alpha \in M$

```
    using rank_closed cons_closed by (simp flip: setclass_iff)
  moreover
  have σ ∈ Vset(?α)
    using Vset_Ord_rank_iff by auto
  moreover
  note ‹q ⊩ φ ([ϱ,σ] @ nenv)›
  ultimately
  have ?P(⟨ϱ,q⟩,?α) by (auto simp del: Vset_rank_iff)
  moreover
  have (μ α. ?P(⟨ϱ,q⟩,α)) = f(⟨ϱ,q⟩)
    unfolding f_def by simp
  ultimately
  obtain τ where τ∈M τ ∈ Vset(f(⟨ϱ,q⟩)) q ⊩ φ ([ϱ,τ] @ nenv)
    using LeastI[of λ α. ?P(⟨ϱ,q⟩,α) ?α] by auto
  with ‹q∈G› ‹ϱ∈M› ‹nenv∈_› ‹arity(φ)≤ 2 #+ length(nenv)›
  have M[G], map(val(P,G),[ϱ,τ] @ nenv) ⊨ φ
    using truth_lemma[OF ‹φ∈_› generic, of [ϱ,τ] @ nenv] by auto
  moreover from ‹x∈c› ‹c∈M[G]›
  have x∈M[G] using transitivity_MG by simp
  moreover
 note ‹M[G],[x,v] @ env⊨ φ› ‹env = map(val(P,G),nenv)› ‹τ∈M› ‹val(P,G,ϱ)=x›
    ‹univalent(##M[G],_,_)› ‹x∈c› ‹v∈M[G]›
  ultimately
  have v=val(P,G,τ)
    using GenExtI[of τ G] unfolding univalent_def by (auto)
  from ‹τ ∈ Vset(f(⟨ϱ,q⟩))› ‹Ord(f(_))›  ‹f(⟨ϱ,q⟩)∈Y›
  have τ ∈ Vset(?sup)
    using Vset_Ord_rank_iff lt_Union_iff[of _ rank(τ)] by auto
  with ‹τ∈M›
  have val(P,G,τ) ∈ val(P,G,?big_name)
    using domain_of_prod[of one {one} {x∈Vset(?sup). x ∈ M} ] def_val[of G
?big_name]
       one_in_G[OF generic] one_in_P  by (auto simp del: Vset_rank_iff)
  with ‹v=val(P,G,τ)›
  have v ∈ val(P,G,{x∈Vset(?sup). x ∈ M} × {one})
    by simp
}
then
have {v. x∈c, ?R(x,v)} ⊆ val(P,G,?big_name) (is ?repl⊆?big)
  by blast
with ‹?big_name∈M›
have ?repl = {v∈?big. ∃x∈c. sats(M[G], φ, [x,v] @ env )} (is _ = ?rhs)
proof(intro equalityI subsetI)
  fix v
  assume v∈?repl
  with ‹?repl⊆?big›
  obtain x where x∈c M[G], [x, v] @ env ⊨ φ v∈?big
    using subsetD by auto
  with ‹univalent(##M[G],_,_)› ‹c∈M[G]›
```

```
    show v ∈ ?rhs
      unfolding univalent_def
     using transitivity_MG ReplaceI[of λ x v. ∃ x∈c. M[G], [x, v] @ env ⊨ φ] by
blast
  next
    fix v
    assume v∈?rhs
    then
    obtain x where
      v∈val(P,G, ?big_name) M[G], [x, v] @ env ⊨ φ x∈c
      by blast
    moreover from this ‹c∈M[G]›
    have v∈M[G] x∈M[G]
      using transitivity_MG GenExtI[OF ‹?big_name∈_›,of G] by auto
    moreover from calculation ‹univalent(##M[G],_,_)›
    have ?R(x,y) ⟹ y = v for y
      unfolding univalent_def by auto
    ultimately
    show v∈?repl
      using ReplaceI[of ?R x v c]
      by blast
  qed
  moreover
  let ?ψ = Exists(And(Member(0,2#+length(env)),φ))
  have v∈M[G] ⟹ (∃ x∈c. M[G], [x,v] @ env ⊨ φ) ⟷ M[G], [v] @ env @ [c]
⊨ ?ψ
    arity(?ψ) ≤ 2 #+ length(env) ?ψ∈formula
    for v
  proof -
    fix v
    assume v∈M[G]
    with ‹c∈M[G]›
    have nth(length(env)#+1,[v]@env@[c]) = c
      using ‹env∈_›nth_concat[of v c M[G] env]
      by auto
      note inMG= ‹nth(length(env)#+1,[v]@env@[c]) = c› ‹c∈M[G]› ‹v∈M[G]›
‹env∈_›
    show (∃ x∈c. M[G], [x,v] @ env ⊨ φ) ⟷ M[G], [v] @ env @ [c] ⊨ ?ψ
    proof
      assume ∃ x∈c. M[G], [x, v] @ env ⊨ φ
      then obtain x where
        x∈c M[G], [x, v] @ env ⊨ φ x∈M[G]
        using transitivity_MG[OF _ ‹c∈M[G]›]
        by auto
      with ‹φ∈_› ‹arity(φ)≤2#+length(env)› inMG
       show M[G], [v] @ env @ [c] ⊨ Exists(And(Member(0, 2 #+ length(env)),
φ))
        using arity_sats_iff[of φ [c] _ [x,v]@env]
        by auto
```

```
    next
      assume M[G], [v] @ env @ [c] ⊨ Exists(And(Member(0, 2 #+ length(env)),
φ))
      with inMG
      obtain x where
        x∈M[G] x∈c M[G], [x,v]@env@[c] ⊨ φ
        by auto
      with ‹φ∈_› ‹arity(φ)≤2#+length(env)› inMG
      show ∃ x∈c. M[G], [x, v] @ env⊨ φ
        using arity_sats_iff[of φ [c] _ [x,v]@env]
        by auto
    qed
  next
    from ‹env∈_› ‹φ∈_›
    show arity(?ψ)≤2#+length(env)
      using pred_mono[OF _ ‹arity(φ)≤2#+length(env)›] lt_trans[OF _ le_refl]
      by (auto simp add:ord_simp_union)
  next
    from ‹φ∈_›
    show ?ψ∈formula by simp
  qed
  moreover from this
  have {v∈?big. ∃ x∈c. M[G], [x,v] @ env ⊨ φ} = {v∈?big. M[G], [v] @ env @ [c]
⊨  ?ψ}
    using transitivity_MG[OF _ GenExtI, OF _ ‹?big_name∈M›]
    by simp
  moreover from calculation and ‹env∈_› ‹c∈_› ‹?big∈M[G]›
  have {v∈?big. M[G] , [v] @ env @ [c] ⊨ ?ψ} ∈ M[G]
    using Collect_sats_in_MG by auto
  ultimately
  show ?thesis by simp
qed

theorem strong_replacement_in_MG:
  assumes
    φ∈formula and arity(φ) ≤ 2 #+ length(env) env ∈ list(M[G])
  shows
    strong_replacement(##M[G],λx v. sats(M[G],φ,[x,v] @ env))
proof -
  let ?R=λx y . M[G], [x, y] @ env ⊨ φ
  {
    fix A
    let ?Y={v . x ∈ A, v∈M[G] ∧ ?R(x,v)}
    assume 1: (##M[G])(A)
      ∀ x[##M[G]]. x ∈ A ⟶ (∀ y[##M[G]]. ∀ z[##M[G]]. ?R(x,y) ∧ ?R(x,z)
⟶ y = z)
    then
    have univalent(##M[G], A, ?R) A∈M[G]
      unfolding univalent_def by simp_all
```

with *assms* ‹A∈_›
have $(\#\#M[G])(?Y)$
using *Replace_sats_in_MG* **by** *auto*
have $b \in ?Y \longleftrightarrow (\exists\, x[\#\#M[G]].\ x \in A \wedge ?R(x,b))$ **if** $(\#\#M[G])(b)$ **for** b
proof(*rule*)
from ‹A∈_›
show $\exists\, x[\#\#M[G]].\ x \in A \wedge ?R(x,b)$ **if** $b \in ?Y$
using *that transitivity_MG* **by** *auto*
next
show $b \in ?Y$ **if** $\exists\, x[\#\#M[G]].\ x \in A \wedge ?R(x,b)$
proof -
from ‹$(\#\#M[G])(b)$›
have $b{\in}M[G]$ **by** *simp*
with *that*
obtain x **where** $(\#\#M[G])(x)$ $x{\in}A$ $b{\in}M[G] \wedge ?R(x,b)$
by *blast*
moreover from *this 1* ‹$(\#\#M[G])(b)$›
have $x{\in}M[G]$ $z{\in}M[G] \wedge ?R(x,z) \Longrightarrow b = z$ **for** z
by *auto*
ultimately
show *?thesis*
using *ReplaceI*[*of* λ x y. $y{\in}M[G] \wedge ?R(x,y)$] **by** *blast*
qed
qed
then
have $\forall\, b[\#\#M[G]].\ b \in ?Y \longleftrightarrow (\exists\, x[\#\#M[G]].\ x \in A \wedge ?R(x,b))$
by *simp*
with ‹$(\#\#M[G])(?Y)$›
have $(\exists\, Y[\#\#M[G]].\ \forall\, b[\#\#M[G]].\ b \in Y \longleftrightarrow (\exists\, x[\#\#M[G]].\ x \in A \wedge ?R(x,b)))$
by *auto*
}
then show *?thesis* **unfolding** *strong_replacement_def univalent_def*
by *auto*
qed

end

end

42 The Axiom of Infinity in $M[G]$

theory *Infinity_Axiom*
imports *Pairing_Axiom Union_Axiom Separation_Axiom*
begin

context *G_generic* **begin**

interpretation *mg_triv*: *M_trivial*$\#\#M[G]$

using *transitivity_MG zero_in_MG generic Union_MG pairing_in_MG*
by *unfold_locales auto*

lemma *infinity_in_MG* : *infinity_ax*(##*M*[*G*])
proof -
from *infinity_ax* **obtain** *I* **where**
Eq1: *I*∈*M* *0* ∈ *I* ∀ *y*∈*M*. *y* ∈ *I* ⟶ *succ*(*y*) ∈ *I*
unfolding *infinity_ax_def* **by** *auto*
then
have *check*(*I*) ∈ *M*
using *check_in_M* **by** *simp*
then
have *I*∈ *M*[*G*]
using *valcheck generic one_in_G one_in_P GenExtI*[*of check*(*I*) *G*] **by** *simp*
with ‹*0*∈*I*›
have *0*∈*M*[*G*] **using** *transitivity_MG* **by** *simp*
with ‹*I*∈*M*›
have *y* ∈ *M* **if** *y* ∈ *I* **for** *y*
using *transitivity*[*OF* _ ‹*I*∈*M*›] *that* **by** *simp*
with ‹*I*∈*M*[*G*]›
have *succ*(*y*) ∈ *I* ∩ *M*[*G*] **if** *y* ∈ *I* **for** *y*
using *that Eq1 transitivity_MG* **by** *blast*
with *Eq1* ‹*I*∈*M*[*G*]› ‹*0*∈*M*[*G*]›
show *?thesis*
unfolding *infinity_ax_def* **by** *auto*
qed

end
end

43 The Axiom of Choice in $M[G]$

theory *Choice_Axiom*
imports *Powerset_Axiom Pairing_Axiom Union_Axiom Extensionality_Axiom*
Foundation_Axiom Powerset_Axiom Separation_Axiom
Replacement_Axiom Interface Infinity_Axiom Relativization
begin

definition
induced_surj :: *i*⇒*i*⇒*i*⇒*i* **where**
induced_surj(*f*,*a*,*e*) ≡ *f*-‘‘(*range*(*f*)-*a*)×{*e*} ∪ *restrict*(*f*,*f*-‘‘*a*)

lemma *domain_induced_surj*: *domain*(*induced_surj*(*f*,*a*,*e*)) = *domain*(*f*)
unfolding *induced_surj_def* **using** *domain_restrict domain_of_prod* **by** *auto*

lemma *range_restrict_vimage*:
assumes *function*(*f*)
shows *range*(*restrict*(*f*,*f*-‘‘*a*)) ⊆ *a*
proof

```
  from assms
  have function(restrict(f,f-``a))
    using function_restrictI by simp
  fix y
  assume y ∈ range(restrict(f,f-``a))
  then
  obtain x where ⟨x,y⟩ ∈ restrict(f,f-``a)  x ∈ f-``a x∈domain(f)
    using domain_restrict domainI[of _ _ restrict(f,f-``a)] by auto
  moreover
  note ‹function(restrict(f,f-``a))›
  ultimately
  have y = restrict(f,f-``a)`x
    using function_apply_equality by blast
  also from ‹x ∈ f-``a›
  have restrict(f,f-``a)`x = f`x
    by simp
  finally
  have y=f`x .
  moreover from assms ‹x∈domain(f)›
  have ⟨x,f`x⟩ ∈ f
    using function_apply_Pair by auto
  moreover
  note assms ‹x ∈ f-``a›
  ultimately
  show y∈a
    using function_image_vimage[of f a] by auto
qed

lemma induced_surj_type:
  assumes
    function(f)
  shows
    induced_surj(f,a,e): domain(f) → {e} ∪ a
    and
    x ∈ f-``a ⟹ induced_surj(f,a,e)`x = f`x
proof -
  let ?f1=f-``(range(f)-a) × {e} and ?f2=restrict(f, f-``a)
  have domain(?f2) = domain(f) ∩ f-``a
    using domain_restrict by simp
  moreover from assms
  have 1: domain(?f1) = f-``(range(f))-f-``a
    using domain_of_prod function_vimage_Diff by simp
  ultimately
  have domain(?f1) ∩ domain(?f2) = 0
    by auto
  moreover
  have function(?f1) relation(?f1) range(?f1) ⊆ {e}
    unfolding function_def relation_def range_def by auto
  moreover from this and assms
```

have *?f1*: *domain*(*?f1*) → *range*(*?f1*)
using *function_imp_Pi* **by** *simp*
moreover from *assms*
have *?f2*: *domain*(*?f2*) → *range*(*?f2*)
using *function_imp_Pi*[*of restrict*(*f*, *f* -‘‘ *a*)] *function_restrictI* **by** *simp*
moreover from *assms*
have *range*(*?f2*) ⊆ *a*
using *range_restrict_vimage* **by** *simp*
ultimately
have *induced_surj*(*f*,*a*,*e*): *domain*(*?f1*) ∪ *domain*(*?f2*) → {*e*} ∪ *a*
unfolding *induced_surj_def* **using** *fun_is_function fun_disjoint_Un fun_weaken_type*
by *simp*
moreover
have *domain*(*?f1*) ∪ *domain*(*?f2*) = *domain*(*f*)
using *domain_restrict domain_of_prod* **by** *auto*
ultimately
show *induced_surj*(*f*,*a*,*e*): *domain*(*f*) → {*e*} ∪ *a*
by *simp*
assume *x* ∈ *f*-‘‘*a*
then
have *?f2*‘*x* = *f*‘*x*
using *restrict* **by** *simp*
moreover from ‹*x* ∈ *f*-‘‘*a*› **and** *1*
have *x* ∉ *domain*(*?f1*)
by *simp*
ultimately
show *induced_surj*(*f*,*a*,*e*)‘*x* = *f*‘*x*
unfolding *induced_surj_def* **using** *fun_disjoint_apply2*[*of x ?f1 ?f2*] **by** *simp*
qed

lemma *induced_surj_is_surj* :
assumes
e∈*a function*(*f*) *domain*(*f*) = *α* ⋀*y*. *y* ∈ *a* ⟹ ∃ *x*∈*α*. *f* ‘ *x* = *y*
shows
induced_surj(*f*,*a*,*e*) ∈ *surj*(*α*,*a*)
unfolding *surj_def*
proof (*intro CollectI ballI*)
from *assms*
show *induced_surj*(*f*,*a*,*e*): *α* → *a*
using *induced_surj_type*[*of f a e*] *cons_eq cons_absorb* **by** *simp*
fix *y*
assume *y* ∈ *a*
with *assms*
have ∃ *x*∈*α*. *f* ‘ *x* = *y*
by *simp*
then
obtain *x* **where** *x*∈*α* *f* ‘ *x* = *y* **by** *auto*
with ‹*y*∈*a*› *assms*
have *x*∈*f*-‘‘*a*

using *vimage_iff function_apply_Pair*[*of f x*] **by** *auto*
with ‹*f* ‘ *x* = *y*› *assms*
have *induced_surj*(*f*, *a*, *e*) ‘ *x* = *y*
using *induced_surj_type* **by** *simp*
with ‹*x*∈α› **show**
∃ *x*∈α. *induced_surj*(*f*, *a*, *e*) ‘ *x* = *y* **by** *auto*
qed

context *G_generic*
begin

definition
upair_name :: *i* ⇒ *i* ⇒ *i* **where**
upair_name(τ,ϱ) ≡ *Upair*(⟨τ,*one*⟩,⟨ϱ,*one*⟩)

lemma *Upair_simp* : *Upair*(*a*,*b*) = {*a*,*b*}
by *auto*

relativize *upair_name is_upair_name*

lemma *upair_name_abs* :
assumes *x*∈*M* *y*∈*M* *z*∈*M*
shows *is_upair_name*(##*M*,*x*,*y*,*z*) ⟷ *z* = *upair_name*(*x*,*y*)
unfolding *is_upair_name_def upair_name_def*
using *assms zero_in_M one_in_M empty_abs pair_abs pair_in_M_iff upair_in_M_iff upair_abs*
Upair_simp
by *simp*

definition
opair_name :: *i* ⇒ *i* ⇒ *i* **where**
opair_name(τ,ϱ) ≡ *upair_name*(*upair_name*(τ,τ),*upair_name*(τ,ϱ))

relativize *opair_name is_opair_name*

lemma *upair_name_closed* :
⟦ *x*∈*M*; *y*∈*M* ⟧ ⟹ *upair_name*(*x*,*y*)∈*M*
unfolding *upair_name_def* **using** *upair_in_M_iff pair_in_M_iff one_in_M*
upair_in_M_iff Upair_simp
by *simp*

definition
upair_name_fm :: [*i*,*i*,*i*] ⇒ *i* **where**
upair_name_fm(*x*,*y*,*o*,*z*) ≡ *Exists*(*Exists*(*And*(*pair_fm*(*x*#+2,*o*#+2,1),
And(*pair_fm*(*y*#+2,*o*#+2,0),*upair_fm*(1,0,*z*#+2)))))

lemma *upair_name_fm_type*[*TC*] :
⟦ *s*∈*nat*;*x*∈*nat*;*y*∈*nat*;*o*∈*nat*⟧ ⟹ *upair_name_fm*(*s*,*x*,*y*,*o*)∈*formula*
unfolding *upair_name_fm_def* **by** *simp*

lemma *sats_upair_name_fm* :
  assumes *x∈nat y∈nat z∈nat o∈nat env∈list(M)nth(o,env)=one*
  shows
   sats(M,upair_name_fm(x,y,o,z),env) ⟷ is_upair_name(##M,nth(x,env),nth(y,env),nth(z,env))
  unfolding *upair_name_fm_def is_upair_name_def*
 using *assms zero_in_M empty_abs pair_abs one_in_M pair_in_M_iff Upair_simp upair_in_M_iff*
  by *auto*

lemma *opair_name_abs* :
  assumes *x∈M y∈M z∈M*
  shows *is_opair_name(##M,x,y,z) ⟷ z = opair_name(x,y)*
  unfolding *is_opair_name_def opair_name_def* **using** *assms upair_name_abs upair_name_closed* **by** *simp*

lemma *opair_name_closed* :
  ⟦ *x∈M*; *y∈M* ⟧ ⟹ *opair_name(x,y)∈M*
  unfolding *opair_name_def* **using** *upair_name_closed* **by** *simp*

definition
  opair_name_fm :: [*i*,*i*,*i*,*i*] ⇒ *i* **where**
 opair_name_fm(x,y,o,z) ≡ Exists(Exists(And(upair_name_fm(x#+2,x#+2,o#+2,1),
          And(upair_name_fm(x#+2,y#+2,o#+2,0),upair_name_fm(1,0,o#+2,z#+2)))))

lemma *opair_name_fm_type*[*TC*] :
   ⟦ *s∈nat;x∈nat;y∈nat;o∈nat*⟧ ⟹ *opair_name_fm(s,x,y,o)∈formula*
  unfolding *opair_name_fm_def* **by** *simp*

lemma *sats_opair_name_fm* :
  assumes *x∈nat y∈nat z∈nat o∈nat env∈list(M)nth(o,env)=one*
  shows
   sats(M,opair_name_fm(x,y,o,z),env) ⟷ is_opair_name(##M,nth(x,env),nth(y,env),nth(z,env))
  unfolding *opair_name_fm_def is_opair_name_def*
  using *assms sats_upair_name_fm*
  by *auto*

lemma *val_upair_name* : *val(P,G,upair_name(τ,ϱ)) = {val(P,G,τ),val(P,G,ϱ)}*
  unfolding *upair_name_def* **using** *val_Upair Upair_simp generic one_in_G one_in_P* **by** *simp*

lemma *val_opair_name* : *val(P,G,opair_name(τ,ϱ)) = ⟨val(P,G,τ),val(P,G,ϱ)⟩*
  unfolding *opair_name_def Pair_def* **using** *val_upair_name* **by** *simp*

lemma *val_RepFun_one*: *val(P,G,{⟨f(x),one⟩ . x∈a}) = {val(P,G,f(x)) . x∈a}*
proof -
  let *?A = {f(x) . x ∈ a}*
  let *?Q = λ⟨x,p⟩ . p = one*
  have *one ∈ P∩G* **using** *generic one_in_G one_in_P* **by** *simp*

have $\{\langle f(x),one\rangle\ .\ x \in a\} = \{t \in ?A \times P\ .\ ?Q(t)\}$
using *one_in_P* **by** *force*
then
have $val(P,G,\{\langle f(x),one\rangle\ .\ x \in a\}) = val(P,G,\{t \in ?A \times P\ .\ ?Q(t)\})$
by *simp*
also
have $... = \{val(P,G,t)\ ..\ t \in ?A\ ,\ \exists p{\in}P{\cap}G\ .\ ?Q(\langle t,p\rangle)\}$
using *val_of_name_alt* **by** *simp*
also
have $... = \{val(P,G,t)\ .\ t \in ?A\ \}$
using ‹$one{\in}P{\cap}G$› **by** *force*
also
have $... = \{val(P,G,f(x))\ .\ x \in a\}$
by *auto*
finally show *?thesis* **by** *simp*
qed

43.1 $M[G]$ is a transitive model of ZF

interpretation *mgzf*: *M_ZF_trans* $M[G]$
using *Transset_MG generic pairing_in_MG Union_MG*
extensionality_in_MG power_in_MG foundation_in_MG
strong_replacement_in_MG separation_in_MG infinity_in_MG
by *unfold_locales simp_all*

definition
$is\_opname\_check :: [i,i,i] \Rightarrow o$ **where**
$is\_opname\_check(s,x,y) \equiv \exists chx{\in}M.\ \exists sx{\in}M.\ is\_check(x,chx) \wedge fun\_apply(\#\#M,s,x,sx) \wedge is\_opair\_name(\#\#M,chx,sx,y)$

definition
$opname\_check\_fm :: [i,i,i,i] \Rightarrow i$ **where**
$opname\_check\_fm(s,x,y,o) \equiv Exists(Exists(And(check\_fm(2\#{+}x,2\#{+}o,1),$
$And(fun\_apply\_fm(2\#{+}s,2\#{+}x,0),opair\_name\_fm(1,0,2\#{+}o,2\#{+}y)))))$

lemma *opname_check_fm_type*$[TC]$:
$[\![\ s{\in}nat;x{\in}nat;y{\in}nat;o{\in}nat]\!] \Longrightarrow opname\_check\_fm(s,x,y,o){\in}formula$
unfolding *opname_check_fm_def* **by** *simp*

lemma *sats_opname_check_fm*:
assumes $x{\in}nat\ y{\in}nat\ z{\in}nat\ o{\in}nat\ env{\in}list(M)\ nth(o,env){=}one$
$y{<}length(env)$
shows
$sats(M,opname\_check\_fm(x,y,z,o),env) \longleftrightarrow is\_opname\_check(nth(x,env),nth(y,env),nth(z,env))$
unfolding *opname_check_fm_def is_opname_check_def*
using *assms sats_check_fm sats_opair_name_fm one_in_M* **by** *simp*

lemma *opname_check_abs* :
assumes *s∈M x∈M y∈M*
shows *is_opname_check(s,x,y) ⟷ y = opair_name(check(x),s‘x)*
unfolding *is_opname_check_def*
using *assms check_abs check_in_M opair_name_abs apply_abs apply_closed*
by *simp*

lemma *repl_opname_check* :
assumes
A∈M f∈M
shows
{opair_name(check(x),f‘x). x∈A}∈M
proof -
have *arity(opname_check_fm(3,0,1,2))= 4*
unfolding *fm_definitions opname_check_fm_def opair_name_fm_def upair_name_fm_def*
by (*simp add:ord_simp_union*)
moreover
have *x∈A ⟹ opair_name(check(x), f ‘ x)∈M* **for** *x*
using *assms opair_name_closed apply_closed transitivity check_in_M*
by *simp*
ultimately
show *?thesis* **using** *assms opname_check_abs*[*of f*] *sats_opname_check_fm*
one_in_M transitivity
Replace_relativized_in_M[*of opname_check_fm*(*3,0,1,2*) [*one,f*] *_ is_opname_check*(*f*)
λx. opair_name(*check*(*x*),*f‘x*)]
by *auto*
qed

theorem *choice_in_MG*:
assumes *choice_ax*(*##M*)
shows *choice_ax*(*##M*[*G*])
proof -
{
fix *a*
assume *a∈M*[*G*]
then
obtain *τ* **where** *τ∈M val*(*P*,*G*,*τ*) = *a*
using *GenExt_def* **by** *auto*
with ‹*τ∈M*›
have *domain*(*τ*)*∈M*
using *domain_closed* **by** *simp*
then
obtain *s α* **where** *s∈surj*(*α*,*domain*(*τ*)) *Ord*(*α*) *s∈M α∈M*
using *assms choice_ax_abs* **by** *auto*
then
have *α∈M*[*G*]
using *M_subset_MG generic one_in_G subsetD* **by** *blast*

let *?A=domain*(τ)$\times P$
let *?g* = {*opair_name*(*check*(β),*s*‘β). $\beta\in\alpha$}
have *?g* $\in M$ **using** ‹$s\in M$› ‹$\alpha\in M$› *repl_opname_check* **by** *simp*
let *?f_dot*={⟨*opair_name*(*check*(β),*s*‘β),*one*⟩. $\beta\in\alpha$}
have *?f_dot* = *?g* $\times$ {*one*} **by** *blast*
from *one_in_M* **have** {*one*} $\in M$ **using** *singleton_closed* **by** *simp*
define *f* **where**
  f $\equiv$ *val*(*P*,*G*,*?f_dot*)
from ‹{*one*}$\in M$› ‹*?g*$\in M$› ‹*?f_dot* = *?g*$\times${*one*}›
have *?f_dot*$\in M$
  using *cartprod_closed* **by** *simp*
then
have *f* $\in M[G]$
  unfolding *f_def* **by** (*blast intro*:*GenExtI*)
have *f* = {*val*(*P*,*G*,*opair_name*(*check*(β),*s*‘β)) . $\beta\in\alpha$}
  unfolding *f_def* **using** *val_RepFun_one* **by** *simp*
also
have ... = {⟨β,*val*(*P*,*G*,*s*‘β)⟩ . $\beta\in\alpha$}
  using *val_opair_name valcheck generic one_in_G one_in_P* **by** *simp*
finally
have *f* = {⟨β,*val*(*P*,*G*,*s*‘β)⟩ . $\beta\in\alpha$} **.**
then
have *1*: *domain*(*f*) = α *function*(*f*)
  unfolding *function_def* **by** *auto*
have *2*: $y \in a \Longrightarrow \exists\, x\in\alpha$. *f* ‘ *x* = *y* **for** *y*
proof -
  fix *y*
  assume
    $y \in a$
  with ‹*val*(*P*,*G*,τ) = *a*›
  obtain σ **where** $\sigma\in$*domain*(τ) *val*(*P*,*G*,σ) = *y*
    using *elem_of_val*[*of y _* τ] **by** *blast*
  with ‹*s*$\in$*surj*(α,*domain*(τ))›
  obtain β **where** $\beta\in\alpha$ *s*‘β = σ
    unfolding *surj_def* **by** *auto*
  with ‹*val*(*P*,*G*,σ) = *y*›
  have *val*(*P*,*G*,*s*‘β) = *y*
    by *simp*
  with ‹*f* = {⟨β,*val*(*P*,*G*,*s*‘β)⟩ . $\beta\in\alpha$}› ‹$\beta\in\alpha$›
  have ⟨β,*y*⟩$\in$*f*
    by *auto*
  with ‹*function*(*f*)›
  have *f*‘β = *y*
    using *function_apply_equality* **by** *simp*
  with ‹$\beta\in\alpha$› **show**
    $\exists\,\beta\in\alpha$. *f* ‘ β = *y*
    by *auto*
qed
then

```
    have ∃α∈(M[G]). ∃f'∈(M[G]). Ord(α) ∧ f' ∈ surj(α,a)
    proof (cases a=0)
      case True
      then
      show ?thesis
        unfolding surj_def using zero_in_MG by auto
    next
      case False
      with ‹a∈M[G]›
      obtain e where e∈a e∈M[G]
        using transitivity_MG by blast
      with 1 and 2
      have induced_surj(f,a,e) ∈ surj(α,a)
        using induced_surj_is_surj by simp
      moreover from ‹f∈M[G]› ‹a∈M[G]› ‹e∈M[G]›
      have induced_surj(f,a,e) ∈ M[G]
        unfolding induced_surj_def
        by (simp flip: setclass_iff)
      moreover note
        ‹α∈M[G]› ‹Ord(α)›
      ultimately show ?thesis by auto
    qed
  }
  then
  show ?thesis using mgzf.choice_ax_abs by simp
qed

end

end
```

44 Ordinals in generic extensions

```
theory Ordinals_In_MG
  imports
    Forcing_Theorems Relative_Univ
begin

context G_generic
begin

lemma rank_val: rank(val(P,G,x)) ≤ rank(x) (is ?Q(x))
proof (induct rule:ed_induction[of ?Q])
  case (1 x)
  have val(P,G,x) = {val(P,G,u). u∈{t∈domain(x). ∃p∈P . ⟨t,p⟩∈x ∧ p ∈ G }}
    using def_val unfolding Sep_and_Replace by blast
  then
   have rank(val(P,G,x)) = (⋃u∈{t∈domain(x). ∃p∈P . ⟨t,p⟩∈x ∧ p ∈ G }.
succ(rank(val(P,G,u))))
```

using *rank*[*of val*(*P*,*G*,*x*)] **by** *simp*
moreover
have *succ*(*rank*(*val*(*P*,*G*, *y*))) ≤ *rank*(*x*) **if** *ed*(*y*, *x*) **for** *y*
using *1*[*OF that*] *rank_ed*[*OF that*] **by** (*auto intro*:*lt_trans1*)
moreover from *this*
have (⋃ *u*∈{*t*∈*domain*(*x*). ∃ *p*∈*P* . ⟨*t*,*p*⟩∈*x* ∧ *p* ∈ *G* }. *succ*(*rank*(*val*(*P*,*G*,*u*))))
≤ *rank*(*x*)
by (*rule_tac UN_least_le*) (*auto*)
ultimately
show *?case* **by** *simp*
qed

lemma *Ord_MG_iff*:
assumes *Ord*(α)
shows α ∈ *M* ⟷ α ∈ *M*[*G*]
proof
show α ∈ *M* ⟹ α ∈ *M*[*G*]
using *generic*[*THEN one_in_G*, *THEN M_subset_MG*] **..**
next
assume α ∈ *M*[*G*]
then
obtain *x* **where** *x*∈*M val*(*P*,*G*,*x*) = α
using *GenExtD* **by** *auto*
then
have *rank*(α) ≤ *rank*(*x*)
using *rank_val* **by** *blast*
with *assms*
have α ≤ *rank*(*x*)
using *rank_of_Ord* **by** *simp*
then
have α ∈ *succ*(*rank*(*x*)) **using** *ltD* **by** *simp*
with ‹*x*∈*M*›
show α ∈ *M*
using *cons_closed transitivity*[*of* α *succ*(*rank*(*x*))]
rank_closed **unfolding** *succ_def* **by** *simp*
qed

end

end

45 The main theorem

theory *Forcing_Main*
imports
Succession_Poset
ZF_Miscellanea
Internal_ZFC_Axioms
Choice_Axiom

Ordinals_In_MG

begin

45.1 The generic extension is countable

lemma (**in** *forcing_data*) *surj_nat_MG* :
  $\exists f.\ f \in surj(\omega, M[G])$
proof -
  let *?f*=$\lambda n{\in}\omega.\ val(P,G,enum`n)$
  have $x \in \omega \Longrightarrow val(P,G,\ enum\ `\ x){\in}\ M[G]$ **for** x
    using *GenExtD*[*THEN iffD2*, *of _ G*] *bij_is_fun*[*OF M_countable*] **by** *force*
  then
  have *?f*: $\omega \to M[G]$
    using *lam_type*[*of* ω $\lambda n.\ val(P,G,enum`n)$ $\lambda\_.M[G]$] **by** *simp*
  moreover
  have $\exists\, n{\in}\omega.\ ?f`n = x$ **if** $x{\in}M[G]$ **for** x
    using *that GenExtD*[*of _ G*] *bij_is_surj*[*OF M_countable*]
    unfolding *surj_def* **by** *auto*
  ultimately
  show *?thesis*
    unfolding *surj_def* **by** *blast*
qed

lemma (**in** *G_generic*) *MG_eqpoll_nat*: $M[G] \approx \omega$
proof -
  interpret *MG*: *M_ZF_trans* $M[G]$
    using *Transset_MG generic pairing_in_MG*
      Union_MG extensionality_in_MG power_in_MG
      foundation_in_MG strong_replacement_in_MG[*simplified*]
      separation_in_MG[*simplified*] *infinity_in_MG*
    by *unfold_locales simp_all*
  obtain f **where** $f \in surj(\omega, M[G])$
    using *surj_nat_MG* **by** *blast*
  then
  have $M[G] \lesssim \omega$
    using *well_ord_surj_imp_lepoll well_ord_Memrel*[*of* ω]
    by *simp*
  moreover
  have $\omega \lesssim M[G]$
    using *MG.nat_into_M subset_imp_lepoll* **by** (*auto del*:*lepollI*)
  ultimately
  show *?thesis* **using** *eqpollI*
    by *simp*
qed

45.2 The main result

theorem *extensions_of_ctms*:
  assumes

```
    M ≈ ω  Transset(M)  M ⊨ ZF
  shows
    ∃N.
      M ⊆ N ∧ N ≈ ω ∧ Transset(N) ∧ N ⊨ ZF ∧ M≠N ∧
      (∀α. Ord(α) ⟶ (α ∈ M ⟷ α ∈ N)) ∧
      ((M, []⊨ ·AC·) ⟶ N ⊨ ZFC)
proof -
  from ‹M ⊨ ZF›
  interpret M_ZF M
    using M_ZF_iff_M_satT
    by simp
  from ‹Transset(M)›
  interpret M_ZF_trans M
    using M_ZF_iff_M_satT
    by unfold_locales
  from ‹M ≈ ω›
  obtain enum where enum ∈ bij(ω,M)
    using eqpoll_sym unfolding eqpoll_def by blast
  then
  interpret M_ctm M enum by unfold_locales
  interpret forcing_data 2^<ω seqle 0 M enum
    using nat_into_M seqspace_closed seqle_in_M
    by unfold_locales simp
  obtain G where M_generic(G)  M ≠ M^(2^<ω)[G]  (is M≠?N)
    using cohen_extension_is_proper
    by blast
  then
  interpret G_generic 2^<ω seqle 0 _ enum G by unfold_locales
  interpret MG: M_ZF ?N
    using generic pairing_in_MG
      Union_MG  extensionality_in_MG power_in_MG
      foundation_in_MG  strong_replacement_in_MG[simplified]
      separation_in_MG[simplified] infinity_in_MG
    by unfold_locales simp_all
  have ?N ⊨ ZF
    using M_ZF_iff_M_satT[of ?N] MG.M_ZF_axioms by simp
  moreover
  have M, []⊨ ·AC· ⟹ ?N ⊨ ZFC
  proof -
    assume M, [] ⊨ ·AC·
    then
    have choice_ax(##M)
      unfolding ZF_choice_fm_def using ZF_choice_auto by simp
    then
    have choice_ax(##?N) using choice_in_MG by simp
    with ‹?N ⊨ ZF›
    show ?N ⊨ ZFC
      using ZF_choice_auto sats_ZFC_iff_sats_ZF_AC
      unfolding ZF_choice_fm_def by simp
```

qed
moreover
note ‹$M \neq ?N$›
moreover
have *Transset*(*?N*) **using** *Transset_MG* **.**
moreover
have $M \subseteq ?N$ **using** *M_subset_MG*[*OF one_in_G*] *generic* **by** *simp*
ultimately
show *?thesis*
using *Ord_MG_iff MG_eqpoll_nat*
by (*rule_tac x*=*?N* **in** *exI*, *simp*)
qed

end

46 Cardinal Arithmetic under Choice

theory *Cardinal_Library_Relative*
imports
ZF_Library_Relative
Delta_System_Lemma.ZF_Library
Replacement_Lepoll
begin

locale *M_library* = *M_ZF_library* + *M_cardinal_AC*
begin

declare *eqpoll_rel_refl* [*simp*]

46.1 Miscellaneous

lemma *cardinal_rel_RepFun_le*:
assumes $S \in A \to B$ $M(S)$ $M(A)$ $M(B)$
shows $|\{S`a \,.\, a \in A\}|^M \leq |A|^M$
proof -
note *assms*
moreover from *this*
have $\{S \,`\, a \,.\, a \in A\} = S``A$
using *image_eq_UN RepFun_def UN_iff* **by** *force*
moreover from *calculation*
have $M(\lambda x \in A.\ S \,`\, x)$ $M(\{S \,`\, a \,.\, a \in A\})$
using *lam_closed*[*of* $\lambda\ x.\ S`x$] *apply_type*[*OF* ‹$S \in \_$›]
transM[*OF* _ ‹$M(B)$›] *image_closed*
by *auto*
moreover from *assms this*
have $(\lambda x \in A.\ S`x) \in$ *surj_rel*($M$,$A$, $\{S`a \,.\, a \in A\}$)
using *mem_surj_abs lam_funtype*[*of* A $\lambda x\ .\ S`x$]
unfolding *surj_def* **by** *auto*
ultimately

show *?thesis*
using *surj_rel_char surj_rel_implies_cardinal_rel_le* **by** *simp*
qed

lemma *subset_imp_le_cardinal_rel*: $A \subseteq B \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow |A|^M \leq |B|^M$
using *subset_imp_lepoll_rel*[*THEN lepoll_rel_imp_cardinal_rel_le*] **.**

lemma *lt_cardinal_rel_imp_not_subset*: $|A|^M < |B|^M \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow \neg\ B \subseteq A$
using *subset_imp_le_cardinal_rel le_imp_not_lt* **by** *blast*

lemma *cardinal_rel_lt_csucc_rel_iff*:
$Card\_rel(M,K) \Longrightarrow M(K) \Longrightarrow M(K') \Longrightarrow |K'|^M < (K^+)^M \longleftrightarrow |K'|^M \leq K$
by (*simp add*: *Card_rel_lt_csucc_rel_iff*)

lemmas *inj_rel_is_fun* = *inj_is_fun*[*OF mem_inj_rel*]

lemma *inj_rel_bij_rel_range*: $f \in inj^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow f \in bij^M(A,range(f))$
using *bij_rel_char inj_rel_char inj_bij_range* **by** *force*

lemma *bij_rel_is_inj_rel*: $f \in bij^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow f \in inj^M(A,B)$
unfolding *bij_rel_def* **by** *simp*

lemma *inj_rel_weaken_type*: $[|\ f \in inj^M(A,B);\ \ B\subseteq D;\ M(A);\ M(B);\ M(D)\ |] ==> f \in inj^M(A,D)$
using *inj_rel_char inj_rel_is_fun inj_weaken_type* **by** *auto*

lemma *bij_rel_converse_bij_rel* [*TC*]: $f \in bij^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) ==> converse(f): bij^M(B,A)$
using *bij_rel_char* **by** *force*

lemma *bij_rel_is_fun_rel*: $f \in bij^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow\ f \in A\rightarrow^M B$
using *bij_rel_char mem_function_space_rel_abs bij_is_fun* **by** *simp*

lemmas *bij_rel_is_fun* = *bij_rel_is_fun_rel*[*THEN mem_function_space_rel*]

lemma *comp_bij_rel*:
$g \in bij^M(A,B) \Longrightarrow f \in bij^M(B,C) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (f\ O\ g) \in bij^M(A,C)$
using *bij_rel_char comp_bij* **by** *force*

lemma *inj_rel_converse_fun*: $f \in inj^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow converse(f) \in range(f)\rightarrow^M A$
proof -
assume $f \in inj^M(A,B)\ M(A)\ M(B)$
then
have $M(f)\ M(converse(f))\ M(range(f))\ f{\in}inj(A,B)$

```
    using inj_rel_char converse_closed range_closed
    by auto
  then
  show ?thesis
    using inj_converse_inj function_space_rel_char inj_is_fun ‹M(A)› by auto
qed

end

locale M_cardinal_UN_nat = M_cardinal_UN _ ω X for X
begin
lemma cardinal_rel_UN_le_nat:
  assumes ⋀i. i∈ω ⟹ |X(i)|^M ≤ ω
  shows |⋃i∈ω. X(i)|^M ≤ ω
proof -
  from assms
  show ?thesis
  by (simp add: cardinal_rel_UN_le InfCard_rel_nat)
qed

end

locale M_cardinal_UN_inj = M_library +
  j:M_cardinal_UN _ J +
  y:M_cardinal_UN _ K λk. if k∈range(f) then X(converse(f)‘k) else 0 for J K
f +
assumes
  f_inj: f ∈ inj_rel(M,J,K)
begin

lemma inj_rel_imp_cardinal_rel_UN_le:
  notes [dest] = InfCard_is_Card Card_is_Ord
  fixes Y
  defines Y(k) ≡ if k∈range(f) then X(converse(f)‘k) else 0
  assumes InfCard^M(K) ⋀i. i∈J ⟹ |X(i)|^M ≤ K
  shows |⋃i∈J. X(i)|^M ≤ K
proof -
  have M(K) M(J) ⋀w x. w ∈ X(x) ⟹ M(x)
    using y.Pi_assumptions j.Pi_assumptions j.X_witness_in_M by simp_all
  then
  have M(f)
    using inj_rel_char f_inj by simp
  note inM = ‹M(f)› ‹M(K)› ‹M(J)› ‹⋀w x. w ∈ X(x) ⟹ M(x)›
  have i∈J ⟹ f‘i ∈ K for i
    using inj_rel_is_fun[OF f_inj] apply_type
      function_space_rel_char by (auto simp add:inM)
  have (⋃i∈J. X(i)) ⊆ (⋃i∈K. Y(i))
  proof (standard, elim UN_E)
    fix x i
```

```
    assume i∈J x∈X(i)
    with ‹i∈J ⟹ f‘i ∈ K›
    have x ∈ Y(f‘i) f‘i ∈ K
      unfolding Y_def
      using inj_is_fun right_inverse f_inj
      by (auto simp add:inM Y_def intro: apply_rangeI)
    then
    show x ∈ (⋃i∈K. Y(i)) by auto
  qed
  then
  have |⋃i∈J. X(i)|^M ≤ |⋃i∈K. Y(i)|^M
    using subset_imp_le_cardinal_rel j.UN_closed y.UN_closed
    unfolding Y_def by (simp add:inM)
  moreover
  note assms ‹⋀i. i∈J ⟹ f‘i ∈ K› inM
  moreover from this
  have k∈range(f) ⟹ converse(f)‘k ∈ J for k
    using inj_rel_converse_fun[OF f_inj]
      range_fun_subset_codomain function_space_rel_char by simp
  ultimately
  show |⋃i∈J. X(i)|^M ≤ K
     using InfCard_rel_is_Card_rel[THEN Card_rel_is_Ord,THEN Ord_0_le,
of K]
    by (rule_tac le_trans[OF _ y.cardinal_rel_UN_le])
      (auto intro:Ord_0_le simp:Y_def)+
qed

end

locale M_cardinal_UN_lepoll = M_library + M_replacement_lepoll _ λ_. X +
  j:M_cardinal_UN _ J for J
begin

— FIXME: this "LEQpoll" should be "LEPOLL"; same correction in Delta System
lemma leqpoll_rel_imp_cardinal_rel_UN_le:
  notes [dest] = InfCard_is_Card Card_is_Ord
  assumes InfCard^M(K) J ≲^M K ⋀i. i∈J ⟹ |X(i)|^M ≤ K
    M(K)
  shows |⋃i∈J. X(i)|^M ≤ K
proof -
  from ‹J ≲^M K›
  obtain f where f ∈ inj_rel(M,J,K) M(f) by blast
  moreover
  let ?Y=λk. if k∈range(f) then X(converse(f)‘k) else 0
  note ‹M(K)›
  moreover from calculation
  have k ∈ range(f) ⟹ converse(f)‘k ∈ J for k
    using mem_inj_rel[THEN inj_converse_fun, THEN apply_type]
      j.Pi_assumptions by blast
```

```
  moreover from ‹M(f)›
  have w ∈ ?Y(x) ⟹ M(x) for w x
    by (cases x∈range(f)) (auto dest:transM)
  moreover from calculation
  interpret M_Pi_assumptions_choice _ K ?Y
    using j.Pi_assumptions lepoll_assumptions
  proof (unfold_locales, auto dest:transM)
    show strong_replacement(M, λy z. False)
      unfolding strong_replacement_def by auto
  qed
  from calculation
  interpret M_cardinal_UN_inj _ _ _ _ f
    using lepoll_assumptions
    by unfold_locales auto
  from assms
  show ?thesis using inj_rel_imp_cardinal_rel_UN_le by simp
qed

end

context M_library
begin

lemma cardinal_rel_lt_csucc_rel_iff':
  includes Ord_dests
  assumes Card_rel(M,κ)
    and types:M(κ) M(X)
  shows κ < |X|^M ⟷ (κ^+)^M ≤ |X|^M
  using assms cardinal_rel_lt_csucc_rel_iff[of κ X] Card_rel_csucc_rel[of κ]
    not_le_iff_lt[of (κ^+)^M |X|^M] not_le_iff_lt[of |X|^M κ]
  by blast

lemma lepoll_rel_imp_subset_bij_rel:
  assumes M(X) M(Y)
  shows X ≲^M Y ⟷ (∃ Z[M]. Z ⊆ Y ∧ Z ≈^M X)
proof
  assume X ≲^M Y
  then
  obtain j where  j ∈ inj_rel(M,X,Y)
    by blast
  with assms
  have range(j) ⊆ Y j ∈ bij_rel(M,X,range(j)) M(range(j)) M(j)
    using inj_rel_bij_rel_range inj_rel_char
      inj_rel_is_fun[THEN range_fun_subset_codomain,of j X Y]
    by auto
  with assms
  have range(j) ⊆ Y X ≈^M range(j)
    unfolding eqpoll_rel_def by auto
  with assms ‹M(j)›
```

show $\exists\, Z[M].\ Z \subseteq Y \wedge Z \approx^M X$
using *eqpoll_rel_sym*[*OF* ‹$X \approx^M$ *range*(j)›]
by *auto*
next
assume $\exists\, Z[M].\ Z \subseteq Y \wedge Z \approx^M X$
then
obtain $Z\ f$ **where** $f \in$ *bij_rel*(M,Z,X) $Z \subseteq Y$ $M(Z)$ $M(f)$
unfolding *eqpoll_rel_def* **by** *blast*
with *assms*
have *converse*(f) $\in$ *inj_rel*(M,X,Y) M(*converse*(f))
using *inj_rel_weaken_type*[*OF bij_rel_converse_bij_rel*[*THEN bij_rel_is_inj_rel*],*of* f Z X Y]
by *auto*
then
show $X \lesssim^M Y$
unfolding *lepoll_rel_def* **by** *auto*
qed

The following result proves to be very useful when combining *cardinal_rel* and *eqpoll_rel* in a calculation.

lemma *cardinal_rel_Card_rel_eqpoll_rel_iff*:
Card_rel(M,κ) $\Longrightarrow$ $M(\kappa)$ $\Longrightarrow$ $M(X)$ $\Longrightarrow$ $|X|^M = \kappa \longleftrightarrow X \approx^M \kappa$
using *Card_rel_cardinal_rel_eq*[*of* κ] *cardinal_rel_eqpoll_rel_iff*[*of* X κ] **by** *auto*

lemma *lepoll_rel_imp_lepoll_rel_cardinal_rel*:
assumes$X \lesssim^M Y$ $M(X)$ $M(Y)$
shows $X \lesssim^M |Y|^M$
using *assms cardinal_rel_Card_rel_eqpoll_rel_iff*[*of* $|Y|^M$ Y]
Card_rel_cardinal_rel
lepoll_rel_eq_trans[*of* _ _ $|Y|^M$] **by** *simp*

lemma *lepoll_rel_Un*:
assumes *InfCard_rel*(M,κ) $A \lesssim^M \kappa$ $B \lesssim^M \kappa$ $M(A)$ $M(B)$ $M(\kappa)$
shows $A \cup B \lesssim^M \kappa$
proof -
from *assms*
have $A \cup B \lesssim^M$ *sum*(A,B)
using *Un_lepoll_rel_sum* **by** *simp*
moreover
note *assms*
moreover from *this*
have $|sum(A,B)|^M \le \kappa \oplus^M \kappa$
using *sum_lepoll_rel_mono*[*of* A κ B κ] *lepoll_rel_imp_cardinal_rel_le*
unfolding *cadd_rel_def* **by** *auto*
ultimately
show *?thesis*
using *InfCard_rel_cdouble_eq Card_rel_cardinal_rel_eq*
InfCard_rel_is_Card_rel Card_rel_le_imp_lepoll_rel[*of sum*(A,B) κ]

lepoll_rel_trans[*of* $A \cup B$]
by *auto*
qed

lemma *cardinal_rel_Un_le*:
assumes *InfCard_rel*(M,κ) $|A|^M \leq \kappa$ $|B|^M \leq \kappa$ $M(\kappa)$ $M(A)$ $M(B)$
shows $|A \cup B|^M \leq \kappa$
using *assms lepoll_rel_Un le_Card_rel_iff InfCard_rel_is_Card_rel* **by** *auto*

lemma *eqpoll_rel_imp_Finite*: $A \approx^M B \Longrightarrow$ *Finite*(A) $\Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow$ *Finite*(B)
proof -
assume $A \approx^M B$ *Finite*(A) $M(A)$ $M(B)$
then obtain f n g **where** $f \in bij(A,B)$ $n \in \omega$ $g \in bij(A,n)$
unfolding *Finite_def eqpoll_def eqpoll_rel_def*
using *bij_rel_char*
by *auto*
then
have g O *converse*(f) $\in bij(B,n)$
using *bij_converse_bij comp_bij* **by** *simp*
with ‹$n \in \_$›
show*Finite*(B)
unfolding *Finite_def eqpoll_def* **by** *auto*
qed

lemma *eqpoll_rel_imp_Finite_iff*: $A \approx^M B \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow$ *Finite*(A) $\longleftrightarrow$ *Finite*(B)
using *eqpoll_rel_imp_Finite eqpoll_rel_sym* **by** *force*

lemma *Finite_cardinal_rel_iff'*: $M(i) \Longrightarrow$ *Finite*($|i|^M$) $\longleftrightarrow$ *Finite*(i)
using *eqpoll_rel_imp_Finite_iff*[*OF cardinal_rel_eqpoll_rel*]
by *auto*

lemma *cardinal_rel_subset_of_Card_rel*:
assumes *Card_rel*(M,γ) $a \subseteq \gamma$ $M(a)$ $M(\gamma)$
shows $|a|^M < \gamma \vee |a|^M = \gamma$
proof -
from *assms*
have $|a|^M < |\gamma|^M \vee |a|^M = |\gamma|^M$
using *subset_imp_le_cardinal_rel*[*THEN le_iff*[*THEN iffD1*]] **by** *simp*
with *assms*
show *?thesis*
using *Card_rel_cardinal_rel_eq* **by** *auto*
qed

lemma *cardinal_rel_cases*:
includes *Ord_dests*
assumes $M(\gamma)$ $M(X)$
shows *Card_rel*(M,γ) $\Longrightarrow |X|^M < \gamma \longleftrightarrow \neg\, |X|^M \geq \gamma$

using *assms not_le_iff_lt Card_rel_is_Ord Ord_cardinal_rel*
by *auto*

end

46.2 Countable and uncountable sets

definition
countable :: $i \Rightarrow o$ **where**
$countable(X) \equiv X \lesssim \omega$

relativize functional *countable countable_rel* **external**
relationalize *countable_rel is_countable*

context *M_library*
begin

lemma *countableI*[*intro*]: $X \lesssim^M \omega \Longrightarrow countable\_rel(M,X)$
unfolding *countable_rel_def* **by** *simp*

lemma *countableD*[*dest*]: $countable\_rel(M,X) \Longrightarrow X \lesssim^M \omega$
unfolding *countable_rel_def* **by** *simp*

lemma *countable_rel_iff_cardinal_rel_le_nat*: $M(X) \Longrightarrow countable\_rel(M,X)$
$\longleftrightarrow |X|^M \leq \omega$
using *le_Card_rel_iff*[*of* ω X] *Card_rel_nat*
unfolding *countable_rel_def* **by** *simp*

lemma *lepoll_rel_countable_rel*: $X \lesssim^M Y \Longrightarrow countable\_rel(M,Y) \Longrightarrow M(X)$
$\Longrightarrow M(Y) \Longrightarrow countable\_rel(M,X)$
using *lepoll_rel_trans*[*of* X Y] **by** *blast*

— Next lemma can be proved without using AC
lemma *surj_rel_countable_rel*:
$countable\_rel(M,X) \Longrightarrow f \in surj\_rel(M,X,Y) \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow M(f)$
$\Longrightarrow countable\_rel(M,Y)$
using *surj_rel_implies_cardinal_rel_le*[*of f X Y*, *THEN le_trans*]
countable_rel_iff_cardinal_rel_le_nat **by** *simp*

lemma *Finite_imp_countable_rel*: $Finite\_rel(M,X) \Longrightarrow M(X) \Longrightarrow countable\_rel(M,X)$
unfolding *Finite_rel_def*
by (*auto intro*:*InfCard_rel_nat nats_le_InfCard_rel*[*of* _ ω,
THEN le_imp_lepoll_rel] *dest*!:*eq_lepoll_rel_trans*[*of* X _ ω])

end

lemma (**in** *M_cardinal_UN_lepoll*) *countable_rel_imp_countable_rel_UN*:
assumes $countable\_rel(M,J)$ $\bigwedge i.\ i \in J \Longrightarrow countable\_rel(M,X(i))$
shows $countable\_rel(M,\bigcup i \in J.\ X(i))$

using *assms leqpoll_rel_imp_cardinal_rel_UN_le*[*of* ω] *InfCard_rel_nat*
InfCard_rel_is_Card_rel j.UN_closed
countable_rel_iff_cardinal_rel_le_nat j.Pi_assumptions
Card_rel_le_imp_lepoll_rel[*of* J ω] *Card_rel_cardinal_rel_eq*[*of* ω]
by *auto*

locale *M_cardinal_library* = *M_library* + *M_replacement* +
assumes
lam_replacement_inj_rel:*lam_replacement*(M, $\lambda x.\ inj^M(fst(x),snd(x))$)
and
cardinal_lib_assms1:
$M(A) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow$
separation(M, $\lambda y.\ \exists\, x{\in}A.\ y = \langle x,\ \mu\ i.\ x \in$ *if_range_F_else_F*($\lambda x.$ *if* $M(x)$ *then* x *else* 0,b,f,i)$\rangle$)
separation(M,*Ord*)
and
cardinal_lib_assms2:
$M(A') \Longrightarrow M(G) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow$
separation(M, $\lambda y.\ \exists\, x{\in}A'.\ y = \langle x,\ \mu\ i.\ x \in$ *if_range_F_else_F*($\lambda a.$ *if* $M(a)$ *then* G`a *else* 0,b,f,i)$\rangle$)
and
cardinal_lib_assms3:
$M(A') \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow M(F) \Longrightarrow$
separation(M, $\lambda y.\ \exists\, x{\in}A'.\ y = \langle x,\ \mu\ i.\ x \in$ *if_range_F_else_F*($\lambda a.$ *if* $M(a)$ *then* F-``$\{a\}$ *else* 0,b,f,i)$\rangle$)
and
cdlt_assms:
$M(G) \Longrightarrow M(Q) \Longrightarrow$ *separation*(M, $\lambda p.\ \forall\, x{\in}G.\ x \in snd(p) \longleftrightarrow (\forall\, s{\in}fst(p).\ \langle s,\ x\rangle \in Q)$)
$M(x) \Longrightarrow M(Q) \Longrightarrow$ *separation*(M, $\lambda a\ .\ \ \forall\, s{\in}x.\ \langle s,\ a\rangle \in Q$)
and
cardinal_lib_assms5 :
$M(\gamma) \Longrightarrow$ *separation*(M, $\lambda Z\ .$ *cardinal_rel*(M,Z) $< \gamma$)
and
cardinal_lib_assms6:
$M(f) \Longrightarrow M(\beta) \Longrightarrow Ord(\beta) \Longrightarrow$
strong_replacement(M, $\lambda x\ y.\ x{\in}\beta \wedge y = \langle x,$ *transrec*(x, $\lambda a\ g.\ f$ ` (g `` a))$\rangle$)
separation(M, $\lambda\ x\ .\ \exists\, a.\ \exists\, b\ .\ x{=}\langle a,b\rangle \wedge a{\neq}b$)

begin

lemma *countable_rel_union_countable_rel*:
assumes $\bigwedge x.\ x \in C \Longrightarrow$ *countable_rel*(M,x) *countable_rel*(M,C) $M(C)$
shows *countable_rel*(M,$\bigcup C$)
proof -
have $x \in$ (*if* $M(i)$ *then* i *else* 0) $\Longrightarrow M(i)$ **for** x i
by (*cases* $M(i)$) *auto*
then
interpret *M_replacement_lepoll* M $\lambda\_\ x.$ *if* $M(x)$ *then* x *else* 0

```
    using  lam_replacement_if[OF lam_replacement_identity
     lam_replacement_constant[OF nonempty], where b=M] lam_replacement_inj_rel
    apply unfold_locales apply (auto simp add: separation_def)
  proof -
    fix b f
    assume M(b) M(f)
    show lam_replacement(M, λx. μ i. x ∈ if_range_F_else_F(λx. if M(x) then
x else 0, b, f, i))
    proof (cases b=0)
      case True
      with ‹M(f)›
      show ?thesis
        using cardinal_lib_assms1
        by (simp_all; rule_tac lam_Least_assumption_ifM_b0)+
    next
      case False
      with ‹M(f)› ‹M(b)›
      show ?thesis
      using cardinal_lib_assms1
      by (rule_tac lam_Least_assumption_ifM_bnot0)  auto
    qed
  qed
  note ‹M(C)›
  moreover
  have  w ∈ (if M(x) then x else 0) ⟹ M(x) for w x
    by (cases M(x)) auto
  ultimately
  interpret M_cardinal_UN_lepoll _ λc. if M(c) then c else 0 C
    using lepoll_assumptions
    by unfold_locales simp_all
  have (if M(i) then i else 0) = i if i∈C for i
    using transM[OF _ ‹M(C)›] that by simp
  then
  show ?thesis
    using assms countable_rel_imp_countable_rel_UN by simp
qed

end

abbreviation
  uncountable_rel :: [i⇒o,i]⇒o where
  uncountable_rel(M,X) ≡ ¬ countable_rel(M,X)

context M_cardinal_library
begin

lemma uncountable_rel_iff_nat_lt_cardinal_rel:
  M(X) ⟹ uncountable_rel(M,X) ⟷ ω < |X|^M
  using countable_rel_iff_cardinal_rel_le_nat not_le_iff_lt by simp
```

lemma *uncountable_rel_not_empty*: *uncountable_rel*(M,X) $\Longrightarrow$ $X \neq 0$
using *empty_lepoll_relI* **by** *auto*

lemma *uncountable_rel_imp_Infinite*: *uncountable_rel*(M,X) $\Longrightarrow$ $M(X) \Longrightarrow$ *Infinite*(X)
using *uncountable_rel_iff_nat_lt_cardinal_rel*[*of X*] *lepoll_rel_nat_imp_Infinite*[*of X*]
cardinal_rel_le_imp_lepoll_rel[*of* ω X] *leI*
by *simp*

lemma *uncountable_rel_not_subset_countable_rel*:
assumes *countable_rel*(M,X) *uncountable_rel*(M,Y) $M(X)$ $M(Y)$
shows $\neg$ ($Y \subseteq X$)
using *assms lepoll_rel_trans subset_imp_lepoll_rel*[*of Y X*]
by *blast*

46.3 Results on Aleph_rels

lemma *nat_lt_Aleph_rel1*: $\omega < \aleph_1{}^M$
by (*simp add*: *Aleph_rel_succ Aleph_rel_zero lt_csucc_rel*)

lemma *zero_lt_Aleph_rel1*: $0 < \aleph_1{}^M$
by (*rule lt_trans*[*of* _ ω], *auto simp add*: *ltI nat_lt_Aleph_rel1*)

lemma *le_Aleph_rel1_nat*: $M(k) \Longrightarrow$ *Card_rel*(M,k) $\Longrightarrow$ $k<\aleph_1{}^M \Longrightarrow k \leq \omega$
by (*simp add*: *Aleph_rel_succ Aleph_rel_zero Card_rel_lt_csucc_rel_iff Card_rel_nat*)

lemma *lesspoll_rel_Aleph_rel_plus_one*:
assumes *Ord*(α)
and *types*:$M(\alpha)$ $M(d)$
shows $d \prec^M \aleph_{succ(\alpha)}{}^M \longleftrightarrow d \lesssim^M \aleph_\alpha{}^M$
using *assms Aleph_rel_succ Card_rel_is_Ord Ord_Aleph_rel lesspoll_rel_csucc_rel*
by *simp*

lemma *cardinal_rel_Aleph_rel* [*simp*]: *Ord*(α) $\Longrightarrow$ $M(\alpha) \Longrightarrow |\aleph_\alpha{}^M|^M = \aleph_\alpha{}^M$
using *Card_rel_cardinal_rel_eq* **by** *simp*

— Could be proved without using AC
lemma *Aleph_rel_lesspoll_rel_increasing*:
includes *Aleph_rel_intros*
assumes $M(b)$ $M(a)$
shows $a < b \Longrightarrow \aleph_a{}^M \prec^M \aleph_b{}^M$
using *assms*
cardinal_rel_lt_iff_lesspoll_rel[*of* $\aleph_a{}^M$ $\aleph_b{}^M$]
Aleph_rel_increasing[*of a b*] *Card_rel_cardinal_rel_eq*[*of* $\aleph_b$]
lt_Ord lt_Ord2 Card_rel_Aleph_rel[*THEN Card_rel_is_Ord*]
by *auto*

lemma *uncountable_rel_iff_subset_eqpoll_rel_Aleph_rel1*:

includes *Ord_dests*
assumes $M(X)$
notes *Aleph_rel_zero*[*simp*] *Card_rel_nat*[*simp*] *Aleph_rel_succ*[*simp*]
shows *uncountable_rel*(M,X) $\longleftrightarrow$ ($\exists\, S[M].\ S \subseteq X \wedge S \approx^M \aleph_1{}^M$)
proof
assume *uncountable_rel*(M,X)
with ‹$M(X)$›
have $\aleph_1{}^M \lesssim^M X$
using *uncountable_rel_iff_nat_lt_cardinal_rel cardinal_rel_lt_csucc_rel_iff'*
cardinal_rel_le_imp_lepoll_rel **by** *auto*
with ‹$M(X)$›
obtain S **where** $M(S)$ $S \subseteq X$ $S \approx^M \aleph_1{}^M$
using *lepoll_rel_imp_subset_bij_rel* **by** *auto*
then
show $\exists\, S[M].\ S \subseteq X \wedge S \approx^M \aleph_1{}^M$
using *cardinal_rel_cong Card_rel_csucc_rel*[*of* ω] *Card_rel_cardinal_rel_eq*
by *auto*
next
note *Aleph_rel_lesspoll_rel_increasing*[*of 1 0*,*simplified*]
assume $\exists\, S[M].\ S \subseteq X \wedge S \approx^M \aleph_1{}^M$
moreover
have *eq*:$\aleph_1{}^M = (\omega^+)^M$ **by** *auto*
moreover from *calculation* ‹$M(X)$›
have *A*:$(\omega^+)^M \lesssim^M X$
using *subset_imp_lepoll_rel*[*THEN* [*2*] *eq_lepoll_rel_trans*, *of* $\aleph_1{}^M$ _ X,
OF eqpoll_rel_sym] **by** *auto*
with ‹$M(X)$›
show *uncountable_rel*(M,X)
using
lesspoll_rel_trans1[*OF lepoll_rel_trans*[*OF A* _] ‹$\omega \prec^M (\omega^+)^M$›]
lesspoll_rel_not_refl
by *auto*
qed

lemma *UN_if_zero*: $M(K) \Longrightarrow (\bigcup x{\in}K.\ \mathit{if}\ M(x)\ \mathit{then}\ G\ `\ x\ \mathit{else}\ 0) = (\bigcup x{\in}K.\ G\ `\ x)$
using *transM*[*of* _ K] **by** *auto*

lemma *mem_F_bound1*:
fixes F G
defines $F \equiv \lambda\_\ x.\ \mathit{if}\ M(x)\ \mathit{then}\ G`x\ \mathit{else}\ 0$
shows $x{\in}F(A,c) \Longrightarrow c \in (\mathit{range}(f) \cup \mathit{domain}(G))$
using *apply_0* **unfolding** *F_def*
by (*cases* $M(c)$, *auto simp*:*F_def drSR_Y_def dC_F_def*)

lemma *lt_Aleph_rel_imp_cardinal_rel_UN_le_nat*: *function*(G) $\Longrightarrow$ *domain*(G) $\lesssim^M \omega \Longrightarrow$
$\forall\, n{\in}\mathit{domain}(G).\ |G`n|^M{<}\aleph_1{}^M \Longrightarrow M(G) \Longrightarrow |\bigcup n{\in}\mathit{domain}(G).\ G`n|^M{\le}\omega$
proof -

assume $M(G)$
moreover from *this*
have $x \in (\mathit{if}\ M(i)\ \mathit{then}\ G\ `\ i\ \mathit{else}\ 0) \Longrightarrow M(i)$ **for** $x\ i$
by (*cases* $M(i)$) *auto*
moreover
have *separation*(M, M) **unfolding** *separation_def* **by** *auto*
ultimately
interpret *M_replacement_lepoll M* $\lambda\_\ x.\ \mathit{if}\ M(x)\ \mathit{then}\ G`x\ \mathit{else}\ 0$
using *lam_replacement_inj_rel cardinal_lib_assms2 mem_F_bound1*[*of* _ _ *G*]
lam_if_then_replacement_apply
by (*unfold_locales*, *simp_all*)
(*rule lam_Least_assumption_general*[**where** $U{=}\lambda\_.\ \mathit{domain}(G)$], *auto*)
note ‹$M(G)$›
moreover
have $w \in (\mathit{if}\ M(x)\ \mathit{then}\ G\ `\ x\ \mathit{else}\ 0) \Longrightarrow M(x)$ **for** $w\ x$
by (*cases* $M(x)$) *auto*
ultimately
interpret *M_cardinal_UN_lepoll* _ $\lambda n.\ \mathit{if}\ M(n)\ \mathit{then}\ G`n\ \mathit{else}\ 0\ \mathit{domain}(G)$
using *lepoll_assumptions1*[**where** $S{=}\mathit{domain}(G)$,*unfolded lepoll_assumptions1_def*]
cardinal_lib_assms2 lepoll_assumptions
by (*unfold_locales*, *auto*)
assume *function*(G)
let *?N*=*domain*(G) **and** *?R*=$\bigcup n{\in}\mathit{domain}(G).\ G`n$
assume *?N* $\lesssim^{M} \omega$
assume *Eq1*: $\forall\, n{\in}?N.\ |G`n|^{M} < \aleph_1{}^{M}$
{
fix n
assume $n{\in}?N$
with *Eq1* ‹$M(G)$›
have $|G`n|^{M} \le \omega$
using *le_Aleph_rel1_nat*[*of* $|G\ `\ n|^{M}$] *leqpoll_rel_imp_cardinal_rel_UN_le*
UN_if_zero[*of domain*(G) G]
by (*auto dest*:*transM*)
}
then
have $n{\in}?N \Longrightarrow |G`n|^{M} \le \omega$ **for** n **.**
moreover
note ‹*?N* $\lesssim^{M} \omega$› ‹$M(G)$›
moreover
have (*if* $M(i)$ *then* $G\ `\ i$ *else* 0) $\subseteq$ $G\ `\ i$ **for** i
by *auto*
with ‹$M(G)$›
have $|\mathit{if}\ M(i)\ \mathit{then}\ G\ `\ i\ \mathit{else}\ 0|^{M} \le |G\ `\ i|^{M}$ **for** i
proof(*cases* $M(i)$)
case *True*
with ‹$M(G)$› **show** *?thesis* **using** *Ord_cardinal_rel*[*OF apply_closed*]
by *simp*
next

```
    case False
    then
    have i∉domain(G)
      using transM[OF _ domain_closed[OF ‹M(G)›]] by auto
    then
    show ?thesis
      using Ord_cardinal_rel[OF apply_closed] apply_0 by simp
  qed
  ultimately
  show ?thesis
    using InfCard_rel_nat leqpoll_rel_imp_cardinal_rel_UN_le[of ω]
      UN_if_zero[of domain(G) G]
      le_trans[of |if M(_) then G ‘ _ else 0|^M |G ‘ _|^M ω]
    by auto blast
qed

lemma Aleph_rel1_eq_cardinal_rel_vimage: f:ℵ_1^M→^Mω ⟹ ∃n∈ω. |f-‘‘{n}|^M
= ℵ_1^M
proof -
  assume f:ℵ_1^M→^Mω
  then
  have function(f) domain(f) = ℵ_1^M range(f)⊆ω f∈ℵ_1^M→ω M(f)
    using mem_function_space_rel[OF ‹f∈_›] domain_of_fun fun_is_function
range_fun_subset_codomain
      function_space_rel_char
    by auto
  let ?G=λn∈range(f). f-‘‘{n}
  from ‹f:ℵ_1^M→ω›
  have range(f) ⊆ ω domain(?G) = range(f)
    using range_fun_subset_codomain
    by simp_all
  from ‹f:ℵ_1^M→ω› ‹M(f)› ‹range(f) ⊆ ω›
  have M(f-‘‘{n}) if n ∈ range(f) for n
    using that transM[of _ ω] by auto
  with ‹M(f)› ‹range(f) ⊆ ω›
  have domain(?G) ≲^M ω M(?G)
    using subset_imp_lepoll_rel lam_closed[of λx . f-‘‘{x}] cardinal_lib_assms4
    by simp_all
  have function(?G) by (simp add:function_lam)
  from ‹f:ℵ_1^M→ω›
  have n∈ω ⟹ f-‘‘{n} ⊆ ℵ_1^M for n
    using Pi_vimage_subset by simp
  with ‹range(f) ⊆ ω›
  have ℵ_1^M = (⋃n∈range(f). f-‘‘{n})
  proof (intro equalityI, intro subsetI)
    fix x
    assume x ∈ ℵ_1^M
    with ‹f:ℵ_1^M→ω› ‹function(f)› ‹domain(f) = ℵ_1^M›
    have x ∈ f-‘‘{f‘x} f‘x ∈ range(f)
```

```
    using function_apply_Pair vimage_iff apply_rangeI by simp_all
  then
  show x ∈ (⋃ n∈range(f). f-‘‘{n}) by auto
qed auto
{
  assume ∀ n∈range(f). |f-‘‘{n}|^M < ℵ_1^M
  then
  have ∀ n∈domain(?G). |?G‘n|^M < ℵ_1^M
    using zero_lt_Aleph_rel1 by (auto)
  with ‹function(?G)› ‹domain(?G) ≲^M ω› ‹M(?G)›
  have |⋃ n∈domain(?G). ?G‘n|^M≤ω
    using lt_Aleph_rel_imp_cardinal_rel_UN_le_nat[of ?G]
    by auto
  then
  have |⋃ n∈range(f). f-‘‘{n}|^M≤ω by simp
  with ‹ℵ_1^M = _›
  have |ℵ_1^M|^M ≤ ω by auto
  then
  have ℵ_1^M ≤ ω
    using Card_rel_Aleph_rel Card_rel_cardinal_rel_eq
    by auto
  then
  have False
    using nat_lt_Aleph_rel1 by (blast dest:lt_trans2)
}
with ‹range(f)⊆ω› ‹M(f)›
obtain n where n∈ω ¬(|f -‘‘ {n}|^M < ℵ_1^M) M(f -‘‘ {n})
  using nat_into_M by auto
moreover from this
have ℵ_1^M ≤ |f-‘‘{n}|^M
  using not_lt_iff_le Card_rel_is_Ord by simp
moreover
note ‹n∈ω ⟹ f-‘‘{n} ⊆ ℵ_1^M›
ultimately
show ?thesis
  using subset_imp_le_cardinal_rel[THEN le_anti_sym, of _ ℵ_1^M]
    Card_rel_Aleph_rel Card_rel_cardinal_rel_eq
  by auto
qed
```

— There is some asymmetry between assumptions and conclusion (*eqpoll_rel* versus *cardinal_rel*)

```
lemma eqpoll_rel_Aleph_rel1_cardinal_rel_vimage:
  assumes Z ≈^M (ℵ_1^M) f ∈ Z →^M ω M(Z)
  shows ∃ n∈ω. |f-‘‘{n}|^M = ℵ_1^M
proof -
  have M(1) M(ω) by simp_all
  then
```

have $M(\aleph_1{}^M)$ **by** *simp*
with *assms* ‹$M(1)$›
obtain g **where** A:$g\in bij\_rel(M,\aleph_1{}^M,Z)$ $M(g)$
using *eqpoll_rel_sym* **unfolding** *eqpoll_rel_def* **by** *blast*
with ‹$f : Z \to^M \omega$› *assms*
have $M(f)$ $converse(g) \in bij\_rel(M,Z, \aleph_1{}^M)$ $f\in Z\to\omega$ $g\in \aleph_1{}^M\to Z$
using *bij_rel_is_fun_rel bij_rel_converse_bij_rel bij_rel_char function_space_rel_char*
by *simp_all*
with ‹$g\in bij\_rel(M,\aleph_1{}^M,Z)$› ‹$M(g)$›
have $f\ O\ g : \aleph_1{}^M \to^M \omega$ $M(converse(g))$
using *comp_fun*[*OF* _ ‹$f\in Z\to$_›,*of g*] *function_space_rel_char*
by *simp_all*
then
obtain n **where** $n\in\omega$ $|(f\ O\ g)\text{-}``\{n\}|^M = \aleph_1{}^M$
using *Aleph_rel1_eq_cardinal_rel_vimage*
by *auto*
with ‹$M(f)$› ‹$M(converse(g))$›
have $M(converse(g)\ ``\ (f\text{-}``\ \{n\}))$ $f\text{-}``\ \{n\} \subseteq Z$
using *image_comp converse_comp Pi_iff*[*THEN iffD1,OF* ‹$f\in Z\to\omega$›] *vimage_subset*
unfolding *vimage_def*
using *transM*[*OF* ‹$n\in\omega$›]
by *auto*
from ‹$n\in\omega$› ‹$|(f\ O\ g)\text{-}``\{n\}|^M = \aleph_1{}^M$›
have $\aleph_1{}^M = |converse(g)\ ``\ (f\text{-}``\{n\})|^M$
using *image_comp converse_comp* **unfolding** *vimage_def*
by *auto*
also from ‹$converse(g) \in bij\_rel(M,Z, \aleph_1{}^M)$› ‹$f\colon Z\to^M \omega$› ‹$M(Z)$› ‹$M(f)$›
‹$M(\aleph_1{}^M)$›
‹$M(converse(g)\ ``\ (f\text{-}``\ \{n\}))$›
have $\ldots = |f\text{-}``\{n\}|^M$
using *range_of_subset_eqpoll_rel*[*of converse*(g) Z _ $f\text{-}``\{n\}$,
OF bij_rel_is_inj_rel[*OF* ‹$converse(g)\in$_›] ‹$f\text{-}``\ \{n\} \subseteq Z$›]
cardinal_rel_cong vimage_closed[*OF singleton_closed*[*OF transM*[*OF* ‹$n\in\omega$›]],*of f*]
by *auto*
finally
show *?thesis* **using** ‹$n\in$_› **by** *auto*
qed

46.4 Applications of transfinite recursive constructions

definition
$rec\_constr :: [i,i] \Rightarrow i$ **where**
$rec\_constr(f,\alpha) \equiv transrec(\alpha,\lambda a\ g.\ f`(g``a))$

The function *rec_constr* allows to perform *recursive constructions*: given a choice function on the powerset of some set, a transfinite sequence is created by successively choosing some new element.

The next result explains its use.

```
lemma rec_constr_unfold: rec_constr(f,α) = f`({rec_constr(f,β). β∈α})
  using def_transrec[OF rec_constr_def, of f α] image_lam by simp

lemma rec_constr_type:
  assumes f:Pow_rel(M,G)→^M G Ord(α) M(G)
  shows M(α) ⟹ rec_constr(f,α) ∈ G
  using assms(2)
proof(induct rule:trans_induct)
  case (step β)
  with assms
  have {rec_constr(f, x) . x ∈ β} = {y . x ∈ β, y=rec_constr(f, x)} (is _ = ?Y)
       M(f)
    using transM[OF _ ‹M(β)›] function_space_rel_char  Ord_in_Ord
    by auto
  moreover from assms this step ‹M(β)› ‹Ord(β)›
  have M({y . x ∈ β, y=<x,rec_constr(f, x)>}) (is M(?Z))
    using strong_replacement_closed[OF cardinal_lib_assms6(1),of f β β,OF _
_ _ _
      univalent_conjI2[where P=λx _ . x∈β,OF univalent_triv]]
      transM[OF _ ‹M(β)›] transM[OF step(2) ‹M(G)›] Ord_in_Ord
    unfolding rec_constr_def
    by auto
  moreover from assms this step ‹M(β)› ‹Ord(β)›
  have ?Y = {snd(y) . y∈?Z}
  proof(intro equalityI, auto)
    fix u
    assume u∈β
    with assms this step ‹M(β)› ‹Ord(β)›
   have <u,rec_constr(f,u)> ∈ ?Z  rec_constr(f, u) = snd(<u,rec_constr(f,u)>)
      by auto
    then
    show ∃x∈{y . x ∈ β, y = ⟨x, rec_constr(f, x)⟩}. rec_constr(f, u) = snd(x)
      using bexI[of _ u] by force
  qed
  moreover from ‹M(?Z)› ‹?Y = _›
  have M(?Y)
    using
    RepFun_closed[OF lam_replacement_imp_strong_replacement[OF lam_replacement_snd]
‹M(?Z)›]
      fst_snd_closed[THEN conjunct2] transM[OF _ ‹M(?Z)›]
    by simp
  moreover from assms step
  have {rec_constr(f, x) . x ∈ β} ∈ Pow(G) (is ?X∈_)
    using transM[OF _ ‹M(β)›] function_space_rel_char
    by auto
  moreover from assms calculation step
  have M(?X)
    by simp
  moreover from calculation ‹M(G)›
```

have *?X∈Pow_rel(M,G)*
using *Pow_rel_char* **by** *simp*
ultimately
have *f‘?X ∈ G*
using *assms apply_type*[*OF mem_function_space_rel*[*of f*],*of Pow_rel*(*M*,*G*)
G ?X]
by *auto*
then
show *?case*
by (*subst rec_constr_unfold*,*simp*)
qed

lemma *rec_constr_closed* :
assumes *f*:*Pow_rel*(*M*,*G*)→^{M} *G Ord*(*α*) *M*(*G*) *M*(*α*)
shows *M*(*rec_constr*(*f*,*α*))
using *transM*[*OF rec_constr_type* ‹*M*(*G*)›] *assms* **by** *auto*

lemma *lambda_rec_constr_closed* :
assumes *Ord*(*γ*) *M*(*γ*) *M*(*f*) *f*:*Pow_rel*(*M*,*G*)→^{M} *G M*(*G*)
shows *M*(*λα∈γ . rec_constr*(*f*,*α*))
using *lam_closed2*[*OF cardinal_lib_assms6*(*1*),*unfolded rec_constr_def*[*symmetric*],*of*
f γ]
rec_constr_type[*OF* ‹*f*∈_› *Ord_in_Ord*[*of γ*]] *transM*[*OF* _ ‹*M*(*G*)›] *assms*
by *simp*

The next lemma is an application of recursive constructions. It works under the assumption that whenever the already constructed subsequence is small enough, another element can be added.

— FIXME: these should be postulated in some locale.

lemma *bounded_cardinal_rel_selection*:
includes *Ord_dests*
assumes
⋀*Z*. |*Z*|^M < *γ* ⟹ *Z* ⊆ *G* ⟹ *M*(*Z*) ⟹ ∃ *a*∈*G*. ∀ *s*∈*Z*. <*s*,*a*>∈*Q* *b*∈*G*
Card_rel(*M*,*γ*)
M(*G*) *M*(*Q*) *M*(*γ*)
shows
∃ *S*[*M*]. *S* : *γ* →^{M} *G* ∧ (∀ *α* ∈ *γ*. ∀ *β* ∈ *γ*. *α*<*β* ⟶ <*S‘α*,*S‘β*>∈*Q*)
proof -
from *assms*
have *M*(*x*) ⟹ *M*({*a* ∈ *G* . ∀ *s*∈*x*. ⟨*s*, *a*⟩ ∈ *Q*}) **for** *x*
using *cdlt_assms* **by** *simp*
let *?cdlt*γ={*Z*∈*Pow_rel*(*M*,*G*) . |*Z*|^M<*γ*} — “cardinal_rel less than *γ*”
and *?inQ*=*λY*.{*a*∈*G*. ∀ *s*∈*Y*. <*s*,*a*>∈*Q*}
from ‹*M*(*G*)› ‹*Card_rel*(*M*,*γ*)› ‹*M*(*γ*)›
have *M*(*?cdlt*γ) *Ord*(*γ*)
using *cardinal_lib_assms5*[*OF* ‹*M*(*γ*)›] *Card_rel_is_Ord*
by *simp_all*
from *assms*

```
  have H:∃ a. a ∈ ?inQ(Y) if Y∈?cdltγ for Y
  proof -
    {
      fix Y
      assume Y∈?cdltγ
      then
      have A:Y∈Pow_rel(M,G) |Y|^M<γ  by simp_all
      then
      have Y⊆G M(Y) using Pow_rel_char[OF ‹M(G)›] by simp_all
      with A
      obtain a where a∈G ∀ s∈Y. <s,a>∈Q
        using assms(1) by force
      with ‹M(G)›
      have ∃ a. a ∈ ?inQ(Y) by auto
    }
    then show ?thesis using that by simp
  qed
  then
  have ∃ f[M]. f ∈ Pi_rel(M,?cdltγ,?inQ) ∧ f ∈ Pi(?cdltγ,?inQ)
  proof -
    from ‹⋀x. M(x) ⟹ M({a ∈ G . ∀ s∈x. ⟨s, a⟩ ∈ Q})› ‹M(G)›
     have x ∈ {Z ∈ Pow^M(G) . |Z|^M < γ} ⟹ M({a ∈ G . ∀ s∈x. ⟨s, a⟩ ∈ Q})
for x
      by (auto dest:transM)
   with‹M(G)› ‹⋀x. M(x) ⟹ M({a ∈ G . ∀ s∈x. ⟨s, a⟩ ∈ Q})› ‹M(Q)› ‹M(?cdltγ)›
    interpret M_Pi_assumptions_choice M ?cdltγ ?inQ
      using cdlt_assms[where Q=Q] lam_replacement_Collect_ball_Pair[THEN
          lam_replacement_imp_strong_replacement] surj_imp_inj_replacement3
        lam_replacement_hcomp2[OF lam_replacement_constant
          lam_replacement_Collect_ball_Pair _ _ lam_replacement_minimum,
          unfolded lam_replacement_def]
        lam_replacement_hcomp lam_replacement_Sigfun[OF
          lam_replacement_Collect_ball_Pair, of G Q, THEN
          lam_replacement_imp_strong_replacement]
      by unfold_locales (blast dest: transM, auto dest:transM)
    show ?thesis using AC_Pi_rel Pi_rel_char H by auto
    qed
  then
   obtain f where f_type:f ∈ Pi_rel(M,?cdltγ,?inQ) f ∈ Pi(?cdltγ,?inQ) and
M(f)
    by auto
  moreover
  define Cb where Cb ≡ λ_∈Pow_rel(M,G)-?cdltγ. b
  moreover from ‹b∈G› ‹M(?cdltγ)› ‹M(G)›
  have Cb ∈ Pow_rel(M,G)-?cdltγ → G M(Cb)
    using lam_closed[of λ_.b Pow_rel(M,G)-?cdltγ]
      tag_replacement transM[OF ‹b∈G›]
    unfolding Cb_def by auto
  moreover
```

note ‹*Card_rel*(*M*,*γ*)›
ultimately
have *f* ∪ *Cb* : (∏ *x*∈*Pow_rel*(*M*,*G*). *?inQ*(*x*) ∪ *G*) **using**
fun_Pi_disjoint_Un[*of f ?cdlt*γ *?inQ Cb Pow_rel*(*M*,*G*)-*?cdlt*γ λ_.*G*]
Diff_partition[*of* {*Z*∈*Pow_rel*(*M*,*G*). |*Z*|^*M*<γ} *Pow_rel*(*M*,*G*), *OF Collect_subset*]
by *auto*
moreover
have *?inQ*(*x*) ∪ *G* = *G* **for** *x* **by** *auto*
moreover from *calculation*
have *f* ∪ *Cb* : *Pow_rel*(*M*,*G*) → *G*
using *function_space_rel_char* **by** *simp*
ultimately
have *f* ∪ *Cb* : *Pow_rel*(*M*,*G*) →^*M* *G*
using *function_space_rel_char* ‹*M*(*f*)› ‹*M*(*Cb*)› *Pow_rel_closed* ‹*M*(*G*)›
by *auto*
define *S* **where** *S*≡λα∈γ. *rec_constr*(*f* ∪ *Cb*, α)
from ‹*f* ∪ *Cb*: *Pow_rel*(*M*,*G*) →^*M* *G*› ‹*Card_rel*(*M*,*γ*)› ‹*M*(*γ*)› ‹*M*(*G*)›
have *S* : *γ* → *G* *M*(*f* ∪ *Cb*)
unfolding *S_def*
using *Ord_in_Ord*[*OF Card_rel_is_Ord*] *rec_constr_type lam_type transM*[*OF* _ ‹*M*(*γ*)›]
function_space_rel_char
by *auto*
moreover from ‹*f*∪*Cb* ∈ _→^*M* *G*› ‹*Card_rel*(*M*,*γ*)› ‹*M*(*γ*)› ‹*M*(*G*)› ‹*M*(*f* ∪ *Cb*)› ‹*Ord*(*γ*)›
have *M*(*S*)
unfolding *S_def*
using *lambda_rec_constr_closed*
by *simp*
moreover
have ∀α∈γ. ∀β∈γ. α < β ⟶ <*S* ‘ α, *S* ‘ β>∈*Q*
proof (*intro ballI impI*)
fix α β
assume β∈γ
with ‹*Card_rel*(*M*,*γ*)› ‹*M*(*S*)› ‹*M*(*γ*)›
have β⊆γ *M*(*S*‘‘β) *M*(β) {*S*‘*x* . *x* ∈ β} = {*restrict*(*S*,β) ‘*x* . *x* ∈ β}
using *transM*[*OF* ‹β∈γ› ‹*M*(*γ*)›] *image_closed Card_rel_is_Ord OrdmemD*
by *auto*
with ‹β∈_› ‹*Card_rel*(*M*,*γ*)› ‹*M*(*γ*)›
have {*rec_constr*(*f* ∪ *Cb*, *x*) . *x*∈β} = {*S*‘*x* . *x* ∈ β}
using *Ord_trans*[*OF* _ _ *Card_rel_is_Ord*, *of* _ β γ]
unfolding *S_def*
by *auto*
moreover from ‹β∈γ› ‹*S* : *γ* → *G*› ‹*Card_rel*(*M*,*γ*)› ‹*M*(*γ*)› ‹*M*(*S*‘‘β)›
have {*S*‘*x* . *x* ∈ β} ⊆ *G* *M*({*S*‘*x* . *x* ∈ β})
using *Ord_trans*[*OF* _ _ *Card_rel_is_Ord*, *of* _ β γ]
apply_type[*of S* γ λ_. *G*]
by(*auto*,*simp add*:*image_fun_subset*[*OF* ‹*S*∈_› ‹β⊆_›])
moreover from ‹*Card_rel*(*M*,*γ*)› ‹β∈γ› ‹*S*∈_› ‹β⊆γ› ‹*M*(*S*)› ‹*M*(β)› ‹*M*(*G*)›

```
‹M(γ)›
  have |{S‘x . x ∈ β}|^M < γ
    using
      ‹{S‘x . x∈β} = {restrict(S,β)‘x . x∈β}›[symmetric]
      cardinal_rel_RepFun_le[of restrict(S,β) β G,
          OF restrict_type2[of S γ λ_.G β] restrict_closed]
      Ord_in_Ord Ord_cardinal_rel
      lt_trans1[of |{S‘x . x ∈ β}|^M |β|^M γ]
      Card_rel_lt_iff[THEN iffD2, of β γ, OF _ _ _ _ ltI]
      Card_rel_is_Ord
    by auto
  moreover
  have ∀ x∈β. <S‘x, f ‘ {S‘x . x ∈ β}> ∈ Q
  proof -
    from calculation and f_type
    have f ‘ {S‘x . x ∈ β} ∈ {a∈G. ∀ x∈β. <S‘x,a>∈Q}
      using apply_type[of f ?cdltγ ?inQ {S‘x . x ∈ β}]
          Pow_rel_char[OF ‹M(G)›]
      by simp
    then
    show ?thesis by simp
  qed
  moreover
  assume α∈γ α < β
  moreover from this
  have α∈β using ltD by simp
  moreover
  note ‹β∈γ› ‹Cb ∈ Pow_rel(M,G)-?cdltγ → G›
  ultimately
  show <S ‘ α, S ‘ β>∈Q
    using fun_disjoint_apply1[of {S‘x . x ∈ β} Cb f]
      domain_of_fun[of Cb] ltD[of α β]
     by (subst (2) S_def, auto) (subst rec_constr_unfold, auto)
 qed
 moreover
 note ‹M(G)› ‹M(γ)›
 ultimately
show ?thesis using function_space_rel_char by auto
qed
```

The following basic result can, in turn, be proved by a bounded-cardinal_rel selection.

```
lemma Infinite_iff_lepoll_rel_nat: M(Z) ⟹ Infinite(Z) ⟷ ω ≲^M Z
proof
  define Distinct where Distinct = {<x,y> ∈ Z×Z . x≠y}
  have Distinct = {xy ∈ Z×Z . ∃ a b. xy = ⟨a, b⟩ ∧ a ≠ b} (is _=?A)
    unfolding Distinct_def by auto
  moreover
  assume Infinite(Z) M(Z)
```

moreover from *calculation*
have $M(Distinct)$
using *cardinal_lib_assms6* **by** *simp*
from ‹*Infinite*(Z)› ‹$M(Z)$›
obtain b **where** $b{\in}Z$
using *Infinite_not_empty* **by** *auto*
{
fix Y
assume $|Y|^M < \omega$ $M(Y)$
then
have *Finite*(Y)
using *Finite_cardinal_rel_iff'* *ltD* *nat_into_Finite* **by** *auto*
with ‹*Infinite*(Z)›
have $Z \neq Y$ **by** *auto*
}
moreover
have $(\bigwedge W.\ M(W) \Longrightarrow |W|^M < \omega \Longrightarrow W \subseteq Z \Longrightarrow \exists\, a{\in}Z.\ \forall\, s{\in}W.\ <s,a>{\in}Distinct)$
proof -
fix W
assume $M(W)$ $|W|^M < \omega$ $W \subseteq Z$
moreover from *this*
have *Finite_rel*(M, W)
using
cardinal_rel_closed[*OF* ‹$M(W)$›] *Card_rel_nat*
lt_Card_rel_imp_lesspoll_rel[*of* ω,*simplified*,*OF* _ ‹$|W|^M < \omega$›]
lesspoll_rel_nat_is_Finite_rel[*of* W]
eqpoll_rel_imp_lepoll_rel *eqpoll_rel_sym*[*OF* *cardinal_rel_eqpoll_rel*,*of* W]
lesspoll_rel_trans1[*of* W $|W|^M$ ω] **by** *auto*
moreover from *calculation*
have $\neg Z{\subseteq}W$
using *equalityI* ‹*Infinite*(Z)› **by** *auto*
moreover from *calculation*
show $\exists\, a{\in}Z.\ \forall\, s{\in}W.\ <s,a>{\in}Distinct$
unfolding *Distinct_def* **by** *auto*
qed
moreover from ‹$b{\in}Z$› ‹$M(Z)$› ‹$M(Distinct)$› *this*
obtain S **where** $S : \omega \rightarrow^M Z$ $M(S)$ $\forall\, \alpha{\in}\omega.\ \forall\, \beta{\in}\omega.\ \alpha < \beta \longrightarrow <S`\alpha,S`\beta> \in Distinct$
using *bounded_cardinal_rel_selection*[*OF* _ ‹$b{\in}Z$› *Card_rel_nat*,*of Distinct*]
by *blast*
moreover from *this*
have $\alpha \in \omega \Longrightarrow \beta \in \omega \Longrightarrow \alpha{\neq}\beta \Longrightarrow S`\alpha \neq S`\beta$ **for** α β
unfolding *Distinct_def*
by (*rule_tac* *lt_neq_symmetry*[*of* ω $\lambda\alpha$ β. $S`\alpha \neq S`\beta$])
auto
moreover from *this* ‹$S{\in}\_$› ‹$M(Z)$›
have $S{\in}inj(\omega,Z)$ **using** *function_space_rel_char* **unfolding** *inj_def* **by** *auto*
ultimately
show $\omega \lesssim^M Z$

unfolding *lepoll_rel_def* **using** *inj_rel_char* ‹$M(Z)$› **by** *auto*
next
assume $\omega \lesssim^M Z\ M(Z)$
then
show *Infinite*(Z) **using** *lepoll_rel_nat_imp_Infinite* **by** *simp*
qed

lemma *Infinite_InfCard_rel_cardinal_rel*: *Infinite*(Z) $\Longrightarrow$ $M(Z)$ $\Longrightarrow$ *InfCard_rel*(M,$|Z|^M$)
using *lepoll_rel_eq_trans eqpoll_rel_sym lepoll_rel_nat_imp_Infinite*
Infinite_iff_lepoll_rel_nat Inf_Card_rel_is_InfCard_rel cardinal_rel_eqpoll_rel
by *simp*

lemma (**in** *M_trans*) *mem_F_bound2*:
fixes $F\ A$
defines $F \equiv \lambda\_\ x.\ if\ M(x)\ then\ A\text{-}``\{x\}\ else\ 0$
shows $x{\in}F(A,c) \Longrightarrow c \in (range(f) \cup range(A))$
using *apply_0* **unfolding** *F_def*
by (*cases* $M(c)$, *auto simp*:*F_def drSR_Y_def dC_F_def*)

lemma *Finite_to_one_rel_surj_rel_imp_cardinal_rel_eq*:
assumes $F \in Finite\_to\_one\_rel(M,Z,Y) \cap surj\_rel(M,Z,Y)$ *Infinite*(Z) $M(Z)$
$M(Y)$
shows $|Y|^M = |Z|^M$
proof -
have *sep_true*: *separation*(M, M) **unfolding** *separation_def* **by** *auto*
note ‹$M(Z)$› ‹$M(Y)$›
moreover from *this assms*
have $M(F)$ $F \in Z \to Y$
unfolding *Finite_to_one_rel_def*
using *function_space_rel_char* **by** *simp_all*
moreover from *this*
have $x \in (if\ M(i)\ then\ F\text{-}``\{i\}\ else\ 0) \Longrightarrow M(i)$ **for** $x\ i$
by (*cases* $M(i)$) *auto*
moreover from *calculation*
interpret *M_replacement_lepoll* $M\ \lambda\_\ x.\ if\ M(x)\ then\ F\text{-}``\{x\}\ else\ 0$
using *lam_replacement_inj_rel mem_F_bound2 cardinal_lib_assms3*
lam_replacement_vimage_sing_fun
lam_replacement_if[*OF* _
lam_replacement_constant[*OF nonempty*],**where** $b{=}M$] *sep_true*
by (*unfold_locales*, *simp_all*)
(*rule lam_Least_assumption_general*[**where** $U{=}\lambda\_.\ range(F)$], *auto*)
have $w \in (if\ M(y)\ then\ F\text{-}``\{y\}\ else\ 0) \Longrightarrow M(y)$ **for** $w\ y$
by (*cases* $M(y)$) *auto*
moreover from ‹$F{\in}\_{\cap}\_$›
have *0*:*Finite*($F\text{-}``\{y\}$) **if** $y{\in}Y$ **for** y
unfolding *Finite_to_one_rel_def*
using *vimage_fun_sing* ‹$F{\in}Z{\to}Y$› *transM*[*OF that* ‹$M(Y)$›] *transM*[*OF* _
‹$M(Z)$›] *that* **by** *simp*
ultimately

```
  interpret M_cardinal_UN_lepoll _ λy. if M(y) then F-‘‘{y} else 0 Y
    using cardinal_lib_assms3 lepoll_assumptions
    by unfold_locales  (auto dest:transM simp del:mem_inj_abs)
  from ‹F∈Z→Y›
  have Z = (⋃ y∈Y. {x∈Z . F‘x = y})
    using apply_type by auto
  then
  show ?thesis
  proof (cases Finite(Y))
    case True
    with ‹Z = (⋃ y∈Y. {x∈Z . F‘x = y})› and assms and ‹F∈Z→Y›
    show ?thesis
    using Finite_RepFun[THEN [2] Finite_Union, of Y λy. F-‘‘{y}] 0 vimage_fun_sing[OF
‹F∈Z→Y›]
      by simp
  next
    case False
    moreover from this ‹M(Y)›
    have Y ≲^M |Y|^M
      using cardinal_rel_eqpoll_rel eqpoll_rel_sym eqpoll_rel_imp_lepoll_rel by
auto
    moreover
    note assms
    moreover from ‹F∈_∩_›
    have Finite({x∈Z . F‘x = y}) M(F-‘‘{y}) if y∈Y for y
      unfolding Finite_to_one_rel_def
      using transM[OF that  ‹M(Y)›] transM[OF _ ‹M(Z)›] vimage_fun_sing[OF
‹F∈Z→Y›] that
      by simp_all
    moreover from calculation
    have |{x∈Z . F‘x = y}|^M ∈ ω if y∈Y for y
    using Finite_cardinal_rel_in_nat that transM[OF that ‹M(Y)›] vimage_fun_sing[OF
‹F∈Z→Y›] that
      by simp
    moreover from calculation
    have |{x∈Z . F‘x = y}|^M ≤ |Y|^M if y∈Y for y
      using Infinite_imp_nats_lepoll_rel[THEN lepoll_rel_imp_cardinal_rel_le,
            of _ |{x∈Z . F‘x = y}|^M]
          that cardinal_rel_idem transM[OF that ‹M(Y)›] vimage_fun_sing[OF
‹F∈Z→Y›]
      by auto
    ultimately
    have |⋃ y∈Y. {x∈Z . F‘x = y}|^M ≤ |Y|^M
      using leqpoll_rel_imp_cardinal_rel_UN_le
        Infinite_InfCard_rel_cardinal_rel[of Y] vimage_fun_sing[OF ‹F∈Z→Y›]
      by(auto simp add:transM[OF _ ‹M(Y)›])
   moreover from ‹F ∈ Finite_to_one_rel(M,Z,Y) ∩ surj_rel(M,Z,Y)› ‹M(Z)›
‹M(F)› ‹M(Y)›
    have |Y|^M ≤ |Z|^M
```

```
    using surj_rel_implies_cardinal_rel_le by auto
  moreover
  note ‹Z = (⋃ y∈Y. {x∈Z . F`x = y})›
  ultimately
  show ?thesis
    using le_anti_sym by auto
 qed
qed

lemma cardinal_rel_map_Un:
 assumes Infinite(X) Finite(b) M(X) M(b)
 shows |{a ∪ b . a ∈ X}|^M = |X|^M
proof -
 have (λa∈X. a ∪ b) ∈ Finite_to_one_rel(M,X,{a ∪ b . a ∈ X})
   (λa∈X. a ∪ b) ∈  surj_rel(M,X,{a ∪ b . a ∈ X})
   M({a ∪ b . a ∈ X})
   unfolding def_surj_rel
 proof
   fix d
   have Finite({a ∈ X . a ∪ b = d}) (is Finite(?Y(b,d)))
     using ‹Finite(b)›
   proof (induct arbitrary:d)
     case 0
     have {a ∈ X . a ∪ 0 = d} = (if d∈X then {d} else 0)
       by auto
     then
     show ?case by simp
   next
     case (cons c b)
     from ‹c ∉ b›
     have ?Y(cons(c,b),d) ⊆ (if c∈d then ?Y(b,d) ∪ ?Y(b,d-{c}) else 0)
       by auto
     with cons
     show ?case
       using subset_Finite
       by simp
   qed
   moreover
   assume d ∈ {x ∪ b . x ∈ X}
   ultimately
   show Finite({a ∈ X . M(a) ∧ (λx∈X. x ∪ b) ` a = d})
     using subset_Finite[of {a ∈ X . M(a) ∧ (λx∈X. x ∪ b) ` a = d}
         {a ∈ X . (λx∈X. x ∪ b) ` a = d}] by auto
 next
   note ‹M(X)› ‹M(b)›
   moreover
   show M(λa∈X. a ∪ b)
     using lam_closed[of λ x . x∪b,OF _ ‹M(X)›] Un_closed[OF transM[OF _
‹M(X)›] ‹M(b)›]
```

tag_union_replacement[*OF* ‹*M*(*b*)›]
by *simp*
moreover from *this*
have $\{a \cup b \,.\, a \in X\} = (\lambda x{\in}X.\ x \cup b)\ ``\ X$
using *image_lam* **by** *simp*
with *calculation*
show $M(\{a \cup b \,.\, a \in X\})$ **by** *auto*
moreover from *calculation*
show $(\lambda a{\in}X.\ a \cup b) \in X \to^M \{a \cup b \,.\, a \in X\}$
using *function_space_rel_char* **by** (*auto intro*:*lam_funtype*)
ultimately
show $(\lambda a{\in}X.\ a \cup b) \in surj^M(X,\{a \cup b \,.\, a \in X\})\ M(\{a \cup b \,.\, a \in X\})$
using *surj_rel_char function_space_rel_char*
unfolding *surj_def* **by** *auto*
next
qed (*simp add*:‹*M*(*X*)›)
moreover from *assms this*
show *?thesis*
using *Finite_to_one_rel_surj_rel_imp_cardinal_rel_eq* **by** *simp*
qed

end

end

47 The Delta System Lemma, Relativized

theory *Delta_System_Relative*
imports
Cardinal_Library_Relative
begin

definition
delta_system :: $i \Rightarrow o$ **where**
$delta\_system(D) \equiv \exists r.\ \forall A{\in}D.\ \forall B{\in}D.\ A \neq B \longrightarrow A \cap B = r$

lemma *delta_systemI*[*intro*]:
assumes $\forall A{\in}D.\ \forall B{\in}D.\ A \neq B \longrightarrow A \cap B = r$
shows $delta\_system(D)$
using *assms* **unfolding** *delta_system_def* **by** *simp*

lemma *delta_systemD*[*dest*]:
$delta\_system(D) \Longrightarrow \exists r.\ \forall A{\in}D.\ \forall B{\in}D.\ A \neq B \longrightarrow A \cap B = r$
unfolding *delta_system_def* **by** *simp*

lemma *delta_system_root_eq_Inter*:
assumes $delta\_system(D)$
shows $\forall A{\in}D.\ \forall B{\in}D.\ A \neq B \longrightarrow A \cap B = \bigcap D$

proof (*clarify*, *intro equalityI*, *auto*)
  fix A' B' x C
  assume *hyp*:$A'\in D$ $B'\in D$ $A'\neq B'$ $x\in A'$ $x\in B'$ $C\in D$
  with *assms*
  obtain r **where** *delta*:$\forall A\in D.\ \forall B\in D.\ A \neq B \longrightarrow A \cap B = r$
    by *auto*
  show $x \in C$
  proof (*cases* $C=A'$)
    case *True*
    with *hyp* **and** *assms*
    show *?thesis* **by** *simp*
  next
    case *False*
    moreover
    note *hyp*
    moreover from *calculation* **and** *delta*
    have $r = C \cap A'$ $A' \cap B' = r$ $x\in r$ **by** *auto*
    ultimately
    show *?thesis* **by** *simp*
  qed
qed

relativize functional *delta_system* *delta_system_rel* **external**

locale *M_delta* = *M_cardinal_library* +
  assumes
    cardinal_replacement:*strong_replacement*(M, $\lambda A\ y.\ y = \langle A, |A|^M\rangle$)
    and
    countable_lepoll_assms:
    $M(G) \Longrightarrow$ *separation*(M, $\lambda p.\ \forall x\in G.\ x \in snd(p) \longleftrightarrow fst(p) \in x$)
    $M(G) \Longrightarrow M(A) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow$ *separation*(M, $\lambda y.\ \exists x\in A.$
                    $y = \langle x, \mu\ i.\ x \in$ *if_range_F_else_F*($\lambda x.\ \{xa \in G\ .\ x \in xa\}$,
b, f, i)$\rangle$)
    and
    disjoint_separation: $M(c) \Longrightarrow$ *separation*(M, $\lambda\ x.\ \exists\, a.\ \exists\, b.\ x=\langle a,b\rangle \wedge a \cap b =$
c)

begin

lemma (**in** *M_trans*) *mem_F_bound6*:
  fixes F G
  defines $F \equiv \lambda\_\ x.$ *Collect*(G, $(\in)(x)$)
  shows $x\in F(G,c) \Longrightarrow c \in (range(f) \cup \bigcup G)$
  using *apply_0* **unfolding** *F_def*
  by (*cases* $M(c)$, *auto simp*:*F_def*)

lemma *delta_system_Aleph_rel1*:
  assumes $\forall\, A\in F.$ *Finite*(A) $F \approx^M \aleph_1{}^M$ $M(F)$
  shows $\exists\, D[M].\ D \subseteq F \wedge$ *delta_system*(D) $\wedge\ D \approx^M \aleph_1{}^M$

proof -
have $M(G) \Longrightarrow M(p) \Longrightarrow M(\{A{\in}G\ .\ p \in A\})$ **for** $G\ p$
proof -
assume $M(G)\ M(p)$
have $\{A{\in}G\ .\ p \in A\} = G \cap (Memrel(\{p\} \cup G)\ ``\ \{p\})$
unfolding *Memrel_def* **by** *auto*
with ‹$M(G)$› ‹$M(p)$›
show *?thesis* **by** *simp*
qed
from ‹$M(F)$›
have $M(\lambda A{\in}F.\ |A|^M)$
using *cardinal_replacement*
by (*rule_tac lam_closed*) (*auto dest*:*transM*)

Since all members are finite,

with ‹$\forall\, A{\in}F.\ Finite(A)$› ‹$M(F)$›
have $(\lambda A{\in}F.\ |A|^M) : F \rightarrow^M \omega$ (**is** *?cards* : _)
by (*simp add*:*mem_function_space_rel_abs*, *rule_tac lam_type*)
(*force dest*:*transM*)
moreover from *this*
have *a*:*?cards* -`` $\{n\} = \{\ A{\in}F\ .\ |A|^M = n\ \}$ **for** n
using *vimage_lam* **by** *auto*
moreover
note ‹$F \approx^M \aleph_1{}^M$› ‹$M(F)$›
moreover from *calculation*

there are uncountably many have the same cardinal:

obtain n **where** $n{\in}\omega\ |?cards\ -``\ \{n\}|^M = \aleph_1{}^M$
using *eqpoll_rel_Aleph_rel1_cardinal_rel_vimage*[*of* F *?cards*] **by** *auto*
moreover
define G **where** $G \equiv ?cards\ -``\ \{n\}$
moreover from *calculation* **and** ‹$M(?cards)$›
have $M(G)$ **by** *blast*
moreover from *calculation*
have $G \subseteq F$ **by** *auto*
ultimately

Therefore, without loss of generality, we can assume that all elements of the family have cardinality $n \in \omega$.

have $A{\in}G \Longrightarrow |A|^M = n$ **and** $G \approx^M \aleph_1{}^M$ **and** $M(G)$ **for** A
using *cardinal_rel_Card_rel_eqpoll_rel_iff* **by** *auto*
with ‹$n{\in}\omega$›

So we prove the result by induction on this n and generalizing G, since the argument requires changing the family in order to apply the inductive hypothesis.

have $\exists\, D[M].\ D \subseteq G \wedge delta\_system(D) \wedge D \approx^M \aleph_1{}^M$
proof (*induct arbitrary*:*G*)

```
    case 0 — This case is impossible
    then
    have G ⊆ {0}
      using cardinal_rel_0_iff_0 by (blast dest:transM)
    with ‹G ≈^M ℵ_1^M› ‹M(G)›
    show ?case
      using nat_lt_Aleph_rel1 subset_imp_le_cardinal_rel[of G {0}]
        lt_trans2 cardinal_rel_Card_rel_eqpoll_rel_iff by auto
  next
    case (succ n)
    have ∀ a∈G. Finite(a)
    proof
      fix a
      assume a ∈ G
      moreover
      note ‹M(G)› ‹n∈ω›
      moreover from calculation
      have M(a) by (auto dest: transM)
      moreover from succ and ‹a∈G›
      have |a|^M = succ(n) by simp
      ultimately
      show Finite(a)
        using Finite_cardinal_rel_iff' nat_into_Finite[of succ(n)]
        by fastforce
    qed
    show ∃ D[M]. D ⊆ G ∧ delta_system(D) ∧ D ≈^M ℵ_1^M
    proof (cases ∃ p[M]. {A∈G . p ∈ A} ≈^M ℵ_1^M)
     case True — the positive case, uncountably many sets with a common element
      then
      obtain p where {A∈G . p ∈ A} ≈^M ℵ_1^M M(p) by blast
      moreover
     note 1=‹M(G)› ‹M(G) ⟹ M(p) ⟹ M({A∈G . p ∈ A})› singleton_closed[OF
‹M(p)›]
      moreover from this
      have M({x - {p} . x ∈ {x ∈ G . p ∈ x}})
        using RepFun_closed[OF lam_replacement_Diff'[THEN
            lam_replacement_imp_strong_replacement]]
          Diff_closed[OF transM[OF _ 1(2)]] by auto
      moreover from 1
      have M(converse(λx∈{x ∈ G . p ∈ x}. x - {p})) (is M(converse(?h)))
        using converse_closed[of ?h] lam_closed[OF diff_Pair_replacement]
          Diff_closed[OF transM[OF _ 1(2)]]
        by auto
      moreover from calculation
      have {A-{p} . A∈{X∈G. p∈X}} ≈^M ℵ_1^M (is ?F ≈^M _)
        using Diff_bij_rel[of {A∈G . p ∈ A} {p}, THEN
          comp_bij_rel[OF bij_rel_converse_bij_rel, where C=ℵ_1^M,
            THEN bij_rel_imp_eqpoll_rel, of _ _ ?F]]
        unfolding eqpoll_rel_def
```

by (*auto simp del*:*mem_bij_abs*)

Now using the hypothesis of the successor case,

moreover from ‹$\bigwedge A.\ A{\in}G \Longrightarrow |A|^M{=}succ(n)$› ‹$\forall\, a{\in}G.\ Finite(a)$›
and *this* ‹$M(G)$›
have $p{\in}A \Longrightarrow A{\in}G \Longrightarrow |A - \{p\}|^M = n$ **for** A
using *Finite_imp_succ_cardinal_rel_Diff*[*of* _ p] **by** (*force dest*: *transM*)
moreover
have $\forall\, a{\in}?F.\ Finite(a)$
proof (*clarsimp*)
fix A
assume $p{\in}A\ A{\in}G$
with ‹$\bigwedge A.\ p \in A \Longrightarrow A \in G \Longrightarrow |A - \{p\}|^M = n$› **and** ‹$n{\in}\omega$› ‹$M(G)$›
have $Finite(|A - \{p\}|^M)$
using *nat_into_Finite* **by** *simp*
moreover from ‹$p{\in}A$› ‹$A{\in}G$› ‹$M(G)$›
have $M(A - \{p\})$ **by** (*auto dest*: *transM*)
ultimately
show $Finite(A - \{p\})$
using *Finite_cardinal_rel_iff'* **by** *simp*
qed
moreover

we may apply the inductive hypothesis to the new family $\{A - \{p\}\ .\ A \in \{X \in G\ .\ p \in X\}\}$:

note ‹$(\bigwedge A.\ A \in ?F \Longrightarrow |A|^M = n) \Longrightarrow ?F \approx^M \aleph_1{}^M \Longrightarrow M(?F) \Longrightarrow$
$\exists\, D[M].\ D \subseteq ?F \wedge delta\_system(D) \wedge D \approx^M \aleph_1{}^M$›
moreover
note *1*=‹$M(G)$› ‹$M(G) \Longrightarrow M(p) \Longrightarrow M(\{A{\in}G\ .\ p \in A\})$› *singleton_closed*[*OF* ‹$M(p)$›]
moreover from *this*
have $M(\{x - \{p\}\ .\ x \in \{x \in G\ .\ p \in x\}\})$
using *RepFun_closed*[*OF lam_replacement_Diff'*[*THEN*
lam_replacement_imp_strong_replacement]]
Diff_closed[*OF transM*[*OF* _ *1*(*2*)]] **by** *auto*
ultimately
obtain D **where** $D{\subseteq}\{A{-}\{p\}\ .\ A{\in}\{X{\in}G.\ p{\in}X\}\}$ $delta\_system(D)$ $D \approx^M \aleph_1{}^M$ $M(D)$
by *auto*
moreover from *this*
obtain r **where** $\forall\, A{\in}D.\ \forall\, B{\in}D.\ A \neq B \longrightarrow A \cap B = r$
by *fastforce*
then
have $\forall\, A{\in}D.\forall\, B{\in}D.\ A{\cup}\{p\} \neq B{\cup}\{p\} \longrightarrow (A \cup \{p\}) \cap (B \cup \{p\}) = r{\cup}\{p\}$
by *blast*
ultimately
have $delta\_system(\{B \cup \{p\}\ .\ B{\in}D\})$ (**is** $delta\_system(?D)$)
by *fastforce*
moreover from ‹$M(D)$› ‹$M(p)$›

have $M(?D)$
using *RepFun_closed un_Pair_replacement transM*[*of* _ *D*] **by** *auto*
moreover from ‹$D \approx^M \aleph_1{}^M$› ‹$M(D)$›
have *Infinite*(D) $|D|^M = \aleph_1{}^M$
using *uncountable_rel_iff_subset_eqpoll_rel_Aleph_rel1*[*THEN iffD2*, *THEN uncountable_rel_imp_Infinite*, *of D*]
apply *auto*[*1*]
using *cardinal_rel_eqpoll_rel_iff*[*of* D $\aleph_1{}^M$] ‹$M(D)$› ‹$D \approx^M \aleph_1{}^M$›
by *simp*
moreover from *this* ‹$M(?D)$› ‹$M(D)$› ‹$M(p)$›
have $?D \approx^M \aleph_1{}^M$
using *cardinal_rel_map_Un*[*of* D $\{p\}$] *naturals_lt_nat*
cardinal_rel_eqpoll_rel_iff[*THEN iffD1*] **by** *simp*
moreover
note ‹$D \subseteq \{A\text{-}\{p\} \,.\, A \in \{X \in G.\ p \in X\}\}$›
have $?D \subseteq G$
proof -
{
fix A
assume $A \in G$ $p \in A$
moreover from *this*
have $A = A - \{p\} \cup \{p\}$
by *blast*
ultimately
have $A\text{-}\{p\} \cup \{p\} \in G$
by *auto*
}
with ‹$D \subseteq \{A\text{-}\{p\} \,.\, A \in \{X \in G.\ p \in X\}\}$›
show *?thesis*
by *blast*
qed
moreover
note ‹$M(?D)$›
ultimately
show $\exists\, D[M].\ D \subseteq G \wedge$ *delta_system*(D) $\wedge\ D \approx^M \aleph_1{}^M$ **by** *auto*
next
case *False*
note ‹$\neg\ (\exists\, p[M].\ \{A \in G \,.\, p \in A\} \approx^M \aleph_1{}^M)$› — the other case
‹$M(G)$› ‹$\bigwedge p.\ M(G) \Longrightarrow M(p) \Longrightarrow M(\{A \in G \,.\, p \in A\})$›
moreover from ‹$G \approx^M \aleph_1{}^M$› **and** *this*
have $M(p) \Longrightarrow \{A \in G \,.\, p \in A\} \lesssim^M \aleph_1{}^M$ (**is** _ $\Longrightarrow$ $?G(p) \lesssim^M$ _) **for** p
by (*auto intro*!:*lepoll_rel_eq_trans*[*OF subset_imp_lepoll_rel*] *dest*:*transM*)
moreover from *calculation*
have $M(p) \Longrightarrow ?G(p) \prec^M \aleph_1{}^M$ **for** p
using ‹$M(G) \Longrightarrow M(p) \Longrightarrow M(\{A \in G \,.\, p \in A\})$›
unfolding *lesspoll_rel_def* **by** *simp*
moreover from *calculation*
have $M(p) \Longrightarrow ?G(p) \lesssim^M \omega$ **for** p
using *lesspoll_rel_Aleph_rel_plus_one*[*of 0*] *Aleph_rel_zero* **by** *auto*

moreover
have $\{A \in G \,.\, S \cap A \neq 0\} = (\bigcup p{\in}S.\ ?G(p))$ **for** S
by *auto*
moreover from *calculation*
have $M(S) \Longrightarrow i \in S \Longrightarrow M(\{x \in G \,.\, i \in x\})$ **for** $i\ S$
by (*auto dest*: *transM*)
moreover
have $M(S) \Longrightarrow$ *countable_rel*$(M,S) \Longrightarrow$ *countable_rel*$(M,\{A \in G \,.\, S \cap A \neq 0\})$ **for** S
proof -
from ‹$M(G)$›
interpret *M_replacement_lepoll M* $\lambda\_\ x.$ *Collect*$(G, (\in)(x))$
using *countable_lepoll_assms*(*2-*) *lam_replacement_inj_rel separation_in_rev*
lam_replacement_Collect[*OF _ _ countable_lepoll_assms*(*1*)] *mem_F_bound6*[*of _ G*]
by *unfold_locales*
(*auto dest*:*transM intro*:*lam_Least_assumption_general*[*of _ _ _ _ _ Union*])
fix S
assume $M(S)$
with ‹$M(G)$› ‹$\bigwedge i.\ M(S) \Longrightarrow i \in S \Longrightarrow M(\{x \in G \,.\, i \in x\})$›
interpret *M_cardinal_UN_lepoll _ ?G S*
using *lepoll_assumptions*
by *unfold_locales* (*auto dest*:*transM*)
assume *countable_rel*(M,S)
with ‹$M(S)$› *calculation*(*6*) *calculation*(*7*,*8*)[*of S*]
show *countable_rel*$(M,\{A \in G \,.\, S \cap A \neq 0\})$
using *InfCard_rel_nat Card_rel_nat*
le_Card_rel_iff[*THEN iffD2*, *THEN* [*3*] *leqpoll_rel_imp_cardinal_rel_UN_le*, *THEN* [*4*] *le_Card_rel_iff*[*THEN iffD1*], *of* ω] *j.UN_closed*
unfolding *countable_rel_def* **by** (*auto dest*: *transM*)
qed
define *Disjoint* **where** $Disjoint = \{<A,B> \in G{\times}G \,.\, B \cap A = 0\}$
have $Disjoint = \{x \in G{\times}G \,.\, \exists\ a\ b.\ x{=}<a,b> \wedge a{\cap}b{=}0\}$
unfolding *Disjoint_def* **by** *force*
with ‹$M(G)$›
have $M(Disjoint)$
using *disjoint_separation* **by** *simp*

For every countable_rel subfamily of G there is another some element disjoint from all of them:

have $\exists\, A{\in}G.\ \forall\, S{\in}X.\ <S,A>{\in}Disjoint$ **if** $|X|^M < \aleph_1{}^M\ X \subseteq G\ M(X)$ **for** X
proof -
note ‹$n{\in}\omega$› ‹$M(G)$›
moreover from *this* **and** ‹$\bigwedge A.\ A{\in}G \Longrightarrow |A|^M = succ(n)$›
have $|A|^M{=}\ succ(n)\ M(A)$ **if** $A{\in}G$ **for** A
using *that Finite_cardinal_rel_eq_cardinal*[*of A*] *Finite_cardinal_rel_iff'*[*of A*]
nat_into_Finite transM[*of A G*] **by** (*auto dest*:*transM*)

ultimately
have $A{\in}G \Longrightarrow \mathit{Finite}(A)$ **for** A
using *cardinal_rel_Card_rel_eqpoll_rel_iff*[*of succ*(n) A]
Finite_cardinal_rel_eq_cardinal[*of* A] *nat_into_Card_rel*[*of succ*(n)]
nat_into_M[*of* n] **unfolding** *Finite_def eqpoll_rel_def* **by** (*auto*)
with ‹$X{\subseteq}G$› ‹$M(X)$›
have $A{\in}X \Longrightarrow \mathit{countable\_rel}(M,A)$ **for** A
using *Finite_imp_countable_rel* **by** (*auto dest*: *transM*)
moreover from ‹$M(X)$›
have $M(\bigcup X)$ **by** *simp*
moreover
note ‹$|X|^M < \aleph_1{}^M$› ‹$M(X)$›
ultimately
have *countable_rel*(M,$\bigcup X$)
using *Card_rel_nat*[*THEN cardinal_rel_lt_csucc_rel_iff*, *of* X]
countable_rel_union_countable_rel[*of* X]
countable_rel_iff_cardinal_rel_le_nat[*of* X] *Aleph_rel_succ*
Aleph_rel_zero **by** *simp*
with ‹$M(\bigcup X)$› ‹$M(\_) \Longrightarrow \mathit{countable\_rel}(M,\_) \Longrightarrow \mathit{countable\_rel}(M,\{A \in G$. $\_ \cap A \neq 0\})$›
have *countable_rel*(M,$\{A \in G$. $(\bigcup X) \cap A \neq 0\}$) **by** *simp*
with ‹$G \approx^M \aleph_1{}^M$› ‹$M(G)$›
obtain B **where** $B{\in}G$ $B \notin \{A \in G$. $(\bigcup X) \cap A \neq 0\}$
using *nat_lt_Aleph_rel1 cardinal_rel_Card_rel_eqpoll_rel_iff*[*of* $\aleph_1{}^M$ G]
uncountable_rel_not_subset_countable_rel
[*of* $\{A \in G$. $(\bigcup X) \cap A \neq 0\}$ G]
uncountable_rel_iff_nat_lt_cardinal_rel[*of* G]
by *force*
then
have $\exists\, A{\in}G.\ \forall\, S{\in}X.\ A \cap S = 0$ **by** *auto*
with ‹$X{\subseteq}G$›
show $\exists\, A{\in}G.\ \forall\, S{\in}X.\ <S,A>{\in}\mathit{Disjoint}$ **unfolding** *Disjoint_def*
using *subsetD* **by** *simp*
qed
moreover from ‹$G \approx^M \aleph_1{}^M$› ‹$M(G)$›
obtain b **where** $b{\in}G$
using *uncountable_rel_iff_subset_eqpoll_rel_Aleph_rel1*
uncountable_rel_not_empty **by** *blast*
ultimately

Hence, the hypotheses to perform a bounded-cardinal selection are satisfied,

obtain S **where** $S{:}\aleph_1{}^M \to^M G$ $\alpha{\in}\aleph_1{}^M \Longrightarrow \beta{\in}\aleph_1{}^M \Longrightarrow \alpha{<}\beta \Longrightarrow <S`\alpha,\ S`\beta> \in \mathit{Disjoint}$
for α β
using *bounded_cardinal_rel_selection*[*of* $\aleph_1{}^M$ G *Disjoint*] ‹$M(\mathit{Disjoint})$›
by *force*
moreover from *this* ‹$n{\in}\omega$› ‹$M(G)$›
have *inM*:$M(S)$ $M(n)$ $\bigwedge x.\ x \in \aleph_1{}^M \Longrightarrow S\ `\ x \in G$ $\bigwedge x.\ x \in \aleph_1{}^M \Longrightarrow M(x)$

using *function_space_rel_char* **by** (*auto dest*: *transM*)
ultimately
have $\alpha \in \aleph_1{}^M \Longrightarrow \beta \in \aleph_1{}^M \Longrightarrow \alpha{\neq}\beta \Longrightarrow S`\alpha \cap S`\beta = 0$ **for** $\alpha\ \beta$
unfolding *Disjoint_def*
using *lt_neq_symmetry*[*of* $\aleph_1{}^M$ $\lambda\alpha\ \beta.\ S`\alpha \cap S`\beta = 0$] *Card_rel_is_Ord*
by *auto* (*blast*)

and a symmetry argument shows that obtained S is an injective $\aleph_1{}^M$-sequence of disjoint elements of G.

moreover from *this* **and** ‹$\bigwedge A.\ A{\in}G \Longrightarrow |A|^M = succ(n)$› *inM*
‹$S : \aleph_1{}^M \rightarrow^M G$› ‹$M(G)$›
have $S \in inj\_rel(M,\aleph_1{}^M,\ G)$
using *def_inj_rel*[*OF Aleph_rel_closed* ‹$M(G)$›, *of 1*]
proof (*clarsimp*)
fix $w\ x$
from *inM*
have $a \in \aleph_1{}^M \Longrightarrow b \in \aleph_1{}^M \Longrightarrow a \neq b \Longrightarrow S\ `\ a \neq S\ `\ b$ **for** $a\ b$
using ‹$\bigwedge A.\ A{\in}G \Longrightarrow |A|^M = succ(n)$›[*THEN* [*4*] *cardinal_rel_succ_not_0*[*THEN* [*4*]
Int_eq_zero_imp_not_eq[*OF calculation*, *of* $\aleph_1{}^M$ $\lambda x.\ x$],
of $\lambda\_.n$], *OF* _ _ _ _ *apply_closed*] **by** *auto*
moreover
assume $w \in \aleph_1{}^M\ x \in \aleph_1{}^M\ S\ `\ w = S\ `\ x$
ultimately
show $w = x$ **by** *blast*
qed
moreover from *this* ‹$M(G)$›
have $range(S) \approx^M \aleph_1{}^M$
using *inj_rel_bij_rel_range eqpoll_rel_sym* **unfolding** *eqpoll_rel_def*
by (*blast dest*: *transM*)
moreover
note ‹$M(G)$›
moreover from *calculation*
have $range(S) \subseteq G$
using *inj_rel_is_fun range_fun_subset_codomain*
by (*fastforce dest*: *transM*)
moreover
note ‹$M(S)$›
find_theorems $M(?S) \Longrightarrow strong\_replacement(M,\ \lambda x\ y.\ y = ?S\ `\ x)$
ultimately
show $\exists\, D[M].\ D \subseteq G \wedge delta\_system(D) \wedge D \approx^M \aleph_1{}^M$
using *inj_rel_is_fun ZF_Library.range_eq_image*[*of* $S\ \aleph_1{}^M\ G$]
image_function[*OF fun_is_function*, *OF inj_rel_is_fun*, *of* $S\ \aleph_1{}^M\ G$]
domain_of_fun[*OF inj_rel_is_fun*, *of* $S\ \aleph_1{}^M\ G$] *apply_replacement*[*of S*]
by (*rule_tac* $x{=}S`\aleph_1{}^M$ **in** *rexI*) (*auto dest*:*transM intro*!:*RepFun_closed*)

This finishes the successor case and hence the proof.

qed
qed

with ‹$G \subseteq F$›
show *?thesis* **by** *blast*
qed

lemma *delta_system_uncountable_rel*:
assumes $\forall A{\in}F.\ Finite(A)$ *uncountable_rel*(M,F) $M(F)$
shows $\exists D[M].\ D \subseteq F \wedge delta\_system(D) \wedge D \approx^M \aleph_1{}^M$
proof -
from *assms*
obtain S **where** $S \subseteq F$ $S \approx^M \aleph_1{}^M$ $M(S)$
using *uncountable_rel_iff_subset_eqpoll_rel_Aleph_rel1*[*of* F] **by** *auto*
moreover from ‹$\forall A{\in}F.\ Finite(A)$› **and** *this*
have $\forall A{\in}S.\ Finite(A)$ **by** *auto*
ultimately
show *?thesis* **using** *delta_system_Aleph_rel1*[*of* S]
by (*auto dest*:*transM*)
qed

end

end

48 Cohen forcing notions

theory *Cohen_Posets_Relative*
imports
Cohen_Posets— FIXME: This theory is going obsolete
Delta_System_Relative
begin

locale *M_cohen* = *M_delta* +
assumes
separation_domain_pair: $M(A) \Longrightarrow separation(M,\ \lambda p.\ \forall x{\in}A.\ x \in snd(p) \longleftrightarrow domain(x) = fst(p))$
and
separation_restrict_eq_dom_eq:
$M(A) \Longrightarrow M(B) \Longrightarrow \forall x[M].\ separation(M,\ \lambda dr.\ \exists r{\in}A\ .\ restrict(r,B) = x \wedge dr{=}domain(r))$
and
separation_restrict_eq_dom_eq_pair:
$M(A) \Longrightarrow M(B) \Longrightarrow M(D) \Longrightarrow$
$separation(M,\ \lambda p.\ \forall x{\in}D.\ x \in snd(p) \longleftrightarrow (\exists r{\in}A.\ restrict(r,\ B) = fst(p) \wedge x = domain(r)))$
and
countable_lepoll_assms2:
$M(A') \Longrightarrow M(A) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow separation(M,\ \lambda y.\ \exists x{\in}A'.\ y = \langle x,\ \mu\ i.\ x \in if\_range\_F\_else\_F(\lambda a.\ \{p \in A\ .\ domain(p) = a\},\ b,\ f,\ i)\rangle)$
and
countable_lepoll_assms3:

$M(A) \Longrightarrow M(f) \Longrightarrow M(b) \Longrightarrow M(D) \Longrightarrow M(r') \Longrightarrow M(A') \Longrightarrow$
separation(M, $\lambda y.\ \exists\, x{\in}A'.\ y = \langle x,\ \mu\ i.\ x \in$ *if_range_F_else_F*(*drSR_Y*(r', D, A), b, f, i)$\rangle$))
and
domain_mem_separation: $M(A) \Longrightarrow$ *separation*(M, $\lambda x\ .$ *domain*(x)$\in A$)
and
domain_eq_separation: $M(p) \Longrightarrow$ *separation*(M, $\lambda x\ .$ *domain*(x) $= p$)
and
restrict_eq_separation: $M(r) \Longrightarrow M(p) \Longrightarrow$ *separation*(M, $\lambda x\ .$ *restrict*(x,r) $= p$)

context *M_cardinal_library*
begin

lemma *lesspoll_nat_imp_lesspoll_rel*:
assumes $A \prec \omega$ $M(A)$
shows $A \prec^M \omega$
proof -
note *assms*
moreover from *this*
obtain f n **where** $f{\in}bij^M(A,n)$ $n{\in}\omega$ $A \approx^M n$
using *lesspoll_nat_is_Finite* **using** *Finite_rel_def*[*of M A*] **by** *auto*
moreover from *calculation*
have $A \lesssim^M \omega$
using *lesspoll_nat_is_Finite Infinite_imp_nats_lepoll_rel*[*of* ω n]
nat_not_Finite eq_lepoll_rel_trans[*of A n* ω]
by *auto*
moreover from *calculation*
have $\neg\ g \in bij^M(A,\omega)$ **for** g
using *mem_bij_rel* **unfolding** *lesspoll_def* **by** *auto*
ultimately
show *?thesis* **unfolding** *lesspoll_rel_def eqpoll_rel_def bij_rel_is_inj_rel rex_def*
by *auto*
qed

lemma *Finite_imp_lesspoll_rel_nat*:
assumes *Finite*(A) $M(A)$
shows $A \prec^M \omega$
using *Finite_imp_lesspoll_nat assms lesspoll_nat_imp_lesspoll_rel* **by** *auto*

lemma *InfCard_rel_lesspoll_rel_Un*:
includes *Ord_dests*
assumes *InfCard_rel*(M,κ) $A \prec^M \kappa$ $B \prec^M \kappa$
and *types*: $M(\kappa)$ $M(A)$ $M(B)$
shows $A \cup B \prec^M \kappa$
proof -
from *assms*
have $|A|^M < \kappa$ $|B|^M < \kappa$
using *lesspoll_rel_cardinal_rel_lt InfCard_rel_is_Card_rel* **by** *auto*

```
  show ?thesis
  proof (cases Finite(A) ∧ Finite(B))
    case True
    with assms
    show ?thesis
      using lesspoll_rel_trans2[OF _ le_imp_lepoll_rel, of _ nat κ]
        Finite_imp_lesspoll_rel_nat[OF Finite_Un]
      unfolding InfCard_rel_def by simp
  next
    case False
    with types
    have InfCard_rel(M,max(|A|^M,|B|^M))
      using Infinite_InfCard_rel_cardinal_rel InfCard_rel_is_Card_rel
        le_trans[of nat] not_le_iff_lt[THEN iffD1, THEN leI, of |A|^M |B|^M]
      unfolding max_def InfCard_rel_def
      by (auto)
    with ‹M(A)› ‹M(B)›
    have |A ∪ B|^M ≤ max(|A|^M,|B|^M)
      using cardinal_rel_Un_le[of max(|A|^M,|B|^M) A B]
        not_le_iff_lt[THEN iffD1, THEN leI, of |A|^M |B|^M ]
      unfolding max_def by simp
    moreover from ‹|A|^M < κ› ‹|B|^M < κ›
    have max(|A|^M,|B|^M) < κ
      unfolding max_def by simp
    ultimately
    have |A ∪ B|^M <  κ
      using lt_trans1 by blast
    moreover
    note ‹InfCard_rel(M,κ)›
    moreover from calculation types
    have |A ∪ B|^M ≺^M κ
      using lt_Card_rel_imp_lesspoll_rel InfCard_rel_is_Card_rel by simp
    ultimately
    show ?thesis
      using cardinal_rel_eqpoll_rel eq_lesspoll_rel_trans[of A ∪ B |A ∪ B|^M κ]
        eqpoll_rel_sym types by simp
  qed
qed

end

locale M_add_reals = M_cohen + add_reals
begin

lemmas zero_lesspoll_rel_kappa = zero_lesspoll_rel[OF zero_lt_kappa]

end
```

declare (**in** *M_trivial*) *compat_in_abs*[*absolut*]

definition
antichain_rel :: [*i*⇒*o*,*i*,*i*,*i*] ⇒ *o* (‹*antichain–'*(_,_,_')›) **where**
antichain_rel(*M*,*P*,*leq*,*A*) ≡ *subset*(*M*,*A*,*P*) ∧ (∀ *p*[*M*]. ∀ *q*[*M*].
p∈*A* ⟶ *q*∈*A* ⟶ *p* ≠ *q*⟶ ¬ *is_compat_in*(*M*,*P*,*leq*,*p*,*q*))

abbreviation
antichain_r_set :: [*i*,*i*,*i*,*i*] ⇒ *o* (‹*antichain–'*(_,_,_')›) **where**
antichain^M(*P*,*leq*,*A*) ≡ *antichain_rel*(##*M*,*P*,*leq*,*A*)

context *M_trivial*
begin

lemma *antichain_abs* [*absolut*]:
⟦ *M*(*A*); *M*(*P*); *M*(*leq*) ⟧ ⟹ *antichain*^{M}(*P*,*leq*,*A*) ⟷ *antichain*(*P*,*leq*,*A*)
unfolding *antichain_rel_def antichain_def* **by** (*simp add*:*absolut*)

end

definition
ccc_rel :: [*i*⇒*o*,*i*,*i*] ⇒ *o* (‹*ccc–'*(_,_')›) **where**
ccc_rel(*M*,*P*,*leq*) ≡ ∀ *A*[*M*]. *antichain_rel*(*M*,*P*,*leq*,*A*) ⟶
(∀ *κ*[*M*]. *is_cardinal*(*M*,*A*,*κ*) ⟶ (∃ *om*[*M*]. *omega*(*M*,*om*) ∧ *le_rel*(*M*,*κ*,*om*)))

abbreviation
ccc_r_set :: [*i*,*i*,*i*]⇒*o* (‹*ccc–'*(_,_')›) **where**
ccc_r_set(*M*) ≡ *ccc_rel*(##*M*)

context *M_cardinals*
begin

lemma *def_ccc_rel*:
shows
ccc^M(*P*,*leq*) ⟷ (∀ *A*[*M*]. *antichain*^{M}(*P*,*leq*,*A*) ⟶ |*A*|^M ≤ *ω*)
using *is_cardinal_iff*
unfolding *ccc_rel_def* **by** (*simp add*:*absolut*)

end

context *M_add_reals*
begin

lemma *lam_replacement_drSR_Y*: $M(A) \Longrightarrow M(D) \Longrightarrow M(r') \Longrightarrow$ *lam_replacement*(M, *drSR_Y*(r',D,A))
using *lam_replacement_drSR_Y separation_restrict_eq_dom_eq separation_restrict_eq_dom_eq_pair*
by *simp*

lemma (**in** *M_trans*) *mem_F_bound3*:
fixes F A
defines $F \equiv dC\_F$
shows $x{\in}F(A,c) \Longrightarrow c \in (range(f) \cup \{domain(x).\ x{\in}A\})$
using *apply_0* **unfolding** *F_def*
by (*cases* $M(c)$, *auto simp*:*F_def drSR_Y_def dC_F_def*)

lemma *ccc_rel_Fn_nat*:
notes *Sep_and_Replace* [*simp*]— FIXME with all *SepReplace* instances
assumes $M(I)$
shows $ccc^M(Fn(nat,I,2),\ Fnle(nat,I,2))$
proof -
from *assms*
have $M(Fn(nat,I,2))$ **using** *Fn_nat_eq_FiniteFun* **by** *simp*
{
fix A
assume $\neg\ |A|^M \leq nat\ M(A)$
then
have $M(\{domain(p)\ .\ p \in A\})$
using *RepFun_closed domain_replacement_simp transM*[*OF* _ ‹$M(A)$›]
by *auto*
assume $A \subseteq Fn(nat,\ I,\ 2)$
moreover from *this*
have *countable_rel*(M,{$p{\in}A.\ domain(p) = d$}) **if** $M(d)$ **for** d
proof (*cases* $d \prec^M nat \wedge\ d \subseteq I$)
case *True*
with ‹$A \subseteq Fn(nat,\ I,\ 2)$› ‹$M(A)$›
have $\{p \in A\ .\ domain(p) = d\} \subseteq d \rightarrow^M 2$
using *domain_of_fun function_space_rel_char*[*of* _ *2*]
by (*auto*,*subgoal_tac* $M(domain(x))$) (*force dest*: *transM*)+— slow
moreover from *True* ‹$M(d)$›
have $Finite(d \rightarrow^M 2)$
using *Finite_Pi*[*THEN* [*2*] *subset_Finite*, *of* _ d λ_. *2*]
lesspoll_rel_nat_is_Finite_rel function_space_rel_char[*of* _ *2*] **by** *auto*
moreover from ‹$M(d)$›
have $M(d \rightarrow^M 2)$ **by** *simp*
moreover from ‹$M(A)$›
have $M(\{p \in A\ .\ domain(p) = d\})$
using *separation_closed domain_eq_separation*[*OF* ‹$M(d)$›] **by** *simp*
ultimately
show *?thesis* **using** *subset_Finite*[*of* _ $d{\rightarrow}^M 2$]
by (*auto intro*!:*Finite_imp_countable_rel*)
next
case *False*

```
    with ‹A ⊆ Fn(nat, I, 2)› ‹M(A)›
    have {p ∈ A . domain(p) = d} = 0
      using function_space_rel_char[of _ 2, OF transM, of _ A]
      apply (intro equalityI)
       apply (clarsimp)
      apply (rule lesspoll_nat_imp_lesspoll_rel[of domain(_), THEN [2] swap])
        apply (auto dest!: domain_of_fun[ of _ _ λ_. 2] dest:transM)
      done
    then
    show ?thesis using empty_lepoll_relI by auto
  qed
  moreover
  have uncountable_rel(M,{domain(p) . p ∈ A})
  proof
    have 1:M({domain(p) . p ∈ A'}) if M(A') for A'— Repeated above
      using that RepFun_closed domain_replacement_simp transM[OF _ that]
      by auto
    moreover
    have  M(x) ⟹ x ∈ A ∧ domain(x) = i ⟹ M(i) for A x i
      by auto
    moreover from calculation
    interpret M_replacement_lepoll M dC_F
   using lam_replacement_dC_F domain_eq_separation separation_domain_pair
       lam_replacement_inj_rel
      unfolding dC_F_def
      apply unfold_locales apply(simp_all)
    proof -
      fix A b f
      assume M(A) M(b) M(f)
      with calculation[of A]
      show lam_replacement(M, λx. μ i. x ∈ if_range_F_else_F(λd. {p ∈ A .
domain(p) = d}, b, f, i))
     using lam_replacement_dC_F separation_domain_pair domain_eq_separation
         mem_F_bound3 countable_lepoll_assms2
       unfolding dC_F_def
      by (rule_tac lam_Least_assumption_general[where U=λ_. {domain(x).
x∈A}])
         (auto)
    qed
    note ‹M({domain(p). p∈A})› ‹M(A)›
    moreover from this
    have x ∈ A ⟹ M({p ∈ A . domain(p) = domain(x)}) for x
      using separation_closed domain_eq_separation transM[OF _ ‹M(A)›] by
simp
    ultimately
    interpret M_cardinal_UN_lepoll _ dC_F(A) {domain(p). p∈A}
      using countable_lepoll_assms2
        lepoll_assumptions transM[of _ A]
        lepoll_assumptions1[OF ‹M(A)› ‹M({domain(p) . p ∈ A})›]
```

```
      domain_eq_separation
    lam_replacement_inj_rel lam_replacement_dC_F[OF _ _ separation_domain_pair]
     unfolding dC_F_def
     apply (unfold_locales)
      apply(simp del:if_range_F_else_F_def,simp)
   apply (rule_tac lam_Least_assumption_general[where U=λ_. {domain(x).
x∈A}], auto)
     done
   from ‹A ⊆ Fn(nat, I, 2)›
   have x:(⋃ d∈{domain(p) . p ∈ A}. {p∈A. domain(p) = d}) = A
     by auto
   moreover
   assume countable_rel(M,{domain(p) . p ∈ A})
   moreover
   note ‹⋀d. M(d) ⟹ countable_rel(M,{p∈A. domain(p) = d})›
   moreover from ‹M(A)›
   have p∈A ⟹ M(domain(p)) for p by (auto dest: transM)
   ultimately
   have countable_rel(M,A)
     using countable_rel_imp_countable_rel_UN
     unfolding dC_F_def
     by auto
   with ‹¬ |A|^M ≤ nat› ‹M(A)›
   show False
     using countable_rel_iff_cardinal_rel_le_nat by simp
 qed
 moreover from ‹A ⊆ Fn(nat, I, 2)› ‹M(A)›
 have p ∈ A ⟹ Finite(domain(p)) for p
   using lesspoll_rel_nat_is_Finite_rel[of domain(p)]
     lesspoll_nat_imp_lesspoll_rel[of domain(p)]
     domain_of_fun[of p _ λ_. 2] by (auto dest:transM)
 moreover
 note ‹M({domain(p). p∈A})›
 ultimately
obtain D where delta_system(D) D ⊆ {domain(p) . p ∈ A} D ≈^M ℵ_1^M M(D)
   using delta_system_uncountable_rel[of {domain(p) . p ∈ A}] by auto
 then
 have delta:∀ d1∈D. ∀ d2∈D. d1 ≠ d2 ⟶ d1 ∩ d2 = ⋂ D
   using delta_system_root_eq_Inter
   by simp
 moreover from ‹D ≈^M ℵ_1^M› ‹M(D)›
 have uncountable_rel(M,D)
   using uncountable_rel_iff_subset_eqpoll_rel_Aleph_rel1 by auto
 moreover from this and ‹D ⊆ {domain(p) . p ∈ A}›
 obtain p1 where p1 ∈ A domain(p1) ∈ D
   using uncountable_rel_not_empty[of D] by blast
 moreover from this and ‹p1 ∈ A ⟹ Finite(domain(p1))›
 have Finite(domain(p1)) using Finite_domain by simp
 moreover
```

define r **where** $r \equiv \bigcap D$
moreover from ‹$M(D)$›
have $M(r)$ $M(r{\times}2)$ **unfolding** *r_def* **by** *simp_all*
ultimately
have *Finite*(r) **using** *subset_Finite*[*of r domain*(*p1*)] **by** *auto*
have *countable_rel*(M,{*restrict*(p,r) . $p{\in}A$})
proof -
note ‹M(*Fn*(*nat*, *I*, *2*))› ‹$M(r)$›
moreover from *this*
have $f{\in}$*Fn*(*nat*, *I*, *2*) $\Longrightarrow$ M(*restrict*(f,r)) **for** f
by (*blast dest*: *transM*)
ultimately
have $f{\in}$*Fn*(*nat*, *I*, *2*) $\Longrightarrow$ *restrict*(f,r) $\in$ *Pow_rel*(M,$r \times 2$) **for** f
using *restrict_subset_Sigma*[*of f _* λ*_. 2 r*] *Pow_rel_char*
by (*auto dest*!:*FnD simp*: *Pi_def*) (*auto dest*:*transM*)
with ‹$A \subseteq$ *Fn*(*nat*, *I*, *2*)›
have {*restrict*(f,r) . $f \in A$ } $\subseteq$ *Pow_rel*(M,$r \times 2$)
by *fast*
moreover from ‹$M(A)$› ‹$M(r)$›
have M({*restrict*(f,r) . $f \in A$ })
using *RepFun_closed restrict_strong_replacement transM*[*OF _* ‹$M(A)$›]
by *auto*
moreover
note ‹*Finite*(r)› ‹$M(r)$›
ultimately
show *?thesis*
using *Finite_Sigma*[*THEN Finite_Pow_rel*, *of r* λ*_. 2*]
by (*intro Finite_imp_countable_rel*) (*auto intro*:*subset_Finite*)
qed
moreover from ‹$M(A)$› ‹$M(D)$›
have M({$p{\in}A$. *domain*(p) $\in D$})
using *domain_mem_separation* **by** *simp*
have *uncountable_rel*(M,{$p{\in}A$. *domain*(p) $\in D$}) (**is** *uncountable_rel*(M,*?X*))
proof
from ‹$D \subseteq$ {*domain*(p) . $p \in A$}›
have ($\lambda p{\in}$*?X*. *domain*(p)) $\in$ *surj*(*?X*, D)
using *lam_type* **unfolding** *surj_def* **by** *auto*
moreover from ‹$M(A)$› ‹M(*?X*)›
have $M(\lambda p{\in}$*?X*. *domain*(p))
using *lam_closed*[*OF domain_replacement* ‹M(*?X*)›] *transM*[*OF _* ‹M(*?X*)›]
by *simp*
moreover
note ‹M(*?X*)› ‹$M(D)$›
moreover from *calculation*
have *surjection*:($\lambda p{\in}$*?X*. *domain*(p)) $\in$ $surj^M$(*?X*, D)
using *surj_rel_char* **by** *simp*
moreover
assume *countable_rel*(M,*?X*)
moreover

```
    note ‹uncountable_rel(M,D)›
    ultimately
    show False
      using surj_rel_countable_rel[OF _ surjection] by auto
  qed
  moreover
   have D = (⋃f∈Pow_rel(M,r×2) . {domain(p) .. p∈A, restrict(p,r) = f ∧
domain(p) ∈ D})
  proof -
    {
      fix z
      assume z ∈ D
      with ‹M(D)›
      have ‹M(z)› by (auto dest:transM)
      from ‹z∈D› ‹D ⊆ _› ‹M(A)›
      obtain p  where domain(p) = z p ∈ A M(p)
        by (auto dest:transM)
      moreover from ‹A ⊆ Fn(nat, I, 2)› ‹M(z)› and this
      have p : z →^M 2
        using domain_of_fun function_space_rel_char by (auto dest!:FnD)
      moreover from this ‹M(z)›
      have p : z → 2
        using domain_of_fun function_space_rel_char by (auto)
      moreover from this ‹M(r)›
      have restrict(p,r) ⊆ r × 2
        using function_restrictI[of p r] fun_is_function[of p z λ_. 2]
          restrict_subset_Sigma[of p z λ_. 2 r] function_space_rel_char
        by (auto simp:Pi_def)
      moreover from ‹M(p)› ‹M(r)›
      have M(restrict(p,r)) by simp
      moreover
      note ‹M(r)›
      ultimately
      have ∃p∈A.  restrict(p,r) ∈ Pow_rel(M,r×2) ∧ domain(p) = z
        using Pow_rel_char by auto
    }
    then
    show ?thesis
      by (intro equalityI) (force)+
  qed
  from ‹M(D)›‹M(r)›
 have M({domain(p) .. p∈A, restrict(p,r) = f ∧ domain(p) ∈ D}) (is M(?Y(A,f)))
    if M(f) M(A) for f A
    using that RepFun_closed domain_replacement_simp
      separation_conj[OF restrict_eq_separation[OF ‹M(r)› ‹M(f)›]
                         domain_mem_separation[OF ‹M(D)›]]
      transM[OF _ ‹M(D)›]
    by simp
  obtain f where uncountable_rel(M,?Y(A,f)) M(f)
```

```
    proof -
      note ‹M(r)›
      moreover from this
      have M(Pow^M(r × 2)) by simp
      moreover
      note ‹M(A)› ‹⋀f A. M(f) ⟹ M(A) ⟹ M(?Y(A,f))› ‹M(D)›
      moreover from calculation
      interpret M_replacement_lepoll M drSR_Y(r,D)
     using countable_lepoll_assms3 lam_replacement_inj_rel lam_replacement_drSR_Y
        apply (unfold_locales, simp_all)
          apply (rule_tac [2] lam_Least_assumption_drSR_Y)
              apply(simp_all add:drSR_Y_def)
      proof -
        fix i A x
        assume ∃xa∈A. restrict(xa, r) = i ∧ domain(xa) ∈ D ∧ x = domain(xa)
M(A) M(r)
        moreover from this
        obtain xa where xa∈A restrict(xa, r) = i by blast
        ultimately
        show M(i) by (auto dest:transM)
      qed
      from calculation
      have x ∈ Pow^M(r × 2) ⟹ M(drSR_Y(r, D, A, x)) for x
        unfolding drSR_Y_def by (auto dest:transM)
      ultimately
      interpret M_cardinal_UN_lepoll _ ?Y(A) Pow_rel(M,r×2)
     using countable_lepoll_assms3 lepoll_assumptions[where S=Pow_rel(M,r×2),
unfolded lepoll_assumptions_defs]
          lam_replacement_drSR_Y
        unfolding drSR_Y_def
            apply unfold_locales apply (simp_all add:lam_replacement_inj_rel
del:Sep_and_Replace if_range_F_else_F_def)
        unfolding drSR_Y_def[symmetric]
          apply (rule_tac lam_Least_assumption_drSR_Y)
        by (simp_all add: del:Sep_and_Replace if_range_F_else_F_def)
          ((fastforce dest:transM[OF _ ‹M(A)›])+)[2]
      {
        from ‹Finite(r)› ‹M(r)›
        have countable_rel(M,Pow_rel(M,r×2))
          using Finite_Sigma[THEN Finite_Pow_rel] Finite_imp_countable_rel
          by simp
        moreover
        assume M(f) ⟹ countable_rel(M,?Y(A,f)) for f
        moreover
        note ‹D = (⋃f∈Pow_rel(M,r×2) .?Y(A,f))› ‹M(r)›
        moreover
        note ‹uncountable_rel(M,D)›
        ultimately
        have False
```

```
      using countable_rel_imp_countable_rel_UN by (auto dest: transM)
    }
    with that
    show ?thesis by auto
  qed
  moreover from this ‹M(A)› and ‹M(f) ⟹ M(A) ⟹ M(?Y(A,f))›
  have M(?Y(A,f)) by blast
  ultimately
  obtain j where j ∈ inj_rel(M,nat, ?Y(A,f)) M(j)
    using uncountable_rel_iff_nat_lt_cardinal_rel[THEN iffD1, THEN leI,
        THEN cardinal_rel_le_imp_lepoll_rel, THEN lepoll_relD]
    by auto
  with ‹M(?Y(A,f))›
  have j'0 ≠ j'1 j'0 ∈ ?Y(A,f) j'1 ∈ ?Y(A,f)
    using inj_is_fun[THEN apply_type, of j nat ?Y(A,f)]
      inj_rel_char
    unfolding inj_def by auto
  then
  obtain p q where domain(p) ≠ domain(q) p ∈ A q ∈ A
    domain(p) ∈ D domain(q) ∈ D
    restrict(p,r) = restrict(q,r) by auto
  moreover from this and delta
  have domain(p) ∩ domain(q) = r unfolding r_def by simp
  moreover
  note ‹A ⊆ Fn(nat, I, 2)›
  moreover from calculation
  have p ∪ q ∈ Fn(nat, I, 2)
    using restrict_eq_imp_compat InfCard_nat by blast
  ultimately
  have ∃ p∈A. ∃ q∈A. p ≠ q ∧ compat_in(Fn(nat, I, 2), Fnle(nat, I, 2), p, q)
    unfolding compat_in_def
    by (rule_tac bexI[of _ p], rule_tac bexI[of _ q]) blast
}
moreover from assms
have M(Fnle(ω,I,2)) by simp
moreover note ‹M(Fn(ω,I,2))›
ultimately
show ?thesis using def_ccc_rel by (auto simp:absolut antichain_def) fastforce
qed

end

end
theory ZF_Trans_Interpretations
  imports
    Cohen_Posets_Relative
    Forcing_Main
    Separation_Instances
    Replacement_Instances
```

begin

lemmas (**in** *M_ZF_trans*) *separation_instances* =
separation_well_ord
separation_obase_equals separation_is_obase
separation_PiP_rel separation_surjP_rel
separation_id_body separation_rvimage_body
separation_radd_body separation_rmult_body
separation_ord_iso_body

lemma (**in** *M_ZF_trans*) *lam_replacement_inj_rel*:
shows
lam_replacement(*##M*, *λp. inj*$^{\#\#M}$(*fst*(*p*),*snd*(*p*)))
using *lam_replacement_iff_lam_closed*[*THEN iffD2*,*of λp. inj*^{M}(*fst*(*p*),*snd*(*p*))]
LambdaPair_in_M[**where** *φ*=*is_inj_fm*(*0*,*1*,*2*) **and** *is_f*=*is_inj*(*##M*) **and**
env=[],*OF*
is_inj_fm_type _ is_inj_iff_sats[*symmetric*] *inj_rel_iff*,*simplified*]
arity_is_inj_fm[*of 0 1 2*] *ord_simp_union transitivity fst_snd_closed*
inj_rel_closed zero_in_M
by *simp*

definition *is_order_body*
where *is_order_body*(*M*,*X*,*x*,*z*) ≡ ∃ *A*[*M*]. *cartprod*(*M*,*X*,*X*,*A*) ∧ *subset*(*M*,*x*,*A*)
∧ *M*(*z*) ∧ *M*(*x*) ∧
is_well_ord(*M*,*X*, *x*) ∧ *is_ordertype*(*M*,*X*, *x*,*z*)

synthesize *is_order_body* **from_definition assuming** *nonempty*
arity_theorem for *is_transitive_fm*
arity_theorem for *is_linear_fm*
arity_theorem for *is_wellfounded_on_fm*
arity_theorem for *is_well_ord_fm*

arity_theorem for *pred_set_fm*
arity_theorem for *image_fm*
definition *omap_wfrec_body* **where**
omap_wfrec_body(*A*,*r*) ≡ (·∃·*image_fm*(*2*, *0*, *1*) ∧
pred_set_fm
(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*A*)))))))), *3*,
succ(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*r*)))))))), *0*) ··)

lemma *type_omap_wfrec_body_fm* :*A*∈*nat* ⟹ *r*∈*nat* ⟹ *omap_wfrec_body*(*A*,*r*)∈*formula*
unfolding *omap_wfrec_body_def* **by** *simp*

lemma *arity_aux* : *A*∈*nat* ⟹ *r*∈*nat* ⟹ *arity*(*omap_wfrec_body*(*A*,*r*)) = (*9*#+*A*)
∪ (*9*#+*r*)
unfolding *omap_wfrec_body_def*
using *arity_image_fm arity_pred_set_fm pred_Un_distrib union_abs2*[*of 3*]
union_abs1

by (*simp,auto simp add:Un_assoc*[*symmetric*] *union_abs1*)

lemma *arity_omap_wfrec*: *A∈nat ⟹ r∈nat ⟹*
arity(*is_wfrec_fm*(*omap_wfrec_body*(*A*,*r*),*succ*(*succ*(*succ*(*r*))), *1*, *0*)) =
(*4*#+*A*) ∪ (*4*#+*r*)
using *Arities.arity_is_wfrec_fm*[*OF _ _ _ _ _ arity_aux*,*of A r 3*#+*r 1 0*]
pred_Un_distrib
union_abs1 union_abs2 type_omap_wfrec_body_fm
by *auto*

lemma *arity_isordermap*: *A∈nat ⟹ r∈nat ⟹d∈nat⟹*
arity(*is_ordermap_fm*(*A*,*r*,*d*)) = *succ*(*d*) ∪ (*succ*(*A*) ∪ *succ*(*r*))
unfolding *is_ordermap_fm_def*
using *arity_lambda_fm*[**where** *i*=(*4*#+*A*) ∪ (*4*#+*r*),*OF _ _ _ _ arity_omap_wfrec*,
unfolded omap_wfrec_body_def] *pred_Un_distrib union_abs1*
by *auto*

lemma *arity_is_ordertype*: *A∈nat ⟹ r∈nat ⟹d∈nat⟹*
arity(*is_ordertype_fm*(*A*,*r*,*d*)) = *succ*(*d*) ∪ (*succ*(*A*) ∪ *succ*(*r*))
unfolding *is_ordertype_fm_def*
using *arity_isordermap arity_image_fm pred_Un_distrib*
by *auto*

arity_theorem for *is_order_body_fm*

lemma *arity_is_order_body*: *arity*(*is_order_body_fm*(*2*,*0*,*1*)) = *3*
using *arity_is_order_body_fm arity_is_ordertype ord_simp_union*
by *simp*

lemma (**in** *M_ZF_trans*) *replacement_is_order_body*:
X∈M ⟹ strong_replacement(##*M*, *is_order_body*(##*M*,*X*))
apply(*rule_tac strong_replacement_cong*[
where *P*=λ *x f*. *M*,[*x*,*f*,*X*] ⊨ *is_order_body_fm*(*2*,*0*,*1*),*THEN iffD1*])
apply(*rule_tac is_order_body_iff_sats*[**where** *env*=[_,_,*X*],*symmetric*])
apply(*simp_all add:zero_in_M*)
apply(*rule_tac replacement_ax*[**where** *env*=[*X*],*simplified*])
apply(*simp_all add*: *arity_is_order_body*)
done

lemma (**in** *M_pre_cardinal_arith*) *is_order_body_abs* :
M(*X*) ⟹ *M*(*x*) ⟹ *M*(*z*) ⟹ *is_order_body*(*M*, *X*, *x*, *z*) ⟷
M(*z*) ∧ *M*(*x*) ∧ *x*∈*Pow_rel*(*M*,*X*×*X*) ∧ *well_ord*(*X*, *x*) ∧ *z* = *ordertype*(*X*, *x*)
using *well_ord_abs is_well_ord_iff_wellordered is_ordertype_iff' ordertype_rel_abs*
well_ord_is_linear subset_abs Pow_rel_char
unfolding *is_order_body_def*
by *simp*

definition *H_order_pred* **where**
H_order_pred(*A*,*r*) ≡ λ*x f* . *f* ‘‘ *Order.pred*(*A*, *x*, *r*)

relationalize *H_order_pred* *is_H_order_pred*

lemma (**in** *M_basic*) *H_order_pred_abs* :
M(*A*) ⟹ *M*(*r*) ⟹ *M*(*x*) ⟹ *M*(*f*) ⟹ *M*(*z*) ⟹
is_H_order_pred(*M*,*A*,*r*,*x*,*f*,*z*) ⟷ *z* = *H_order_pred*(*A*,*r*,*x*,*f*)
unfolding *is_H_order_pred_def H_order_pred_def*
by *simp*

synthesize *is_H_order_pred* **from_definition assuming** *nonempty*

definition *order_pred_wfrec_body* **where**
order_pred_wfrec_body(*M*,*A*,*r*,*z*,*x*) ≡ ∃ *y*[*M*].
pair(*M*, *x*, *y*, *z*) ∧
(∃ *f*[*M*].
(∀ *z*[*M*].
z ∈ *f* ⟷
(∃ *xa*[*M*].
∃ *y*[*M*].
∃ *xaa*[*M*].
∃ *sx*[*M*].
∃ *r_sx*[*M*].
∃ *f_r_sx*[*M*].
pair(*M*, *xa*, *y*, *z*) ∧
pair(*M*, *xa*, *x*, *xaa*) ∧
upair(*M*, *xa*, *xa*, *sx*) ∧
pre_image(*M*, *r*, *sx*, *r_sx*) ∧
restriction(*M*, *f*, *r_sx*, *f_r_sx*) ∧
xaa ∈ *r* ∧ (∃ *a*[*M*]. *image*(*M*, *f_r_sx*, *a*, *y*) ∧
pred_set(*M*, *A*, *xa*, *r*, *a*)))) ∧
(∃ *a*[*M*]. *image*(*M*, *f*, *a*, *y*) ∧ *pred_set*(*M*, *A*, *x*, *r*, *a*)))

synthesize *order_pred_wfrec_body* **from_definition**
arity_theorem for *order_pred_wfrec_body_fm*

lemma (**in** *M_ZF_trans*) *wfrec_replacement_order_pred*:
A∈*M* ⟹ *r*∈*M* ⟹ *wfrec_replacement*(##*M*, λ*x g z*. *is_H_order_pred*(##*M*,*A*,*r*,*x*,*g*,*z*)
, *r*)
unfolding *wfrec_replacement_def is_wfrec_def M_is_recfun_def is_H_order_pred_def*
apply(*rule_tac strong_replacement_cong*[
where *P*=λ *x f*. *M*,[*x*,*f*,*r*,*A*] ⊨ *order_pred_wfrec_body_fm*(*3*,*2*,*1*,*0*),*THEN*
iffD1])
apply(*subst order_pred_wfrec_body_def*[*symmetric*])
apply(*rule_tac order_pred_wfrec_body_iff_sats*[**where** *env*=[_,_,*r*,*A*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)

apply(*rule_tac replacement_ax*[**where** *env*=[*r*,*A*],*simplified*])
apply(*simp_all add*: *arity_order_pred_wfrec_body_fm ord_simp_union*)
done

lemma (**in** *M_ZF_trans*) *wfrec_replacement_order_pred'*:
A∈*M* ⟹ *r*∈*M* ⟹ *wfrec_replacement*(##*M*, λ*x g z*. *z* = *H_order_pred*(*A*,*r*,*x*,*g*)
, *r*)
using *wfrec_replacement_cong*[*OF H_order_pred_abs*[*of A r*,*rule_format*] *refl*,*THEN*
iffD1,
OF _ _ _ _ _ *wfrec_replacement_order_pred*[*of A r*]]
by *simp*

sublocale *M_ZF_trans* ⊆ *M_pre_cardinal_arith* ##*M*
using *separation_instances wfrec_replacement_order_pred'*[*unfolded H_order_pred_def*]
replacement_is_order_eq_map[*unfolded order_eq_map_def*] *banach_replacement*
by *unfold_locales simp_all*

lemma (**in** *M_ZF_trans*) *replacement_ordertype*:
X∈*M* ⟹ *strong_replacement*(##*M*, λ*x z*. *z* ∈ *M* ∧ *x* ∈ *M* ∧ *x* ∈ Pow^M(*X* ×
X) ∧ *well_ord*(*X*, *x*) ∧ *z* = *ordertype*(*X*, *x*))
using *strong_replacement_cong*[*THEN iffD1*,*OF* _ *replacement_is_order_body*,*simplified*]
is_order_body_abs
unfolding *is_order_body_def*
by *simp*

lemma *arity_is_jump_cardinal_body*: *arity*(*is_jump_cardinal_body'_fm*(*0*,*1*)) =
2
unfolding *is_jump_cardinal_body'_fm_def*
using *arity_is_ordertype arity_is_well_ord_fm arity_is_Pow_fm arity_cartprod_fm*
arity_Replace_fm[**where** *i*=*5*] *ord_simp_union*
by *simp*

lemma (**in** *M_ZF_trans*) *replacement_is_jump_cardinal_body*:
strong_replacement(##*M*, *is_jump_cardinal_body'*(##*M*))
apply(*rule_tac strong_replacement_cong*[
where *P*=λ *x f*. *M*,[*x*,*f*] ⊨ *is_jump_cardinal_body'_fm*(*0*,*1*),*THEN iffD1*])
apply(*rule_tac is_jump_cardinal_body'_iff_sats*[**where** *env*=[_,_],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac replacement_ax*[**where** *env*=[],*simplified*])
apply(*simp_all add*: *arity_is_jump_cardinal_body*)
done

lemma (**in** *M_pre_cardinal_arith*) *univalent_aux2*: *M*(*X*) ⟹ *univalent*(*M*,*Pow_rel*(*M*,*X*×*X*),
λ*r z*. *M*(*z*) ∧ *M*(*r*) ∧ *is_well_ord*(*M*, *X*, *r*) ∧ *is_ordertype*(*M*, *X*, *r*, *z*))
using *is_well_ord_iff_wellordered*
is_ordertype_iff[*of* _ *X*]
trans_on_subset[*OF well_ord_is_trans_on*]
well_ord_is_wf[*THEN wf_on_subset_A*] *mem_Pow_rel_abs*

unfolding *univalent_def*
by (*simp*)

lemma (**in** *M_pre_cardinal_arith*) *is_jump_cardinal_body_abs* :
M(*X*) ⟹ *M*(*c*) ⟹ *is_jump_cardinal_body'*(*M*, *X*, *c*) ⟷ *c* = *jump_cardinal_body'_rel*(*M*,*X*)
using *well_ord_abs is_well_ord_iff_wellordered is_ordertype_iff' ordertype_rel_abs*
well_ord_is_linear subset_abs Pow_rel_iff Replace_abs[*of Pow_rel*(*M*,*X*×*X*),*OF*
_ _
univalent_aux2]
unfolding *is_jump_cardinal_body'_def jump_cardinal_body'_rel_def*
by *simp*

lemma (**in** *M_ZF_trans*) *replacement_jump_cardinal_body*:
strong_replacement(##*M*, λ*x z*. *z* ∈ *M* ∧ *x* ∈ *M* ∧ *z* = *jump_cardinal_body*(##*M*,
x))
using *strong_replacement_cong*[*THEN iffD1*,*OF _ replacement_is_jump_cardinal_body*,*simplified*]

jump_cardinal_body_eq is_jump_cardinal_body_abs
by *simp*

sublocale *M_ZF_trans* ⊆ *M_pre_aleph* ##*M*
using *replacement_ordertype replacement_jump_cardinal_body HAleph_wfrec_repl*
lam_replacement_min[*unfolded lam_replacement_def*]
lam_replacement_imp_strong_replacement lam_replacement_vimage_sing_fun
lam_replacement_Sigfun[*OF lam_replacement_vimage_sing_fun*]
vimage_closed singleton_closed surj_imp_inj_replacement1
by *unfold_locales* (*simp_all add*: *transrec_replacement_def*
wfrec_replacement_def is_wfrec_def M_is_recfun_def flip:*setclass_iff*)

arity_theorem intermediate for *is_HAleph_fm*
lemma *arity_is_HAleph_fm*: *arity*(*is_HAleph_fm*(*2*, *1*, *0*)) = *3*
using *arity_fun_apply_fm*[*of 11 0 1*,*simplified*]
arity_is_HAleph_fm' arity_ordinal_fm arity_is_If_fm
arity_empty_fm arity_is_Limit_fm
arity_is_If_fm
arity_is_Limit_fm arity_empty_fm
arity_Replace_fm[**where** *i=12* **and** *v=10* **and** *n=3*]
pred_Un_distrib ord_simp_union
by *simp*

lemma *arity_is_Aleph*: *arity*(*is_Aleph_fm*(*0*, *1*)) = *2*
unfolding *is_Aleph_fm_def*
using *arity_transrec_fm*[*OF _ _ _ _ arity_is_HAleph_fm*] *ord_simp_union*
by *simp*

lemma (**in** *M_ZF_trans*) *replacement_is_aleph*:
strong_replacement(##*M*, λ*x y*. *Ord*(*x*) ∧ *is_Aleph*(##*M*,*x*,*y*))
apply(*rule_tac strong_replacement_cong*[
where *P*=λ *x y*. *M*,[*x*,*y*] ⊨ *And*(*ordinal_fm*(*0*),*is_Aleph_fm*(*0*,*1*)),*THEN*

iffD1])
apply (*auto simp add*: *ordinal_iff_sats*[**where** *env*=[_,_],*symmetric*])
apply(*rule_tac is_Aleph_iff_sats*[**where** *env*=[_,_],*THEN iffD2*],*simp_all add*:*zero_in_M*)
apply(*rule_tac is_Aleph_iff_sats*[**where** *env*=[_,_],*THEN iffD1*],*simp_all add*:*zero_in_M*)
apply(*rule_tac replacement_ax*[**where** *env*=[],*simplified*])
apply(*simp_all add*:*arity_is_Aleph ord_simp_union is_Aleph_fm_type*)
done

lemma (**in** *M_ZF_trans*) *replacement_aleph_rel*:
shows *strong_replacement*(##*M*, λ*x y*. *Ord*(*x*) ∧ *y* = ℵ$_x{}^M$)
using *strong_replacement_cong*[*THEN iffD2*,*OF _ replacement_is_aleph*,**where** *P1*=λ*x y* . *Ord*(*x*) ∧ *y*=*Aleph_rel*(##*M*,*x*)]
is_Aleph_iff
by *auto*

sublocale *M_ZF_trans* ⊆ *M_aleph* ##*M*
by (*unfold_locales*,*simp add*: *replacement_aleph_rel*)

sublocale *M_ZF_trans* ⊆ *M_FiniteFun* ##*M*
using *separation_supset_body separation_cons_like_rel*
replacement_range replacement_omega_funspace
by (*unfold_locales*,*simp_all*)

sublocale *M_ZFC_trans* ⊆ *M_AC* ##*M*
using *choice_ax* **by** (*unfold_locales*, *simp_all*)

sublocale *M_ZFC_trans* ⊆ *M_cardinal_AC* ##*M* **..**

definition *toplevel1_body* :: [*i*,*i*,*i*] ⇒ *o* **where**
toplevel1_body(*Q*,*x*) ≡ λ*a*. ∀ *s*∈*x*. ⟨*s*, *a*⟩ ∈ *Q*

relativize functional *toplevel1_body toplevel1_body_rel*
relationalize *toplevel1_body_rel is_toplevel1_body*

synthesize *is_toplevel1_body* **from_definition assuming** *nonempty*
arity_theorem for *is_toplevel1_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_toplevel1_body*:
(##*M*)(*A*) ⟹ (##*M*)(*B*) ⟹ *separation*(##*M*, *is_toplevel1_body*(##*M*,*A*,*B*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*,*A*,*B*] ⊨ *is_toplevel1_body_fm*(*1*,*2*,*0*),*THEN iffD1*])
apply(*rule_tac is_toplevel1_body_iff_sats*[**where** *env*=[_,*A*,*B*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*B*],*simplified*])
apply(*simp_all add*:*arity_is_toplevel1_body_fm ord_simp_union is_toplevel1_body_fm_type*)

done

lemma (**in** *M_ZF_trans*) *toplevel1_body_abs*:
assumes $(\#\#M)(A)$ $(\#\#M)(B)$ $(\#\#M)(x)$
shows *is_toplevel1_body*$(\#\#M,A,B,x) \longleftrightarrow$ *toplevel1_body*(A,B,x)
using *assms pair_in_M_iff apply_closed transM*
unfolding *toplevel1_body_def is_toplevel1_body_def*
by (*auto*)

lemma (**in** *M_ZF_trans*) *separation_toplevel1_body*:
$(\#\#M)(Q) \Longrightarrow (\#\#M)(x) \Longrightarrow$ *separation*$(\#\#M,\ \lambda a.\ \forall\, s{\in}x.\ \langle s,\ a\rangle \in Q)$
using *separation_is_toplevel1_body toplevel1_body_abs*
separation_cong[**where** *P*=*is_toplevel1_body*$(\#\#M,Q,x)$ **and** *M*=*##M*,*THEN iffD1*]
unfolding *toplevel1_body_def*
by *simp*

definition *toplevel2_body* :: $[i,i] \Rightarrow o$ **where**
toplevel2_body$(x) \equiv \lambda a.\ |a| < x$

relativize functional *toplevel2_body toplevel2_body_rel*
relationalize *toplevel2_body_rel is_toplevel2_body*

synthesize *is_toplevel2_body* **from_definition assuming** *nonempty*
arity_theorem for *is_toplevel2_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_toplevel2_body*:
$(\#\#M)(A) \Longrightarrow$ *separation*$(\#\#M,$ *is_toplevel2_body*$(\#\#M,A))$
apply(*rule_tac separation_cong*[
where $P{=}\lambda\ x\ .\ M,[x,A] \models$ *is_toplevel2_body_fm*$(1,0)$,*THEN iffD1*])
apply(*rule_tac is_toplevel2_body_iff_sats*[**where** *env*=[_,*A*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*],*simplified*])
apply(*simp_all add*:*arity_is_toplevel2_body_fm ord_simp_union is_toplevel2_body_fm_type*)
done

lemma (**in** *M_ZF_trans*) *toplevel2_body_abs*:
assumes $(\#\#M)(A)$ $(\#\#M)(x)$
shows *is_toplevel2_body*$(\#\#M,A,x) \longleftrightarrow$ *toplevel2_body_rel*$(\#\#M,A,x)$
using *assms pair_in_M_iff apply_closed transM is_cardinal_iff*
cardinal_rel_closed
unfolding *toplevel2_body_rel_def is_toplevel2_body_def*
by (*auto simp*:*absolut*)

lemma (**in** *M_ZF_trans*) *separation_toplevel2_body*:
$(\#\#M)(x) \Longrightarrow$ *separation*$(\#\#M,\ \lambda a.\ |a|^M < x)$
using *separation_is_toplevel2_body toplevel2_body_abs*
separation_cong[**where** *P*=*is_toplevel2_body*$(\#\#M,x)$ **and** *M*=*##M*,*THEN*

iffD1]
unfolding *toplevel2_body_rel_def*
by *simp*

definition *toplevel3_body* :: $i \Rightarrow o$ **where**
$toplevel3\_body \equiv \lambda x.\ \exists\, a\ b.\ x = \langle a,\ b\rangle \wedge a \neq b$

relativize functional *toplevel3_body toplevel3_body_rel*
relationalize *toplevel3_body_rel is_toplevel3_body*

synthesize *is_toplevel3_body* **from_definition**
arity_theorem for *is_toplevel3_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_toplevel3_body*:
separation(##*M*, *is_toplevel3_body*(##*M*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*] $\models$ *is_toplevel3_body_fm*(*0*),*THEN iffD1*])
apply(*rule_tac is_toplevel3_body_iff_sats*[**where** *env*=[_],*symmetric*])
apply(*simp_all*)
apply(*rule_tac separation_ax*[**where** *env*=[],*simplified*])
apply(*simp_all add*:*arity_is_toplevel3_body_fm ord_simp_union is_toplevel3_body_fm_type*)
done

lemma (**in** *M_ZF_trans*) *toplevel3_body_abs*:
assumes (##*M*)(*x*)
shows *is_toplevel3_body*(##*M*,*x*) $\longleftrightarrow$ *toplevel3_body*(*x*)
using *assms pair_in_M_iff apply_closed*
unfolding *toplevel3_body_def is_toplevel3_body_def*
by (*auto*)

lemma (**in** *M_ZF_trans*) *separation_toplevel3_body*:
separation(##*M*, λx . $\exists\, a\ b.\ x = \langle a,\ b\rangle \wedge a \neq b$)
using *separation_is_toplevel3_body toplevel3_body_abs*
separation_cong[**where** *P*=*is_toplevel3_body*(##*M*) **and** *M*=##*M*,*THEN*
iffD1]
unfolding *toplevel3_body_def*
by *simp*

definition *toplevel4_body* :: [*i*,*i*] $\Rightarrow$ *o* **where**
$toplevel4\_body(R) \equiv \lambda z.\ \exists\, a\ b.\ z = \langle a,\ b\rangle \wedge a \cap b = R$

relativize functional *toplevel4_body toplevel4_body_rel*
relationalize *toplevel4_body_rel is_toplevel4_body*

synthesize *is_toplevel4_body* **from_definition assuming** *nonempty*
arity_theorem for *is_toplevel4_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_toplevel4_body*:
 (##*M*)(*A*) ⟹ *separation*(##*M*, *is_toplevel4_body*(##*M*,*A*))
  apply(*rule_tac separation_cong*[
        where *P*=λ *x* . *M*,[*x*,*A*] ⊨ *is_toplevel4_body_fm*(*1*,*0*),*THEN iffD1*])
   apply(*rule_tac is_toplevel4_body_iff_sats*[**where** *env*=[_,*A*],*symmetric*])
  apply(*simp_all add*:*nonempty*[*simplified*])
  apply(*rule_tac separation_ax*[**where** *env*=[*A*],*simplified*])
   apply(*simp_all add*:*arity_is_toplevel4_body_fm ord_simp_union is_toplevel4_body_fm_type*)
  done

lemma (**in** *M_ZF_trans*) *toplevel4_body_abs*:
  assumes (##*M*)(*R*) (##*M*)(*x*)
  shows *is_toplevel4_body*(##*M*,*R*,*x*) ⟷ *toplevel4_body*(*R*,*x*)
  using *assms pair_in_M_iff is_Int_abs*
  unfolding *toplevel4_body_def is_toplevel4_body_def*
  by (*auto*)

lemma (**in** *M_ZF_trans*) *separation_toplevel4_body*:
 (##*M*)(*R*) ⟹ *separation*
        (##*M*, λ*x*. ∃ *a b*. *x* = ⟨*a*, *b*⟩ ∧ *a* ∩ *b* = *R*)
  using *separation_is_toplevel4_body toplevel4_body_abs*
  unfolding *toplevel4_body_def*
  by (*rule_tac separation_cong*[
        where *P*=*is_toplevel4_body*(##*M*,*R*),*THEN iffD1*])

definition *toplevel5_body* :: [*i*,*i*] ⇒ *o* **where**
  toplevel5_body(*R*) ≡ λ*x*. *domain*(*x*) ∈ *R*

relativize functional *toplevel5_body toplevel5_body_rel*
relationalize *toplevel5_body_rel is_toplevel5_body*

synthesize *is_toplevel5_body* **from_definition assuming** *nonempty*
arity_theorem for *is_toplevel5_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_toplevel5_body*:
 (##*M*)(*A*) ⟹ *separation*(##*M*, *is_toplevel5_body*(##*M*,*A*))
  apply(*rule_tac separation_cong*[
        where *P*=λ *x* . *M*,[*x*,*A*] ⊨ *is_toplevel5_body_fm*(*1*,*0*),*THEN iffD1*])
   apply(*rule_tac is_toplevel5_body_iff_sats*[**where** *env*=[_,*A*],*symmetric*])
  apply(*simp_all add*:*nonempty*[*simplified*])
  apply(*rule_tac separation_ax*[**where** *env*=[*A*],*simplified*])
   apply(*simp_all add*:*arity_is_toplevel5_body_fm ord_simp_union is_toplevel5_body_fm_type*)
  done

lemma (**in** *M_ZF_trans*) *toplevel5_body_abs*:
  assumes (##*M*)(*R*) (##*M*)(*x*)
  shows *is_toplevel5_body*(##*M*,*R*,*x*) ⟷ *toplevel5_body*(*R*,*x*)
  using *assms pair_in_M_iff is_Int_abs*

unfolding *toplevel5_body_def is_toplevel5_body_def*
by (*auto simp:domain_closed*[*simplified*])

lemma (**in** *M_ZF_trans*) *separation_toplevel5_body*:
(##*M*)(*R*) ⟹ *separation*
(##*M*, λ*x*. *domain*(*x*) ∈ *R*)
using *separation_is_toplevel5_body toplevel5_body_abs*
unfolding *toplevel5_body_def*
by (*rule_tac separation_cong*[
where *P*=*is_toplevel5_body*(##*M*,*R*),*THEN iffD1*])

definition *toplevel7_body* :: [*i*,*i*,*i*] ⇒ *o* **where**
toplevel7_body(*Q*,*x*) ≡ λ*a*. *restrict*(*a*, *Q*) = *x*

relativize functional *toplevel7_body toplevel7_body_rel*
relationalize *toplevel7_body_rel is_toplevel7_body*

synthesize *is_toplevel7_body* **from_definition assuming** *nonempty*
arity_theorem for *is_toplevel7_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_toplevel7_body*:
(##*M*)(*A*) ⟹ (##*M*)(*B*) ⟹ *separation*(##*M*, *is_toplevel7_body*(##*M*,*A*,*B*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*,*A*,*B*] ⊨ *is_toplevel7_body_fm*(*1*,*2*,*0*),*THEN iffD1*])
apply(*rule_tac is_toplevel7_body_iff_sats*[**where** *env*=[_,*A*,*B*],*symmetric*])
apply(*simp_all add:zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*B*],*simplified*])
apply(*simp_all add:arity_is_toplevel7_body_fm ord_simp_union is_toplevel7_body_fm_type*)
done

lemma (**in** *M_ZF_trans*) *toplevel7_body_abs*:
assumes (##*M*)(*A*) (##*M*)(*B*) (##*M*)(*x*)
shows *is_toplevel7_body*(##*M*,*A*,*B*,*x*) ⟷ *toplevel7_body*(*A*,*B*,*x*)
using *assms pair_in_M_iff apply_closed transM*
unfolding *toplevel7_body_def is_toplevel7_body_def*
by (*auto*)

lemma (**in** *M_ZF_trans*) *separation_toplevel7_body*:
(##*M*)(*Q*) ⟹ (##*M*)(*x*) ⟹ *separation*(##*M*, λ*a*. *restrict*(*a*, *Q*) = *x*)
using *separation_is_toplevel7_body toplevel7_body_abs*
separation_cong[**where** *P*=*is_toplevel7_body*(##*M*,*Q*,*x*) **and** *M*=##*M*,*THEN*
iffD1]
unfolding *toplevel7_body_def*
by *simp*

definition *toplevel8_body* :: [*i*,*i*] ⇒ *o* **where**
toplevel8_body(*R*) ≡ λ*z*. *R* ∈ *domain*(*z*)

relativize functional *toplevel8_body toplevel8_body_rel*
relationalize *toplevel8_body_rel is_toplevel8_body*

synthesize *is_toplevel8_body* **from_definition assuming** *nonempty*
arity_theorem for *is_toplevel8_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_toplevel8_body*:
 $(\#\#M)(A) \Longrightarrow$ *separation*$(\#\#M,$ *is_toplevel8_body*$(\#\#M,A))$
 apply(*rule_tac separation_cong*[
 where $P=\lambda\ x\ .\ M,[x,A] \models$ *is_toplevel8_body_fm*$(1,0)$,*THEN iffD1*])
 apply(*rule_tac is_toplevel8_body_iff_sats*[**where** *env*=[_,*A*],*symmetric*])
 apply(*simp_all add*:*nonempty*[*simplified*])
 apply(*rule_tac separation_ax*[**where** *env*=[*A*],*simplified*])
 apply(*simp_all add*:*arity_is_toplevel8_body_fm ord_simp_union is_toplevel8_body_fm_type*)
 done

lemma (**in** *M_ZF_trans*) *toplevel8_body_abs*:
 assumes $(\#\#M)(R)$ $(\#\#M)(x)$
 shows *is_toplevel8_body*$(\#\#M,R,x) \longleftrightarrow$ *toplevel8_body*(R,x)
 using *assms pair_in_M_iff is_Int_abs*
 unfolding *toplevel8_body_def is_toplevel8_body_def*
 by (*auto simp*:*domain_closed*[*simplified*])

lemma (**in** *M_ZF_trans*) *separation_toplevel8_body*:
 $(\#\#M)(R) \Longrightarrow$ *separation*
 $(\#\#M,\ \lambda z.\ R \in$ *domain*$(z))$
 using *separation_is_toplevel8_body toplevel8_body_abs*
 unfolding *toplevel8_body_def*
 by (*rule_tac separation_cong*[
 where *P*=*is_toplevel8_body*$(\#\#M,R)$,*THEN iffD1*])

definition *toplevel9_body* :: $[i,i,i] \Rightarrow o$ **where**
 toplevel9_body$(Q,x) \equiv \lambda z.\ \exists\, n{\in}\omega.\ \langle\langle Q,\ n\rangle,\ 1\rangle \in z \wedge \langle\langle x,\ n\rangle,\ 0\rangle \in z$

relativize functional *toplevel9_body toplevel9_body_rel*
relationalize *toplevel9_body_rel is_toplevel9_body*

synthesize *is_toplevel9_body* **from_definition assuming** *nonempty*
arity_theorem for *is_toplevel9_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_toplevel9_body*:
 $(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow$ *separation*$(\#\#M,$ *is_toplevel9_body*$(\#\#M,A,B))$
 apply(*rule_tac separation_cong*[
 where $P=\lambda\ x\ .\ M,[x,A,B] \models$ *is_toplevel9_body_fm*$(1,2,0)$,*THEN iffD1*])
 apply(*rule_tac is_toplevel9_body_iff_sats*[**where** *env*=[_,*A*,*B*],*symmetric*])
 apply(*simp_all add*:*zero_in_M*)
 apply(*rule_tac separation_ax*[**where** *env*=[*A*,*B*],*simplified*])

apply(*simp_all add*:*arity_is_toplevel9_body_fm ord_simp_union is_toplevel9_body_fm_type*)
done

lemma (**in** *M_ZF_trans*) *toplevel9_body_abs*:
assumes (##*M*)(*A*) (##*M*)(*B*) (##*M*)(*x*)
shows *is_toplevel9_body*(##*M*,*A*,*B*,*x*) ⟷ *toplevel9_body*(*A*,*B*,*x*)
using *assms pair_in_M_iff apply_closed transM*
unfolding *toplevel9_body_def is_toplevel9_body_def*
by (*auto simp*:*nat_into_M*[*simplified*] *M_nat*[*simplified*])

lemma (**in** *M_ZF_trans*) *separation_toplevel9_body*:
(##*M*)(*Q*) ⟹ (##*M*)(*x*) ⟹ *separation*(##*M*, λ*z*. ∃ *n*∈ω. ⟨⟨*Q*, *n*⟩, *1*⟩ ∈ *z* ∧
⟨⟨*x*, *n*⟩, *0*⟩ ∈ *z*)
using *separation_is_toplevel9_body toplevel9_body_abs*
separation_cong[**where** *P*=*is_toplevel9_body*(##*M*,*Q*,*x*) **and** *M*=##*M*,*THEN*
iffD1]
unfolding *toplevel9_body_def*
by *simp*

definition *toplevel10_body* :: [*i*,*i*,*i*] ⇒ *o* **where**
toplevel10_body(*A*,*r*) ≡ λ*y*. ∃ *x*∈*A*. *y* = ⟨*x*, *restrict*(*x*, *r*)⟩

relativize functional *toplevel10_body toplevel10_body_rel*
relationalize *toplevel10_body_rel is_toplevel10_body*

synthesize *is_toplevel10_body* **from_definition assuming** *nonempty*
arity_theorem for *is_toplevel10_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_toplevel10_body*:
(##*M*)(*A*) ⟹ (##*M*)(*r*) ⟹ *separation*(##*M*, *is_toplevel10_body*(##*M*,*A*,*r*))
apply(*rule_tac separation_cong*[
where *P*=λ *x* . *M*,[*x*,*A*,*r*] ⊨ *is_toplevel10_body_fm*(*1*,*2*,*0*),*THEN iffD1*])
apply(*rule_tac is_toplevel10_body_iff_sats*[**where** *env*=[_,*A*,*r*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[*A*,*r*],*simplified*])
apply(*simp_all add*:*arity_is_toplevel10_body_fm ord_simp_union is_toplevel10_body_fm_type*)
done

lemma (**in** *M_ZF_trans*) *toplevel10_body_abs*:
assumes (##*M*)(*A*) (##*M*)(*r*) (##*M*)(*x*)
shows *is_toplevel10_body*(##*M*,*A*,*r*,*x*) ⟷ *toplevel10_body*(*A*,*r*,*x*)
using *assms pair_in_M_iff apply_closed transM*
unfolding *toplevel10_body_def is_toplevel10_body_def*
by *auto*

lemma (**in** *M_ZF_trans*) *separation_toplevel10_body*:
(##*M*)(*A*) ⟹ (##*M*)(*r*) ⟹ *separation*(##*M*, λ*y*. ∃ *x*∈*A*. *y* = ⟨*x*, *restrict*(*x*,

$r)\rangle)$
using *separation_is_toplevel10_body toplevel10_body_abs*
separation_cong[**where** P=*is_toplevel10_body*(\#\#M,A,r) **and** M=\#\#M,*THEN iffD1*]
unfolding *toplevel10_body_def*
by *simp*

definition *toplevel11_body* :: [i,i] ⇒ o **where**
toplevel11_body(A) ≡ λp. (∀ x∈A. (x ∈ *snd*(p) ⟷ *domain*(x) = *fst*(p)))

relativize functional *toplevel11_body toplevel11_body_rel*
relationalize *toplevel11_body_rel is_toplevel11_body*

synthesize *is_toplevel11_body* **from_definition assuming** *nonempty*
arity_theorem for *is_toplevel11_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_toplevel11_body*:
(\#\#M)(A) ⟹ *separation*(\#\#M, *is_toplevel11_body*(\#\#M,A))
apply(*rule_tac separation_cong*[
where P=λ x . M,[x,A] ⊨ *is_toplevel11_body_fm*(1,0),*THEN iffD1*])
apply(*rule_tac is_toplevel11_body_iff_sats*[**where** *env*=[_,A],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac separation_ax*[**where** *env*=[A],*simplified*])
apply(*simp_all add*:*arity_is_toplevel11_body_fm ord_simp_union is_toplevel11_body_fm_type*)
done

lemma (**in** *M_ZF_trans*) *toplevel11_body_abs*:
assumes (\#\#M)(A) (\#\#M)(x)
shows *is_toplevel11_body*(\#\#M,A,x) ⟷ *toplevel11_body*(A,x)
using *assms domain_closed domain_abs zero_in_M transM*[*of* _ A] *transitivity fst_snd_closed fst_abs snd_abs*
unfolding *toplevel11_body_def is_toplevel11_body_def*
by *auto*

lemma (**in** *M_ZF_trans*) *separation_toplevel11_body*:
(\#\#M)(A) ⟹ *separation*(\#\#M, λp. ∀ x∈A. x ∈ *snd*(p) ⟷ *domain*(x) = *fst*(p))
using *separation_is_toplevel11_body toplevel11_body_abs*
separation_cong[**where** P=*is_toplevel11_body*(\#\#M,A) **and** M=\#\#M,*THEN iffD1*]
unfolding *toplevel11_body_def*
by *simp*

definition *toplevel12_body*
where *toplevel12_body*(G,p) ≡ ∀ x∈G. x ∈ *snd*(p) ⟷ *fst*(p) ∈ x

relativize functional *toplevel12_body toplevel12_body_rel*
relationalize *toplevel12_body_rel is_toplevel12_body*

synthesize *is_toplevel12_body* **from_definition**
arity_theorem for *is_toplevel12_body_fm*

lemma (**in** *M_ZF_trans*) *separation_is_toplevel12_body*:
 $(\#\#M)(G) \Longrightarrow$ *separation*$(\#\#M,$ *is_toplevel12_body*$(\#\#M,G))$
 apply(*rule_tac separation_cong*[
 where $P{=}\lambda\ x\ .\ M,[x,G] \models$ *is_toplevel12_body_fm*$(1,0)$,*THEN iffD1*])
 apply(*rule_tac is_toplevel12_body_iff_sats*[**where** *env*=[_,*G*],*symmetric*])
 apply(*simp_all*)
 apply(*rule_tac separation_ax*[**where** *env*=[*G*],*simplified*])
 apply(*simp_all add*:*arity_is_toplevel12_body_fm ord_simp_union is_toplevel12_body_fm_type*)
 done

lemma (**in** *M_ZF_trans*) *toplevel12_body_abs*:
 assumes $(\#\#M)(G)$ $(\#\#M)(x)$
 shows *is_toplevel12_body*$(\#\#M,G,x) \longleftrightarrow$ *toplevel12_body*(G,x)
 using *assms pair_in_M_iff fst_abs snd_abs transitivity fst_snd_closed*
 unfolding *toplevel12_body_def is_toplevel12_body_def*
 by *force*

lemma (**in** *M_ZF_trans*) *separation_toplevel12_body*:
 $(\#\#M)(G) \Longrightarrow$ *separation*
 $(\#\#M,\ \lambda p.\ \forall\, x{\in}G.\ x \in snd(p) \longleftrightarrow fst(p) \in x)$
 using *toplevel12_body_abs separation_is_toplevel12_body*
 unfolding *toplevel12_body_def*
 by (*rule_tac separation_cong*[
 where *P*=*is_toplevel12_body*$(\#\#M,G)$,*THEN iffD1*])

end
theory *Cardinal_Preservation*
 imports
 Cohen_Posets_Relative
 Forcing_Main
 ZF_Trans_Interpretations
begin

context *forcing_notion*
begin

definition
 antichain :: $i{\Rightarrow}o$ **where**
 $antichain(A) \equiv A{\subseteq}P \wedge (\forall\, p{\in}A.\ \forall\, q{\in}A.\ p \neq q \longrightarrow p \perp q)$

definition
 ccc :: *o* **where**
 $ccc \equiv \forall\, A.\ antichain(A) \longrightarrow |A| \leq \omega$

end

locale *M_trivial_notion* = *M_trivial* + *forcing_notion*
begin

abbreviation
antichain_r′ :: *i* ⇒ *o* **where**
antichain_r′(*A*) ≡ *antichain_rel*(*M*,*P*,*leq*,*A*)

lemma *antichain_abs′* [*absolut*]:
⟦ *M*(*A*); *M*(*P*); *M*(*leq*) ⟧ ⟹ *antichain_r′*(*A*) ⟷ *antichain*(*A*)
unfolding *antichain_rel_def antichain_def compat_def*
by (*simp add*:*absolut*)

end

— MOVE THIS to an appropriate place

The following interpretation makes the simplifications from the locales *M_trans*, *M_trivial*, etc., available for *M*[*G*]

sublocale *forcing_data* ⊆ *M_trivial_notion* ##*M* **..**

context *forcing_data*
begin

lemma *antichain_abs″* [*absolut*]: *A*∈*M* ⟹ *antichain_r′*(*A*) ⟷ *antichain*(*A*)
using *P_in_M leq_in_M*
unfolding *antichain_rel_def antichain_def compat_def*
by (*auto simp add*:*absolut transitivity*)

end

lemma (**in** *forcing_notion*) *Incompatible_imp_not_eq*: ⟦ *p* ⊥ *q*; *p*∈*P*; *q*∈*P* ⟧⟹ *p* ≠ *q*
using *refl_leq* **by** *blast*

lemma (**in** *forcing_data*) *inconsistent_imp_incompatible*:
assumes *p* ⊩ *φ env q* ⊩ *Neg*(*φ*) *env p*∈*P q*∈*P*
arity(*φ*) ≤ *length*(*env*) *φ* ∈ *formula env* ∈ *list*(*M*)
shows *p* ⊥ *q*
proof
assume *compat*(*p*,*q*)
then
obtain *d* **where** *d* ≼ *p d* ≼ *q d* ∈ *P* **by** *blast*
moreover
note *assms*
moreover from *calculation*
have *d* ⊩ *φ env d* ⊩ *Neg*(*φ*) *env*
using *strengthening_lemma* **by** *simp_all*
ultimately

show *False*
using *Forces_Neg*[*of d env* φ] *refl_leq P_in_M*
by (*auto dest*:*transM*; *drule_tac bspec*; *auto dest*:*transM*)
qed

notation (**in** *forcing_data*) *check* ($\langle\_^{v}\rangle$ [*101*] *100*)

context *G_generic* **begin**

— NOTE: The following bundled additions to the simpset might be of use later on, perhaps add them globally to some appropriate locale.
lemmas *generic_simps* = *generic*[*THEN one_in_G*, *THEN valcheck*, *OF one_in_P*]
generic[*THEN one_in_G*, *THEN M_subset_MG*, *THEN subsetD*]
check_in_M GenExtI P_in_M
lemmas *generic_dests* = *M_genericD*[*OF generic*] *M_generic_compatD*[*OF generic*]

bundle *G_generic_lemmas* = *generic_simps*[*simp*] *generic_dests*[*dest*]

end

sublocale *G_generic* ⊆ *ext*:*M_ZF_trans M*[*G*]
using *Transset_MG generic pairing_in_MG Union_MG*
extensionality_in_MG power_in_MG foundation_in_MG
strong_replacement_in_MG separation_in_MG infinity_in_MG
by *unfold_locales simp_all*

sublocale *G_generic_AC* ⊆ *ext*:*M_ZFC_trans M*[*G*]
using *choice_ax choice_in_MG*
by *unfold_locales*

lemma (**in** *forcing_data*) *forces_neq_apply_imp_incompatible*:
assumes
$p \Vdash \cdot 0\text{`}1 \text{ is } 2\cdot\ [f,a,b^v]$
$q \Vdash \cdot 0\text{`}1 \text{ is } 2\cdot\ [f,a,b'^v]$
$b \neq b'$
— More general version: taking general names b^v and b'^v, satisfying $p \Vdash \cdot\neg\cdot 0 = 1\cdot\cdot\ [b^v,\ b'^v]$ and $q \Vdash \cdot\neg\cdot 0 = 1\cdot\cdot\ [b^v,\ b'^v]$.
and
types:$f{\in}M\ a{\in}M\ b{\in}M\ b'{\in}M\ p{\in}P\ q{\in}P$
shows
$p \perp q$
proof -
{
fix *G*
assume *M_generic*(*G*)
then
interpret *G_generic* _ _ _ _ _ *G* **by** *unfold_locales*
include *G_generic_lemmas*
— FIXME: make a locale containg two *M_ZF_trans* instances, one for *M* and

one for $M[G]$
assume $q \in G$
with *assms* ‹*M_generic*(G)›
have $M[G]$, *map*(*val*(P,G),[f,a,b'^v]) $\models$ ·0‘1 *is* 2·
using *truth_lemma*[*of* ·0‘1 *is* 2· G [f,a,b'^v]]
by (*auto simp add*:*ord_simp_union arity_fun_apply_fm fun_apply_type*)
with ‹$b \neq b'$› *types*
have $M[G]$, *map*(*val*(P,G),[f,a,b^v]) $\models$ ·¬·0‘1 *is* 2··
using *GenExtI* **by** *auto*
}
with *types*
have $q \Vdash$ ·¬·0‘1 *is* 2·· [f,a,b^v]
using *definition_of_forcing*[**where** φ=·¬·0‘1 *is* 2··] *check_in_M*
by (*auto simp add*:*ord_simp_union arity_fun_apply_fm*)
with ‹$p \Vdash$ ·0‘1 *is* 2· [f,a,b^v]› **and** *types*
show $p \perp q$
using *check_in_M inconsistent_imp_incompatible*
by (*simp add*:*ord_simp_union arity_fun_apply_fm fun_apply_type*)
qed

context *G_generic_AC* **begin**

context
includes *G_generic_lemmas*
begin

— Simplifying simp rules (because of the occurrence of "##")
lemmas *sharp_simps* = *Card_rel_Union Card_rel_cardinal_rel Collect_abs Cons_abs Cons_in_M_iff Diff_closed Equal_abs Equal_in_M_iff Finite_abs Forall_abs Forall_in_M_iff Inl_abs Inl_in_M_iff Inr_abs Inr_in_M_iff Int_closed Inter_abs Inter_closed M_nat Member_abs Member_in_M_iff Memrel_closed Nand_abs Nand_in_M_iff Nil_abs Nil_in_M Ord_cardinal_rel Pow_rel_closed Un_closed Union_abs Union_closed and_abs and_closed apply_abs apply_closed bij_rel_closed bijection_abs bool_of_o_abs bool_of_o_closed cadd_rel_0 cadd_rel_closed cardinal_rel_0_iff_0 cardinal_rel_closed cardinal_rel_idem cartprod_abs cartprod_closed cmult_rel_0 cmult_rel_1 cmult_rel_closed comp_closed composition_abs cons_abs cons_closed converse_abs converse_closed csquare_lam_closed csquare_rel_closed depth_closed domain_abs domain_closed eclose_abs eclose_closed empty_abs field_abs field_closed finite_funspace_closed finite_ordinal_abs formula_N_abs formula_N_closed formula_abs formula_case_abs formula_case_closed formula_closed formula_functor_abs fst_closed function_abs function_space_rel_closed hd_abs image_abs image_closed inj_rel_closed injection_abs inter_abs irreflexive_abs is_depth_apply_abs is_eclose_n_abs is_funspace_abs iterates_closed le_abs length_abs length_closed lepoll_rel_refl limit_ordinal_abs linear_rel_abs list_N_abs list_N_closed list_abs list_case'_closed list_case_abs list_closed list_functor_abs lt_abs*

mem_bij_abs mem_eclose_abs mem_inj_abs mem_list_abs membership_abs
minimum_closed nat_case_abs nat_case_closed nonempty not_abs
not_closed nth_abs number1_abs number2_abs number3_abs omega_abs
or_abs or_closed order_isomorphism_abs ordermap_closed
ordertype_closed ordinal_abs pair_abs pair_in_M_iff powerset_abs
pred_closed pred_set_abs quasilist_abs quasinat_abs radd_closed
rall_abs range_abs range_closed relation_abs restrict_closed
restriction_abs rex_abs rmult_closed rtrancl_abs rtrancl_closed
rvimage_closed separation_closed setdiff_abs singleton_abs
singleton_in_M_iff snd_closed strong_replacement_closed subset_abs
succ_in_M_iff successor_abs successor_ordinal_abs sum_abs sum_closed
surj_rel_closed surjection_abs tl_abs trancl_abs trancl_closed
transitive_rel_abs transitive_set_abs typed_function_abs union_abs
upair_abs upair_in_M_iff vimage_abs vimage_closed well_ord_abs
mem_formula_abs fst_abs snd_abs nth_closed

— NOTE: there is a theorem missing from those above
lemmas *mg_sharp_simps = ext.Card_rel_Union ext.Card_rel_cardinal_rel*
ext.Collect_abs ext.Cons_abs ext.Cons_in_M_iff ext.Diff_closed
ext.Equal_abs ext.Equal_in_M_iff ext.Finite_abs ext.Forall_abs
ext.Forall_in_M_iff ext.Inl_abs ext.Inl_in_M_iff ext.Inr_abs
ext.Inr_in_M_iff ext.Int_closed ext.Inter_abs ext.Inter_closed
ext.M_nat ext.Member_abs ext.Member_in_M_iff ext.Memrel_closed
ext.Nand_abs ext.Nand_in_M_iff ext.Nil_abs ext.Nil_in_M
ext.Ord_cardinal_rel ext.Pow_rel_closed ext.Un_closed
ext.Union_abs ext.Union_closed ext.and_abs ext.and_closed
ext.apply_abs ext.apply_closed ext.bij_rel_closed
ext.bijection_abs ext.bool_of_o_abs ext.bool_of_o_closed
ext.cadd_rel_0 ext.cadd_rel_closed ext.cardinal_rel_0_iff_0
ext.cardinal_rel_closed ext.cardinal_rel_idem ext.cartprod_abs
ext.cartprod_closed ext.cmult_rel_0 ext.cmult_rel_1
ext.cmult_rel_closed ext.comp_closed ext.composition_abs
ext.cons_abs ext.cons_closed ext.converse_abs ext.converse_closed
ext.csquare_lam_closed ext.csquare_rel_closed ext.depth_closed
ext.domain_abs ext.domain_closed ext.eclose_abs ext.eclose_closed
ext.empty_abs ext.field_abs ext.field_closed
ext.finite_funspace_closed ext.finite_ordinal_abs ext.formula_N_abs
ext.formula_N_closed ext.formula_abs ext.formula_case_abs
ext.formula_case_closed ext.formula_closed ext.formula_functor_abs
ext.fst_closed ext.function_abs ext.function_space_rel_closed
ext.hd_abs ext.image_abs ext.image_closed ext.inj_rel_closed
ext.injection_abs ext.inter_abs ext.irreflexive_abs
ext.is_depth_apply_abs ext.is_eclose_n_abs ext.is_funspace_abs
ext.iterates_closed ext.le_abs ext.length_abs ext.length_closed
ext.lepoll_rel_refl ext.limit_ordinal_abs ext.linear_rel_abs
ext.list_N_abs ext.list_N_closed ext.list_abs
ext.list_case'_closed ext.list_case_abs ext.list_closed
ext.list_functor_abs ext.lt_abs ext.mem_bij_abs ext.mem_eclose_abs
ext.mem_inj_abs ext.mem_list_abs ext.membership_abs

ext.minimum_closed ext.nat_case_abs ext.nat_case_closed
ext.nonempty ext.not_abs ext.not_closed ext.nth_abs
ext.number1_abs ext.number2_abs ext.number3_abs ext.omega_abs
ext.or_abs ext.or_closed ext.order_isomorphism_abs
ext.ordermap_closed ext.ordertype_closed ext.ordinal_abs
ext.pair_abs ext.pair_in_M_iff ext.powerset_abs ext.pred_closed
ext.pred_set_abs ext.quasilist_abs ext.quasinat_abs
ext.radd_closed ext.rall_abs ext.range_abs ext.range_closed
ext.relation_abs ext.restrict_closed ext.restriction_abs
ext.rex_abs ext.rmult_closed ext.rtrancl_abs ext.rtrancl_closed
ext.rvimage_closed ext.separation_closed ext.setdiff_abs
ext.singleton_abs ext.singleton_in_M_iff ext.snd_closed
ext.strong_replacement_closed ext.subset_abs ext.succ_in_M_iff
ext.successor_abs ext.successor_ordinal_abs ext.sum_abs
ext.sum_closed ext.surj_rel_closed ext.surjection_abs ext.tl_abs
ext.trancl_abs ext.trancl_closed ext.transitive_rel_abs
ext.transitive_set_abs ext.typed_function_abs ext.union_abs
ext.upair_abs ext.upair_in_M_iff ext.vimage_abs ext.vimage_closed
ext.well_ord_abs ext.mem_formula_abs ext.nth_closed

declare *sharp_simps*[*simp del*, *simplified setclass_iff*, *simp*]
— The following was motivated by the fact that $[\![(\#\#M[G])(?f); (\#\#M[G])(?a)]\!] \Longrightarrow (\#\#M[G])(?f\ `\ ?a)$ did not simplify appropriately
NOTE: $[\![(\#\#M)(?p); (\#\#M)(?x)]\!] \Longrightarrow is\_fst(\#\#M, ?p, ?x) \longleftrightarrow ?x = fst(?p)$ and $[\![(\#\#M)(?p); (\#\#M)(?y)]\!] \Longrightarrow is\_snd(\#\#M, ?p, ?y) \longleftrightarrow ?y = snd(?p)$ not in mgzf interpretation.
declare *mg_sharp_simps*[*simp del*, *simplified setclass_iff*, *simp*]

— Kunen IV.2.31
lemma *forces_below_filter*:
  assumes $M[G]$, $map(val(P,G),env) \models \varphi\ p \in G$
    $arity(\varphi) \leq length(env)\ \varphi \in formula\ env \in list(M)$
  shows $\exists q{\in}G.\ q \preceq p \wedge q \Vdash \varphi\ env$
proof -
  note *assms*
  moreover from *this*
  obtain r **where** $r \Vdash \varphi\ env\ r{\in}G$
    using *generic truth_lemma*[*of* φ _ *env*]
    by *blast*
  moreover from *this* **and** ‹$p{\in}G$›
  obtain q **where** $q \preceq p\ q \preceq r\ q \in G$ **by** *auto*
  ultimately
  show *?thesis*
    using *strengthening_lemma*[*of* $r\ \varphi$ _ *env*] **by** *blast*
qed

abbreviation
  $fm\_leq :: [i,i,i] \Rightarrow i$ (‹·_≼_·›) **where**
  $fm\_leq(A,l,B) \equiv leq\_fm(l,A,B)$

simple_rename *ren_F* **src** [*x_P*, *x_leq*, *x_o*, *x_f*, *y_c*, *x_bc*, *p*, *x*, *b*]
tgt [*x_bc*, *y_c*,*b*,*x*, *x_P*, *x_leq*, *x_o*, *x_f*, *p*]

simple_rename *ren_G* **src** [*x*,*x_P*, *x_leq*, *x_one*, *x_f*,*x_p*,*y*,*x_B*]
tgt [*x*,*y*,*x_P*, *x_leq*, *x_one*, *x_f*,*x_p*,*x_B*]

simple_rename *ren_F_aux* **src** [*q*,*x_P*, *x_leq*, *x_one*, *f_dot*, *x_a*, *x_bc*,*x_p*,*x_b*]
tgt [*x_bc*, *q*, *x_b*, *x_P*, *x_leq*, *x_one*, *f_dot*,*x_a*,*x_p*]

simple_rename *ren_G_aux* **src** [*x_b*, *x_P*, *x_leq*, *x_one*, *f_dot*,*x_a*,*x_p*,*y*]
tgt [*x_b*, *y*, *x_P*, *x_leq*, *x_one*, *f_dot*,*x_a*,*x_p*]

lemma *ccc_fun_closed_lemma_aux*:
assumes *f_dot*∈*M p*∈*M a*∈*M b*∈*M*
shows {*q* ∈ *P* . *q* ≼ *p* ∧ (*M*, [*q*, *P*, *leq*, *one*, *f_dot*, a^v, b^v] ⊨ *forces*(·0‘1 *is* 2·
))} ∈ *M*
proof -
let *?app_fm*=·0‘1 *is* 2·
let *?ψ*=··0 $\preceq^2$ 7· ∧ *forces*(*?app_fm*) ·
let *?Q*=λ *q* . *q* ≼ *p* ∧ (*M*, [*q*, *P*, *leq*, *one*, *f_dot*, a^v, b^v] ⊨ *forces*(*?app_fm*))
have *?app_fm* ∈ *formula arity*(*?app_fm*) = 3
using *arity_fun_apply_fm union_abs1*
by *simp_all*
then
have *arity*(*forces*(*?app_fm*)) ≤ 7
using *arity_forces*[*OF* ‹*?app_fm*∈_›] **by** *simp_all*
then
have *arity*(*?ψ*) ≤ 8 *?ψ*∈*formula*
using *arity_leq_fm union_abs2 union_abs1 le_trans*
by *simp_all*
with ‹*a*∈*M*› ‹*b*∈*M*›
have a^v∈*M* b^v∈*M*
by *simp_all*
note *types*=‹*f_dot*∈_› ‹*p*∈*M*› *P_in_M leq_in_M one_in_M* ‹a^v∈*M*› ‹b^v∈*M*›
then
have *sats*:(*M*,[*q*, *P*, *leq*, *one*, *f_dot*,a^v, b^v,*p*] ⊨*?ψ*) ⟷ *?Q*(*q*) **if** *q*∈*P* **for** *q*
proof -
note *types′*= *types transitivity*[*OF* ‹*q*∈_› ‹*P*∈_›]
with *types′* ‹*arity*(*forces*(_))≤_›
have
?Q(*q*) ⟷ *q* ≼ *p* ∧ (*M*, [*q*, *P*, *leq*, *one*, *f_dot*, a^v, b^v,*p*] ⊨ *forces*(*?app_fm*))
using *arity_sats_iff*[*of* _ [*p*] _ [_, _, _, _, _, _, _]]
by *simp*
also from *types′*
have ... ⟷ (*M*, [*q*,*P*, *leq*, *one*, *f_dot*, a^v, b^v,*p*] ⊨ *?ψ*)
(**is** _ ⟷ _, *?η* ⊨ _)
unfolding *leq_fm_def* **using** *transitivity*[*of* _ *P*]
by *auto*

ultimately
show *?thesis* **by** *simp*
qed
with *types* ‹*?ψ*∈_› ‹*arity*(*?ψ*) ≤ *8*›
show {*q*∈*P*. *?Q*(*q*)}∈*M*
using *separation_in_M*[**where** *Q*=*?Q* **and** *env*=[*P*, *leq*, *one*, *f_dot*,a^v, b^v,*p*]]
by *simp*
qed

lemma *ccc_fun_closed_lemma_aux2*:
assumes *B*∈*M f_dot*∈*M p*∈*M a*∈*M*
shows (##*M*)(λ*b*∈*B*. {*q* ∈ *P* . *q* ≼ *p* ∧ (*M*, [*q*, *P*, *leq*, *one*, *f_dot*, a^v, b^v] ⊨ *forces*(·*0*‘*1 is 2*·))})
proof -
let *?app_fm*=·*0*‘*1 is 2*·
let *?Q*=λ *b q* . *q* ≼ *p* ∧ (*M*, [*q*, *P*, *leq*, *one*, *f_dot*, a^v, b^v] ⊨ *forces*(*?app_fm*))
from *assms*
have *closed*:{*q*∈*P* . *?Q*(*b*,*q*)} ∈ *M* **if** *b*∈*B* **for** *b*
using *ccc_fun_closed_lemma_aux transitivity*[*OF* _ ‹*B*∈_›] *that*
by *simp*
let *?ψ*=(··*0* $\preceq^2$ *7*· ∧ *forces*(*?app_fm*) ·)
let *?G*=(·∃··2^v*5 is 0*· ∧ *ren*(*?ψ*) ‘ *9* ‘ *9* ‘ *ren_F_aux_fn* ··)
have *?app_fm* ∈ *formula arity*(*?app_fm*) = *3*
using *arity_fun_apply_fm union_abs1*
by *simp_all*
then
have *arity*(*forces*(*?app_fm*)) ≤ *7*
using *arity_forces*[*OF* ‹*?app_fm*∈_›] **by** *simp_all*
then
have *arity*(*?ψ*) ≤ *9*
using *arity_leq_fm union_abs2 union_abs1 le_trans*
by *simp*
then
have *ren*(*?ψ*)‘*9*‘*9*‘*ren_F_aux_fn* ∈ *formula arity*(*ren*(*?ψ*)‘*9*‘*9*‘*ren_F_aux_fn*) ≤ *9*
using *arity_ren ren_tc ren_F_aux_thm check_fm_type leq_fm_type ren_F_aux_fn_def pred_le*
by *simp_all*
with ‹*arity*(*forces*(_))≤*7*›
have *arity*(*?G*) ≤ *8* *?G*∈*formula*
using *check_fm_type arity_check_fm pred_Un_distrib pred_le Un_le*
by *simp_all*
have *pred*(*arity*(*?G*)) ≤ *9* *pred*(*arity*(*?G*))∈*nat*
using *pred_le*[*OF* _ ‹*arity*(*?G*)≤*8*›] *le_trans pred_type*[*OF* _ ‹*arity*(*?G*)≤*8*›]
by *simp_all*
have *pred*(*arity*(*?G*)) ≤ *8*
using *le_trans pred_le* ‹*arity*(*?G*)≤*8*›
by *simp_all*
note *types*=‹*f_dot*∈_› ‹*p*∈*M*› *P_in_M leq_in_M one_in_M* ‹*a*∈*M*› ‹*B*∈*M*›

from ‹$a{\in}M$›
have $a^v{\in}M$ **by** *simp*
have *sats*:$(M,\ [q,b,\ P,\ leq,\ one,\ f\_dot,\ a^v,\ p]\ {\models}\ ?G) \longleftrightarrow ?Q(b,q)$ **if** $b{\in}B\ q{\in}M$ **for** $b\ q$
proof -
from *that*
have $b^v{\in}M\ b{\in}M$
using *transitivity*[*OF* _ ‹$B{\in}$_›] **by** *simp_all*
note *types*′=‹$a^v{\in}M$› *types* ‹$b{\in}M$› ‹$b^v{\in}M$› *that*
from *types*′ ‹*arity*(*forces*(_))≤7›
have $?Q(b,q) \longleftrightarrow$
$(q \preceq p\ \wedge\ (M,\ [q,P,leq,one,f\_dot,a^v,b^v,p,b]\ {\models}\ forces(?app\_fm)))$
using *arity_sats_iff*[*of* _ [*p*,*b*] _ [_, _, _, _, _,_, _]] *transitivity*[*of* _ *P*]
by *simp*
also from *types*′
have ... $\longleftrightarrow (M,\ [q,P,\ leq,\ one,\ f\_dot,\ a^v,\ b^v,p,b]\ {\models}\ ?\psi)$
(**is** _ $\longleftrightarrow$ _, $?\eta\ {\models}$ _)
using *sats_leq_fm*[*of 2 0 7*] *leq_abs*
by *simp*
also from *types*′ ‹*arity*(*forces*(_))≤_› ‹*arity*($?\psi$)≤ 9› *ren_F_aux_fn_def*
have ... $\longleftrightarrow M,\ [b^v,q,b,\ P,\ leq,\ one,\ f\_dot,a^v,p]\ {\models}\ ren(?\psi)$‘9‘9‘*ren_F_aux_fn*
(**is** _ $\longleftrightarrow$ _, $?\eta'\ {\models}\ ?\psi'$)
using *sats_iff_sats_ren*[*of* $?\psi$ *9 9* $?\eta$ *M* $?\eta'$]*ren_F_aux_thm*(*1*)[**where** *A*=*M*] *ren_F_aux_thm*
by *auto*
also from *assms types*′
have ... $\longleftrightarrow M,\ [q,b,P,\ leq,\ one,\ f\_dot,a^v,p]\ {\models}\ ?G$ (**is** _ $\longleftrightarrow M,?\eta''{\models}$_)
using *sats_check_fm*[*of 5* [_]@$?\eta''$ *2 0*] *check_abs*
by *simp*
ultimately
show *?thesis*
by *simp*
qed
then
have *sats*′:$(M,[q,b,P,\ leq,\ one,\ f\_dot,a^v,p,y]\ {\models}?G) \longleftrightarrow ?Q(b,q)$
if $b{\in}B\ q{\in}M\ y{\in}M$ **for** $b\ q\ y$
proof -
from *that* ‹$?G{\in}$_› ‹*arity*($?G$)≤8› *types*
have $(M,[q,b,\ P,\ leq,\ one,\ f\_dot,a^v,p,y]\ {\models}?G) \longleftrightarrow M,[q,b,P,\ leq,\ one,\ f\_dot,a^v,p]\ {\models}?G$
using *transitivity*[*of b B*] *transitivity*[*of q P*]
arity_sats_iff[*of* $?G$ [*y*] *M* [*q*,*b*, *P*, *leq*, *one*, *f_dot*,a^v,*p*]]
by *simp*
then show *?thesis* **using** *sats*[*OF* ‹$b{\in}B$› ‹$q{\in}M$›] *that*
by *simp*
qed
from *types*
have *sats_col*:$(M,[b,P,\ leq,\ one,\ f\_dot,a^v,p,y]\ {\models}Collect\_fm(1,?G,7)) \longleftrightarrow$
$is\_Collect(\#\#M,P,?Q(b),y)$ **if** $b{\in}B\ y{\in}M$ **for** $b\ y$

```
    using that sats'[OF ‹b∈B›] sats_Collect_fm[of M ?Q(b)]
      transitivity[OF ‹b∈B› ‹B∈M›]
    by simp
  from ‹pred(arity(?G))∈nat› ‹?G∈_›
  have Collect_fm(1,?G,7) ∈ formula arity(Collect_fm(1,?G,7)) ≤ 8
    using  arity_Collect_fm pred_le union_abs1 Un_le[OF _ ‹pred(arity(?G))≤
8 ›,of 8 ]
    by (simp_all)
  then
  have arity(ren(Collect_fm(1,?G,7))‘8‘8‘ren_G_aux_fn) ≤ 8
    ren(Collect_fm(1,?G,7))‘8‘8‘ren_G_aux_fn ∈ formula (is ?Γ∈_)
   using ren_tc[of Collect_fm(1,?G,7)] ren_G_aux_thm(2) ren_G_aux_fn_def
arity_ren
    by simp_all
  from ‹arity(Collect_fm(1,?G,7))≤_› ‹Collect_fm(1,?G,7)∈_› types
  have (M,[b,y,P, leq, one, f_dot,a^v,p] ⊨ ?Γ) ⟷
        M,[b,P, leq, one, f_dot,a^v,p,y] ⊨ Collect_fm(1,?G,7) if b∈B y∈M for b y
    using sats_iff_sats_ren[of Collect_fm(1,?G,7) 8 8 _ M [b,y,P, leq, one,
f_dot,a^v,p]]
      ren_G_aux_thm(1)[where A=M] ren_G_aux_thm(2) transitivity[of b B]
that ren_G_aux_fn_def
    by simp
  then
 have (M,[b,y,P, leq, one, f_dot,a^v,p] ⊨ren(Collect_fm(1,?G,7))‘8‘8‘ren_G_aux_fn)
⟷
            y = {q∈P . ?Q(b,q)} if b∈B y∈M for y b
    using that iff_trans[OF _ sats_col]
      Collect_abs types
    by auto
  with assms types ‹?Γ∈_› ‹arity(?Γ)≤_›
  show ?thesis
    using closed Lambda_in_M[where env=[P, leq, one, f_dot,a^v,p]
       and φ=ren(Collect_fm(1,?G,7))‘8‘8‘ren_G_aux_fn]
    by simp
qed

lemma ccc_fun_closed_lemma:
  assumes A∈M B∈M f_dot∈M p∈M
  shows (λa∈A. {b∈B. ∃ q∈P. q ≼ p ∧ (q ⊩ ·0‘1 is 2· [f_dot, a^v, b^v])}) ∈ M
proof -
  let ?app_fm=·0‘1 is 2·
  let ?ψ=(·∃··0 ∈ 1· ∧ · ·0 ≼^2 7· ∧ forces(?app_fm) ···)
  let ?G=(·∃··2^v 5 is 0· ∧ (·∃··2^v 6 is 0· ∧ ren(?ψ) ‘ 9 ‘ 9 ‘ ren_F_fn··)··)
 let ?Q=λ x b . (∃ q∈P. q ≼ p ∧ (M, [q, P, leq, one, f_dot, x^v, b^v] ⊨ forces(?app_fm)))
  have ?app_fm ∈ formula arity(?app_fm) = 3
    using arity_fun_apply_fm union_abs1
    by simp_all
  then
  have arity(forces(?app_fm)) ≤ 7
```

```
    using arity_forces[OF ‹?app_fm∈_›] by simp_all
  then
  have arity(?ψ) ≤ 9
    using arity_leq_fm union_abs2 union_abs1 le_trans
    by simp
  then
  have ren(?ψ)‘9‘9‘ren_F_fn ∈ formula pred(pred(arity(ren(?ψ)‘9‘9‘ren_F_fn)))
≤ 7
   using arity_ren ren_tc ren_F_thm check_fm_type leq_fm_type ren_F_fn_def
pred_le
    by simp_all
  with ‹arity(forces(_))≤7›
  have arity(?G) ≤ 7 ?G∈formula
    using check_fm_type arity_check_fm pred_Un_distrib Un_le
    by simp_all
  have pred(arity(?G)) ≤ 8 pred(arity(?G))∈nat
    using pred_le[OF _ ‹arity(?G)≤7›] le_trans pred_type[OF _ ‹arity(?G)≤7›]
    by simp_all
  note types=‹f_dot∈_› ‹p∈M› P_in_M leq_in_M one_in_M ‹A∈M› ‹B∈M›
  {fix x
    assume x∈A
    with ‹A∈M›
    have x∈M x^v∈M
      using transitivity[of x A]
      by simp_all
    {
      fix b
      assume b∈M
      then
      have b∈M
        using transitivity[OF _ ‹B∈_›] check_in_M by simp_all
      note types'=‹x∈A› ‹x∈M› ‹x^v∈M› types ‹b∈M›
      from types' ‹arity(forces(_))≤7›
      have
        ?Q(x,b) ⟷
          (∃ q∈P. q ≼ p ∧ (M, [q,P,leq,one,f_dot,x^v,b^v,p,x,b] ⊨ forces(?app_fm)))
        using arity_sats_iff[of _ [p,x,b] _ [_, _, _, _, _, _, _]] transitivity[of _
P]
        by simp
      also from types'
      have ... ⟷ (M, [P, leq, one, f_dot, x^v, b^v,p,x,b] ⊨ ?ψ)
        (is _ ⟷ _, ?η ⊨ _)
        unfolding leq_fm_def using transitivity[of _ P]
        by auto
      also from types' ‹arity(forces(_))≤_› ‹arity(?ψ)≤ 9› ren_F_fn_def
      have ... ⟷ M, [b^v, x^v, b, x, P, leq, one, f_dot, p] ⊨ ren(?ψ)‘9‘9‘ren_F_fn
        (is _ ⟷ _, ?η' ⊨ ?ψ')
        using sats_iff_sats_ren[of ?ψ 9 9 ?η M ?η']ren_F_thm(1)[where A=M]
ren_F_thm
```

```
      by auto
    also from assms types'
    have ... ⟷ M, [b,x,P, leq, one, f_dot,p] ⊨ ?G (is _ ⟷ M,?η''⊨_)
        using sats_check_fm[of 5 [_]@?η''] sats_check_fm[of 6 [_,_]@?η''] check_abs
      by simp
    ultimately
    have ?Q(x,b) ⟷ M,?η''⊨?G by simp
  }
  then
  have sats:(M,[b, x, P, leq, one, f_dot,p] ⊨?G) ⟷ ?Q(x,b) if b∈M for b
    using that by simp
  from types ‹x∈M› ‹arity(?G) ≤ 7› ‹?G∈_›
  have sep:{b∈B. ?Q(x,b)}∈M
    using sats Collect_in_M[where env=[x, P, leq, one, f_dot,p]]
    by simp
  note sats sep
 }
 moreover from this
 have sats:(M,[b, x, P, leq, one, f_dot,p] ⊨?G) ⟷ ?Q(x,b) if b∈M x∈A for b x
  using that by simp
 moreover from calculation
 have closed:{b∈B. ?Q(x,b)}∈M if x∈A for x using that by simp
 have sats':(M,[b, x,P, leq, one, f_dot,p,y,B] ⊨?G) ⟷ ?Q(x,b)
  if b∈M x∈A y∈M for b x y
 proof -
  from that ‹?G∈_› ‹arity(?G)≤7› types
  have (M,[b, x,P, leq, one, f_dot,p,y,B] ⊨?G) ⟷ M,[b, x,P, leq, one, f_dot,p]
⊨?G
    using transitivity[of x A] arity_sats_iff[of ?G [y,B] M [b, x,P, leq, one,
f_dot,p]]
    by simp
  then show ?thesis using sats that
    by simp
 qed
 from types
 have sats_col:(M,[x,P, leq, one, f_dot,p,y,B] ⊨Collect_fm(7,?G,6)) ⟷
          is_Collect(##M,B,λz.?Q(x,z),y) if x∈A y∈M for x y
  using that sats'[of _ x y] sats_Collect_fm[of M λb.?Q(x,b)] transitivity[of x A]
  by simp
 from ‹pred(arity(?G))∈nat› ‹?G∈_›
 have Collect_fm(7,?G,6) ∈ formula arity(Collect_fm(7,?G,6)) ≤ 8
  using union_abs2[OF ‹pred(arity(?G)) ≤ 8›] arity_Collect_fm union_abs2
  by simp_all
 then
 have arity(ren(Collect_fm(7,?G,6))‘8‘8‘ren_G_fn) ≤ 8
  ren(Collect_fm(7,?G,6))‘8‘8‘ren_G_fn ∈ formula (is ?Γ∈_)
  using ren_tc[of Collect_fm(7,?G,6)] ren_G_thm(2) ren_G_fn_def arity_ren
  by simp_all
```

from ‹*arity*(*Collect_fm*(7,?*G*,6))≤_› ‹*Collect_fm*(7,?*G*,6)∈_› *types*
have (*M*,[*x*,*y*,*P*, *leq*, *one*, *f_dot*,*p*,*B*] ⊨?Γ) ⟷
M,[*x*,*P*, *leq*, *one*, *f_dot*,*p*,*y*,*B*] ⊨ *Collect_fm*(7,?*G*,6) **if** *x*∈*A* *y*∈*M* **for** *x y*
using *sats_iff_sats_ren*[*of Collect_fm*(7,?*G*,6) *8 8 _ M* [*x*,*y*,*P*, *leq*, *one*, *f_dot*,*p*,*B*]]
ren_G_thm(1)[**where** *A*=*M*] *ren_G_thm*(2) *transitivity*[*of x A*] *that ren_G_fn_def*
by *simp*
then
have (*M*,[*x*,*y*,*P*, *leq*, *one*, *f_dot*,*p*,*B*] ⊨*ren*(*Collect_fm*(7,?*G*,6))‘8‘8‘*ren_G_fn*) ⟷
y = {*z*∈*B* . ?*Q*(*x*,*z*)} **if** *x*∈*A* *y*∈*M* **for** *y x*
using *that iff_trans*[*OF* _ *sats_col*]
Collect_abs types
by *auto*
with *assms types* ‹?Γ∈_› ‹*arity*(?Γ)≤_›
show *?thesis*
using *closed Lambda_in_M*[**where** *env*=[*P*, *leq*, *one*, *f_dot*,*p*,*B*] **and** *φ*=?Γ]
by *simp*
qed

— Kunen IV.3.5
lemma *ccc_fun_approximation_lemma*:
notes *le_trans*[*trans*]
assumes *ccc*^{M}(*P*,*leq*) *A*∈*M* *B*∈*M* *f*∈*M*[*G*] *f* : *A* → *B*
shows
∃ *F*∈*M*. *F* : *A* → *Pow*(*B*) ∧ (∀ *a*∈*A*. *f*‘*a* ∈ *F*‘*a* ∧ |*F*‘*a*|^M ≤ *ω*)
proof -
from ‹*f*∈*M*[*G*]›
obtain *f_dot* **where** *f* = *val*(*P*,*G*,*f_dot*) *f_dot*∈*M* **using** *GenExtD* **by** *force*
with *assms*
obtain *p* **where** *p* ⊩ ·0:1→2· [*f_dot*, *A*^{v}, *B*^{v}] *p*∈*G* *p*∈*M*
using *transitivity*[*OF M_genericD P_in_M*]
generic truth_lemma[*of* ·0:1→2· *G* [*f_dot*, *A*^{v}, *B*^{v}]]
by (*auto simp add*:*ord_simp_union arity_typed_function_fm*
— NOTE: type-checking is not performed here by the Simplifier
typed_function_type)
define *F* **where** *F*≡λ*a*∈*A*. {*b*∈*B*. ∃ *q*∈*P*. *q* ≼ *p* ∧ (*q* ⊩ ·0‘1 *is* 2· [*f_dot*, *a*^{v}, *b*^{v}])}
from *assms* ‹*f_dot*∈_› ‹*p*∈*M*›
have *F* ∈ *M*
unfolding *F_def* **using** *ccc_fun_closed_lemma* **by** *simp*
moreover
have *f*‘*a* ∈ *F*‘*a* **if** *a* ∈ *A* **for** *a*
proof -
note ‹*f*: *A* → *B*› ‹*a* ∈ *A*›
moreover from *this*
have *f* ‘ *a* ∈ *B* **by** *simp*
moreover

```
    note ‹f∈M[G]› ‹A∈M›
    moreover from calculation
    have M[G], [f, a, f‘a] ⊨ ·0‘1 is 2·
      by (auto dest:transM)
    moreover
    note ‹B∈M› ‹f = val(P,G,f_dot)›
    moreover from calculation
    have a∈M val(P,G, f_dot)‘a∈M
      by (auto dest:transM)
    moreover
    note ‹f_dot∈M› ‹p∈G›
    ultimately
    obtain q where q ≼ p q ⊩ ·0‘1 is 2· [f_dot, a^v, (f‘a)^v] q∈G
      using forces_below_filter[of ·0‘1 is 2· [f_dot, a^v, (f‘a)^v] p]
      by (auto simp add: ord_simp_union arity_fun_apply_fm
          fun_apply_type)
    with ‹f‘a ∈ B›
    have f‘a ∈ {b∈B . ∃ q∈P. q ≼ p ∧ q ⊩ ·0‘1 is 2· [f_dot, a^v, b^v]}
      by blast
    with ‹a∈A›
    show ?thesis unfolding F_def by simp
  qed
  moreover
  have |F‘a|^M ≤ ω if a ∈ A for a
  proof -
    let ?Q=λb. {q∈P. q ≼ p ∧ (q ⊩ ·0‘1 is 2· [f_dot, a^v, b^v])}
    from ‹F ∈ M› ‹a∈A› ‹A∈M›
    have F‘a ∈ M a∈M
      using transitivity[OF _ ‹A∈M›] by simp_all
    moreover
    have 2:⋀x. x∈F‘a ⟹ x∈M
      using transitivity[OF _ ‹F‘a∈M›] by simp
    moreover
    have 3:⋀x. x∈F‘a ⟹ (##M)(?Q(x))
    using ccc_fun_closed_lemma_aux[OF ‹f_dot∈M› ‹p∈M› ‹a∈M› 2] transitivity[of
_ F‘a]
      by simp
    moreover
    have 4:lam_replacement(##M,λb. {q ∈ P . q ≼ p ∧ (M, [q, P, leq, one, f_dot,
a^v, b^v] ⊨ forces(·0‘1 is 2· ))})
      using ccc_fun_closed_lemma_aux2[OF _ ‹f_dot∈M› ‹p∈M› ‹a∈M›]
        lam_replacement_iff_lam_closed[THEN iffD2]
        ccc_fun_closed_lemma_aux[OF ‹f_dot∈M› ‹p∈M› ‹a∈M›]
      by simp
    ultimately
    interpret M_Pi_assumptions_choice ##M F‘a ?Q
      using Pi_replacement1[OF _ 3] lam_replacement_Sigfun[OF 4]
        lam_replacement_imp_strong_replacement
          ccc_fun_closed_lemma_aux[OF ‹f_dot∈M› ‹p∈M› ‹a∈M›]
```

```
        lam_replacement_product
      lam_replacement_hcomp2[OF lam_replacement_constant 4 _ _ lam_replacement_minimum,unfolded
lam_replacement_def]
      by unfold_locales simp_all
  from ‹F‘a ∈ M›
  interpret M_Pi_assumptions2 ##M F‘a ?Q λ_ . P
    using P_in_M lam_replacement_imp_strong_replacement[OF
        lam_replacement_Sigfun[OF lam_replacement_constant]]
      Pi_replacement1 transM[of _ F‘a]
    by unfold_locales simp_all
  from ‹p ⊩ ·0:1→2· [f_dot, A^v, B^v]› ‹a∈A›
  have ∃ y. y ∈ ?Q(b) if b ∈ F‘a for b
    using that unfolding F_def by auto
  then
  obtain q where q ∈ Pi^M(F‘a,?Q) q∈M using AC_Pi_rel by auto
  moreover
  note ‹F‘a ∈ M›
  moreover from calculation
  have q : F‘a →^M P
    using Pi_rel_weaken_type def_function_space_rel by auto
  moreover from calculation
  have q : F‘a → range(q) q : F‘a → P q : F‘a →^M range(q)
    using mem_function_space_rel_abs range_of_fun by simp_all
  moreover
  have q‘b ⊥ q‘c if b ∈ F‘a c ∈ F‘a b ≠ c
   — For the next step, if the premise b ≠ c is first, the proof breaks down badly
    for b c
  proof -
    from ‹b ∈ F‘a› ‹c ∈ F‘a› ‹q ∈ Pi^M(F‘a,?Q)› ‹q∈M›
    have q‘b ⊩ ·0‘1 is 2· [f_dot, a^v, b^v]
      q‘c ⊩ ·0‘1 is 2· [f_dot, a^v, c^v]
      using mem_Pi_rel_abs[of q] apply_type[of _ _  ?Q]
      by simp_all
    with ‹b ≠ c› ‹q : F‘a → P› ‹a∈A› ‹b∈_› ‹c∈_›
      ‹A∈M› ‹f_dot∈M› ‹F‘a∈M›
    show ?thesis
      using forces_neq_apply_imp_incompatible
        transitivity[of _ A] transitivity[of _ F‘a]
      by auto
  qed
  moreover from calculation
  have antichain(range(q))
    using Pi_range_eq[of _  _ λ_ . P]
    unfolding antichain_def by auto
  moreover from this and ‹q∈M›
  have antichain_r'(range(q))
    by (simp add:absolut)
  moreover from calculation
  have q‘b ≠ q‘c if b ≠ c b ∈ F‘a c ∈ F‘a for b c
```

```
      using that Incompatible_imp_not_eq apply_type
        mem_function_space_rel_abs by simp
    ultimately
    have q ∈ inj^M(F‘a,range(q))
      using def_inj_rel by auto
    with ‹F‘a ∈ M› ‹q∈M›
    have |F‘a|^M ≤ |range(q)|^M
      using def_lepoll_rel
      by (rule_tac lepoll_rel_imp_cardinal_rel_le) auto
    also from ‹antichain_r'(range(q))› ‹ccc^M(P,leq)› ‹q∈M›
    have |range(q)|^M ≤ ω
      using def_ccc_rel by simp
    finally
    show ?thesis .
  qed
  moreover
  have F : A → Pow(B)
    unfolding F_def by (rule_tac lam_type) blast
  ultimately
  show ?thesis by auto
qed

end

end

end
theory Not_CH
  imports
    Cardinal_Preservation
begin

definition
  Add_subs :: [i,i] ⇒ i where
  Add_subs(κ,α) ≡ Fn(ω,κ×α,2)

locale M_master = M_cohen +
  assumes
  domain_separation: M(x) ⟹ separation(M, λz. x ∈ domain(z))
  and
  inj_dense_separation: M(x) ⟹ M(w) ⟹
    separation(M, λz. ∃n∈ω.    ⟨⟨w, n⟩, 1⟩ ∈ z ∧ ⟨⟨x, n⟩, 0⟩ ∈ z)
  and
  lam_apply_replacement: M(A) ⟹ M(f) ⟹
      strong_replacement(M, λx y. y = ⟨x, λn∈A. f ‘ ⟨x, n⟩⟩)
  and
  UN_lepoll_assumptions:
  M(A) ⟹ M(b) ⟹ M(f) ⟹ M(A') ⟹ separation(M, λy. ∃x∈A'. y = ⟨x, μ
i. x∈if_range_F_else_F(( ‘)(A), b, f, i)⟩)
```

begin

lemma (**in** *M_FiniteFun*) *Fn_nat_closed*:
 assumes $M(A)$ $M(B)$ **shows** $M(Fn(\omega,A,B))$
 using *assms Fn_nat_eq_FiniteFun*
 by *simp*

lemma *Aleph_rel2_closed*[*intro*,*simp*]: $M(\aleph_2{}^M)$
 using *nat_into_M*[*of 2*] *nat_into_Ord* **by** *simp*

end

locale *M_master_sub* = *M_master* + *N*:*M_master N* **for** *N* +
 assumes
  M_imp_N: $M(x) \Longrightarrow N(x)$ **and**
  Ord_iff: $Ord(x) \Longrightarrow M(x) \longleftrightarrow N(x)$

sublocale *M_master_sub* $\subseteq$ *M_N_Perm*
 using *M_imp_N* **by** *unfold_locales*

context *M_master_sub*
begin

lemma *cardinal_rel_le_cardinal_rel*: $M(X) \Longrightarrow |X|^N \leq |X|^M$
 using *M_imp_N N.lepoll_rel_cardinal_rel_le*[*OF lepoll_rel_transfer Card_rel_is_Ord*]
  cardinal_rel_eqpoll_rel[*THEN eqpoll_rel_sym*, *THEN eqpoll_rel_imp_lepoll_rel*]
 by *simp*

lemma *Aleph_rel_sub_closed*: $Ord(\alpha) \Longrightarrow M(\alpha) \Longrightarrow N(\aleph_\alpha{}^M)$
 using *Aleph_rel2_closed Ord_iff*[*THEN iffD1*,
  OF Card_rel_Aleph_rel[*THEN Card_rel_is_Ord*]]
 by *simp*

lemma *Card_rel_imp_Card_rel*: $M(\kappa) \Longrightarrow Card^N(\kappa) \Longrightarrow Card^M(\kappa)$
 using *N.Card_rel_is_Ord*[*of* κ] *M_imp_N Ord_cardinal_rel_le*[*of* κ]
  cardinal_rel_le_cardinal_rel[*of* κ] *le_anti_sym*
 unfolding *Card_rel_def* **by** *auto*

lemma *csucc_rel_le_csucc_rel*:
 assumes $Ord(\kappa)$ $M(\kappa)$
 shows $(\kappa^+)^M \leq (\kappa^+)^N$
proof -
 note *assms*
 moreover from *this*
 have $N(L) \wedge Card^N(L) \wedge \kappa < L \Longrightarrow M(L) \wedge Card^M(L) \wedge \kappa < L$
  (**is** *?P*$(L) \Longrightarrow$ *?Q*(L)) **for** *L*
  using *M_imp_N Ord_iff*[*THEN iffD2*, *of L*] *N.Card_rel_is_Ord lt_Ord*
   Card_rel_imp_Card_rel **by** *auto*

moreover from *assms*
have $N((\kappa^+)^N)\ Card^N((\kappa^+)^N)\ \kappa < (\kappa^+)^N$
using *N.lt_csucc_rel*[*of* κ] *N.Card_rel_csucc_rel*[*of* κ] *M_imp_N* **by** *simp_all*
ultimately
show *?thesis*
using *M_imp_N Least_antitone*[*of _ ?P ?Q*] **unfolding** *csucc_rel_def* **by** *blast*
qed

lemma *Aleph_rel_le_Aleph_rel*: $Ord(\alpha) \Longrightarrow M(\alpha) \Longrightarrow \aleph_\alpha{}^M \leq \aleph_\alpha{}^N$
proof (*induct rule:trans_induct3*)
case *0*
then
show *?case*
using *Aleph_rel_zero N.Aleph_rel_zero* **by** *simp*
next
case (*succ x*)
then
have $\aleph_x{}^M \leq \aleph_x{}^N\ Ord(x)\ M(x)$ **by** *simp_all*
moreover from *this*
have $(\aleph_x{}^{M+})^M \leq (\aleph_x{}^{N+})^M$
using *M_imp_N Ord_iff*[*THEN iffD2*, *OF N.Card_rel_is_Ord*]
by (*intro csucc_rel_le_mono*) *simp_all*
moreover from *calculation*
have $(\aleph_x{}^{N+})^M \leq (\aleph_x{}^{N+})^N$
using *M_imp_N N.Card_rel_is_Ord Ord_iff*[*THEN iffD2*, *OF N.Card_rel_is_Ord*]
by (*intro csucc_rel_le_csucc_rel*) *auto*
ultimately
show *?case*
using *M_imp_N Aleph_rel_succ N.Aleph_rel_succ csucc_rel_le_csucc_rel le_trans* **by** *auto*
next
case (*limit x*)
then
show *?case*
using *M_imp_N Aleph_rel_limit N.Aleph_rel_limit*
by *simp* (*blast dest*: *transM intro*!:*le_implies_UN_le_UN*)
qed

end

lemmas (**in** *M_ZFC_trans*) *sep_instances* =
separation_toplevel1_body separation_toplevel2_body separation_toplevel3_body
separation_toplevel4_body separation_toplevel5_body separation_toplevel6_body
separation_toplevel7_body separation_toplevel8_body separation_toplevel9_body
separation_toplevel10_body separation_toplevel11_body separation_Ord
separation_toplevel12_body separation_insnd_ballPair
separation_restrict_eq_dom_eq separation_restrict_eq_dom_eq_pair
separation_ifrangeF_body separation_ifrangeF_body2 separation_ifrangeF_body3

separation_ifrangeF_body4 separation_ifrangeF_body5 separation_ifrangeF_body6 separation_ifrangeF_body7

lemmas (**in** *M_ZF_trans*) *repl_instances = lam_replacement_inj_rel lam_replacement_cardinal*[*unfolded lam_replacement_def*] *replacement_trans_apply_image replacement_abs_apply_pair*

sublocale *M_ZFC_trans* ⊆ *M_master* ##*M*
by *unfold_locales* (*simp_all add*:*repl_instances sep_instances del*:*setclass_iff*)

context *M_ctm_AC*
begin

— FIXME: using notation as if *Add_subs* were used
lemma *ccc_Add_subs_Aleph_2*: $ccc^M(Fn(\omega,\aleph_2{}^M \times \omega,2),Fnle(\omega,\aleph_2{}^M \times \omega,2))$
proof -
interpret *M_add_reals* ##*M* $\aleph_2{}^M \times \omega$
by *unfold_locales blast*
show *?thesis*
using *ccc_rel_Fn_nat* **by** *fast*
qed

end

sublocale *G_generic_AC* ⊆ *M_master_sub* ##*M* ##(*M*[*G*])
using *M_subset_MG*[*OF one_in_G*] *generic Ord_MG_iff*
by *unfold_locales auto*

lemma (**in** *M_trans*) *mem_F_bound4*:
fixes *F A*
defines *F* ≡ (‘)
shows *x*∈*F*(*A*,*c*) ⟹ *c* ∈ (*range*(*f*) ∪ *domain*(*A*))
using *apply_0* **unfolding** *F_def*
by (*cases M*(*c*), *auto simp*:*F_def*)

lemma (**in** *M_trans*) *mem_F_bound5*:
fixes *F A*
defines *F* ≡ λ_ *x*. *A*‘*x*
shows *x*∈*F*(*A*,*c*) ⟹ *c* ∈ (*range*(*f*) ∪ *domain*(*A*))
using *apply_0* **unfolding** *F_def*
by (*cases M*(*c*), *auto simp*:*F_def drSR_Y_def dC_F_def*)

context *G_generic_AC* **begin**

context
includes *G_generic_lemmas*
begin

lemma *G_in_MG*: $G \in M[G]$
using *G_in_Gen_Ext*[*OF _ one_in_G*, *OF _ generic*]
by *blast*

lemma *ccc_preserves_Aleph_succ*:
assumes $ccc^M(P,leq)$ $Ord(z)$ $z \in M$
shows $Card^{M[G]}({\aleph_{succ(z)}}^M)$
proof (*rule ccontr*)
assume $\neg$ $Card^{M[G]}({\aleph_{succ(z)}}^M)$
moreover
note ‹$z \in M$›
moreover from *this* **and** ‹$Ord(z)$›
have $Ord({\aleph_{succ(z)}}^M)$
using *Card_rel_Aleph_rel*[*of* $succ(z)$, *THEN Card_rel_is_Ord*]
by *fastforce*
ultimately
obtain α f **where** $\alpha < {\aleph_{succ(z)}}^M$ $f \in surj^{M[G]}(\alpha, {\aleph_{succ(z)}}^M)$
using *ext.lt_surj_rel_empty_imp_Card_rel M_subset_MG*[*OF one_in_G*, *OF generic*]
Aleph_rel_closed[*of* $succ(z)$] ‹$Ord(z)$› **by** *simp blast*
moreover from *this* **and** ‹$z \in M$› ‹$Ord(z)$›
have $\alpha \in M$ $f \in M[G]$
using *Aleph_rel_closed*[*of* $succ(z)$] *ext.trans_surj_rel_closed*
by (*auto dest*:*transM ext.transM dest*!:*ltD*)
moreover
note ‹$ccc^M(P,leq)$› ‹$z \in M$›
ultimately
obtain F **where** $F:\alpha \to Pow({\aleph_{succ(z)}}^M)$ $\forall \beta \in \alpha.\ f`\beta \in F`\beta$ $\forall \beta \in \alpha.\ |F`\beta|^M \leq \omega$
$F \in M$
using *ccc_fun_approximation_lemma*[*of* α ${\aleph_{succ(z)}}^M$ f]
ext.mem_surj_abs[*of* f α ${\aleph_{succ(z)}}^M$] ‹$Ord(z)$›
Aleph_rel_closed[*of* $succ(z)$] *surj_is_fun*[*of* f α ${\aleph_{succ(z)}}^M$] **by** *auto*
then
have $\beta \in \alpha \Longrightarrow |F`\beta|^M \leq {\aleph_0}^M$ **for** β
using *Aleph_rel_zero* **by** *simp*
interpret *M_replacement_lepoll* ##*M* (`)
using *UN_lepoll_assumptions lam_replacement_apply lam_replacement_inj_rel*
mem_F_bound4 apply_0
unfolding *lepoll_assumptions_defs*
proof (*unfold_locales*,
rule_tac [*3*] *lam_Least_assumption_general*[**where** *U*=*domain*, *OF _ mem_F_bound4*], *simp_all*)
fix A i x
assume $A \in M$ $x \in M$ $x \in A$ ` i
then
show $i \in M$
using *apply_0*[*of* i A] *transM*[*of _* *domain*(A), *simplified*]
by *force*

```
  qed
  from ‹α ∈ M› ‹F:α→Pow(ℵ_succ(z)^M)› ‹F∈M›
  interpret M_cardinal_UN_lepoll ##M λβ. F‘β α
    using Aleph_rel_closed[of 0] UN_lepoll_assumptions lepoll_assumptions
      lam_replacement_apply lam_replacement_inj_rel
  proof (unfold_locales, auto dest:transM simp del:if_range_F_else_F_def)
    show w ∈ F ‘ x ⟹ x ∈ M for w x
    proof -
      fix w x
      assume w ∈ F‘x
      then
      have x ∈ domain(F)
        using apply_0 by auto
      with ‹F:α→Pow(ℵ_succ(z)^M)›
      have x ∈ α
        using domain_of_fun by simp
      with ‹α ∈ M›
      show x ∈ M by (auto dest:transM)
    qed
   show w ∈ F ‘ x ⟹ x ∈ M for w x — FIXME: why two times?
    proof -
      fix w x
      assume w ∈ F‘x
      then
      have x ∈ domain(F)
        using apply_0 by auto
      with ‹F:α→Pow(ℵ_succ(z)^M)›
      have x ∈ α
        using domain_of_fun by simp
      with ‹α ∈ M›
      show x ∈ M by (auto dest:transM)
    qed
  next
    fix f b
    assume b∈M f∈M
    with ‹F∈M›
    show lam_replacement(##M, λx. μ i. x ∈ if_range_F_else_F((‘)(F), b, f,
i))
      using UN_lepoll_assumptions mem_F_bound5
        by (rule_tac lam_Least_assumption_general[where U=domain, OF _
mem_F_bound5])
       simp_all
  qed
  from ‹α < ℵ_succ(z)^M› ‹α ∈ M› assms
  have α ≲^M ℵ_z^M
    using
      Aleph_rel_zero
      cardinal_rel_lt_csucc_rel_iff[of ℵ_z^M α]
      le_Card_rel_iff[of ℵ_z^M α]
```

```
    Aleph_rel_succ[of z] Card_rel_lt_iff[of α ℵ_succ(z)^M]
    lt_Ord[of α ℵ_succ(z)^M]
    csucc_rel_closed[of ℵ_z^M] Card_rel_csucc_rel[of ℵ_z^M]
    Aleph_rel_closed[of z]
    Card_rel_Aleph_rel[THEN Card_rel_is_Ord, OF _ _ Aleph_rel_closed]
  by simp
with ⟨α < ℵ_succ(z)^M⟩ ⟨∀β∈α. |F‘β|^M ≤ ω⟩ ⟨α ∈ M⟩ assms
have |⋃β∈α. F‘β|^M ≤ ℵ_z^M
  using InfCard_rel_Aleph_rel[of z] Aleph_rel_zero
    subset_imp_lepoll_rel[THEN lepoll_rel_imp_cardinal_rel_le,
      of ⋃β∈α. F‘β ℵ_z^M]
    Aleph_rel_closed[of z] Aleph_rel_succ
    Aleph_rel_increasing[THEN leI, THEN [2] le_trans, of _ 0 z]
    Ord_0_lt_iff[THEN iffD1, of z]
  by (cases 0<z; rule_tac leqpoll_rel_imp_cardinal_rel_UN_le) (auto, force)
moreover
note ⟨z∈M⟩ ⟨Ord(z)⟩
moreover from ⟨∀β∈α. f‘β ∈ F‘β⟩ ⟨f ∈ surj^M[G](α, ℵ_succ(z)^M)⟩
  ⟨α ∈ M⟩ ⟨f ∈ M[G]⟩ and this
have ℵ_succ(z)^M ⊆ (⋃β∈α. F‘β)
  using Aleph_rel_closed[of succ(z)] ext.mem_surj_abs
  by (force simp add:surj_def)
moreover from ⟨F ∈ M⟩ ⟨α ∈ M⟩
have (⋃x∈α. F ‘ x) ∈ M
 using j.B_replacement— NOTE: it didn't require (##M)(⋃i∈α. F ‘ i) before!
  by (intro Union_closed[simplified] RepFun_closed[simplified])
    (auto dest:transM)
ultimately
have ℵ_succ(z)^M ≤ ℵ_z^M
  using subset_imp_le_cardinal_rel[of ℵ_succ(z)^M ⋃β∈α. F‘β]
    Aleph_rel_closed[of succ(z)] le_trans
  by auto
with assms
show False
  using Aleph_rel_increasing not_le_iff_lt[of ℵ_succ(z)^M ℵ_z^M]
    Card_rel_Aleph_rel[THEN Card_rel_is_Ord, OF _ _ Aleph_rel_closed]
  by auto
qed

end

end

context M_ctm
begin

abbreviation
  Add :: i where
```

Add ≡ *Fn*(ω, $\aleph_2{}^M$ × ω, *2*)

end

locale *add_generic* = *G_generic_AC Fn*(ω, $\aleph_2{}^{\#\#M}$ × ω, *2*) *Fnle*(ω, $\aleph_2{}^{\#\#M}$ × ω, *2*) *0*

sublocale *add_generic* ⊆ *cohen_data* ω $\aleph_2{}^M$ × ω *2* **by** *unfold_locales auto*

context *add_generic*
begin

notation *Leq* (**infixl** ≼ *50*)
notation *Incompatible* (**infixl** ⊥ *50*)
notation *GenExt_at_P* (_[_] [*71*,*1*])

lemma *Add_subs_preserves_Aleph_succ*: *Ord*(*z*) ⟹ *z*∈*M* ⟹ *Card*$^{M[G]}$($\aleph_{succ(z)}{}^M$)
  using *ccc_preserves_Aleph_succ ccc_Add_subs_Aleph_2*
  by *auto*

lemma *Aleph_rel_nats_MG_eq_Aleph_rel_nats_M*:
  includes *G_generic_lemmas*
  assumes *z* ∈ ω
  shows $\aleph_z{}^{M[G]} = \aleph_z{}^M$
  using *assms*
proof (*induct*)
  case *0*
  have $\aleph_0{}^{M[G]} = \omega$
    using *ext.Aleph_rel_zero* **.**
  also
  have ω = $\aleph_0{}^M$
    using *Aleph_rel_zero* **by** *simp*
  finally
  show *?case* **.**
next
  case (*succ z*)
  then
  have $\aleph_{succ(z)}{}^M \leq \aleph_{succ(z)}{}^{M[G]}$
    using *Aleph_rel_le_Aleph_rel nat_into_M* **by** *simp*
  moreover from ‹*z* ∈ ω›
  have $\aleph_z{}^M \in M[G]$ $\aleph_{succ(z)}{}^M \in M[G]$
    using *Aleph_rel_closed nat_into_M* **by** *simp_all*
  moreover from *this* **and** ‹$\aleph_z{}^{M[G]} = \aleph_z{}^M$› ‹*z* ∈ ω›
  have $\aleph_{succ(z)}{}^{M[G]} \leq \aleph_{succ(z)}{}^M$
    using *ext.Aleph_rel_succ nat_into_M*
      Add_subs_preserves_Aleph_succ[*THEN ext.csucc_rel_le*, *of z*]
      Aleph_rel_increasing[*of z succ*(*z*)]
    by *simp*
  ultimately

show *?case* **using** *le_anti_sym* **by** *blast*
qed

abbreviation
f_G :: *i* (‹f_G›) **where**
$f_G \equiv \bigcup G$

abbreviation
dom_dense :: *i⇒i* **where**
dom_dense(*x*) ≡ { *p∈Add* . *x* ∈ *domain*(*p*) }

— FIXME write general versions of this for *Fn*(ω, *I*, *J*) in a context with a generic filter for it
lemma *dense_dom_dense*: $x \in \aleph_2^M \times \omega \Longrightarrow$ *dense*(*dom_dense*(*x*))
proof
fix *p*
assume $x \in \aleph_2^M \times \omega$ *p* ∈ *Add*
show ∃ *d∈dom_dense*(*x*). *d* ≼ *p*
proof (*cases x* ∈ *domain*(*p*))
case *True*
with ‹$x \in \aleph_2^M \times \omega$› ‹*p* ∈ *Add*›
show *?thesis* **using** *refl_leq* **by** *auto*
next
case *False*
note ‹*p* ∈ *Add*›
moreover from *this* **and** *False* **and** ‹$x \in \aleph_2^M \times \omega$›
have *cons*(⟨*x*,*0*⟩, *p*) ∈ *Add*
using *FiniteFun.consI*[*of x* $\aleph_2^M \times \omega$ *0 2 p*]
Fn_nat_eq_FiniteFun **by** *auto*
moreover from ‹*p* ∈ *Add*›
have *x*∈*domain*(*cons*(⟨*x*,*0*⟩, *p*)) **by** *simp*
ultimately
show *?thesis*
by (*fastforce del*:*FnD*)
qed
qed

lemma *dom_dense_closed*[*intro*,*simp*]: $x \in \aleph_2^M \times \omega \Longrightarrow$ *dom_dense*(*x*) ∈ *M*
using *Fn_nat_closed Aleph_rel2_closed domain_separation*[*of x*] *nat_into_M*
by (*rule_tac separation_closed*[*simplified*], *blast dest*:*transM*) *simp*

lemma *domain_f_G*: **assumes** $x \in \aleph_2^M$ *y* ∈ ω
shows ⟨*x*, *y*⟩ ∈ *domain*(f_G)
proof -
from *assms*
have *dense*(*dom_dense*(⟨*x*, *y*⟩)) **using** *dense_dom_dense* **by** *simp*
with *assms*
obtain *p* **where** *p*∈*dom_dense*(⟨*x*, *y*⟩) *p*∈*G*

using *generic*[*THEN M_generic_denseD*, *of dom_dense*(⟨*x*, *y*⟩)]
by *auto*
then
show ⟨*x*, *y*⟩ ∈ *domain*(f_G) **by** *blast*
qed

— MOVE THIS to `Cohen_Posets.thy`
lemma *Fn_nat_subset_Pow*: *Fn*(ω,*I*,*J*) ⊆ *Pow*(*I*×*J*)
using *subset_trans*[*OF FiniteFun.dom_subset Fin.dom_subset*]
Fn_nat_eq_FiniteFun **by** *simp*

lemma *f_G_funtype*:
includes *G_generic_lemmas*
shows $f_G : \aleph_2^M \times \omega \to 2$
using *generic domain_f_G*
unfolding *Pi_def*
proof (*auto*)
show *x* ∈ *B* ⟹ *B* ∈ *G* ⟹ *x* ∈ ($\aleph_2^M$ × ω) × *2* **for** *B x*
using *Fn_nat_subset_Pow* **by** *blast*
show *function*(f_G)
using *Un_filter_is_function generic*
unfolding *M_generic_def* **by** *fast*
qed

abbreviation
inj_dense :: *i*⇒*i*⇒*i* **where**
inj_dense(*w*,*x*) ≡
{ *p*∈*Add* . (∃ *n*∈ω. <<*w*,*n*>,*1*> ∈ *p* ∧ <<*x*,*n*>,*0*> ∈ *p*) }

— FIXME write general versions of this for *Fn*(ω, *I*, *J*) in a context with a generic filter for it
lemma *dense_inj_dense*:
assumes *w* ∈ $\aleph_2^M$ *x* ∈ $\aleph_2^M$ *w* ≠ *x*
shows *dense*(*inj_dense*(*w*,*x*))
proof
fix *p*
assume *p* ∈ *Add*
then
obtain *n* **where** <*w*,*n*> ∉ *domain*(*p*) <*x*,*n*> ∉ *domain*(*p*) *n* ∈ ω
proof -
{
assume <*w*,*n*> ∈ *domain*(*p*) ∨ <*x*,*n*> ∈ *domain*(*p*) **if** *n* ∈ ω **for** *n*
then
have ω ⊆ *range*(*domain*(*p*)) **by** *blast*
then
have ¬ *Finite*(*p*)
using *Finite_range Finite_domain subset_Finite nat_not_Finite*
by *auto*
with ‹*p* ∈ *Add*›

```
      have False
        using Fn_nat_eq_FiniteFun FiniteFun.dom_subset[of ℵ₂^M × ω 2]
          Fin_into_Finite by auto
    }
    with that— the shape of the goal puts assumptions in this variable
    show ?thesis by auto
  qed
  moreover
  note ‹p ∈ Add› assms
  moreover from calculation
  have cons(⟨⟨x,n⟩,0⟩, p) ∈ Add
    using FiniteFun.consI[of ⟨x,n⟩ ℵ₂^M × ω 0 2 p]
      Fn_nat_eq_FiniteFun by auto
  ultimately
  have cons(⟨⟨w,n⟩,1⟩, cons(⟨⟨x,n⟩,0⟩, p) ) ∈ Add
    using FiniteFun.consI[of ⟨w,n⟩ ℵ₂^M × ω 1 2 cons(⟨⟨x,n⟩,0⟩, p)]
      Fn_nat_eq_FiniteFun by auto
  with ‹n ∈ ω›
  show ∃ d∈inj_dense(w,x). d ≼ p
    using ‹p ∈ Add› by (intro bexI) auto
qed

lemma inj_dense_closed[intro,simp]:
  w ∈ ℵ₂^M ⟹ x ∈ ℵ₂^M ⟹ inj_dense(w,x) ∈ M
  using Fn_nat_closed Aleph_rel2_closed domain_separation[of x] nat_into_M
    inj_dense_separation transM[OF _ Aleph_rel2_closed]
  by (rule_tac separation_closed[simplified]; simp_all)

lemma Aleph_rel2_new_reals:
  assumes w ∈ ℵ₂^M x ∈ ℵ₂^M w ≠ x
  shows (λn∈ω. f_G ‘ ⟨w, n⟩) ≠ (λn∈ω. f_G ‘ ⟨x, n⟩)
proof -
  from assms
  have dense(inj_dense(w,x)) using dense_inj_dense by simp
  with assms
  obtain p where p∈inj_dense(w,x) p∈G
    using generic[THEN M_generic_denseD, of inj_dense(w,x)]
    by blast
  then
  obtain n where n ∈ ω ⟨⟨w, n⟩, 1⟩ ∈ p ⟨⟨x, n⟩, 0⟩ ∈ p
    by blast
  moreover from this and ‹p∈G›
  have ⟨⟨w, n⟩, 1⟩ ∈ f_G ⟨⟨x, n⟩, 0⟩ ∈ f_G by auto
  moreover from calculation
  have f_G ‘ ⟨w, n⟩ = 1 f_G ‘ ⟨x, n⟩ = 0
    using f_G_funtype apply_equality
    by auto
  ultimately
  have (λn∈ω. f_G ‘ ⟨w, n⟩) ‘ n ≠ (λn∈ω. f_G ‘ ⟨x, n⟩) ‘ n
```

```
    by simp
  then
  show ?thesis by fastforce
qed

definition
  h_G :: i (‹h_G›) where
  h_G ≡ λα∈ℵ_2^M. λn∈ω. f_G‘<α,n>

lemma h_G_in_MG[simp]:
  includes G_generic_lemmas
  shows h_G ∈ M[G]
  using Aleph_rel2_closed
    ext.lam_apply_replacement ext.apply_replacement2
    ext.Union_closed[simplified, OF G_in_MG]
    — The "simplified" here is because of the setclass ocurrences
    ext.nat_in_M Aleph_rel2_closed ext.nat_into_M
  unfolding h_G_def
  by (rule_tac ext.lam_closed[simplified] | auto dest:transM)+

lemma h_G_inj_Aleph_rel2_reals: h_G ∈ inj^M[G](ℵ_2^M, ω →^M[G] 2)
  using Aleph_rel_sub_closed
proof (intro ext.mem_inj_abs[THEN iffD2])
  show h_G ∈ inj(ℵ_2^M, ω →^M[G] 2)
    unfolding inj_def
  proof (intro ballI CollectI impI)
    show h_G ∈ ℵ_2^M → ω →^M[G] 2
      using f_G_funtype G_in_MG ext.nat_into_M unfolding h_G_def
        apply (intro lam_type ext.mem_function_space_rel_abs[THEN iffD2],
simp_all)
      apply (rule_tac ext.lam_closed[simplified], simp_all)
       apply (rule ext.apply_replacement2)
        apply (auto dest:ext.transM[OF _ Aleph_rel_sub_closed])
      done
    fix w x
    assume w ∈ ℵ_2^M x ∈ ℵ_2^M h_G ‘ w = h_G ‘ x
    then
    show w = x
      unfolding h_G_def using Aleph_rel2_new_reals by auto
  qed
qed simp_all

lemma Aleph2_extension_le_continuum_rel:
  includes G_generic_lemmas
  shows ℵ_2^M[G] ≤ 2↑ℵ_0^M[G],M[G]
proof -
  have ℵ_2^M ∈ M[G] Ord(ℵ_2^M)
    using Card_rel_Aleph_rel[THEN Card_rel_is_Ord, of 2]
      Aleph_rel2_closed
```

```
    by auto
  moreover from this
  have ℵ_2^M ≲^M[G] ω →^M[G] 2
    using ext.def_lepoll_rel[of ℵ_2^M ω →^M[G] 2,
        OF _ ext.function_space_rel_closed]
      h_G_inj_Aleph_rel2_reals h_G_in_MG ext.nat_into_M ext.M_nat
    by auto
  moreover from calculation
  have ℵ_2^M ≲^M[G] |ω →^M[G] 2|^M[G]
    using ext.lepoll_rel_imp_lepoll_rel_cardinal_rel by simp
  ultimately
  have |ℵ_2^M|^M[G] ≤ 2^↑ℵ_0^M[G],M[G]
    using ext.lepoll_rel_imp_cardinal_rel_le[of ℵ_2^M ω →^M[G] 2,
        OF _ _ ext.function_space_rel_closed] ext.nat_into_M ext.M_nat
      ext.Aleph_rel_zero Aleph_rel_nats_MG_eq_Aleph_rel_nats_M
    unfolding cexp_rel_def by simp
  then
  show ℵ_2^M[G] ≤ 2^↑ℵ_0^M[G],M[G]
    using Aleph_rel_nats_MG_eq_Aleph_rel_nats_M
      ext.Card_rel_Aleph_rel[of 2, THEN ext.Card_rel_cardinal_rel_eq]
      ext.Aleph_rel2_closed
    by simp
qed

lemma Aleph_rel_lt_continuum_rel: ℵ_1^M[G] < 2^↑ℵ_0^M[G],M[G]
  using Aleph2_extension_le_continuum_rel
    ext.Aleph_rel_increasing[of 1 2] le_trans by auto

corollary not_CH: ℵ_1^M[G] ≠ 2^↑ℵ_0^M[G],M[G]
  using Aleph_rel_lt_continuum_rel by auto

end

definition
  ContHyp :: o where
  ContHyp ≡ ℵ_1 = 2^↑ℵ_0

relativize functional ContHyp ContHyp_rel
notation ContHyp_rel (‹CH—›)
relationalize ContHyp_rel is_ContHyp

context M_master
begin

is_iff_rel for ContHyp
 using is_cexp_iff is_Aleph_iff[of 0] is_Aleph_iff[of 1] unfolding is_ContHyp_def
ContHyp_rel_def
  by auto (rule_tac x=0 in rexI, auto)
```

end

synthesize *is_ContHyp* **from_definition assuming** *nonempty*
arity_theorem for *is_ContHyp_fm*

notation *is_ContHyp_fm* (‹·CH·›)

theorem *ctm_of_not_CH*:
  assumes
    $M \approx \omega$ *Transset*(M) $M \models ZFC$
  shows
    $\exists N.$
      $M \subseteq N \wedge N \approx \omega \wedge$ *Transset*(N) $\wedge\ N \models ZFC \cup \{\cdot\neg\cdot CH\cdot\cdot\} \wedge$
      $(\forall \alpha.$ *Ord*(α) $\longrightarrow (\alpha \in M \longleftrightarrow \alpha \in N))$
proof -
  from ‹$M \models ZFC$›
  interpret *M_ZFC M*
    using *M_ZFC_iff_M_satT*
    by *simp*
  from ‹*Transset*(M)›
  interpret *M_ZF_trans M*
    using *M_ZF_iff_M_satT*
    by *unfold_locales*
  from ‹$M \approx \omega$›
  obtain *enum* **where** *enum* $\in$ *bij*(ω,M)
    using *eqpoll_sym* **unfolding** *eqpoll_def* **by** *blast*
  then
  interpret *M_ctm_AC M enum* **by** *unfold_locales*
  interpret *forcing_data Fn*(ω,$\aleph_2{}^M \times \omega$,*2*) *Fnle*($\omega$,$\aleph_2{}^M \times \omega$,*2*) *0 M enum*
  proof -
    interpret *cohen_data* ω $\aleph_2{}^M \times \omega$ *2* **by** *unfold_locales auto*
    show *forcing_data*(*Fn*(ω, $\aleph_2{}^M \times \omega$, *2*), *Fnle*($\omega$, $\aleph_2{}^M \times \omega$, *2*), *0*, *M*, *enum*)
      using *nat_into_M*[*of 2*] *M_nat*
        Fn_nat_closed[*OF cartprod_closed*, *OF Aleph_rel_closed*, *of 2* ω *2*]
        Fnle_nat_closed[*OF cartprod_closed*, *OF Aleph_rel_closed*, *of 2* ω *2*]
      by (*unfold_locales*, *simp_all*)
  qed
  obtain *G* **where** *M_generic*(*G*)
    using *generic_filter_existence*[*OF one_in_P*]
    by *auto*
  moreover from *this*
  interpret *add_generic M enum G* **by** *unfold_locales*
  have $\neg$ ($\aleph_1{}^{M[G]} = 2^{\uparrow \aleph_0{}^{M[G]}, M[G]}$)
    using *not_CH* **.**
  then
  have $M[G]$, [] $\models \cdot\neg\cdot CH\cdot\cdot$
    using *ext.is_ContHyp_iff*
    by (*simp add*:*ContHyp_rel_def*)
  then

have $M[G] \models ZFC \cup \{\cdot\neg\cdot CH\cdot\cdot\}$
using *M_ZFC_iff_M_satT*[*of* $M[G]$] *ext.M_ZFC_axioms* **by** *auto*
moreover
have *Transset*($M[G]$) **using** *Transset_MG* **.**
moreover
have $M \subseteq M[G]$ **using** *M_subset_MG*[*OF one_in_G*] *generic* **by** *simp*
ultimately
show *?thesis*
using *Ord_MG_iff MG_eqpoll_nat*
by (*rule_tac x*=$M[G]$ **in** *exI*, *simp*)
qed

end

49 From M to V

theory *Absolute_Versions*
imports
Not_CH
ZF.Cardinal_AC
begin

49.1 Locales of a class M hold in $\mathcal{V}$

interpretation *V*: *M_trivial* $\mathcal{V}$
using *Union_ax_absolute upair_ax_absolute*
by *unfold_locales auto*

lemmas *bad_simps* = *V.nonempty V.Forall_in_M_iff V.Inl_in_M_iff V.Inr_in_M_iff*
V.succ_in_M_iff V.singleton_in_M_iff V.Equal_in_M_iff V.Member_in_M_iff
V.Nand_in_M_iff
V.Cons_in_M_iff V.pair_in_M_iff V.upair_in_M_iff

lemmas *bad_M_trivial_simps*[*simp del*] = *V.Forall_in_M_iff V.Equal_in_M_iff*
V.nonempty

lemmas *bad_M_trivial_rules*[*rule del*] = *V.pair_in_MI V.singleton_in_MI*
V.pair_in_MD V.nat_into_M
V.depth_closed V.length_closed V.nat_case_closed V.separation_closed
V.Un_closed V.strong_replacement_closed V.nonempty

interpretation *V*:*M_basic* $\mathcal{V}$
using *power_ax_absolute separation_absolute replacement_absolute*
by *unfold_locales auto*

interpretation *V*:*M_eclose* $\mathcal{V}$
by *unfold_locales* (*auto intro*:*separation_absolute replacement_absolute*
simp:*iterates_replacement_def wfrec_replacement_def*)

lemmas *bad_M_basic_rules*[*simp del*, *rule del*] =
V.cartprod_closed V.finite_funspace_closed V.converse_closed
V.list_case'_closed V.pred_closed

interpretation *V*:*M_cardinal_arith* $\mathcal{V}$
by *unfold_locales* (*auto intro*:*separation_absolute replacement_absolute*
simp add:*iterates_replacement_def wfrec_replacement_def lam_replacement_def*)

lemmas *bad_M_cardinals_rules*[*simp del*, *rule del*] =
V.iterates_closed V.M_nat V.trancl_closed V.rvimage_closed

interpretation *V*:*M_cardinal_arith_jump* $\mathcal{V}$
by *unfold_locales* (*auto intro*:*separation_absolute replacement_absolute*
simp:*wfrec_replacement_def*)

lemma *choice_ax_Universe*: *choice_ax*($\mathcal{V}$)
proof -
{
fix *x*
obtain *f* **where** $f \in surj(|x|,x)$
using *cardinal_eqpoll* **unfolding** *eqpoll_def bij_def* **by** *fast*
moreover
have *Ord*($|x|$) **by** *simp*
ultimately
have $\exists\, a.\ Ord(a) \wedge (\exists f.\ f \in surj(a,x))$
by *fast*
}
then
show *?thesis* **unfolding** *choice_ax_def rall_def rex_def*
by *simp*
qed

interpretation *V*:*M_master* $\mathcal{V}$
using *choice_ax_Universe*
by *unfold_locales* (*auto intro*:*separation_absolute replacement_absolute*
simp:*lam_replacement_def transrec_replacement_def wfrec_replacement_def*
is_wfrec_def M_is_recfun_def)

named_theorems *V_simps*

— To work systematically, ASCII versions of "_absolute" theorems as those below are preferable.
lemma *eqpoll_rel_absolute*[*V_simps*]: $x \approx^{\mathcal{V}} y \longleftrightarrow x \approx y$
unfolding *eqpoll_def* **using** *V.def_eqpoll_rel* **by** *auto*

lemma *cardinal_rel_absolute*[*V_simps*]: $|x|^{\mathcal{V}} = |x|$
unfolding *cardinal_def cardinal_rel_def* **by** (*simp add*:*V_simps*)

lemma *Card_rel_absolute*[*V_simps*]:$Card^{\mathcal{V}}(a) \longleftrightarrow Card(a)$
unfolding *Card_rel_def Card_def* **by** (*simp add*:*V_simps*)

lemma *csucc_rel_absolute*[*V_simps*]:$(a^+)^{\mathcal{V}} = a^+$
unfolding *csucc_rel_def csucc_def* **by** (*simp add*:*V_simps*)

lemma *function_space_rel_absolute*[*V_simps*]:$x \to^{\mathcal{V}} y = x \to y$
using *V.function_space_rel_char* **by** (*simp add*:*V_simps*)

lemma *cexp_rel_absolute*[*V_simps*]:$x^{\uparrow y,\mathcal{V}} = x^{\uparrow y}$
unfolding *cexp_rel_def cexp_def* **by** (*simp add*:*V_simps*)

lemma *HAleph_rel_absolute*[*V_simps*]:*HAleph_rel*($\mathcal{V}$,*a*,*b*) = *HAleph*(*a*,*b*)
unfolding *HAleph_rel_def HAleph_def* **by** (*auto simp add*:*V_simps*)

lemma *Aleph_rel_absolute*[*V_simps*]: $Ord(x) \Longrightarrow \aleph_x{}^{\mathcal{V}} = \aleph_x$
proof -
assume $Ord(x)$
have $\aleph_x{}^{\mathcal{V}} = transrec(x, \lambda a\ b.\ HAleph\_rel(\mathcal{V},a,b))$
unfolding *Aleph_rel_def* **by** *simp*
also
have $\ldots = transrec(x, HAleph)$
by (*simp add*:*V_simps*)
also from ‹$Ord(x)$›
have $\ldots = \aleph_x$
using *Aleph'_eq_Aleph* **unfolding** *Aleph'_def* **by** *simp*
finally
show *?thesis* **.**
qed

Example of absolute lemmas obtained from the relative versions. Note the *only* declarations

lemma *Ord_cardinal_idem'*: $Ord(A) \Longrightarrow ||A|| = |A|$
using *V.Ord_cardinal_rel_idem* **by** (*simp only*:*V_simps*)

lemma *Aleph_succ'*: $Ord(\alpha) \Longrightarrow \aleph_{succ(\alpha)} = \aleph_\alpha{}^+$
using *V.Aleph_rel_succ* **by** (*simp only*:*V_simps*)

These two results are new, first obtained in relative form (not ported).

lemma *csucc_cardinal*:
assumes $Ord(\kappa)$ **shows** $|\kappa|^+ = \kappa^+$
using *assms V.csucc_rel_cardinal_rel* **by** (*simp add*:*V_simps*)

lemma *csucc_le_mono*:
assumes $\kappa \leq \nu$ **shows** $\kappa^+ \leq \nu^+$
using *assms V.csucc_rel_le_mono* **by** (*simp add*:*V_simps*)

Example of transferring results from a transitive model to $\mathcal{V}$

lemma (**in** *M_Perm*) *eqpoll_rel_transfer_absolute*:

assumes $M(A)$ $M(B)$ $A \approx^M B$
shows $A \approx B$
proof -
interpret *M_N_Perm M* $\mathcal{V}$
by (*unfold_locales*, *simp only*:*V_simps*)
from *assms*
show *?thesis* **using** *eqpoll_rel_transfer*
by (*simp only*:*V_simps*)
qed

The "relationalized" *CH* with respect to $\mathcal{V}$ corresponds to the real *CH*.

lemma *is_ContHyp_iff_CH*: *is_ContHyp*($\mathcal{V}$) $\longleftrightarrow$ *ContHyp*
using *V.is_ContHyp_iff*
by (*auto simp add*:*ContHyp_rel_def ContHyp_def V_simps*)

end

50 Main definitions of the development

theory *Definitions_Main*
imports
Not_CH
Absolute_Versions
begin

This theory gathers the main definitions of the Forcing session.

It might be considered as the bare minimum reading requisite to trust that our development indeed formalizes the theory of forcing. This should be mathematically clear since this is the only known method for obtaining proper extensions of ctms while preserving the ordinals.

The main theorem of this session and all of its relevant definitions appear in Section 50.3. The reader trusting all the libraries in which our development is based, might jump directly there. But in case one wants to dive deeper, the following sections treat some basic concepts in the ZF logic (Section 50.1) and in the ZF-Constructible library (Section 50.2) on which our definitions are built.

declare [[*show_question_marks*=*false*]]
no_notation *add* (**infixl** ‹#+› *65*)
notation *add* (**infixl** ‹+$_\omega$› *65*)
hide_const (**open**) *Order.pred*

50.1 ZF

For the basic logic ZF we restrict ourselves to just a few concepts.

thm *bij_def*[*unfolded inj_def surj_def*]

$bij(A,\ B) \equiv$
$\{f \in A \to B\ .\ \forall\, w{\in}A.\ \forall\, x{\in}A.\ f\ `\ w = f\ `\ x \longrightarrow w = x\} \cap$
$\{f \in A \to B\ .\ \forall\, y{\in}B.\ \exists\, x{\in}A.\ f\ `\ x = y\}$

thm *eqpoll_def*

$A \approx B \equiv \exists f.\ f \in bij(A,\ B)$

thm *Transset_def*

$Transset(i) \equiv \forall\, x{\in}i.\ x \subseteq i$

thm *Ord_def*

$Ord(i) \equiv Transset(i) \wedge (\forall\, x{\in}i.\ Transset(x))$

thm *lt_def*

$i < j \equiv i \in j \wedge Ord(j)$

With the concepts of empty set and successor in place,

lemma *empty_def'*: $\forall\, x.\ x \notin 0$ **by** *simp*
lemma *succ_def'*: $succ(i) = i \cup \{i\}$ **by** *blast*

we can define the set of natural numbers ω. In the sources, it is defined as a fixpoint, but here we just write its characterization as the first limit ordinal.

thm *Limit_nat*[*unfolded Limit_def*] *nat_le_Limit*[*unfolded Limit_def*]

$Ord(\omega) \wedge 0 < \omega \wedge (\forall\, y.\ y < \omega \longrightarrow succ(y) < \omega)$
$Ord(i) \wedge 0 < i \wedge (\forall\, y.\ y < i \longrightarrow succ(y) < i) \Longrightarrow \omega \leq i$

Then, addition and predecessor are inductively characterized as follows:

thm *add_0_right add_succ_right pred_0 pred_succ_eq*

$m +_\omega succ(n) = succ(m +_\omega n)$
$m \in \omega \Longrightarrow m +_\omega 0 = m$
$pred(0) = 0$
$pred(succ(y)) = y$

Lists on a set A can be characterized by being recursively generated from the empty list [] and the operation *Cons* that adds a new element to the left end; the induction theorem for them show that the characterization is "complete".

thm *Nil Cons list.induct*

$$[] \in list(A)$$
$$[\![a \in A;\ l \in list(A)]\!] \Longrightarrow Cons(a,\ l) \in list(A)$$
$$[\![x \in list(A);\ P([]);\ \bigwedge a\ l.\ [\![a \in A;\ l \in list(A);\ P(l)]\!] \Longrightarrow P(Cons(a,\ l))]\!]$$
$$\Longrightarrow P(x)$$

Length, concatenation, and nth element of lists are recursively characterized as follows.

thm *length.simps app.simps nth_0 nth_Cons*

$$length([]) = 0$$
$$length(Cons(a,\ l)) = succ(length(l))$$
$$[] \ @\ ys = ys$$
$$Cons(a,\ l) \ @\ ys = Cons(a,\ l \ @\ ys)$$
$$nth(0,\ Cons(a,\ l)) = a$$
$$n \in \omega \Longrightarrow nth(succ(n),\ Cons(a,\ l)) = nth(n,\ l)$$

We have the usual Haskell-like notation for iterated applications of *Cons*:

lemma *Cons_app*: $[a,b,c] = Cons(a,Cons(b,Cons(c,[])))$ **..**

Relative quantifiers restrict the range of the bound variable to a class M of type $i \Rightarrow o$; that is, a truth-valued function with set arguments.

lemma $\forall\, x[M].\ P(x) \equiv \forall\, x.\ M(x) \longrightarrow P(x)$
$\exists\, x[M].\ P(x) \equiv \exists\, x.\ M(x) \wedge P(x)$
unfolding *rall_def rex_def* **.**

Finally, a set can be viewed ("cast") as a class using the following function of type $i \Rightarrow i \Rightarrow o$.

thm *setclass_iff*

$$(\#\#A)(x) \longleftrightarrow x \in A$$

50.2 Relative concepts

A list of relative concepts (mostly from the ZF-Constructible library) follows next.

thm *big_union_def*

$big\_union(M,\ A,\ z) \equiv \forall\, x[M].\ x \in z \longleftrightarrow (\exists\, y[M].\ y \in A \wedge x \in y)$

thm *upair_def*

$upair(M,\ a,\ b,\ z) \equiv a \in z \wedge b \in z \wedge (\forall\, x[M].\ x \in z \longrightarrow x = a \vee x = b)$

thm *pair_def*

$pair(M,\ a,\ b,\ z) \equiv$
$\exists\, x[M].\ upair(M,\ a,\ a,\ x) \wedge (\exists\, y[M].\ upair(M,\ a,\ b,\ y) \wedge upair(M,\ x,\ y,\ z))$

thm *successor_def*[*unfolded is_cons_def union_def*]

$successor(M,\ a,\ z) \equiv$
$\exists\, x[M].\ upair(M,\ a,\ a,\ x) \wedge (\forall\, xa[M].\ xa \in z \longleftrightarrow xa \in x \vee xa \in a)$

thm *empty_def*

$empty(M,\ z) \equiv \forall\, x[M].\ x \notin z$

thm *transitive_set_def*[*unfolded subset_def*]

$transitive\_set(M,\ a) \equiv \forall\, x[M].\ x \in a \longrightarrow (\forall\, xa[M].\ xa \in x \longrightarrow xa \in a)$

thm *ordinal_def*

$ordinal(M,\ a) \equiv$
$transitive\_set(M,\ a) \wedge (\forall\, x[M].\ x \in a \longrightarrow transitive\_set(M,\ x))$

thm *image_def*

$image(M,\ r,\ A,\ z) \equiv$
$\forall\, y[M].\ y \in z \longleftrightarrow (\exists\, w[M].\ w \in r \wedge (\exists\, x[M].\ x \in A \wedge pair(M,\ x,\ y,\ w)))$

thm *fun_apply_def*

$is\_apply(M,\ f,\ x,\ y) \equiv$
$\exists\, xs[M].$
$\quad \exists\, fxs[M].\ upair(M,\ x,\ x,\ xs) \wedge image(M,\ f,\ xs,\ fxs) \wedge big\_union(M,\ fxs,\ y)$

thm *is_function_def*

$$
\begin{array}{l}
is\_function(M,\ r) \equiv \\
\forall\, x[M]. \\
\quad \forall\, y[M]. \\
\qquad \forall\, y'[M]. \\
\qquad\quad \forall\, p[M]. \\
\qquad\qquad \forall\, p'[M]. \\
\qquad\qquad\quad pair(M,\ x,\ y,\ p) \longrightarrow \\
\qquad\qquad\quad pair(M,\ x,\ y',\ p') \longrightarrow p \in r \longrightarrow p' \in r \longrightarrow y = y'
\end{array}
$$

thm *is_relation_def*

$$is\_relation(M,\ r) \equiv \forall\, z[M].\ z \in r \longrightarrow (\exists\, x[M].\ \exists\, y[M].\ pair(M,\ x,\ y,\ z))$$

thm *is_domain_def*

$$
\begin{array}{l}
is\_domain(M,\ r,\ z) \equiv \\
\forall\, x[M].\ x \in z \longleftrightarrow (\exists\, w[M].\ w \in r \wedge (\exists\, y[M].\ pair(M,\ x,\ y,\ w)))
\end{array}
$$

thm *typed_function_def*

$$
\begin{array}{l}
typed\_function(M,\ A,\ B,\ r) \equiv \\
is\_function(M,\ r) \wedge \\
is\_relation(M,\ r) \wedge \\
is\_domain(M,\ r,\ A) \wedge \\
(\forall\, u[M].\ u \in r \longrightarrow (\forall\, x[M].\ \forall\, y[M].\ pair(M,\ x,\ y,\ u) \longrightarrow y \in B))
\end{array}
$$

thm *is_function_space_def*[*unfolded is_funspace_def*]
function_space_rel_def surjection_def

$$
\begin{array}{l}
is\_function\_space(M,\ A,\ B,\ fs) \equiv \\
M(fs) \wedge (\forall\, f[M].\ f \in fs \longleftrightarrow typed\_function(M,\ A,\ B,\ f)) \\
A \rightarrow^{M} B \equiv THE\ d.\ is\_function\_space(M,\ A,\ B,\ d) \\
surjection(M,\ A,\ B,\ f) \equiv \\
typed\_function(M,\ A,\ B,\ f) \wedge \\
(\forall\, y[M].\ y \in B \longrightarrow (\exists\, x[M].\ x \in A \wedge is\_apply(M,\ f,\ x,\ y)))
\end{array}
$$

Relative version of the *ZFC* axioms

thm *extensionality_def*

$$extensionality(M) \equiv \forall\, x[M].\ \forall\, y[M].\ (\forall\, z[M].\ z \in x \longleftrightarrow z \in y) \longrightarrow x = y$$

thm *foundation_ax_def*

$$foundation\_ax(M) \equiv \forall\, x[M].\ (\exists\, y[M].\ y \in x) \longrightarrow (\exists\, y[M].\ y \in x \wedge \neg\ (\exists\, z[M].\ z \in x \wedge z \in y))$$

thm *upair_ax_def*

$$upair\_ax(M) \equiv \forall\, x[M].\ \forall\, y[M].\ \exists\, z[M].\ upair(M,\ x,\ y,\ z)$$

thm *Union_ax_def*

$$Union\_ax(M) \equiv \forall\, x[M].\ \exists\, z[M].\ big\_union(M,\ x,\ z)$$

thm *power_ax_def*[*unfolded powerset_def subset_def*]

$$power\_ax(M) \equiv \forall\, x[M].\ \exists\, z[M].\ \forall\, xa[M].\ xa \in z \longleftrightarrow (\forall\, xb[M].\ xb \in xa \longrightarrow xb \in x)$$

thm *infinity_ax_def*

$$infinity\_ax(M) \equiv \exists\, I[M].\ (\exists\, z[M].\ empty(M,\ z) \wedge z \in I) \wedge (\forall\, y[M].\ y \in I \longrightarrow (\exists\, sy[M].\ successor(M,\ y,\ sy) \wedge sy \in I))$$

thm *choice_ax_def*

$$choice\_ax(M) \equiv \forall\, x[M].\ \exists\, a[M].\ \exists\, f[M].\ ordinal(M,\ a) \wedge surjection(M,\ a,\ x,\ f)$$

thm *separation_def*

$$separation(M,\ P) \equiv \forall\, z[M].\ \exists\, y[M].\ \forall\, x[M].\ x \in y \longleftrightarrow x \in z \wedge P(x)$$

thm *univalent_def*

$$univalent(M,\ A,\ P) \equiv \forall\, x[M].\ x \in A \longrightarrow (\forall\, y[M].\ \forall\, z[M].\ P(x,\ y) \wedge P(x,\ z) \longrightarrow y = z)$$

thm *strong_replacement_def*

$strong\_replacement(M,\ P) \equiv$
$\forall\, A[M].$
$\quad univalent(M,\ A,\ P) \longrightarrow (\exists\, Y[M].\ \forall\, b[M].\ b \in Y \longleftrightarrow (\exists\, x[M].\ x \in A \wedge P(x,\ b)))$

Internalized formulas

thm *Member Equal Nand Forall formula.induct*

$[\![x \in \omega;\ y \in \omega]\!] \Longrightarrow \cdot x \in y\cdot \in formula$
$[\![x \in \omega;\ y \in \omega]\!] \Longrightarrow \cdot x = y\cdot \in formula$
$[\![p \in formula;\ q \in formula]\!] \Longrightarrow \cdot\neg(p \wedge q)\cdot \in formula$
$p \in formula \Longrightarrow (\cdot\forall\, p\cdot) \in formula$
$[\![x \in formula;\ \bigwedge x\ y.\ [\![x \in \omega;\ y \in \omega]\!] \Longrightarrow P(\cdot x \in y\cdot);$
$\bigwedge x\ y.\ [\![x \in \omega;\ y \in \omega]\!] \Longrightarrow P(\cdot x = y\cdot);$
$\bigwedge p\ q.\ [\![p \in formula;\ P(p);\ q \in formula;\ P(q)]\!] \Longrightarrow P(\cdot\neg(p \wedge q)\cdot);$
$\bigwedge p.\ [\![p \in formula;\ P(p)]\!] \Longrightarrow P((\cdot\forall\, p\cdot))]\!]$
$\Longrightarrow P(x)$

thm *arity.simps*

$arity(\cdot x \in y\cdot) = succ(x) \cup succ(y)$
$arity(\cdot x = y\cdot) = succ(x) \cup succ(y)$
$arity(\cdot\neg(p \wedge q)\cdot) = arity(p) \cup arity(q)$
$arity((\cdot\forall\, p\cdot)) = pred(arity(p))$

We have the satisfaction relation between $\in$-models and first order formulas (given a "environment" list representing the assignment of free variables),

thm *mem_iff_sats equal_iff_sats sats_Nand_iff sats_Forall_iff*

$[\![nth(i,\ env) = x;\ nth(j,\ env) = y;\ env \in list(A)]\!]$
$\Longrightarrow x \in y \longleftrightarrow A,\ env \models \cdot i \in j\cdot$
$[\![nth(i,\ env) = x;\ nth(j,\ env) = y;\ env \in list(A)]\!]$
$\Longrightarrow x = y \longleftrightarrow A,\ env \models \cdot i = j\cdot$
$env \in list(A) \Longrightarrow (A,\ env \models \cdot\neg(p \wedge q)\cdot) \longleftrightarrow \neg\ ((A,\ env \models p) \wedge (A,\ env \models q))$
$env \in list(A) \Longrightarrow (A,\ env \models (\cdot\forall\, p\cdot)) \longleftrightarrow (\forall\, x{\in}A.\ A,\ Cons(x,\ env) \models p)$

as well as the satisfaction of an arbitrary set of sentences.

thm *satT_def*

$A \models \Phi \equiv \forall\, \varphi{\in}\Phi.\ A,\ [] \models \varphi$

The internalized (viz. as elements of the set *formula*) version of the axioms follow next.

thm *ZF_union_iff_sats ZF_power_iff_sats ZF_pairing_iff_sats*
ZF_foundation_iff_sats ZF_extensionality_iff_sats
ZF_infinity_iff_sats sats_ZF_separation_fm_iff
sats_ZF_replacement_fm_iff ZF_choice_iff_sats

$Union\_ax(\#\#A) \longleftrightarrow A,\ [] \models \cdot Union\ Ax\cdot$
$power\_ax(\#\#A) \longleftrightarrow A,\ [] \models \cdot Powerset\ Ax\cdot$
$upair\_ax(\#\#A) \longleftrightarrow A,\ [] \models \cdot Pairing\cdot$
$foundation\_ax(\#\#A) \longleftrightarrow A,\ [] \models \cdot Foundation\cdot$
$extensionality(\#\#A) \longleftrightarrow A,\ [] \models \cdot Extensionality\cdot$
$infinity\_ax(\#\#A) \longleftrightarrow A,\ [] \models \cdot Infinity\cdot$
$\varphi \in formula \Longrightarrow$
$(M,\ [] \models \cdot Separation(\varphi)\cdot) \longleftrightarrow$
$(\forall\ env{\in}list(M).$
$\quad arity(\varphi) \leq 1 +_\omega length(env) \longrightarrow separation(\#\#M,\ \lambda x.\ M,\ [x]\ @\ env \models \varphi))$
$\varphi \in formula \Longrightarrow$
$(M,\ [] \models \cdot Replacement(\varphi)\cdot) \longleftrightarrow$
$(\forall\ env{\in}list(M).$
$\quad arity(\varphi) \leq 2 +_\omega length(env) \longrightarrow$
$\quad strong\_replacement(\#\#M,\ \lambda x\ y.\ M,\ [x,\ y]\ @\ env \models \varphi))$
$choice\_ax(\#\#A) \longleftrightarrow A,\ [] \models \cdot AC\cdot$

thm *ZF_fin_def ZF_inf_def ZF_def ZFC_fin_def ZFC_def*

$ZF\_fin \equiv$
$\{\cdot Extensionality\cdot,\ \cdot Foundation\cdot,\ \cdot Pairing\cdot,\ \cdot Union\ Ax\cdot,\ \cdot Infinity\cdot,$
$\ \cdot Powerset\ Ax\cdot\}$
$ZF\_inf \equiv \{\cdot Separation(p)\cdot\ .\ p \in formula\} \cup \{\cdot Replacement(p)\cdot\ .\ p \in formula\}$
$ZF \equiv ZF\_inf \cup ZF\_fin$
$ZFC\_fin \equiv ZF\_fin \cup \{\cdot AC\cdot\}$
$ZFC \equiv ZF\_inf \cup ZFC\_fin$

50.3 Forcing

thm *extensions_of_ctms*

$[\![M \approx \omega;\ Transset(M);\ M \models ZF]\!]$
$\Longrightarrow \exists N.\ M \subseteq N \wedge$
$\qquad N \approx \omega \wedge$
$\qquad Transset(N) \wedge$
$\qquad N \models ZF \wedge$
$\qquad M \neq N \wedge (\forall \alpha.\ Ord(\alpha) \longrightarrow \alpha \in M \longleftrightarrow \alpha \in N) \wedge ((M,\ [] \models \cdot AC\cdot) \longrightarrow$
$N \models ZFC)$

In order to state the defining property of the relative equipotence relation, we work under the assumptions of the locale *M_cardinals*. They comprise a finite set of instances of Separation and Replacement to prove closure properties of the transitive class *M*.

lemma (**in** *M_cardinals*) *eqpoll_def'*:
assumes $M(A)$ $M(B)$ **shows** $A \approx^M B \longleftrightarrow (\exists f[M].\ f \in bij(A,B))$

using *assms* **unfolding** *eqpoll_rel_def* **by** *auto*

Below, μ denotes the minimum operator on the ordinals.

lemma *cardinalities_defs*:
fixes $M{::}i{\Rightarrow}o$
shows
$|A|^M \equiv \mu\ i.\ M(i) \wedge i \approx^M A$
$Card^M(\alpha) \equiv \alpha = |\alpha|^M$
$\kappa^{\uparrow\nu,M} \equiv |\nu \rightarrow^M \kappa|^M$
$(\kappa^+)^M \equiv \mu\ x.\ M(x) \wedge Card^M(x) \wedge \kappa < x$
$CH^M \equiv \aleph_1{}^M = 2^{\uparrow\aleph_0{}^M,M}$
unfolding *cardinal_rel_def cexp_rel_def*
csucc_rel_def Card_rel_def ContHyp_rel_def **.**

context *M_aleph*
begin

As in the previous Lemma *eqpoll_def′*, we are now under the assumptions of the locale *M_aleph*. The axiom instances included are sufficient to state and prove the defining properties of the relativized *Aleph* function (in particular, the required ability to perform transfinite recursions).

thm *Aleph_rel_zero Aleph_rel_succ Aleph_rel_limit*

$\aleph_0{}^M = \omega$
$[\![Ord(\alpha);\ M(\alpha)]\!] \Longrightarrow \aleph_{succ(\alpha)}{}^M = (\aleph_\alpha{}^{M+})^M$
$[\![Limit(\alpha);\ M(\alpha)]\!] \Longrightarrow \aleph_\alpha{}^M = (\bigcup j{\in}\alpha.\ \aleph_j{}^M)$

end

lemma *ContHyp_rel_def′*:
fixes $N{::}i{\Rightarrow}o$
shows
$CH^N \equiv \aleph_1{}^N = 2^{\uparrow\aleph_0{}^N,N}$
unfolding *ContHyp_rel_def* **.**

Under appropriate hypothesis (this time, from the locale *M_master*), CH^M is equivalent to its fully relational version *is_ContHyp*. As a sanity check, we see that if the transitive class is indeed $\mathcal{V}$, we recover the original *CH*.

thm *M_master.is_ContHyp_iff is_ContHyp_iff_CH*[*unfolded ContHyp_def*]

$M\_master(M) \Longrightarrow is\_ContHyp(M) \longleftrightarrow CH^M$
$is\_ContHyp(\mathcal{V}) \longleftrightarrow \aleph_1 = 2^{\uparrow\aleph_0}$

In turn, the fully relational version evaluated on a nonempty transitive A is equivalent to the satisfaction of the first-order formula $\cdot CH\cdot$.

thm *is_ContHyp_iff_sats*

$[\![env \in list(A);\ 0 \in A]\!] \Longrightarrow is\_ContHyp(\#\#A) \longleftrightarrow A,\ env \models \cdot CH\cdot$

thm *ctm_of_not_CH*

$[\![M \approx \omega;\ Transset(M);\ M \models ZFC]\!]$
$\Longrightarrow \exists N.\ M \subseteq N \wedge$
$N \approx \omega \wedge$
$Transset(N) \wedge N \models ZFC \cup \{\cdot\neg\cdot CH\cdot\cdot\} \wedge (\forall \alpha.\ Ord(\alpha) \longrightarrow \alpha \in M \longleftrightarrow \alpha \in N)$

end

References

[1] E. Gunther, M. Pagano, P. Sánchez Terraf, First steps towards a formalization of forcing, in: Proceedings of the 13th Workshop on Logical and Semantic Frameworks with Applications, LSFA 2018, Fortaleza, Brazil, September 26-28, 2018, pp. 119–136 (2018).

[2] E. Gunther, M. Pagano, P. Sánchez Terraf, Mechanization of Separation in Generic Extensions, *arXiv e-prints* **1901.03313** (2019).

[3] E. Gunther, M. Pagano, P. Sánchez Terraf, Formalization of Forcing in Isabelle/ZF, *arXiv e-prints* **2001.09715** (2020).

[4] L.C. Paulson, K. Grabczewski, Mechanizing set theory, *J. Autom. Reasoning* **17**: 291–323 (1996).