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

*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.

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$(A,r,p,q) \equiv \exists\, d{\in}A\ .\ \langle d,p\rangle{\in}r \wedge \langle d,q\rangle{\in}r$

definition
is_compat_in :: $[i{\Rightarrow}o,i,i,i,i] \Rightarrow o$ **where**
is_compat_in$(M,A,r,p,q) \equiv \exists\, d[M].\ d{\in}A \wedge (\exists\, dp[M].\ pair(M,d,p,dp) \wedge dp{\in}r \wedge$
$(\exists\, dq[M].\ pair(M,d,q,dq) \wedge dq{\in}r))$

lemma *compat_inI* :
$[\![\ d{\in}A\ ;\ \langle d,p\rangle{\in}r\ ;\ \langle d,g\rangle{\in}r\ ]\!] \Longrightarrow$ *compat_in*(A,r,p,g)
⟨*proof*⟩

lemma *refl_compat*:
$[\![\ refl(A,r)\ ;\ \langle p,q\rangle \in r \mid p{=}q \mid \langle q,p\rangle \in r\ ;\ p{\in}A\ ;\ q{\in}A]\!] \Longrightarrow$ *compat_in*(A,r,p,q)
⟨*proof*⟩

lemma *chain_compat*:
$refl(A,r) \Longrightarrow linear(A,r) \Longrightarrow (\forall\, p{\in}A.\forall\, q{\in}A.$ *compat_in*$(A,r,p,q))$
⟨*proof*⟩

lemma *subset_fun_image*: $f{:}N{\to}P \Longrightarrow f``N{\subseteq}P$
⟨*proof*⟩

lemma *refl_monot_domain*: $refl(B,r) \Longrightarrow A{\subseteq}B \Longrightarrow refl(A,r)$
⟨*proof*⟩

locale *forcing_notion* =
fixes *P leq one*
assumes *one_in_P*: $one \in P$
and *leq_preord*: *preorder_on*(P,leq)
and *one_max*: $\forall\, p{\in}P.\ \langle p,one\rangle{\in}leq$
begin

abbreviation *Leq* :: $[i,\ i] \Rightarrow o$ (**infixl** $\preceq$ *50*)
where $x \preceq y \equiv \langle x,y\rangle{\in}leq$

lemma *refl_leq*:
$r{\in}P \Longrightarrow r{\preceq}r$
⟨*proof*⟩

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

definition
dense :: $i{\Rightarrow}o$ **where**
$dense(D) \equiv \forall\, p{\in}P.\ \exists\, d{\in}D\ .\ d{\preceq}p$

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 :: $i{\Rightarrow}i{\Rightarrow}o$ **where**
dense_below$(D,q) \equiv \forall\, p{\in}P.\ p{\preceq}q \longrightarrow (\exists\, d{\in}D.\ d{\in}P \wedge d{\preceq}p)$

lemma *P_dense*: $dense(P)$
⟨*proof*⟩

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*: $a \preceq b \Longrightarrow b \preceq c \Longrightarrow a \in P \Longrightarrow b \in P \Longrightarrow c \in P \Longrightarrow a \preceq c$
⟨*proof*⟩

lemma *leq_transD'*: $A \subseteq P \Longrightarrow a \preceq b \Longrightarrow b \preceq c \Longrightarrow a \in A \Longrightarrow b \in P \Longrightarrow c \in P \Longrightarrow$
$a \preceq c$
⟨*proof*⟩

lemma *compatD*[*dest!*]: $compat(p,q) \Longrightarrow \exists d \in P.\ d \preceq p \wedge d \preceq q$
⟨*proof*⟩

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

lemma *compatI*[*intro!*]: $d \in P \Longrightarrow d \preceq p \Longrightarrow d \preceq q \Longrightarrow compat(p,q)$
⟨*proof*⟩

lemma *denseD* [*dest*]: $dense(D) \Longrightarrow p \in P \Longrightarrow\ \exists d \in D.\ d \preceq\ p$
⟨*proof*⟩

lemma *denseI* [*intro!*]: $[\![\ \bigwedge p.\ p \in P \Longrightarrow \exists d \in D.\ d \preceq\ p\ ]\!] \Longrightarrow dense(D)$
⟨*proof*⟩

lemma *dense_belowD* [*dest*]:
assumes $dense\_below(D,p)\ q \in P\ q \preceq p$
shows $\exists d \in D.\ d \in P \wedge d \preceq q$
⟨*proof*⟩

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 $dense\_below(D,p)$
⟨*proof*⟩

lemma *dense_below_cong*: $p \in P \Longrightarrow D = D' \Longrightarrow dense\_below(D,p) \longleftrightarrow dense\_below(D',p)$
⟨*proof*⟩

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

$\langle proof \rangle$

lemma *dense_below_mono*: $p \in P \Longrightarrow D \subseteq D' \Longrightarrow dense\_below(D,p) \Longrightarrow dense\_below(D',p)$
$\langle proof \rangle$

lemma *dense_below_under*:
assumes $dense\_below(D,p)$ $p \in P$ $q \in P$ $q \preceq p$
shows $dense\_below(D,q)$
$\langle proof \rangle$

lemma *ideal_dense_below*:
assumes $\bigwedge q.\ q \in P \Longrightarrow q \preceq p \Longrightarrow q \in D$
shows $dense\_below(D,p)$
$\langle proof \rangle$

lemma *dense_below_dense_below*:
assumes $dense\_below(\{q \in P.\ dense\_below(D,q)\},p)$ $p \in P$
shows $dense\_below(D,p)$
$\langle proof \rangle$

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* : $filter(G) \Longrightarrow x \in G \Longrightarrow x \in P$
$\langle proof \rangle$

lemma *filter_leqD* : $filter(G) \Longrightarrow x \in G \Longrightarrow y \in P \Longrightarrow x \preceq y \Longrightarrow y \in G$
$\langle proof \rangle$

lemma *filter_imp_compat*: $filter(G) \Longrightarrow p \in G \Longrightarrow q \in G \Longrightarrow compat(p,q)$
$\langle proof \rangle$

lemma *low_bound_filter*: — says the compatibility is attained inside G
assumes $filter(G)$ **and** $p \in G$ **and** $q \in G$
shows $\exists r \in G.\ r \preceq p \wedge r \preceq q$
$\langle proof \rangle$

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*] : $p \in P \Longrightarrow a \in A \Longrightarrow a \preceq p \Longrightarrow p \in upclosure(A)$
$\langle proof \rangle$

lemma *upclosureE* [*elim*] :

p∈*upclosure*(*A*) ⟹ (⋀*x a*. *x*∈*P* ⟹ *a*∈*A* ⟹ *a*⪯*x* ⟹ *R*) ⟹ *R*
⟨*proof*⟩

lemma *upclosureD* [*dest*] :
p∈*upclosure*(*A*) ⟹ ∃ *a*∈*A*.(*a*⪯*p*) ∧ *p*∈*P*
⟨*proof*⟩

lemma *upclosure_increasing* :
assumes *A*⊆*P*
shows *increasing*(*upclosure*(*A*))
⟨*proof*⟩

lemma *upclosure_in_P*: *A* ⊆ *P* ⟹ *upclosure*(*A*) ⊆ *P*
⟨*proof*⟩

lemma *A_sub_upclosure*: *A* ⊆ *P* ⟹ *A*⊆*upclosure*(*A*)
⟨*proof*⟩

lemma *elem_upclosure*: *A*⊆*P* ⟹ *x*∈*A* ⟹ *x*∈*upclosure*(*A*)
⟨*proof*⟩

lemma *closure_compat_filter*:
assumes *A*⊆*P* (∀ *p*∈*A*.∀ *q*∈*A*. *compat_in*(*A*,*leq*,*p*,*q*))
shows *filter*(*upclosure*(*A*))
⟨*proof*⟩

lemma *aux_RS1*: *f* ∈ *N* → *P* ⟹ *n*∈*N* ⟹ *f*‘*n* ∈ *upclosure*(*f* ‘‘*N*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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 :: $i \Rightarrow o$ **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*⟩

end

— TODO: already in ZF Library
lemma *Pi_rangeD*:
assumes $f{\in}Pi(A,B)\ b \in range(f)$
shows $\exists a{\in}A.\ f`a = b$
⟨*proof*⟩

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 $n{\in}nat\ p{\in}P\ P \subseteq range(enum)\ enum{:}nat \to M$
$\bigwedge x\ k.\ x{\in}P \Longrightarrow k{\in}nat \Longrightarrow \exists q{\in}P.\ q \preceq x \wedge q \in \mathcal{D}\ `\ k$
shows
$f(succ(n)) \in P \wedge f(succ(n)) \preceq f(n) \wedge f(succ(n)) \in \mathcal{D}\ `\ n$
⟨*proof*⟩

lemma *countable_RS_sequence*:
fixes *p enum*

defines $f \equiv \lambda n{\in}nat.\ RS\_seq(n,P,leq,p,enum,\mathcal{D})$
and $Q(q,k,m) \equiv enum`m{\preceq}\ q \wedge enum`m \in \mathcal{D}\ `\ k$
assumes $n{\in}nat$ $p{\in}P$ $P \subseteq range(enum)$ $enum{:}nat{\to}M$
shows
$f`0 = p$ $f`succ(n){\preceq}\ f`n \wedge f`succ(n) \in \mathcal{D}\ `\ n$ $f`succ(n) \in P$
⟨*proof*⟩

lemma *RS_seq_type*:
assumes $n \in nat$ $p{\in}P$ $P \subseteq range(enum)$ $enum{:}nat{\to}M$
shows $RS\_seq(n,P,leq,p,enum,\mathcal{D}) \in P$
⟨*proof*⟩

lemma *RS_seq_funtype*:
assumes $p{\in}P$ $P \subseteq range(enum)$ $enum{:}nat{\to}M$
shows $(\lambda n{\in}nat.\ RS\_seq(n,P,leq,p,enum,\mathcal{D}))\colon nat \to P$
⟨*proof*⟩

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 $dc\_witness :: i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow i$
primrec
wit0 : $dc\_witness(0,A,a,s,R) = a$
witrec : $dc\_witness(succ(n),A,a,s,R) = s`\{x{\in}A.\ \langle dc\_witness(n,A,a,s,R),x\rangle{\in}R\ \}$

lemma *witness_into_A* [*TC*]:
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 $dc\_witness(n,\ A,\ a,\ s,\ R){\in}A$
⟨*proof*⟩

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*⟩

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*⟩

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 $\exists f \in nat \to A. \forall n \in nat. f`n = dc\_witness(n,A,a,s,R)$
  ⟨*proof*⟩

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*⟩

lemma *aux_DC_on_AxNat2* : $\forall x \in A \times nat. \exists y \in A. \langle x,\langle y,succ(snd(x)) \rangle \rangle \in R \Longrightarrow$
  $\forall x \in A \times nat. \exists y \in A \times nat. \langle x,y \rangle \in \{\langle a,b \rangle \in R. snd(b) = succ(snd(a))\}$
  ⟨*proof*⟩

lemma *infer_snd* : $c \in A \times B \Longrightarrow snd(c) = k \Longrightarrow c = \langle fst(c),k \rangle$
  ⟨*proof*⟩

corollary *DC_on_A_x_nat* :
  assumes $(\forall x \in A \times nat. \exists y \in A. \langle x,\langle y,succ(snd(x)) \rangle \rangle \in R)$ $a \in A$
  shows $\exists f \in nat \to A. f`0 = a \wedge (\forall n \in nat. \langle \langle f`n,n \rangle,\langle f`succ(n),succ(n) \rangle \rangle \in R)$ (**is**
$\exists x \in \_. ?P(x)$)
⟨*proof*⟩

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 $\forall x \in A \times nat . \exists y \in A. \langle x,\langle y,succ(snd(x)) \rangle \rangle \in R$
  ⟨*proof*⟩

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

lemma *sequence_DC*:
  assumes $\forall x \in A. \forall n \in nat. \exists y \in A. \langle x,y \rangle \in S`n$
  shows $\forall a \in A. (\exists f \in nat \to A. f`0 = a \wedge (\forall n \in nat. \langle f`n,f`succ(n) \rangle \in S`succ(n)))$
  ⟨*proof*⟩

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*⟩

lemma *DC_imp_RS_sequence*:
 assumes $p{\in}P$
 shows $\exists f.\ f\colon nat{\to}P \wedge f\ `\ 0 = p \wedge$
  $(\forall\, n{\in}nat.\ f\ `\ succ(n){\preceq}\ f\ `\ n \wedge f\ `\ succ(n) \in \mathcal{D}\ `\ n)$
⟨*proof*⟩

theorem *rasiowa_sikorski*:
 $p{\in}P \Longrightarrow \exists\, G.\ p{\in}G \wedge D\_generic(G)$
 ⟨*proof*⟩

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* : $m \in nat \Longrightarrow\ succ(n) \in succ(m) \Longrightarrow n \in m$
 ⟨*proof*⟩

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

lemma *in_n_in_nat* : $m \in nat \Longrightarrow n \in m \Longrightarrow n \in nat$
 ⟨*proof*⟩

lemma *in_succ_in_nat* : $m \in nat \Longrightarrow n \in succ(m) \Longrightarrow n \in nat$
 ⟨*proof*⟩

lemma *ltI_neg* : $x \in nat \Longrightarrow j \le x \Longrightarrow j \neq x \Longrightarrow j < x$
 ⟨*proof*⟩

lemma *succ_pred_eq* : $m \in nat \Longrightarrow m \neq 0 \Longrightarrow succ(pred(m)) = m$
 ⟨*proof*⟩

lemma *succ_ltI* : $succ(j) < n \Longrightarrow j < n$
⟨*proof*⟩

lemma *succ_In* : $n \in nat \Longrightarrow succ(j) \in n \Longrightarrow j \in n$
⟨*proof*⟩

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

lemma *succpred_leI* : $n \in nat \Longrightarrow n \le succ(pred(n))$
⟨*proof*⟩

lemma *succpred_n0* : $succ(n) \in p \Longrightarrow p \neq 0$
⟨*proof*⟩

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

lemma *succ_in* : $succ(x) \le y \Longrightarrow x \in y$
⟨*proof*⟩

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

lemma *pred_type* : $m \in nat \Longrightarrow n \le m \Longrightarrow n \in nat$
⟨*proof*⟩

lemma *pred_le* : $m \in nat \Longrightarrow n \le succ(m) \Longrightarrow pred(n) \le m$
⟨*proof*⟩

lemma *pred_le2* : $n \in nat \Longrightarrow m \in nat \Longrightarrow pred(n) \le m \Longrightarrow n \le succ(m)$
⟨*proof*⟩

lemma *Un_leD1* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow Ord(k) \Longrightarrow i \cup j \le k \Longrightarrow i \le k$
⟨*proof*⟩

lemma *Un_leD2* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow Ord(k) \Longrightarrow i \cup j \le k \Longrightarrow j \le k$
⟨*proof*⟩

lemma *gt1* : $n \in nat \Longrightarrow i \in n \Longrightarrow i \neq 0 \Longrightarrow i \neq 1 \Longrightarrow 1 < i$
⟨*proof*⟩

lemma *pred_mono* : $m \in nat \Longrightarrow n \le m \Longrightarrow pred(n) \le pred(m)$
⟨*proof*⟩

lemma *succ_mono* : $m \in nat \Longrightarrow n \le m \Longrightarrow succ(n) \le succ(m)$
⟨*proof*⟩

lemma *union_abs1* :
$[\![i \le j]\!] \Longrightarrow i \cup j = j$

⟨*proof*⟩

lemma *union_abs2* :
  $[\![i \leq j]\!] \Longrightarrow j \cup i = j$
  ⟨*proof*⟩

lemma *ord_un_max* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow i \cup j = max(i,j)$
  ⟨*proof*⟩

lemma *ord_max_ty* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow Ord(max(i,j))$
  ⟨*proof*⟩

lemmas *ord_simp_union* = *ord_un_max ord_max_ty max_def*

lemma *le_succ* : $x \in nat \Longrightarrow x \leq succ(x)$ ⟨*proof*⟩

lemma *le_pred* : $x \in nat \Longrightarrow pred(x) \leq x$
  ⟨*proof*⟩

lemma *Un_le_compat* : $o \leq p \Longrightarrow q \leq r \Longrightarrow Ord(o) \Longrightarrow Ord(p) \Longrightarrow Ord(q) \Longrightarrow Ord(r) \Longrightarrow o \cup q \leq p \cup r$
  ⟨*proof*⟩

lemma *Un_le* : $p \leq r \Longrightarrow q \leq r \Longrightarrow$
    $Ord(p) \Longrightarrow Ord(q) \Longrightarrow Ord(r) \Longrightarrow$
    $p \cup q \leq r$
  ⟨*proof*⟩

lemma *Un_leI3* : $o \leq r \Longrightarrow p \leq r \Longrightarrow q \leq r \Longrightarrow$
    $Ord(o) \Longrightarrow Ord(p) \Longrightarrow Ord(q) \Longrightarrow Ord(r) \Longrightarrow$
    $o \cup p \cup q \leq r$
  ⟨*proof*⟩

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*⟩

lemma *pred_Un*:
  $x \in nat \Longrightarrow y \in nat \Longrightarrow Arith.pred(succ(x) \cup y) = x \cup Arith.pred(y)$
  $x \in nat \Longrightarrow y \in nat \Longrightarrow Arith.pred(x \cup succ(y)) = Arith.pred(x) \cup y$
  ⟨*proof*⟩

lemma *le_natI* : $j \leq n \Longrightarrow n \in nat \Longrightarrow j \in nat$
  ⟨*proof*⟩

lemma *le_natE* : $n \in nat \Longrightarrow j < n \Longrightarrow j \in n$
  ⟨*proof*⟩

lemma *leD* : **assumes** $n \in nat$ $j \leq n$
shows $j < n \mid j = n$
⟨*proof*⟩

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*⟩

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

lemma *max_cong2* :
assumes $x \leq y$ $Ord(y)$ $Ord(z)$ $Ord(x)$
shows $max(x,z) \leq max(y,z)$
⟨*proof*⟩

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*⟩

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*⟩

lemma *oadd_lt_mono2* :
assumes $Ord(n)$ $Ord(\alpha)$ $Ord(\beta)$ $\alpha < \beta$ $x < n$ $y < n$ $0 < n$
shows $n ** \alpha ++ x < n ** \beta ++ y$
⟨*proof*⟩
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*)}}
⟨*proof*⟩

lemma *SepReplace_subset* : *A*⊆*A*′⟹ {*b* .. *x*∈*A*, *Q*}⊆{*b* .. *x*∈*A*′, *Q*}
⟨*proof*⟩

lemma *SepReplace_iff* [*simp*]: *y*∈{*b*(*x*) .. *x*∈*A*, *P*(*x*)} ⟷ (∃ *x*∈*A*. *y*=*b*(*x*) & *P*(*x*))
⟨*proof*⟩

lemma *SepReplace_dom_implies* :
(⋀ *x* . *x* ∈*A* ⟹ *b*(*x*) = *b*′(*x*))⟹ {*b*(*x*) .. *x*∈*A*, *Q*(*x*)}={*b*′(*x*) .. *x*∈*A*, *Q*(*x*)}
⟨*proof*⟩

lemma *SepReplace_pred_implies* :
∀ *x*. *Q*(*x*)⟶ *b*(*x*) = *b*′(*x*)⟹ {*b*(*x*) .. *x*∈*A*, *Q*(*x*)}={*b*′(*x*) .. *x*∈*A*, *Q*(*x*)}
⟨*proof*⟩

lemma *funcI* : *f* ∈ *A* → *B* ⟹ *a* ∈ *A* ⟹ *b*= *f* ‘ *a* ⟹ ⟨*a*, *b*⟩ ∈ *f*
⟨*proof*⟩

lemma *vimage_fun_sing*:
assumes *f*∈*A*→*B* *b*∈*B*
shows {*a*∈*A* . *f*‘*a*=*b*} = *f*-‘‘{*b*}
⟨*proof*⟩

lemma *image_fun_subset*: *S*∈*A*→*B* ⟹ *C*⊆*A*⟹ {*S* ‘ *x* . *x*∈ *C*} = *S*‘‘*C*
⟨*proof*⟩

lemma *subset_Diff_Un*: *X* ⊆ *A* ⟹ *A* = (*A* - *X*) ∪ *X* ⟨*proof*⟩

lemma *Diff_bij*:
assumes ∀ *A*∈*F*. *X* ⊆ *A* **shows** (λ*A*∈*F*. *A*-*X*) ∈ *bij*(*F*, {*A*-*X*. *A*∈*F*})
⟨*proof*⟩

lemma *function_space_nonempty*:
assumes *b*∈*B*
shows (λ*x*∈*A*. *b*) : *A* → *B*
⟨*proof*⟩

lemma *vimage_lam*: (λ*x*∈*A*. *f*(*x*)) -‘‘ *B* = { *x*∈*A* . *f*(*x*) ∈ *B* }

⟨*proof*⟩

lemma *range_fun_subset_codomain*:
assumes $h{:}B \to C$
shows $range(h) \subseteq C$
⟨*proof*⟩

lemma *Pi_rangeD*:
assumes $f{\in}Pi(A,B)$ $b \in range(f)$
shows $\exists\, a{\in}A.\ f`a = b$
⟨*proof*⟩

lemma *Pi_range_eq*: $f \in Pi(A,B) \Longrightarrow range(f) = \{f \text{ ` } x\ .\ x \in A\}$
⟨*proof*⟩

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

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

lemma *minimum_in*: $[\![\ well\_ord(A,r);\ B{\subseteq}A;\ B{\neq}0\ ]\!] \Longrightarrow minimum(r,B) \in B$
⟨*proof*⟩

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*⟩

lemma *well_ord_surj_imp_lepoll*:
assumes $well\_ord(A,r)$ $h \in surj(A,B)$
shows $B{\lesssim}A$
⟨*proof*⟩
lemma *surj_imp_well_ord*:
assumes $well\_ord(A,r)$ $h \in surj(A,B)$
shows $\exists\, s.\ well\_ord(B,s)$
⟨*proof*⟩

lemma *Pow_sing* : $Pow(\{a\}) = \{0,\{a\}\}$
⟨*proof*⟩

lemma *Pow_cons*:
shows $Pow(cons(a,A)) = Pow(A) \cup \{\{a\} \cup X\ .\ X{:}\ Pow(A)\}$
⟨*proof*⟩

lemma *app_nm* :
assumes $n{\in}nat$ $m{\in}nat$ $f{\in}n{\to}m$ $x \in nat$

shows *f*‘*x* ∈ *nat*
⟨*proof*⟩

lemma *Upair_eq_cons*: *Upair*(*a*,*b*) = {*a*,*b*}
⟨*proof*⟩

lemma *converse_apply_eq* : *converse*(*f*) ‘ *x* = ⋃(*f* -‘‘ {*x*})
⟨*proof*⟩

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*^∗

lemma *fld_restrict_eq* : *a* ∈ *A* ⟹ (*r* ∩ *A*×*A*)-‘‘{*a*} = (*r*-‘‘{*a*} ∩ *A*)
⟨*proof*⟩

lemma *fld_restrict_mono* : *relation*(*r*) ⟹ *A* ⊆ *B* ⟹ *r* ∩ *A*×*A* ⊆ *r* ∩ *B*×*B*
⟨*proof*⟩

lemma *fld_restrict_dom* :
assumes *relation*(*r*) *domain*(*r*) ⊆ *A* *range*(*r*)⊆ *A*
shows *r*∩ *A*×*A* = *r*
⟨*proof*⟩

definition *tr_down* :: [*i*,*i*] ⇒ *i*
where *tr_down*(*r*,*a*) = (*r*^+)-‘‘{*a*}

lemma *tr_downD* : *x* ∈ *tr_down*(*r*,*a*) ⟹ ⟨*x*,*a*⟩ ∈ *r*^+
⟨*proof*⟩

lemma *pred_down* : *relation*(*r*) ⟹ *r*-‘‘{*a*} ⊆ *tr_down*(*r*,*a*)
⟨*proof*⟩

lemma *tr_down_mono* : *relation*(*r*) ⟹ *x* ∈ *r*-‘‘{*a*} ⟹ *tr_down*(*r*,*x*) ⊆ *tr_down*(*r*,*a*)
⟨*proof*⟩

lemma *rest_eq* :
assumes *relation*(*r*) **and** *r*-‘‘{*a*} ⊆ *B* **and** *a* ∈ *B*
shows *r*-‘‘{*a*} = (*r*∩*B*×*B*)-‘‘{*a*}
⟨*proof*⟩

lemma *wfrec_restr_eq* : *r*′ = *r* ∩ *A*×*A* ⟹ *wfrec*[*A*](*r*,*a*,*H*) = *wfrec*(*r*′,*a*,*H*)

⟨*proof*⟩

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*⟩

lemmas *wfrec_tr_down* = *wfrec_restr*[*OF* _ _ _ *subset_refl*]

lemma *wfrec_trans_restr* : $relation(r) \Longrightarrow wf(r) \Longrightarrow trans(r) \Longrightarrow r\text{-``}\{a\}\subseteq A \Longrightarrow a \in A \Longrightarrow$
$wfrec(r, a, H) = wfrec[A](r, a, H)$
⟨*proof*⟩

lemma *field_trancl* : $field(r\hat{}+) = field(r)$
⟨*proof*⟩

definition
Rrel :: $[i \Rightarrow i \Rightarrow o,i] \Rightarrow i$ **where**
$Rrel(R,A) \equiv \{z \in A \times A. \exists x\ y.\ z = \langle x, y\rangle \wedge R(x,y)\}$

lemma *RrelI* : $x \in A \Longrightarrow y \in A \Longrightarrow R(x,y) \Longrightarrow \langle x,y\rangle \in Rrel(R,A)$
⟨*proof*⟩

lemma *Rrel_mem*: $Rrel(mem,x) = Memrel(x)$
⟨*proof*⟩

lemma *relation_Rrel*: $relation(Rrel(R,d))$
⟨*proof*⟩

lemma *field_Rrel*: $field(Rrel(R,d)) \subseteq d$
⟨*proof*⟩

lemma *Rrel_mono* : $A \subseteq B \Longrightarrow Rrel(R,A) \subseteq Rrel(R,B)$
⟨*proof*⟩

lemma *Rrel_restr_eq* : $Rrel(R,A) \cap B \times B = Rrel(R,A \cap B)$
⟨*proof*⟩

lemma *field_Memrel* : $field(Memrel(A)) \subseteq A$

⟨*proof*⟩

lemma *restrict_trancl_Rrel*:
assumes $R(w,y)$
shows $restrict(f,Rrel(R,d)\text{-``}\{y\})\text{`}w$
$= restrict(f,(Rrel(R,d)\hat{}+)\text{-``}\{y\})\text{`}w$

⟨*proof*⟩

lemma *restrict_trans_eq*:
  assumes $w \in y$
  shows *restrict*(*f*,*Memrel*(*eclose*({*x*}))-``{*y*})`*w*
    = *restrict*(*f*,(*Memrel*(*eclose*({*x*}))^+)-``{*y*})`*w*
  ⟨*proof*⟩

lemma *wf_eq_trancl*:
  assumes $\bigwedge f\ y\ .\ H(y,restrict(f,R\text{-}``\{y\})) = H(y,restrict(f,R\hat{}+\text{-}``\{y\}))$
  shows $wfrec(R,\ x,\ H) = wfrec(R\hat{}+,\ x,\ H)$ (**is** *wfrec*(*?r*,_,_) = *wfrec*(*?r'*,_,_))
⟨*proof*⟩

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*⟩
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*,β) *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*⟩

lemma *ordermap_restr_eq*:
  assumes *well_ord*(*X*,*r*)
  shows $ordermap(X,\ r) = ordermap(X,\ r \cap X{\times}X)$
⟨*proof*⟩

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

This theory encapsulates some ML utilities

⟨*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*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *top_iff_sats* [*iff_sats*]:
env ∈ *list*(*A*) ⟹ *0* ∈ *A* ⟹ *sats*(*A*, *Exists*(*Equal*(*0*,*0*)), *env*)
⟨*proof*⟩

lemma *prefix1_iff_sats*[*iff_sats*]:

assumes
$x \in nat$ $env \in list(A)$ $0 \in A$ $a \in A$
shows
$a = nth(x,env) \longleftrightarrow sats(A,\ Equal(0,x\#+1),\ Cons(a,env))$
$nth(x,env) = a \longleftrightarrow sats(A,\ Equal(x\#+1,0),\ Cons(a,env))$
$a \in nth(x,env) \longleftrightarrow sats(A,\ Member(0,x\#+1),\ Cons(a,env))$
$nth(x,env) \in a \longleftrightarrow sats(A,\ Member(x\#+1,0),\ Cons(a,env))$
⟨*proof*⟩

lemma *prefix2_iff_sats*[*iff_sats*]:
assumes
$x \in nat$ $env \in list(A)$ $0 \in A$ $a \in A$ $b \in A$
shows
$b = nth(x,env) \longleftrightarrow sats(A,\ Equal(1,x\#+2),\ Cons(a,Cons(b,env)))$
$nth(x,env) = b \longleftrightarrow sats(A,\ Equal(x\#+2,1),\ Cons(a,Cons(b,env)))$
$b \in nth(x,env) \longleftrightarrow sats(A,\ Member(1,x\#+2),\ Cons(a,Cons(b,env)))$
$nth(x,env) \in b \longleftrightarrow sats(A,\ Member(x\#+2,1),\ Cons(a,Cons(b,env)))$
⟨*proof*⟩

lemma *prefix3_iff_sats*[*iff_sats*]:
assumes
$x \in nat$ $env \in list(A)$ $0 \in A$ $a \in A$ $b \in A$ $c \in A$
shows
$c = nth(x,env) \longleftrightarrow sats(A,\ Equal(2,x\#+3),\ Cons(a,Cons(b,Cons(c,env))))$
$nth(x,env) = c \longleftrightarrow sats(A,\ Equal(x\#+3,2),\ Cons(a,Cons(b,Cons(c,env))))$
$c \in nth(x,env) \longleftrightarrow sats(A,\ Member(2,x\#+3),\ Cons(a,Cons(b,Cons(c,env))))$
$nth(x,env) \in c \longleftrightarrow sats(A,\ Member(x\#+3,2),\ Cons(a,Cons(b,Cons(c,env))))$
⟨*proof*⟩

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*: $[\![nth(n,l) = x;\ n \in nat]\!] \Longrightarrow nth(succ(n),\ Cons(a,l)) = x$
⟨*proof*⟩

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*: $(\mu\ \alpha.\ R(\alpha)) = (\mu\ \alpha.\ Ord(\alpha) \wedge R(\alpha))$
⟨*proof*⟩

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*⟩

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*$(q,i) \in formula$
⟨*proof*⟩

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
    $[\![y \in nat;\ env \in list(A)\ ;\ 0{\in}A]\!]$
    $\Longrightarrow sats(A,\ least\_fm(p,y),\ env) \longleftrightarrow$
        $least(\#\#A,\ P,\ nth(y,env))$
  ⟨*proof*⟩

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
  $[\![nth(j,env) = y;\ j \in nat;\ env \in list(A);\ 0{\in}A]\!]$
   $\Longrightarrow least(\#\#A,\ is\_Q,\ y) \longleftrightarrow sats(A,\ least\_fm(q,j),\ env)$
  ⟨*proof*⟩

lemma *least_conj*: $a{\in}M \Longrightarrow least(\#\#M,\ \lambda x.\ x{\in}M \wedge Q(x),a) \longleftrightarrow least(\#\#M,Q,a)$
  ⟨*proof*⟩

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*⟩

lemma *least_abs*:
  assumes $\bigwedge x.\ Q(x) \Longrightarrow Ord(x) \Longrightarrow \exists\, y[M].\ Q(y) \wedge Ord(y)\ M(a)$
  shows $least(M,Q,a) \longleftrightarrow a = (\mu\ x.\ Q(x))$
  ⟨*proof*⟩

lemma *Least_closed*:
  assumes $\bigwedge x.\ Q(x) \Longrightarrow Ord(x) \Longrightarrow \exists\, y[M].\ Q(y) \wedge Ord(y)$
  shows $M(\mu\ x.\ Q(x))$
  ⟨*proof*⟩

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 $least(M,Q,a) \longleftrightarrow a = (\mu\ x.\ Q(x))$
  ⟨*proof*⟩

lemma *Least_closed'*:

assumes $\bigwedge x.\ Q(x) \Longrightarrow M(x)$
shows $M(\mu\ x.\ Q(x))$
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *is_If_fm_iff_sats* [*iff_sats*]:
assumes *Qsats*: *Q* ⟷ *A*, *env* ⊨ φ **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 $is\_If(\#\#A,Q,ta,fa,ra) \longleftrightarrow A,\ env \models is\_If\_fm(\varphi,t,f,r)$
⟨*proof*⟩

lemma *arity_is_If_fm* [*arity*]:
$\varphi \in formula \Longrightarrow t \in nat \Longrightarrow f \in nat \Longrightarrow r \in nat \Longrightarrow$
$arity(is\_If\_fm(\varphi,\ t,\ f,\ r)) = arity(\varphi) \cup succ(t) \cup succ(r) \cup succ(f)$
⟨*proof*⟩

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*⟩

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
$is\_recursor(M,a,is\_b,k,r) \longleftrightarrow r = recursor(a,b,k)$
⟨*proof*⟩

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*);
∀ *x*[*M*]. ∀ *g*[*M*]. *function*(*g*) ⟶ *M*(*H*(*x*,*g*));
r-''{*a*}⊆*A*; *a* ∈ *A*|]
==> *is_wfrec_on*(*M*,*MH*,*A*,*r*,*a*,*z*) ⟷ *z*=*wfrec*[*A*](*r*,*a*,*H*)
⟨*proof*⟩

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*⟩

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*⟩

declare *relative_abs*[*Rel*]

declare *datatype_abs*[*Rel*]

⟨*ML*⟩

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*] :
  $\llbracket$ *M*(*x*) ; *M*(*s*) $\rrbracket \Longrightarrow$ *is_singleton*(*M*,*x*,*s*) $\longleftrightarrow$ *s* = {*x*}
  ⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_trivial*) *singleton_closed* [*simp*]:
  M(*x*) $\Longrightarrow$ *M*({*x*})
  ⟨*proof*⟩

lemma (**in** *M_trivial*) *Upair_closed*[*simp*]: *M*(*a*) $\Longrightarrow$ *M*(*b*) $\Longrightarrow$ *M*(*Upair*(*a*,*b*))
  ⟨*proof*⟩

lemma (**in** *M_trivial*) *upair_closed*[*simp*] : *M*(*x*) $\Longrightarrow$ *M*(*y*) $\Longrightarrow$ *M*({*x*,*y*})
  ⟨*proof*⟩

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*) $\Longrightarrow$ *M*(*z*) $\Longrightarrow$ *is_f*(*a*,*z*) $\longleftrightarrow$ *z* = *f*(*a*) **and**
    is_g_abs:⋀*a z*. *M*(*a*) $\Longrightarrow$ *M*(*z*) $\Longrightarrow$ *is_g*(*a*,*z*) $\longleftrightarrow$ *z* = *g*(*a*) **and**
    g_closed:⋀*a*. *M*(*a*) $\Longrightarrow$ *M*(*g*(*a*))
    M(*a*) *M*(*w*)
  shows
    is_hcomp(*M*,*is_f*,*is_g*,*a*,*w*) $\longleftrightarrow$ *w* = *f*(*g*(*a*))
  ⟨*proof*⟩

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* $\Longrightarrow$ *b*∈*nat* $\Longrightarrow$ *z*∈*M* $\Longrightarrow$
        is_f(*nth*(*a*,*Cons*(*z*,*env*)),*nth*(*b*,*Cons*(*z*,*env*))) $\longleftrightarrow$ *sats*(*M*,*pf*(*a*,*b*),*Cons*(*z*,*env*))
    and
    g_iff_sats:⋀*a b z*. *a*∈*nat* $\Longrightarrow$ *b*∈*nat* $\Longrightarrow$ *z*∈*M* $\Longrightarrow$
        is_g(*nth*(*a*,*Cons*(*z*,*env*)),*nth*(*b*,*Cons*(*z*,*env*))) $\longleftrightarrow$ *sats*(*M*,*pg*(*a*,*b*),*Cons*(*z*,*env*))
    and

$a{\in}nat$ $w{\in}nat$ $env{\in}list(M)$
shows
$sats(M,hcomp\_fm(pf,pg,a,w),env) \longleftrightarrow is\_hcomp(\#\#M,is\_f,is\_g,nth(a,env),nth(w,env))$
⟨*proof*⟩

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
$hcomp\_r(M,is\_f,is\_g,a,w) \longleftrightarrow w = f(g(a))$
⟨*proof*⟩

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*⟩

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*⟩

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*$,a,b,w) \longleftrightarrow w = f(g1(a,b),g2(a,b))$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*$(M,a,b) \longleftrightarrow a{<}b$
⟨*proof*⟩

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*$(M,a,b) \longleftrightarrow a{\le}b$
⟨*proof*⟩

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
$d{=}d'$
⟨*proof*⟩

lemma *is_Pow_witness*: $M(r) \Longrightarrow \exists\, d[M].$ *is_Pow*(M,r,d)

⟨proof⟩

lemma *is_Pow_closed* : ⟦ $M(r)$;*is_Pow*(M,r,d) ⟧ $\Longrightarrow M(d)$
⟨proof⟩

lemma *Pow_rel_closed*[*intro*,*simp*]: $M(r) \Longrightarrow M($*Pow_rel*$(M,r))$
⟨proof⟩

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⟩

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*$(M,r) \longleftrightarrow A \subseteq r$
⟨proof⟩

lemma *Pow_rel_char*: $M(r) \Longrightarrow$ *Pow_rel*$(M,r) = \{A \in Pow(r).\ M(A)\}$
⟨proof⟩

lemma *mem_Pow_rel_abs*: $M(a) \Longrightarrow M(r) \Longrightarrow a \in$ *Pow_rel*$(M,r) \longleftrightarrow a \in Pow(r)$
⟨proof⟩

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)
⟨proof⟩

end

definition — FIX THIS: not completely relational. Can it be?
Sigfun :: [*i*,*i*⇒*i*]⇒*i* **where**
Sigfun(*x*,*B*) ≡ ⋃ *y*∈*B*(*x*). {⟨*x*,*y*⟩}

lemma *Sigma_Sigfun*: *Sigma*(*A*,*B*) = ⋃ {*Sigfun*(*x*,*B*) . *x*∈*A*}
⟨*proof*⟩

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*) ≡ *M*(*Sd*) ∧ (∃ *RB*[*M*]. *is_Replace*(*M*,*B*(*x*),λ*y z*. *z*={⟨*x*,*y*⟩},*RB*)
∧ *big_union*(*M*,*RB*,*Sd*))

context *M_trivial*
begin

lemma *is_Sigfun_abs*:
assumes
strong_replacement(*M*,λ*y z*. *z*={⟨*x*,*y*⟩})
M(*x*) *M*(*B*(*x*)) *M*(*Sd*)
shows
is_Sigfun(*M*,*x*,*B*,*Sd*) ⟷ *Sd* = *Sigfun*(*x*,*B*)
⟨*proof*⟩

lemma *Sigfun_closed*:
assumes
strong_replacement(*M*, λ*y z*. *y* ∈ *B*(*x*) ∧ *z* = {⟨*x*, *y*⟩})
M(*x*) *M*(*B*(*x*))
shows
M(*Sigfun*(*x*,*B*))
⟨*proof*⟩

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*) ≡ *M*(*S*) ∧ (∃ *RSf*[*M*].
is_Replace(*M*,*A*,λ*x z*. *z*=*Sigfun*(*x*,*B*),*RSf*) ∧ *big_union*(*M*,*RSf*,*S*))

locale *M_Pi* = *M_basic* +
assumes
Pi_separation: *M*(*A*) ⟹ *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*⟩

lemma *Sigma_closed*[*intro*,*simp*]: $M(Sigma(A,B))$
⟨*proof*⟩

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 :: $[i{\Rightarrow}o,i,i{\Rightarrow}i] \Rightarrow i$ (‹*Pi*–'(_,_')›) **where**
Pi_rel$(M,A,B) \equiv$ *THE d. is_Pi*(M,A,B,d)

abbreviation
Pi_r_set :: $[i,i,i{\Rightarrow}i] \Rightarrow i$ (‹*Pi*–'(_,_')›) **where**
Pi_r_set$(M,A,B) \equiv$ *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'*
  ⟨*proof*⟩

lemma *is_Pi_witness*: ∃ *d*[*M*]. *is_Pi*(*M*,*A*,*B*,*d*)
  ⟨*proof*⟩

lemma *is_Pi_closed* : *is_Pi*(*M*,*A*,*B*,*d*) ⟹ *M*(*d*)
  ⟨*proof*⟩

lemma *Pi_rel_closed*[*intro*,*simp*]: *M*(*Pi_rel*(*M*,*A*,*B*))
⟨*proof*⟩

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*) ⟷ *d* = *Pi_rel*(*M*,*A*,*B*)
⟨*proof*⟩

lemma *def_Pi_rel*:
    Pi_rel(*M*,*A*,*B*) = {*f*∈*Pow_rel*(*M*,*Sigma*(*A*,*B*)). *A*⊆*domain*(*f*) ∧ *function*(*f*)}
⟨*proof*⟩

lemma *Pi_rel_char*: *Pi_rel*(*M*,*A*,*B*) = {*f*∈*Pi*(*A*,*B*). *M*(*f*)}
  ⟨*proof*⟩

lemma *mem_Pi_rel_abs*:
  assumes *M*(*f*)
  shows *f* ∈ *Pi_rel*(*M*,*A*,*B*) ⟷ *f* ∈ *Pi*(*A*,*B*)
  ⟨*proof*⟩

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*) ⟹ *N*(*x*)
begin

lemma *Pi_rel_transfer*: $Pi^M(A,B) \subseteq Pi^N(A,B)$
⟨*proof*⟩

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*]: $Pi^M(0,B) = \{0\}$
⟨*proof*⟩

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
$f \in Pi\_rel(M,A,B) \longleftrightarrow function(f) \wedge f \subseteq Sigma(A,B) \wedge A \subseteq domain(f)$
⟨*proof*⟩

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*⟩

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 $f \in Pi^M(A,B)$
⟨*proof*⟩

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 $f \in Pi^M(A,C)$
⟨*proof*⟩

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*)}
⟨*proof*⟩

lemma *arity_pair_fm* [*arity*] : ⟦ *t1*∈*nat* ; *t2*∈*nat* ; *p*∈*nat* ⟧ ⟹
arity(*pair_fm*(*t1*,*t2*,*p*)) = ⋃ {*succ*(*t1*),*succ*(*t2*),*succ*(*p*)}
⟨*proof*⟩

lemma *arity_composition_fm* [*arity*] :
⟦ *r*∈*nat* ; *s*∈*nat* ; *t*∈*nat* ⟧ ⟹ *arity*(*composition_fm*(*r*,*s*,*t*)) = ⋃ {*succ*(*r*), *succ*(*s*), *succ*(*t*)}
⟨*proof*⟩

lemma *arity_domain_fm* [*arity*] :
⟦ *r*∈*nat* ; *z*∈*nat* ⟧ ⟹ *arity*(*domain_fm*(*r*,*z*)) = *succ*(*r*) ∪ *succ*(*z*)
⟨*proof*⟩

lemma *arity_range_fm* [*arity*] :
⟦ *r*∈*nat* ; *z*∈*nat* ⟧ ⟹ *arity*(*range_fm*(*r*,*z*)) = *succ*(*r*) ∪ *succ*(*z*)
⟨*proof*⟩

lemma *arity_union_fm* [*arity*] :
⟦ *x*∈*nat* ; *y*∈*nat* ; *z*∈*nat* ⟧ ⟹ *arity*(*union_fm*(*x*,*y*,*z*)) = ⋃ {*succ*(*x*), *succ*(*y*), *succ*(*z*)}
⟨*proof*⟩

lemma *arity_image_fm* [*arity*] :
$[\![x \in nat ; y \in nat ; z \in nat]\!] \Longrightarrow arity(image\_fm(x,y,z)) = \bigcup \{succ(x), succ(y), succ(z)\}$
⟨*proof*⟩

lemma *arity_pre_image_fm* [*arity*] :
$[\![x \in nat ; y \in nat ; z \in nat]\!] \Longrightarrow arity(pre\_image\_fm(x,y,z)) = \bigcup \{succ(x), succ(y), succ(z)\}$
⟨*proof*⟩

lemma *arity_big_union_fm* [*arity*] :
$[\![x \in nat ; y \in nat]\!] \Longrightarrow arity(big\_union\_fm(x,y)) = succ(x) \cup succ(y)$
⟨*proof*⟩

lemma *arity_fun_apply_fm* [*arity*] :
$[\![x \in nat ; y \in nat ; f \in nat]\!] \Longrightarrow$
$arity(fun\_apply\_fm(f,x,y)) = succ(f) \cup succ(x) \cup succ(y)$
⟨*proof*⟩

lemma *arity_field_fm* [*arity*] :
$[\![r \in nat ; z \in nat]\!] \Longrightarrow arity(field\_fm(r,z)) = succ(r) \cup succ(z)$
⟨*proof*⟩

lemma *arity_empty_fm* [*arity*]:
$[\![r \in nat]\!] \Longrightarrow arity(empty\_fm(r)) = succ(r)$
⟨*proof*⟩

lemma *arity_cons_fm* [*arity*] :
$[\![x \in nat;y \in nat;z \in nat]\!] \Longrightarrow arity(cons\_fm(x,y,z)) = succ(x) \cup succ(y) \cup succ(z)$
⟨*proof*⟩

lemma *arity_succ_fm* [*arity*] :
$[\![x \in nat;y \in nat]\!] \Longrightarrow arity(succ\_fm(x,y)) = succ(x) \cup succ(y)$
⟨*proof*⟩

lemma *arity_number1_fm* [*arity*] :
$[\![r \in nat]\!] \Longrightarrow arity(number1\_fm(r)) = succ(r)$
⟨*proof*⟩

lemma *arity_function_fm* [*arity*] :
$[\![r \in nat]\!] \Longrightarrow arity(function\_fm(r)) = succ(r)$
⟨*proof*⟩

lemma *arity_relation_fm* [*arity*] :
$[\![r \in nat]\!] \Longrightarrow arity(relation\_fm(r)) = succ(r)$
⟨*proof*⟩

lemma *arity_restriction_fm* [*arity*] :

$\llbracket$ *r*∈*nat* ; *z*∈*nat* ; *A*∈*nat* $\rrbracket$ ⟹ *arity*(*restriction_fm*(*A*,*z*,*r*)) = *succ*(*A*) ∪ *succ*(*r*) ∪ *succ*(*z*)
⟨*proof*⟩

lemma *arity_typed_function_fm* [*arity*] :
$\llbracket$ *x*∈*nat* ; *y*∈*nat* ; *f*∈*nat* $\rrbracket$ ⟹
arity(*typed_function_fm*(*f*,*x*,*y*)) = ⋃ {*succ*(*f*), *succ*(*x*), *succ*(*y*)}
⟨*proof*⟩

lemma *arity_subset_fm* [*arity*] :
$\llbracket$*x*∈*nat* ; *y*∈*nat*$\rrbracket$ ⟹ *arity*(*subset_fm*(*x*,*y*)) = *succ*(*x*) ∪ *succ*(*y*)
⟨*proof*⟩

lemma *arity_transset_fm* [*arity*] :
$\llbracket$*x*∈*nat*$\rrbracket$ ⟹ *arity*(*transset_fm*(*x*)) = *succ*(*x*)
⟨*proof*⟩

lemma *arity_ordinal_fm* [*arity*] :
$\llbracket$*x*∈*nat*$\rrbracket$ ⟹ *arity*(*ordinal_fm*(*x*)) = *succ*(*x*)
⟨*proof*⟩

lemma *arity_limit_ordinal_fm* [*arity*] :
$\llbracket$*x*∈*nat*$\rrbracket$ ⟹ *arity*(*limit_ordinal_fm*(*x*)) = *succ*(*x*)
⟨*proof*⟩

lemma *arity_finite_ordinal_fm* [*arity*] :
$\llbracket$*x*∈*nat*$\rrbracket$ ⟹ *arity*(*finite_ordinal_fm*(*x*)) = *succ*(*x*)
⟨*proof*⟩

lemma *arity_omega_fm* [*arity*] :
$\llbracket$*x*∈*nat*$\rrbracket$ ⟹ *arity*(*omega_fm*(*x*)) = *succ*(*x*)
⟨*proof*⟩

lemma *arity_cartprod_fm* [*arity*] :
$\llbracket$ *A*∈*nat* ; *B*∈*nat* ; *z*∈*nat* $\rrbracket$ ⟹ *arity*(*cartprod_fm*(*A*,*B*,*z*)) = *succ*(*A*) ∪ *succ*(*B*) ∪ *succ*(*z*)
⟨*proof*⟩

lemma *arity_singleton_fm* [*arity*] :
$\llbracket$*x*∈*nat* ; *t*∈*nat*$\rrbracket$ ⟹ *arity*(*singleton_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
⟨*proof*⟩

lemma *arity_Memrel_fm* [*arity*] :
$\llbracket$*x*∈*nat* ; *t*∈*nat*$\rrbracket$ ⟹ *arity*(*Memrel_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
⟨*proof*⟩

lemma *arity_quasinat_fm* [*arity*] :
$\llbracket$*x*∈*nat*$\rrbracket$ ⟹ *arity*(*quasinat_fm*(*x*)) = *succ*(*x*)

⟨proof⟩

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*))))
⟨proof⟩

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*)))))
⟨proof⟩

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*))
⟨proof⟩

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⟩

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*(*pred*(*i*)))))))))))
⟨proof⟩

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⟩

lemma *arity_mem_eclose_fm* [*arity*] :
assumes *x*∈*nat* *t*∈*nat*
shows *arity*(*mem_eclose_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
⟨proof⟩

lemma *arity_is_eclose_fm* [*arity*] :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*is_eclose_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
⟨proof⟩

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*))
⟨proof⟩

schematic_goal *arity_least_fm′*:
assumes
i ∈ *nat* *q* ∈ *formula*
shows
arity(*least_fm*(*q*,*i*)) ≡ *?ar*
⟨*proof*⟩

lemma *arity_least_fm* [*arity*] :
assumes
i ∈ *nat* *q* ∈ *formula*
shows
arity(*least_fm*(*q*,*i*)) = *succ*(*i*) ∪ *pred*(*arity*(*q*))
⟨*proof*⟩

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*)))
⟨*proof*⟩

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*))))
⟨*proof*⟩

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*))
⟨*proof*⟩

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

Discipline for *fst* ⟨*ML*⟩

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*))
⟨*ML*⟩
notation *fst_fm* (‹·*fst′*(_′) *is* _·›)

⟨*ML*⟩

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

⟨*ML*⟩

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*))
⟨*ML*⟩
notation *snd_fm* (‹·*snd*′(_′) *is* _·›)
⟨*ML*⟩

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

⟨*ML*⟩

context *M_trans*
begin

lemma *fst_snd_closed*:
assumes *M*(*p*)
shows *M*(*fst*(*p*)) ∧ *M*(*snd*(*p*))
⟨*proof*⟩

lemma *fst_closed*[*intro*,*simp*]: *M*(*x*) ⟹ *M*(*fst*(*x*))
⟨*proof*⟩

lemma *snd_closed*[*intro*,*simp*]: *M*(*x*) ⟹ *M*(*snd*(*x*))
⟨*proof*⟩

lemma *fst_abs* [*absolut*]:
⟦*M*(*p*); *M*(*x*) ⟧ ⟹ *is_fst*(*M*,*p*,*x*) ⟷ *x* = *fst*(*p*)
⟨*proof*⟩

lemma *snd_abs* [*absolut*]:
⟦*M*(*p*); *M*(*y*) ⟧ ⟹ *is_snd*(*M*,*p*,*y*) ⟷ *y* = *snd*(*p*)
⟨*proof*⟩

lemma *empty_rel_abs* : *M*(*x*) ⟹ *M*(*0*) ⟹ *x* = *0* ⟷ *x* = (*THE d*. *M*(*d*) ∧ *empty*(*M*, *d*))
⟨*proof*⟩

lemma *fst_rel_abs*:

⟦$M(p)$ ⟧ ⟹ *fst*(p) = *fst_rel*(M,p)
⟨*proof*⟩

lemma *snd_rel_abs*:
⟦$M(p)$ ⟧ ⟹ *snd*(p) = *snd_rel*(M,p)
⟨*proof*⟩

end

⟨*ML*⟩
context *M_trans*
begin

lemma *minimum_closed*[*simp*,*intro*]:
assumes $M(A)$
shows M(*minimum*(r,A))
⟨*proof*⟩

lemma *first_abs* :
assumes $M(B)$
shows *first*(z,B,r) ⟷ *first_rel*(M,z,B,r)
⟨*proof*⟩
lemma *minimum_abs*:
assumes $M(B)$
shows *minimum*(r,B) = *minimum_rel*(M,r,B)
⟨*proof*⟩

end

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

definition
is_function_space :: [*i*⇒*o*,*i*,*i*,*i*] ⇒ *o* **where**
is_function_space(M,A,B,*fs*) ≡ M(*fs*) ∧ *is_funspace*(M,A,B,*fs*)

definition
function_space_rel :: [*i*⇒*o*,*i*,*i*] ⇒ *i* **where**
function_space_rel(M,A,B) ≡ *THE d. is_function_space*(M,A,B,d)

⟨*ML*⟩

abbreviation
function_space_r :: [*i*,*i*⇒*o*,*i*] ⇒ *i* (‹_ →- _› [*61*,*1*,*61*] *60*) **where**
$A \to^M B$ ≡ *function_space_rel*(M,A,B)

abbreviation
function_space_r_set :: [*i*,*i*,*i*] ⇒ *i* (‹_ →- _› [*61*,*1*,*61*] *60*) **where**
function_space_r_set(A,M) ≡ *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
    $d=d'$
  ⟨*proof*⟩

lemma *is_function_space_witness*:
  assumes $M(A)$ $M(B)$
  shows $\exists\, d[M]$. *is_function_space*(M,A,B,d)
⟨*proof*⟩

lemma *is_function_space_closed* :
 is_function_space$(M,A,B,d) \Longrightarrow M(d)$
  ⟨*proof*⟩
lemma *function_space_rel_closed*[*intro*,*simp*]:
  assumes $M(x)$ $M(y)$
  shows $M($*function_space_rel*$(M,x,y))$
⟨*proof*⟩

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*⟩

lemma *def_function_space_rel*:
  assumes $M(A)$ $M(y)$
  shows *function_space_rel*$(M,A,y) =$ *Pi_rel*$(M,A,\lambda\_.\ y)$
⟨*proof*⟩

lemma *function_space_rel_char*:
  assumes $M(A)$ $M(y)$
  shows *function_space_rel*$(M,A,y) = \{f \in A \to y.\ M(f)\}$
⟨*proof*⟩

lemma *mem_function_space_rel_abs*:
  assumes $M(A)$ $M(y)$ $M(f)$
  shows $f \in$ *function_space_rel*$(M,A,y) \longleftrightarrow f \in A \to y$
  ⟨*proof*⟩

end

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

lemma *function_space_rel_transfer*: $M(A) \Longrightarrow M(B) \Longrightarrow$
    $function\_space\_rel(M,A,B) \subseteq function\_space\_rel(N,A,B)$
  ⟨*proof*⟩

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:: $[i \Rightarrow o,i,i] \Rightarrow o$ **where**
  $injP\_rel(M,A,f) \equiv \forall w[M].\ \forall x[M].\ \forall fw[M].\ \forall fx[M].\ w \in A \wedge x \in A \wedge$
    $is\_apply(M,f,w,fw) \wedge is\_apply(M,f,x,fx) \wedge fw=fx \longrightarrow w=x$

⟨*ML*⟩

context *M_basic*
begin

— I'm undecided on keeping the relative quantifiers here. Same with *surjP* below. It might relieve from changing $?P(?x) \Longrightarrow \exists x.\ ?P(x)$ $(\bigwedge x.\ ?P(x)) \Longrightarrow \forall x.\ ?P(x)$ to $[\![?P(?x);\ ?M(?x)]\!] \Longrightarrow \exists x[?M].\ ?P(x)$ $(\bigwedge x.\ ?M(x) \Longrightarrow ?P(x)) \Longrightarrow \forall x[?M].\ ?P(x)$ in some proofs. I wonder if this escalates well. Assuming that all terms appearing in the "def_" theorem are in *M* and using $[\![?y \in ?x;\ M(?x)]\!] \Longrightarrow M(?y)$, it might do.
lemma *def_injP_rel*:
  assumes
    $M(A)$ $M(f)$
  shows
    $injP\_rel(M,A,f) \longleftrightarrow (\forall w[M].\ \forall x[M].\ w \in A \wedge x \in A \wedge f`w=f`x \longrightarrow w=x)$
  ⟨*proof*⟩

end

13.3 Discipline for *inj*

term *function_space_rel*

⟨*ML*⟩

definition
is_inj :: $[i\Rightarrow o,i,i,i]\Rightarrow o$ **where**
is_inj$(M,A,B,I) \equiv M(I) \wedge (\exists F[M].$ *is_function_space*$(M,A,B,F) \wedge$
is_Collect$(M,F,$*injP_rel*$(M,A),I))$

declare *typed_function_iff_sats Collect_iff_sats* [*iff_sats*]

⟨*ML*⟩
notation *is_function_space_fm* (‹·_ → _ *is* _·›)

⟨*ML*⟩
notation *is_inj_fm* (‹·*inj*′(_,_′) *is* _·›)

⟨*ML*⟩

lemma *arity_is_inj_fm*[*arity*]:
$A \in nat \Longrightarrow$
$B \in nat \Longrightarrow I \in nat \Longrightarrow arity($*is_inj_fm*$(A, B, I)) = succ(A) \cup succ(B) \cup succ(I)$
⟨*proof*⟩

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

abbreviation
inj_r_set :: $[i,i,i] \Rightarrow i$ (‹*inj*−′(_,_′)›) **where**
inj_r_set$(M) \equiv$ *inj_rel*$(\#\#M)$

locale *M_inj* = *M_Pi* +
assumes
injP_separation: $M(r) \Longrightarrow$ *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
$d=d'$
⟨*proof*⟩

lemma *is_inj_witness*: $M(r) \Longrightarrow M(B)\Longrightarrow \exists d[M].$ *is_inj*(M,r,B,d)

⟨*proof*⟩

lemma *is_inj_closed* :
is_inj(M,x,y,d) $\Longrightarrow$ $M(d)$
⟨*proof*⟩

lemma *inj_rel_closed*[*intro*,*simp*]:
assumes $M(x)$ $M(y)$
shows M(*inj_rel*(M,x,y))
⟨*proof*⟩

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*⟩

lemma *def_inj_rel*:
assumes $M(A)$ $M(B)$
shows *inj_rel*(M,A,B) =
{f ∈ *function_space_rel*(M,A,B). ∀ w[M]. ∀ x[M]. w∈A ∧ x∈A ∧ f‘w = f‘x ⟶ w=x}
(**is** _ = *Collect*(_,?P))
⟨*proof*⟩

lemma *inj_rel_char*:
assumes $M(A)$ $M(B)$
shows *inj_rel*(M,A,B) = {f ∈ *inj*(A,B). $M(f)$}
⟨*proof*⟩

end

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

lemma *inj_rel_transfer*: $M(A)$ $\Longrightarrow$ $M(B)$ $\Longrightarrow$ *inj_rel*(M,A,B) ⊆ *inj_rel*(N,A,B)
⟨*proof*⟩

end

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

⟨*ML*⟩

context *M_basic*
begin

lemma *def_surjP_rel*:
  assumes
    $M(A)\ M(B)\ M(f)$
  shows
    $surjP\_rel(M,A,B,f) \longleftrightarrow (\forall y[M].\ \exists x[M].\ y \in B \longrightarrow x \in A \wedge f`x=y)$
  ⟨*proof*⟩

end

13.4 Discipline for *surj*

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

⟨*ML*⟩
notation *is_surj_fm* (‹·surj′(_,_′) is _·›)

definition
  surj_rel :: $[i \Rightarrow o,i,i] \Rightarrow i$ (‹surj–′(_,_′)›) **where**
  $surj\_rel(M,A,B) \equiv THE\ d.\ is\_surj(M,A,B,d)$

abbreviation
  surj_r_set :: $[i,i,i] \Rightarrow i$ (‹surj–′(_,_′)›) **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
    $d=d'$
  ⟨*proof*⟩

lemma *is_surj_witness*: $M(r) \Longrightarrow M(B) \Longrightarrow \exists\, d[M].\ is\_surj(M,r,B,d)$
  ⟨*proof*⟩

lemma *is_surj_closed* :
 $is\_surj(M,x,y,d) \Longrightarrow M(d)$
  ⟨*proof*⟩

lemma *surj_rel_closed*[*intro*,*simp*]:
  assumes $M(x)$ $M(y)$
  shows $M(surj\_rel(M,x,y))$
⟨*proof*⟩

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*⟩

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*⟩

lemma *surj_rel_char*:
  assumes $M(A)$ $M(B)$
  shows $surj\_rel(M,A,B) = \{f \in surj(A,B).\ M(f)\}$
⟨*proof*⟩

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*: $M(A) \Longrightarrow M(B) \Longrightarrow surj\_rel(M,A,B) \subseteq surj\_rel(N,A,B)$
  ⟨*proof*⟩

end

definition

is_Int :: [*i*⇒*o*,*i*,*i*,*i*]⇒*o* **where**
is_Int(*M*,*A*,*B*,*I*) ≡ *M*(*I*) ∧ (∀ *x*[*M*]. *x* ∈ *I* ⟷ *x* ∈ *A* ∧ *x* ∈ *B*)

⟨*ML*⟩
notation *is_Int_fm* (‹_ ∩ _ *is* _›)

context *M_basic*
begin

lemma *is_Int_closed* :
is_Int(*M*,*A*,*B*,*I*) ⟹ *M*(*I*)
⟨*proof*⟩

lemma *is_Int_abs*:
assumes
M(*A*) *M*(*B*) *M*(*I*)
shows
is_Int(*M*,*A*,*B*,*I*) ⟷ *I* = *A* ∩ *B*
⟨*proof*⟩

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*⟩

Note: ⟦*M*(*?A*); *M*(*?B*)⟧ ⟹ *M*(*?A* ∩ *?B*) already in *ZF-Constructible.Relative*.

end

13.5 Discipline for *bij*

⟨*ML*⟩
notation *is_bij_fm* (‹·*bij′*(_,_′) *is* _·›)

abbreviation
bij_r_class :: [*i*⇒*o*,*i*,*i*] ⇒ *i* (‹*bij*−′(_,_′)›) **where**
bij_r_class ≡ *bij_rel*

abbreviation
bij_r_set :: [*i*,*i*,*i*] ⇒ *i* (‹*bij*−′(_,_′)›) **where**
bij_r_set(*M*) ≡ *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)
⟨*proof*⟩

lemma *bij_rel_closed*[*intro*,*simp*]:
assumes $M(x)$ $M(y)$
shows M(*bij_rel*(M,x,y))
⟨*proof*⟩

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) $\longleftrightarrow$ d = *bij_rel*(M,x,y)
⟨*proof*⟩

lemma *def_bij_rel*:
assumes $M(A)$ $M(B)$
shows *bij_rel*(M,A,B) = *inj_rel*(M,A,B) $\cap$ *surj_rel*(M,A,B)
⟨*proof*⟩

lemma *bij_rel_char*:
assumes $M(A)$ $M(B)$
shows *bij_rel*(M,A,B) = {f $\in$ *bij*(A,B). M(f)}
⟨*proof*⟩

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*: $M(A)$ $\Longrightarrow$ $M(B)$ $\Longrightarrow$ *bij_rel*(M,A,B) $\subseteq$ *bij_rel*(N,A,B)
⟨*proof*⟩

end

13.6 Discipline for (≈)

⟨*ML*⟩
notation *is_eqpoll_fm* (‹_ ≈ _›)

context *M_Perm* **begin**

⟨*ML*⟩

⟨proof⟩

end

abbreviation
eqpoll_r :: [*i*,*i*⇒*o*,*i*] => *o* (‹_ ≈- _› [*51*,*1*,*51*] *50*) **where**
$A \approx^M B \equiv$ *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*))
⟨proof⟩

end

context *M_N_Perm*
begin

lemma *eqpoll_rel_transfer*: **assumes** $A \approx^M B$ *M*(*A*) *M*(*B*)
shows $A \approx^N B$
⟨proof⟩

end

13.7 Discipline for (≲)

⟨ML⟩
notation *is_lepoll_fm* (‹·_ ≲ _·›)
⟨ML⟩

context *M_inj* **begin**

⟨ML⟩
⟨proof⟩

end

abbreviation
lepoll_r :: [*i*,*i*⇒*o*,*i*] => *o* (‹_ ≲- _› [*51*,*1*,*51*] *50*) **where**

$A \lesssim^M B \equiv$ *lepoll_rel*(*M*,*A*,*B*)

abbreviation
lepoll_r_set :: [*i*,*i*,*i*] ⇒ *o* (‹_ ≲– _› [*51*,*1*,*51*] *50*) **where**
lepoll_r_set(*A*,*M*) ≡ *lepoll_rel*(##*M*,*A*)

context *M_Perm*
begin

lemma *def_lepoll_rel*:
assumes
M(*A*) *M*(*B*)
shows
lepoll_rel(*M*,*A*,*B*) ⟷ (∃*f*[*M*]. *f* ∈ *inj_rel*(*M*,*A*,*B*))
⟨*proof*⟩

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*⟩

end

13.8 Discipline for (≺)

⟨*ML*⟩
notation *is_lesspoll_fm* (‹·_ ≺ _·›)
⟨*ML*⟩

context *M_Perm* **begin**

⟨*ML*⟩
⟨*proof*⟩

end

abbreviation
lesspoll_r :: [*i*,*i*⇒*o*,*i*] => *o* (‹_ ≺– _› [*51*,*1*,*51*] *50*) **where**
$A \prec^M B \equiv$ *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* ∈ *M*
  shows
    {*x*∈*b* . *P*(*x*)} = {*x*∈*b* . *P*(*x*)} ∩ *M*
    ⟨*proof*⟩

lemma (**in** *M_trivial*) *family_union_closed*: ⟦*strong_replacement*(*M*, *λx y. y* = *f*(*x*)); *M*(*A*); ∀ *x*∈*A*. *M*(*f*(*x*))⟧
      ⟹ *M*(⋃ *x*∈*A*. *f*(*x*))
  ⟨*proof*⟩

lemma (**in** *M_trivial*) *family_union_closed'*: ⟦*strong_replacement*(*M*, *λx y. x*∈*A* ∧ *y* = *f*(*x*)); *M*(*A*); ∀ *x*∈*A*. *M*(*f*(*x*))⟧
      ⟹ *M*(⋃ *x*∈*A*. *f*(*x*))
  ⟨*proof*⟩

definition
  HVfrom :: [*i*⇒*o*,*i*,*i*,*i*] ⇒ *i* **where**
  HVfrom(*M*,*A*,*x*,*f*) ≡ *A* ∪ (⋃ *y*∈*x*. {*a*∈*Pow*(*f*‘*y*). *M*(*a*)})

definition
  is_powapply :: [*i*⇒*o*,*i*,*i*,*i*] ⇒ *o* **where**
  is_powapply(*M*,*f*,*y*,*z*) ≡ *M*(*z*) ∧ (∃ *fy*[*M*]. *fun_apply*(*M*,*f*,*y*,*fy*) ∧ *powerset*(*M*,*fy*,*z*))

lemma *is_powapply_closed*: *is_powapply*(*M*,*f*,*y*,*z*) ⟹ *M*(*z*)
  ⟨*proof*⟩

definition
is_HVfrom :: [*i*⇒*o*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_HVfrom(*M*,*A*,*x*,*f*,*h*) ≡ ∃ *U*[*M*]. ∃ *R*[*M*]. *union*(*M*,*A*,*U*,*h*)
∧ *big_union*(*M*,*R*,*U*) ∧ *is_Replace*(*M*,*x*,*is_powapply*(*M*,*f*),*R*)

definition
is_Vfrom :: [*i*⇒*o*,*i*,*i*,*i*] ⇒ *o* **where**
is_Vfrom(*M*,*A*,*i*,*V*) ≡ *is_transrec*(*M*,*is_HVfrom*(*M*,*A*),*i*,*V*)

definition
is_Vset :: [*i*⇒*o*,*i*,*i*] ⇒ *o* **where**
is_Vset(*M*,*i*,*V*) ≡ ∃ *z*[*M*]. *empty*(*M*,*z*) ∧ *is_Vfrom*(*M*,*z*,*i*,*V*)

14.1 Formula synthesis

schematic_goal *sats_is_powapply_fm_auto*:
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*,*?ipa_fm*(*f*,*y*,*z*),*env*)
⟨*proof*⟩

schematic_goal *is_powapply_iff_sats*:
assumes
nth(*f*,*env*) = *ff* *nth*(*y*,*env*) = *yy* *nth*(*z*,*env*) = *zz* *0*∈*A*
f ∈ *nat* *y* ∈ *nat* *z* ∈ *nat* *env* ∈ *list*(*A*)
shows
is_powapply(##*A*,*ff*,*yy*,*zz*) ⟷ *sats*(*A*, *?is_one_fm*(*a*,*r*), *env*)
⟨*proof*⟩

definition
Hrank :: [*i*,*i*] ⇒ *i* **where**
Hrank(*x*,*f*) = (⋃ *y*∈*x*. *succ*(*f*‘*y*))

definition
PHrank :: [*i*⇒*o*,*i*,*i*,*i*] ⇒ *o* **where**
PHrank(*M*,*f*,*y*,*z*) ≡ *M*(*z*) ∧ (∃ *fy*[*M*]. *fun_apply*(*M*,*f*,*y*,*fy*) ∧ *successor*(*M*,*fy*,*z*))

definition
is_Hrank :: [*i*⇒*o*,*i*,*i*,*i*] ⇒ *o* **where**
is_Hrank(*M*,*x*,*f*,*hc*) ≡ (∃ *R*[*M*]. *big_union*(*M*,*R*,*hc*) ∧*is_Replace*(*M*,*x*,*PHrank*(*M*,*f*),*R*))

definition

rrank :: *i* ⇒ *i* **where**
rrank(*a*) ≡ *Memrel*(*eclose*({*a*}))^+

lemma (**in** *M_eclose*) *wf_rrank* : *M*(*x*) ⟹ *wf*(*rrank*(*x*))
⟨*proof*⟩

lemma (**in** *M_eclose*) *trans_rrank* : *M*(*x*) ⟹ *trans*(*rrank*(*x*))
⟨*proof*⟩

lemma (**in** *M_eclose*) *relation_rrank* : *M*(*x*) ⟹ *relation*(*rrank*(*x*))
⟨*proof*⟩

lemma (**in** *M_eclose*) *rrank_in_M* : *M*(*x*) ⟹ *M*(*rrank*(*x*))
⟨*proof*⟩

14.2 Absoluteness results

locale *M_eclose_pow* = *M_eclose* +
assumes
power_ax : *power_ax*(*M*) **and**
powapply_replacement : *M*(*f*) ⟹ *strong_replacement*(*M*,*is_powapply*(*M*,*f*))
and
HVfrom_replacement : ⟦ *M*(*i*) ; *M*(*A*) ⟧ ⟹
transrec_replacement(*M*,*is_HVfrom*(*M*,*A*),*i*) **and**
PHrank_replacement : *M*(*f*) ⟹ *strong_replacement*(*M*,*PHrank*(*M*,*f*)) **and**
is_Hrank_replacement : *M*(*x*) ⟹ *wfrec_replacement*(*M*,*is_Hrank*(*M*),*rrank*(*x*))

begin

lemma *is_powapply_abs*: ⟦*M*(*f*); *M*(*y*)⟧ ⟹ *is_powapply*(*M*,*f*,*y*,*z*) ⟷ *M*(*z*) ∧ *z* = {*x*∈*Pow*(*f*‘*y*). *M*(*x*)}
⟨*proof*⟩

lemma ⟦*M*(*A*); *M*(*x*); *M*(*f*); *M*(*h*) ⟧ ⟹
is_HVfrom(*M*,*A*,*x*,*f*,*h*) ⟷
(∃ *R*[*M*]. *h* = *A* ∪ ⋃ *R* ∧ *is_Replace*(*M*, *x*,λ*x y*. *y* = {*x* ∈ *Pow*(*f* ‘ *x*) . *M*(*x*)}, *R*))
⟨*proof*⟩

lemma *Replace_is_powapply*:
assumes
M(*R*) *M*(*A*) *M*(*f*)
shows
is_Replace(*M*, *A*, *is_powapply*(*M*, *f*), *R*) ⟷ *R* = *Replace*(*A*,*is_powapply*(*M*,*f*))
⟨*proof*⟩

lemma *powapply_closed*:
⟦ *M*(*y*) ; *M*(*f*) ⟧ ⟹ *M*({*x* ∈ *Pow*(*f* ‘ *y*) . *M*(*x*)})
⟨*proof*⟩

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*⟩

lemma *RepFun_powapply_closed*:
  assumes
    $M(f)$ $M(A)$
  shows
    $M(Replace(A,is\_powapply(M,f)))$
⟨*proof*⟩

lemma *Union_powapply_closed*:
  assumes
    $M(x)$ $M(f)$
  shows
    $M(\bigcup y\in x. \{a\in Pow(f`y). M(a)\})$
⟨*proof*⟩

lemma *relation2_HVfrom*: $M(A) \Longrightarrow relation2(M,is\_HVfrom(M,A),HVfrom(M,A))$
  ⟨*proof*⟩

lemma *HVfrom_closed* :
  $M(A) \Longrightarrow \forall x[M]. \forall g[M]. function(g) \longrightarrow M(HVfrom(M,A,x,g))$
  ⟨*proof*⟩

lemma *transrec_HVfrom*:
  assumes $M(A)$
  shows $Ord(i) \Longrightarrow \{x\in Vfrom(A,i). M(x)\} = transrec(i,HVfrom(M,A))$
⟨*proof*⟩

lemma *Vfrom_abs*: $[\![M(A); M(i); M(V); Ord(i)]\!] \Longrightarrow is\_Vfrom(M,A,i,V) \longleftrightarrow V = \{x\in Vfrom(A,i). M(x)\}$
  ⟨*proof*⟩

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

lemma *Vset_abs*: $[\![M(i); M(V); Ord(i)]\!] \Longrightarrow is\_Vset(M,i,V) \longleftrightarrow V = \{x\in Vset(i). M(x)\}$
  ⟨*proof*⟩

lemma *Vset_closed*: $[\![M(i); Ord(i)]\!] \Longrightarrow M(\{x\in Vset(i). M(x)\})$
  ⟨*proof*⟩

lemma *Hrank_trancl*:$Hrank(y, restrict(f,Memrel(eclose(\{x\}))-``\{y\}))$

$= Hrank(y,\ restrict(f,(Memrel(eclose(\{x\}))\hat{}+)\text{-}``\{y\}))$
⟨*proof*⟩

lemma *rank_trancl*: $rank(x) = wfrec(rrank(x),\ x,\ Hrank)$
⟨*proof*⟩

lemma *univ_PHrank* : $[\![\ M(z)\ ;\ M(f)\ ]\!] \Longrightarrow univalent(M,z,PHrank(M,f))$
⟨*proof*⟩

lemma *PHrank_abs* :
$[\![\ M(f)\ ;\ M(y)\ ]\!] \Longrightarrow PHrank(M,f,y,z) \longleftrightarrow M(z) \wedge z = succ(f`y)$
⟨*proof*⟩

lemma *PHrank_closed* : $PHrank(M,f,y,z) \Longrightarrow M(z)$
⟨*proof*⟩

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*⟩

lemma *RepFun_PHrank*:
assumes
$M(R)\ M(A)\ M(f)$
shows
$Replace(A,PHrank(M,f)) = RepFun(A,\lambda y.\ succ(f`y))$
⟨*proof*⟩

lemma *RepFun_PHrank_closed* :
assumes
$M(f)\ M(A)$
shows
$M(Replace(A,PHrank(M,f)))$
⟨*proof*⟩

lemma *relation2_Hrank* :
$relation2(M,is\_Hrank(M),Hrank)$
⟨*proof*⟩

lemma *Union_PHrank_closed*:
assumes
$M(x)\ M(f)$
shows
$M(\bigcup y\in x.\ succ(f`y))$
⟨*proof*⟩

lemma *is_Hrank_closed* :
  $M(A) \Longrightarrow \forall x[M]. \forall g[M]. function(g) \longrightarrow M(Hrank(x,g))$
  ⟨*proof*⟩

lemma *rank_closed*: $M(a) \Longrightarrow M(rank(a))$
  ⟨*proof*⟩

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*⟩

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 :: $[i,i,i,i] \Rightarrow i$ **where**
  $union\_fun(f,g,m,p) \equiv \lambda j \in m \cup p\ .\ if\ j{\in}m\ then\ f`j\ else\ g`j$

lemma *union_fun_type*:
  assumes $f \in m \to n$
    $g \in p \to q$
  shows $union\_fun(f,g,m,p) \in m \cup p \to n \cup q$
⟨*proof*⟩

lemma *union_fun_action* :
  assumes
    $env \in list(M)$
    $env' \in list(M)$
    $length(env) = m \cup p$
    $\forall\ i\ .\ i \in m \longrightarrow\ nth(f`i,env') = nth(i,env)$
    $\forall\ j\ .\ j \in p \longrightarrow nth(g`j,env') = nth(j,env)$
  shows $\forall\ i\ .\ i \in m \cup p \longrightarrow$
      $nth(i,env) = nth(union\_fun(f,g,m,p)`i,env')$
⟨*proof*⟩

lemma *id_fn_type* :
assumes $n \in nat$
shows $id(n) \in n \to n$
$\langle proof \rangle$

lemma *id_fn_action*:
assumes $n \in nat$ $env \in list(M)$
shows $\bigwedge j . j < n \Longrightarrow nth(j,env) = nth(id(n)`j,env)$
$\langle proof \rangle$

definition
rsum :: $[i,i,i,i,i] \Rightarrow i$ **where**
$rsum(f,g,m,n,p) \equiv \lambda j \in m\#+p . if\ j<m\ then\ f`j\ else\ (g`(j\#-m))\#+n$

lemma *sum_inl*:
assumes $m \in nat$ $n \in nat$
$f \in m \to n$ $x \in m$
shows $rsum(f,g,m,n,p)`x = f`x$
$\langle proof \rangle$

lemma *sum_inr*:
assumes $m \in nat$ $n \in nat$ $p \in nat$
$g \in p \to q$ $m \le x$ $x < m\#+p$
shows $rsum(f,g,m,n,p)`x = g`(x\#-m)\#+n$
$\langle proof \rangle$

lemma *sum_action* :
assumes $m \in nat$ $n \in nat$ $p \in nat$ $q \in nat$
$f \in m \to n$ $g \in p \to q$
$env \in list(M)$
$env' \in list(M)$
$env1 \in list(M)$
$env2 \in list(M)$
$length(env) = m$
$length(env1) = p$
$length(env') = n$
$\bigwedge i . i < m \Longrightarrow nth(i,env) = nth(f`i,env')$
$\bigwedge j. j < p \Longrightarrow nth(j,env1) = nth(g`j,env2)$
shows $\forall\ i . i < m\#+p \longrightarrow$
$nth(i,env@env1) = nth(rsum(f,g,m,n,p)`i,env'@env2)$
$\langle proof \rangle$

lemma *sum_type* :
assumes $m \in nat$ $n \in nat$ $p \in nat$ $q \in nat$
$f \in m \to n$ $g \in p \to q$
shows $rsum(f,g,m,n,p) \in (m\#+p) \to (n\#+q)$

⟨proof⟩

lemma *sum_type_id* :
assumes
$f \in length(env) \to length(env')$
$env \in list(M)$
$env' \in list(M)$
$env1 \in list(M)$
shows
$rsum(f,id(length(env1)),length(env),length(env'),length(env1)) \in (length(env)\#+length(env1)) \to (length(env')\#+length(env1))$
⟨proof⟩

lemma *sum_type_id_aux2* :
assumes
$f \in m \to n$
$m \in nat$ $n \in nat$
$env1 \in list(M)$
shows
$rsum(f,id(length(env1)),m,n,length(env1)) \in (m\#+length(env1)) \to (n\#+length(env1))$
⟨proof⟩

lemma *sum_action_id* :
assumes
$env \in list(M)$
$env' \in list(M)$
$f \in length(env) \to length(env')$
$env1 \in list(M)$
$\bigwedge i . i < length(env) \Longrightarrow nth(i,env) = nth(f`i,env')$
shows $\bigwedge i . i < length(env)\#+length(env1) \Longrightarrow nth(i,env@env1) = nth(rsum(f,id(length(env1)),length(env),length(env'),length(env1))`i,env'@env1)$
⟨proof⟩

lemma *sum_action_id_aux* :
assumes
$f \in m \to n$
$env \in list(M)$
$env' \in list(M)$
$env1 \in list(M)$
$length(env) = m$
$length(env') = n$
$length(env1) = p$
$\bigwedge i . i < m \Longrightarrow nth(i,env) = nth(f`i,env')$
shows $\bigwedge i . i < m\#+length(env1) \Longrightarrow nth(i,env@env1) = nth(rsum(f,id(length(env1)),m,n,length(env1))`i,env'@env1)$
⟨proof⟩

definition
  $sum\_id :: [i,i] \Rightarrow i$ **where**
  $sum\_id(m,f) \equiv rsum(\lambda x\in 1.x,f,1,1,m)$

lemma $sum\_id0 : m\in nat \Longrightarrow sum\_id(m,f)`0 = 0$
  $\langle proof \rangle$

lemma $sum\_idS : p\in nat \Longrightarrow q\in nat \Longrightarrow f\in p\to q \Longrightarrow x \in p \Longrightarrow sum\_id(p,f)`(succ(x)) = succ(f`x)$
  $\langle proof \rangle$

lemma $sum\_id\_tc\_aux$:
  $p \in nat \Longrightarrow q \in nat \Longrightarrow f \in p \to q \Longrightarrow sum\_id(p,f) \in 1\#+p \to 1\#+q$
  $\langle proof \rangle$

lemma sum_id_tc :
  $n \in nat \Longrightarrow m \in nat \Longrightarrow f \in n \to m \Longrightarrow sum\_id(n,f) \in succ(n) \to succ(m)$
  $\langle proof \rangle$

15.2 Renaming of formulas

consts $ren :: i\Rightarrow i$
primrec
  $ren(Member(x,y)) =$
    $(\lambda\ n \in nat\ .\ \lambda\ m \in nat.\ \lambda f \in n \to m.\ Member\ (f`x,\ f`y))$

$ren(Equal(x,y)) =$
    $(\lambda\ n \in nat\ .\ \lambda\ m \in nat.\ \lambda f \in n \to m.\ Equal\ (f`x,\ f`y))$

$ren(Nand(p,q)) =$
    $(\lambda\ n \in nat\ .\ \lambda\ m \in nat.\ \lambda f \in n \to m.\ Nand\ (ren(p)`n`m`f,\ ren(q)`n`m`f))$

$ren(Forall(p)) =$
    $(\lambda\ n \in nat\ .\ \lambda\ m \in nat.\ \lambda f \in n \to m.\ Forall\ (ren(p)`succ(n)`succ(m)`sum\_id(n,f)))$

lemma $arity\_meml : l \in nat \Longrightarrow Member(x,y) \in formula \Longrightarrow arity(Member(x,y)) \le l \Longrightarrow x \in l$
  $\langle proof \rangle$
lemma $arity\_memr : l \in nat \Longrightarrow Member(x,y) \in formula \Longrightarrow arity(Member(x,y)) \le l \Longrightarrow y \in l$
  $\langle proof \rangle$
lemma $arity\_eql : l \in nat \Longrightarrow Equal(x,y) \in formula \Longrightarrow arity(Equal(x,y)) \le l \Longrightarrow x \in l$
  $\langle proof \rangle$
lemma $arity\_eqr : l \in nat \Longrightarrow Equal(x,y) \in formula \Longrightarrow arity(Equal(x,y)) \le l \Longrightarrow y \in l$
  $\langle proof \rangle$
lemma $nand\_ar1 : p \in formula \Longrightarrow q\in formula \Longrightarrow arity(p) \le arity(Nand(p,q))$
  $\langle proof \rangle$

lemma *nand_ar2* : $p \in formula \Longrightarrow q{\in}formula \Longrightarrow arity(q) \le arity(Nand(p,q))$
⟨*proof*⟩

lemma *nand_ar1D* : $p \in formula \Longrightarrow q{\in}formula \Longrightarrow arity(Nand(p,q)) \le n \Longrightarrow arity(p) \le n$
⟨*proof*⟩

lemma *nand_ar2D* : $p \in formula \Longrightarrow q{\in}formula \Longrightarrow arity(Nand(p,q)) \le n \Longrightarrow arity(q) \le n$
⟨*proof*⟩

lemma *ren_tc* : $p \in formula \Longrightarrow$
$(\bigwedge n\ m\ f\ .\ n \in nat \Longrightarrow m \in nat \Longrightarrow f \in n{\rightarrow}m \Longrightarrow ren(p)`n`m`f \in formula)$
⟨*proof*⟩

lemma *arity_ren* :
fixes p
assumes $p \in formula$
shows $\bigwedge n\ m\ f\ .\ n \in nat \Longrightarrow m \in nat \Longrightarrow f \in n{\rightarrow}m \Longrightarrow arity(p) \le n \Longrightarrow arity(ren(p)`n`m`f){\le}m$
⟨*proof*⟩

lemma *arity_forallE* : $p \in formula \Longrightarrow m \in nat \Longrightarrow arity(Forall(p)) \le m \Longrightarrow arity(p) \le succ(m)$
⟨*proof*⟩

lemma *env_coincidence_sum_id* :
assumes $m \in nat$ $n \in nat$
$\varrho \in list(A)$ $\varrho' \in list(A)$
$f \in n \rightarrow m$
$\bigwedge i\ .\ i < n \Longrightarrow nth(i,\varrho) = nth(f`i,\varrho')$
$a \in A$ $j \in succ(n)$
shows $nth(j,Cons(a,\varrho)) = nth(sum\_id(n,f)`j,Cons(a,\varrho'))$
⟨*proof*⟩

lemma *sats_iff_sats_ren* :
assumes $\varphi \in formula$
shows $[\![\ n \in nat\ ;\ m \in nat\ ;\ \varrho \in list(M)\ ;\ \varrho' \in list(M)\ ;\ f \in n \rightarrow m\ ;$
$arity(\varphi) \le n\ ;$
$\bigwedge i\ .\ i < n \Longrightarrow nth(i,\varrho) = nth(f`i,\varrho')\ ]\!] \Longrightarrow$
$sats(M,\varphi,\varrho) \longleftrightarrow sats(M,ren(\varphi)`n`m`f,\varrho')$
⟨*proof*⟩

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*⟩
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*))
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *Transset_intf* :
Transset(*M*) ⟹ *y*∈*x* ⟹ *x* ∈ *M* ⟹ *y* ∈ *M*
⟨*proof*⟩

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: φ ∈ *formula* ⟹ *env* ∈ *list*(*M*) ⟹
arity(φ) ≤ *1* #+ *length*(*env*) ⟹
separation(##*M*,λ*x*. *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*⟩

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
⟨*proof*⟩

sublocale *M_ZF_trans* $\subseteq$ *M_trivial* ##M
⟨*proof*⟩

16.2 Interface with *M_basic*

definition *Intersection* **where**
Intersection(N,B,x) $\equiv$ ($\forall$ y[N]. $y \in B \longrightarrow x \in y$)

⟨*ML*⟩
⟨*proof*⟩
⟨*ML*⟩

context *M_ZF_trans*
begin

lemma *inter_sep_intf* :
assumes
$A \in M$
shows
separation(##M,λx . $\forall$ $y \in M$. $y \in A \longrightarrow x \in y$)
⟨*proof*⟩

schematic_goal *diff_fm_auto*:
  assumes
    $nth(i,env) = x$ $nth(j,env) = B$
    $i \in nat$ $j \in nat$ $env \in list(A)$
  shows
    $x \notin B \longleftrightarrow sats(A,?dfm(i,j),env)$
  ⟨*proof*⟩

lemma *diff_sep_intf* :
  assumes
    $B \in M$
  shows
    $separation(\#\#M,\lambda x . x \notin B)$
⟨*proof*⟩

schematic_goal *cprod_fm_auto*:
  assumes
    $nth(i,env) = z$ $nth(j,env) = B$ $nth(h,env) = C$
    $i \in nat$ $j \in nat$ $h \in nat$ $env \in list(A)$
  shows
    $(\exists x \in A. x \in B \wedge (\exists y \in A. y \in C \wedge pair(\#\#A,x,y,z))) \longleftrightarrow sats(A,?cpfm(i,j,h),env)$
  ⟨*proof*⟩

lemma *cartprod_sep_intf* :
  assumes
    $A \in M$
    and
    $B \in M$
  shows
    $separation(\#\#M,\lambda z. \exists x \in M. x \in A \wedge (\exists y \in M. y \in B \wedge pair(\#\#M,x,y,z)))$
⟨*proof*⟩

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
    $(\exists p \in A. p \in r \ \& \ (\exists x \in A. x \in B \ \& \ pair(\#\#A,x,y,p))) \longleftrightarrow sats(A,?imfm(i,j,h),env)$
  ⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

lemma *restrict_sep_intf* :
assumes
A∈*M*
shows
separation(##*M*,λ*z*. ∃ *x*∈*M*. *x*∈*A* & (∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*z*)))
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *comp_sep_intf* :
assumes
R∈*M*
and
S∈*M*
shows
separation(##*M*,λ*xz*. ∃ *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*⟩

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*)
⟨*proof*⟩

lemma *pred_sep_intf*:
assumes
R∈*M*
and
X∈*M*
shows
separation(##*M*, λ*y*. ∃ *p*∈*M*. *p*∈*R* & *pair*(##*M*,*y*,*X*,*p*))
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *memrel_sep_intf*:
separation(##*M*, λ*z*. ∃ *x*∈*M*. ∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*z*) & *x* ∈ *y*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *rtrancl_separation_intf*:
assumes
r∈*M*
and
A∈*M*
shows
separation (*##M*, *rtran_closure_mem*(*##M*,*A*,*r*))
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

⟨*ML*⟩

lemma *arity_tran_closure_fm* :
⟦*x*∈*nat*;*f*∈*nat*⟧ ⟹ *arity*(*trans_closure_fm*(*x*,*f*)) = *succ*(*x*) ∪ *succ*(*f*)
⟨*proof*⟩

schematic_goal *wellfounded_trancl_fm_auto*:
assumes

$nth(i,env) = p$ $nth(j,env) = r$ $nth(k,env) = B$
$i \in nat$ $j \in nat$ $k \in nat$ $env \in list(A)$
shows
wellfounded_trancl$(\#\#A,B,r,p) \longleftrightarrow sats(A,?wtf(i,j,k),env)$
⟨*proof*⟩

context *M_ZF_trans*
begin

lemma *wftrancl_separation_intf*:
assumes
$r \in M$ **and** $Z \in M$
shows
separation $(\#\#M,$ *wellfounded_trancl*$(\#\#M,Z,r))$
⟨*proof*⟩

Proof that $nat \in M$

lemma *finite_sep_intf*: $separation(\#\#M, \lambda x.\ x \in nat)$
⟨*proof*⟩

lemma *nat_subset_I'*:
$[\![I \in M ; 0 \in I ; \bigwedge x.\ x \in I \Longrightarrow succ(x) \in I]\!] \Longrightarrow nat \subseteq I$
⟨*proof*⟩

lemma *nat_subset_I*: $\exists I \in M.\ nat \subseteq I$
⟨*proof*⟩

lemma *nat_in_M*: $nat \in M$
⟨*proof*⟩

end

sublocale *M_ZF_trans* $\subseteq$ *M_trancl* $\#\#M$
⟨*proof*⟩

16.4 Interface with *M_eclose*

lemma *repl_sats*:
assumes
sat:$\bigwedge x\ z.\ x \in M \Longrightarrow z \in M \Longrightarrow sats(M,\varphi,Cons(x,Cons(z,env))) \longleftrightarrow P(x,z)$
shows
strong_replacement$(\#\#M,\lambda x\ z.\ sats(M,\varphi,Cons(x,Cons(z,env)))) \longleftrightarrow$
strong_replacement$(\#\#M,P)$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *list_repl1_intf*:
assumes
$A \in M$
shows

iterates_replacement(##*M*, *is_list_functor*(##*M*,*A*), *0*)
⟨*proof*⟩

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*⟩

lemma (**in** *M_ZF_trans*) *formula_repl1_intf* :
iterates_replacement(##*M*, *is_formula_functor*(##*M*), *0*)
⟨*proof*⟩

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*⟩

lemma (**in** *M_ZF_trans*) *eclose_repl1_intf*:
assumes
A∈*M*
shows
iterates_replacement(##*M*, *big_union*(##*M*), *A*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *list_repl2_intf*:
assumes
A∈*M*
shows
strong_replacement(##*M*,λ*n y*. *n*∈*nat* & *is_iterates*(##*M*, *is_list_functor*(##*M*,*A*), *0*, *n*, *y*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *formula_repl2_intf*:
strong_replacement(##*M*,λ*n y*. *n*∈*nat* & *is_iterates*(##*M*, *is_formula_functor*(##*M*), *0*, *n*, *y*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *eclose_repl2_intf*:
assumes
A∈*M*
shows
strong_replacement(##*M*,λ*n y*. *n*∈*nat* & *is_iterates*(##*M*, *big_union*(##*M*), *A*, *n*, *y*))
⟨*proof*⟩

sublocale *M_ZF_trans* ⊆ *M_datatypes* ##*M*
⟨*proof*⟩

sublocale *M_ZF_trans* ⊆ *M_eclose* ##*M*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *powapply_repl* :
assumes
f∈*M*
shows
strong_replacement(##*M*,*is_powapply*(##*M*,*f*))
⟨*proof*⟩

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*
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *phrank_repl* :
assumes
f∈*M*
shows
strong_replacement(##*M*,*PHrank*(##*M*,*f*))
⟨*proof*⟩

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*
⟨*proof*⟩

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*))
⟨*proof*⟩

declare *is_Hrank_fm_def*[*fm_definitions add*]
declare *PHrank_fm_def*[*fm_definitions add*]

lemma (**in** *M_ZF_trans*) *wfrec_rank* :
assumes
X∈*M*
shows
wfrec_replacement(##*M*,*is_Hrank*(##*M*),*rrank*(*X*))
⟨*proof*⟩

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*
⟨*proof*⟩

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*))
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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)
  ⟨*proof*⟩

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)
  ⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *trans_repl_HVFrom* :
  assumes
    A∈M i∈M
  shows
    transrec_replacement(##M,is_HVfrom(##M,A),i)
⟨*proof*⟩

sublocale *M_ZF_trans ⊆ M_eclose_pow ##M*
  ⟨*proof*⟩

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*⟩
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*⟩

lemma *Replace_in_M* :
  assumes

f_fm: $\varphi \in$ *formula* **and**
f_ar: *arity*(φ)$\leq$ *2 #+ length*(*env*) **and**
fsats: $\bigwedge$*x y*. *x*∈*A* ⟹ *y*∈*M* ⟹ (*M*,[*x*,*y*]@*env* ⊨ φ) ⟷ *y* = *f*(*x*) **and**
fclosed: $\bigwedge$*x*. *x*∈*A* ⟹ *f*(*x*) ∈ *M* **and**
A∈*M env*∈*list*(*M*)
shows {*f*(*x*) . *x*∈*A*}∈*M*
⟨*proof*⟩

lemma *Replace_relativized_in_M* :
assumes
f_fm: $\varphi \in$ *formula* **and**
f_ar: *arity*(φ)$\leq$ *2 #+ length*(*env*) **and**
fsats: $\bigwedge$*x y*. *x*∈*A* ⟹ *y*∈*M* ⟹ (*M*,[*x*,*y*]@*env* ⊨ φ) ⟷ *is_f*(*x*,*y*) **and**
fabs: $\bigwedge$*x y*. *x*∈*A* ⟹ *y*∈*M* ⟹ *is_f*(*x*,*y*) ⟷ *y* = *f*(*x*) **and**
fclosed: $\bigwedge$*x*. *x*∈*A* ⟹ *f*(*x*) ∈ *M* **and**
A∈*M env*∈*list*(*M*)
shows {*f*(*x*) . *x*∈*A*}∈*M*
⟨*proof*⟩

definition *ϱ_repl* :: *i*⇒*i* **where**
ϱ_repl(*l*) ≡ *rsum*({⟨*0*, *1*⟩, ⟨*1*, *0*⟩}, *id*(*l*), *2*, *3*, *l*)

lemma *f_type* : {⟨*0*, *1*⟩, ⟨*1*, *0*⟩} ∈ *2* → *3*
⟨*proof*⟩

lemma *ren_type* :
assumes *l*∈*nat*
shows *ϱ_repl*(*l*) : *2#+l* → *3#+l*
⟨*proof*⟩

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*⟩

lemma *Lambda_in_M* :
assumes
f_fm: $\varphi \in$ *formula* **and**
f_ar: *arity*(φ)$\leq$ *2 #+ length*(*env*) **and**
fsats: $\bigwedge$*x y*. *x*∈*A* ⟹ *y*∈*M* ⟹ (*M*,[*x*,*y*]@*env* ⊨ φ) ⟷ *is_f*(*x*,*y*) **and**
fabs: $\bigwedge$*x y*. *x*∈*A* ⟹ *y*∈*M* ⟹ *is_f*(*x*,*y*) ⟷ *y* = *f*(*x*) **and**
fclosed: $\bigwedge$*x*. *x*∈*A* ⟹ *f*(*x*) ∈ *M* **and**
A∈*M env*∈*list*(*M*)
shows (λ*x*∈*A* . *f*(*x*)) ∈*M*
⟨*proof*⟩

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
⟨*proof*⟩

lemma *ren_type'* :
assumes *l*∈*nat*
shows ϱ_pair_repl(l) : 3#+l → 4#+l
⟨*proof*⟩

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*⟩

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*⟩

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*: $P \in M$
    and *leq_in_M*: $leq \in M$

begin

lemma *P_sub_M* : $P \subseteq M$
  ⟨*proof*⟩

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*]: $M\_generic(G) \Longrightarrow x \in G \Longrightarrow x \in P$
  ⟨*proof*⟩

lemma *M_generic_leqD* [*dest*]: $M\_generic(G) \Longrightarrow p \in G \Longrightarrow q \in P \Longrightarrow p \preceq q \Longrightarrow q \in G$
  ⟨*proof*⟩

lemma *M_generic_compatD* [*dest*]: $M\_generic(G) \Longrightarrow p \in G \Longrightarrow r \in G \Longrightarrow \exists q \in G.\ q \preceq p \wedge q \preceq r$
  ⟨*proof*⟩

lemma *M_generic_denseD* [*dest*]: $M\_generic(G) \Longrightarrow dense(D) \Longrightarrow D \subseteq P \Longrightarrow D \in M \Longrightarrow \exists q \in G.\ q \in D$
  ⟨*proof*⟩

lemma *G_nonempty*: $M\_generic(G) \Longrightarrow G \neq 0$
⟨*proof*⟩

lemma *one_in_G* :
  assumes $M\_generic(G)$
  shows $one \in G$
⟨*proof*⟩

lemma *G_subset_M*: $M\_generic(G) \Longrightarrow G \subseteq M$
  ⟨*proof*⟩

declare *iff_trans* [*trans*]

lemma *generic_filter_existence*:
  p∈*P* ⟹ ∃ *G*. *p*∈*G* ∧ *M_generic*(*G*)
⟨*proof*⟩

lemma *one_in_M*: *one* ∈ *M*
  ⟨*proof*⟩

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*)
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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*))
  ⟨*proof*⟩

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* : $x \in eclose(a) \Longrightarrow x \in eclose(\{a\})$
⟨*proof*⟩

lemma *ecloseE* :
assumes $x \in eclose(A)$
shows $x \in A \vee (\exists\ B \in A\ .\ x \in eclose(B))$
⟨*proof*⟩

lemma *eclose_singE* : $x \in eclose(\{a\}) \Longrightarrow x = a \vee x \in eclose(a)$
⟨*proof*⟩

lemma *in_eclose_sing* :
assumes $x \in eclose(\{a\})$ $a \in eclose(z)$
shows $x \in eclose(\{z\})$
⟨*proof*⟩

lemma *in_dom_in_eclose* :
assumes $x \in domain(z)$
shows $x \in eclose(z)$
⟨*proof*⟩

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*!]: $t \in domain(x) \Longrightarrow ed(t,x)$
⟨*proof*⟩

lemma *edD*[*dest*!]: $ed(t,x) \Longrightarrow t \in domain(x)$
⟨*proof*⟩

lemma *rank_ed*:
assumes $ed(y,x)$
shows $succ(rank(y)) \le rank(x)$

⟨*proof*⟩

lemma *edrel_dest* [*dest*]: $x \in edrel(A) \Longrightarrow \exists\ a\in A.\ \exists\ b \in A.\ x = \langle a,b\rangle$
⟨*proof*⟩

lemma *edrelD* : $x \in edrel(A) \Longrightarrow \exists\ a\in A.\ \exists\ b \in A.\ x = \langle a,b\rangle \wedge a \in domain(b)$
⟨*proof*⟩

lemma *edrelI* [*intro*!]: $x\in A \Longrightarrow y\in A \Longrightarrow x \in domain(y) \Longrightarrow \langle x,y\rangle\in edrel(A)$
⟨*proof*⟩

lemma *edrel_trans*: $Transset(A) \Longrightarrow y\in A \Longrightarrow x \in domain(y) \Longrightarrow \langle x,y\rangle\in edrel(A)$
⟨*proof*⟩

lemma *domain_trans*: $Transset(A) \Longrightarrow y\in A \Longrightarrow x \in domain(y) \Longrightarrow x\in A$
⟨*proof*⟩

lemma *relation_edrel* : $relation(edrel(A))$
⟨*proof*⟩

lemma *field_edrel* : $field(edrel(A))\subseteq A$
⟨*proof*⟩

lemma *edrel_sub_memrel*: $edrel(A) \subseteq trancl(Memrel(eclose(A)))$
⟨*proof*⟩

lemma *wf_edrel* : $wf(edrel(A))$
⟨*proof*⟩

lemma *ed_induction*:
assumes $\bigwedge x.\ [\![\bigwedge y.\ \ ed(y,x) \Longrightarrow Q(y)\ ]\!] \Longrightarrow Q(x)$
shows $Q(a)$
⟨*proof*⟩

lemma *dom_under_edrel_eclose*: $edrel(eclose(\{x\}))$ -`` $\{x\} = domain(x)$
⟨*proof*⟩

lemma *ed_eclose* : $\langle y,z\rangle \in edrel(A) \Longrightarrow y \in eclose(z)$
⟨*proof*⟩

lemma *tr_edrel_eclose* : $\langle y,z\rangle \in edrel(eclose(\{x\}))$^+ $\Longrightarrow y \in eclose(z)$
⟨*proof*⟩

lemma *restrict_edrel_eq* :
assumes $z \in domain(x)$
shows $edrel(eclose(\{x\})) \cap eclose(\{z\})\times eclose(\{z\}) = edrel(eclose(\{z\}))$
⟨*proof*⟩

lemma *tr_edrel_subset* :
assumes $z \in domain(x)$
shows $tr\_down(edrel(eclose(\{x\})),z) \subseteq eclose(\{z\})$
⟨*proof*⟩

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 :: $[i,i,i] \Rightarrow i$ (_−[_] [71,1])
where $M^P[G] \equiv \{val(P,G,\tau). \tau \in M\}$

abbreviation (**in** *forcing_notion*)
GenExt_at_P :: $i \Rightarrow i \Rightarrow i$ (_[_] [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*:
$check(x) = wfrec(Memrel(eclose(\{x\})), x, Hcheck)$
⟨*proof*⟩

definition
rcheck :: $i \Rightarrow i$ **where**
$rcheck(x) \equiv Memrel(eclose(\{x\}))\hat{}+$

lemma *Hcheck_trancl*:$Hcheck(y, restrict(f,Memrel(eclose(\{x\}))\text{-}``\{y\}))$
$= Hcheck(y, restrict(f,(Memrel(eclose(\{x\}))\hat{}+)\text{-}``\{y\}))$
⟨*proof*⟩

lemma *check_trancl*: $check(x) = wfrec(rcheck(x), x, Hcheck)$

⟨*proof*⟩

lemma *rcheck_in_M* :
$x \in M \Longrightarrow rcheck(x) \in M$
⟨*proof*⟩

lemma *aux_def_check*: $x \in y \Longrightarrow$
$wfrec(Memrel(eclose(\{y\})), x, Hcheck) =$
$wfrec(Memrel(eclose(\{x\})), x, Hcheck)$
⟨*proof*⟩

lemma *def_check* : $check(y) = \{ \langle check(w),one\rangle . w \in y\}$
⟨*proof*⟩

lemma *def_checkS* :
fixes n
assumes $n \in nat$
shows $check(succ(n)) = check(n) \cup \{\langle check(n),one\rangle\}$
⟨*proof*⟩

lemma *field_Memrel2* :
assumes $x \in M$
shows $field(Memrel(eclose(\{x\}))) \subseteq M$
⟨*proof*⟩

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*⟩

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*⟩

lemma *val_mono* : $x\subseteq y \Longrightarrow val(P,G,x) \subseteq val(P,G,y)$
⟨*proof*⟩

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*⟩

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*⟩

lemma *val_of_name_alt* :
  $val(P,G,\{x \in A \times P.\ Q(x)\}) = \{val(P,G,t) \ ..\ t \in A\ ,\ \exists p \in P \cap G\ .\ \ Q(\langle t,p \rangle)\ \}$
  ⟨*proof*⟩

lemma *val_only_names*: $val(P,F,\tau) = val(P,F,\{x \in \tau.\ \exists t \in domain(\tau).\ \exists p \in P.\ x=\langle t,p \rangle\})$
  (**is** _ $= val(P,F,?name)$)
⟨*proof*⟩

lemma *val_only_pairs*: $val(P,F,\tau) = val(P,F,\{x \in \tau.\ \exists t\ p.\ x=\langle t,p \rangle\})$
⟨*proof*⟩

lemma *val_subset_domain_times_range*: $val(P,F,\tau) \subseteq val(P,F,domain(\tau) \times range(\tau))$
  ⟨*proof*⟩

lemma *val_subset_domain_times_P*: $val(P,F,\tau) \subseteq val(P,F,domain(\tau) \times P)$
  ⟨*proof*⟩

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*⟩

lemma *elem_of_val*: $x \in val(P,G,\pi) \Longrightarrow \exists \vartheta \in domain(\pi).\ val(P,G,\vartheta) = x$
  ⟨*proof*⟩

lemma *elem_of_val_pair*: $x \in val(P,G,\pi) \Longrightarrow \exists \vartheta.\ \exists p \in G.\ \ \langle \vartheta,p \rangle \in \pi \wedge val(P,G,\vartheta)$
$= x$
  ⟨*proof*⟩

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*⟩

lemma *GenExtD*:
  $x \in M[G] \Longrightarrow \exists \tau \in M.\ x = val(P,G,\tau)$
  ⟨*proof*⟩

lemma *GenExtI*:
  $x \in M \Longrightarrow val(P,G,x) \in M[G]$
  ⟨*proof*⟩

lemma *Transset_MG* : $Transset(M[G])$
⟨*proof*⟩

lemmas *transitivity_MG* = *Transset_intf*[*OF Transset_MG*]

lemma *check_n_M* :
  fixes n

assumes $n \in nat$
shows $check(n) \in M$
⟨*proof*⟩

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*⟩

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*]:
$[\![\ x \in nat;\ y \in nat;\ z \in nat;\ u \in nat\ ]\!] \Longrightarrow PHcheck\_fm(x,y,z,u) \in formula$
⟨*proof*⟩

lemma *sats_PHcheck_fm* [*simp*]:
$[\![\ x \in nat;\ y \in nat;\ z \in nat;\ u \in nat\ ;\ env \in list(M)]\!]$
$\Longrightarrow sats(M,PHcheck\_fm(x,y,z,u),env) \longleftrightarrow$
$PHcheck(nth(x,env),nth(y,env),nth(z,env),nth(u,env))$
⟨*proof*⟩

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*]:
$[\![\ x \in nat;\ y \in nat;\ z \in nat;\ u \in nat\ ]\!] \Longrightarrow is\_Hcheck\_fm(x,y,z,u) \in formula$
⟨*proof*⟩

lemma *sats_is_Hcheck_fm* [*simp*]:
⟦ $x \in nat$; $y \in nat$; $z \in nat$; $u \in nat$; $env \in list(M)$⟧
⟹ $sats(M,is\_Hcheck\_fm(x,y,z,u),env) \longleftrightarrow$
$is\_Hcheck(nth(x,env),nth(y,env),nth(z,env),nth(u,env))$
⟨*proof*⟩

lemma *wfrec_Hcheck* :
assumes
$X \in M$
shows
$wfrec\_replacement(\#\#M,is\_Hcheck(one),rcheck(X))$
⟨*proof*⟩

lemma *repl_PHcheck* :
assumes
$f \in M$
shows
$strong\_replacement(\#\#M,PHcheck(one,f))$
⟨*proof*⟩

lemma *univ_PHcheck* : ⟦ $z \in M$; $f \in M$ ⟧ ⟹ $univalent(\#\#M,z,PHcheck(one,f))$
⟨*proof*⟩

lemma *relation2_Hcheck* :
$relation2(\#\#M,is\_Hcheck(one),Hcheck)$
⟨*proof*⟩

lemma *PHcheck_closed* :
⟦$z \in M$; $f \in M$; $x \in z$; $PHcheck(one,f,x,y)$ ⟧ ⟹ $(\#\#M)(y)$
⟨*proof*⟩

lemma *Hcheck_closed* :
$\forall y \in M.\ \forall g \in M.\ function(g) \longrightarrow Hcheck(y,g) \in M$
⟨*proof*⟩

lemma *wf_rcheck* : $x \in M \Longrightarrow wf(rcheck(x))$
⟨*proof*⟩

lemma *trans_rcheck* : $x \in M \Longrightarrow trans(rcheck(x))$
⟨*proof*⟩

lemma *relation_rcheck* : $x \in M \Longrightarrow relation(rcheck(x))$
⟨*proof*⟩

lemma *check_in_M* : $x \in M \Longrightarrow check(x) \in M$
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

⟨*ML*⟩

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*⟩

definition
check_fm :: [*i*,*i*,*i*] ⇒ *i* **where**
[*fm_definitions*] :
check_fm(*x*,*o*,*z*) ≡ *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* (⋅_^{v}_ *is* _⋅)

lemma *check_fm_type*[*TC*] :
⟦*x*∈*nat*;*o*∈*nat*;*z*∈*nat*⟧ ⟹ *check_fm*(*x*,*o*,*z*)∈*formula*
⟨*proof*⟩

⟨*ML*⟩

lemma *arity_is_Hcheck_fm* :
  assumes $m\in nat$ $n\in nat$ $p\in nat$ $o\in nat$
  shows $arity(is\_Hcheck\_fm(m,n,p,o)) = succ(o) \cup succ(n) \cup succ(p) \cup succ(m)$
  ⟨*proof*⟩

lemma *arity_check_fm* :
  assumes $m\in nat$ $n\in nat$ $o\in nat$
  shows $arity(check\_fm(m,n,o)) = succ(o) \cup succ(n) \cup succ(m)$
  ⟨*proof*⟩

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*⟩

lemma *check_replacement*:
  $\{check(x). x\in P\} \in M$
⟨*proof*⟩

lemma *pair_check* : $[\![p\in M ; y\in M]\!] \Longrightarrow (\exists c\in M. is\_check(p,c) \wedge pair(\#\#M,c,p,y)) \longleftrightarrow y = \langle check(p),p\rangle$
  ⟨*proof*⟩

lemma *M_subset_MG* : $one \in G \Longrightarrow M \subseteq M[G]$
  ⟨*proof*⟩

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*⟩

lemma *val_G_dot* :
  assumes $G \subseteq P$
    $one \in G$
  shows $val(P,G,G\_dot) = G$
⟨*proof*⟩

lemma *G_in_Gen_Ext* :
  assumes $G \subseteq P$ **and** *one* $\in G$
  shows $G \in M[G]$
  ⟨*proof*⟩

end

locale *G_generic* = *forcing_data* +
  fixes $G :: i$
  assumes *generic* : *M_generic*(G)
begin

lemma *zero_in_MG* :
  $0 \in M[G]$
⟨*proof*⟩

lemma *G_nonempty*: $G{\neq}0$
⟨*proof*⟩

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*⇒*i* **where**
  name2(x) ≡ *fst*(*snd*(*snd*(x)))

definition
  cond_of :: $i \Rightarrow i$ **where**
  $cond\_of(x) \equiv snd(snd(snd((x))))$

lemma *components_simp*:
  $ftype(\langle f,n1,n2,c\rangle) = f$
  $name1(\langle f,n1,n2,c\rangle) = n1$
  $name2(\langle f,n1,n2,c\rangle) = n2$
  $cond\_of(\langle f,n1,n2,c\rangle) = c$
  ⟨*proof*⟩

definition *eclose_n* :: $[i \Rightarrow i,i] \Rightarrow i$ **where**
  $eclose\_n(name,x) = eclose(\{name(x)\})$

definition
  ecloseN :: $i \Rightarrow i$ **where**
  $ecloseN(x) = eclose\_n(name1,x) \cup eclose\_n(name2,x)$

lemma *components_in_eclose* :
  $n1 \in ecloseN(\langle f,n1,n2,c\rangle)$
  $n2 \in ecloseN(\langle f,n1,n2,c\rangle)$
  ⟨*proof*⟩

lemmas *names_simp* = *components_simp*(*2*) *components_simp*(*3*)

lemma *ecloseNI1* :
  assumes $x \in eclose(n1) \vee x \in eclose(n2)$
  shows $x \in ecloseN(\langle f,n1,n2,c\rangle)$
  ⟨*proof*⟩

lemmas *ecloseNI* = *ecloseNI1*

lemma *ecloseN_mono* :
  assumes $u \in ecloseN(x)$ $name1(x) \in ecloseN(y)$ $name2(x) \in ecloseN(y)$
  shows $u \in ecloseN(y)$
⟨*proof*⟩

definition
  is_ftype :: $(i \Rightarrow o) \Rightarrow i \Rightarrow i \Rightarrow o$ **where**
  $is\_ftype \equiv is\_fst$

definition
  ftype_fm :: $[i,i] \Rightarrow i$ **where**
  $ftype\_fm \equiv fst\_fm$

lemma *is_ftype_iff_sats* [*iff_sats*]:
  assumes
    $nth(a,env) = aa$ $nth(b,env) = bb$ $a \in nat$ $b \in nat$ $env \in list(A)$

shows
is_ftype(##*A*,*aa*,*bb*) ⟷ *sats*(*A*,*ftype_fm*(*a*,*b*), *env*)
⟨*proof*⟩

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*))
⟨*proof*⟩

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*)
⟨*proof*⟩

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*))
⟨*proof*⟩

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*))
⟨*proof*⟩

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*)
⟨*proof*⟩

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*))
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*))
⟨*proof*⟩

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*)
⟨*proof*⟩

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 ($ftype(x) = 0 \wedge ftype(y) = 1$) $\vee$ ($ftype(x) = 1 \wedge ftype(y) = 0$)
⟨*proof*⟩

lemma *frecRI1*: $s \in domain(n1) \vee s \in domain(n2) \Longrightarrow frecR(\langle 1, s, n1, q\rangle, \langle 0, n1, n2, q'\rangle)$
⟨*proof*⟩

lemma *frecRI1'*: $s \in domain(n1) \cup domain(n2) \Longrightarrow frecR(\langle 1, s, n1, q\rangle, \langle 0, n1, n2, q'\rangle)$
⟨*proof*⟩

lemma *frecRI2*: $s \in domain(n1) \vee s \in domain(n2) \Longrightarrow frecR(\langle 1, s, n2, q\rangle, \langle 0, n1, n2, q'\rangle)$
⟨*proof*⟩

lemma *frecRI2'*: $s \in domain(n1) \cup domain(n2) \Longrightarrow frecR(\langle 1, s, n2, q\rangle, \langle 0, n1, n2, q'\rangle)$
⟨*proof*⟩

lemma *frecRI3*: $\langle s, r\rangle \in n2 \Longrightarrow frecR(\langle 0, n1, s, q\rangle, \langle 1, n1, n2, q'\rangle)$
⟨*proof*⟩

lemma *frecRI3'*: $s \in domain(n2) \Longrightarrow frecR(\langle 0, n1, s, q\rangle, \langle 1, n1, n2, q'\rangle)$
⟨*proof*⟩

lemma *frecR_iff* :
$frecR(x,y) \longleftrightarrow$
$(ftype(x) = 1 \wedge ftype(y) = 0$
$\wedge (name1(x) \in domain(name1(y)) \cup domain(name2(y)) \wedge (name2(x) = name1(y) \vee name2(x) = name2(y))))$
$\vee (ftype(x) = 0 \wedge ftype(y) = 1 \wedge name1(x) = name1(y) \wedge name2(x) \in domain(name2(y)))$
⟨*proof*⟩

lemma *frecR_D1* :
$frecR(x,y) \Longrightarrow ftype(y) = 0 \Longrightarrow ftype(x) = 1 \wedge$
$(name1(x) \in domain(name1(y)) \cup domain(name2(y)) \wedge (name2(x) = name1(y) \vee name2(x) = name2(y)))$
⟨*proof*⟩

lemma *frecR_D2* :
$frecR(x,y) \Longrightarrow ftype(y) = 1 \Longrightarrow ftype(x) = 0 \wedge$
$ftype(x) = 0 \wedge ftype(y) = 1 \wedge name1(x) = name1(y) \wedge name2(x) \in domain(name2(y))$
⟨*proof*⟩

lemma *frecR_DI* :

assumes *frecR*(⟨*a*,*b*,*c*,*d*⟩,⟨*ftype*(*y*),*name1*(*y*),*name2*(*y*),*cond_of*(*y*)⟩)
shows *frecR*(⟨*a*,*b*,*c*,*d*⟩,*y*)
⟨*proof*⟩

⟨*ML*⟩

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*)
⟨*proof*⟩

⟨*ML*⟩

lemma *eq_ftypep_not_frecrR*:
assumes *ftype*(*x*) = *ftype*(*y*)
shows ¬ *frecR*(*x*,*y*)
⟨*proof*⟩

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*))
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *frecR_le_rnk_names* :
assumes *frecR*(*x*,*y*)
shows *rank_names*(*x*)≤*rank_names*(*y*)
⟨*proof*⟩

definition

Γ :: $i \Rightarrow i$ **where**
$\Gamma(x) = \mathit{3}$ ** $\mathit{rank\_names}(x)$ ++ $\mathit{type\_form}(x)$

lemma Γ_*type* [*TC*]:
shows $\mathit{Ord}(\Gamma(x))$
⟨*proof*⟩

lemma Γ_*mono* :
assumes $\mathit{frecR}(x,y)$
shows $\Gamma(x) < \Gamma(y)$
⟨*proof*⟩

definition
frecrel :: $i \Rightarrow i$ **where**
$\mathit{frecrel}(A) \equiv \mathit{Rrel}(\mathit{frecR},A)$

lemma *frecrelI* :
assumes $x \in A$ $y \in A$ $\mathit{frecR}(x,y)$
shows $\langle x,y\rangle \in \mathit{frecrel}(A)$
⟨*proof*⟩

lemma *frecrelD* :
assumes $\langle x,y\rangle \in \mathit{frecrel}(A1\times A2\times A3\times A4)$
shows $\mathit{ftype}(x) \in A1$ $\mathit{ftype}(x) \in A1$
$\mathit{name1}(x) \in A2$ $\mathit{name1}(y) \in A2$ $\mathit{name2}(x) \in A3$ $\mathit{name2}(x) \in A3$
$\mathit{cond\_of}(x) \in A4$ $\mathit{cond\_of}(y) \in A4$
$\mathit{frecR}(x,y)$
⟨*proof*⟩

lemma *wf_frecrel* :
shows $\mathit{wf}(\mathit{frecrel}(A))$
⟨*proof*⟩

lemma *core_induction_aux*:
fixes *A1 A2* :: *i*
assumes
$\mathit{Transset}(A1)$
$\bigwedge\tau\ \vartheta\ p.\ \ p \in A2 \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![\ q\in A2\ ;\ \sigma\in \mathit{domain}(\vartheta)]\!] \Longrightarrow Q(0,\tau,\sigma,q)]\!] \Longrightarrow Q(1,\tau,\vartheta,p)$
$\bigwedge\tau\ \vartheta\ p.\ \ p \in A2 \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![\ q\in A2\ ;\ \sigma\in \mathit{domain}(\tau) \cup \mathit{domain}(\vartheta)]\!] \Longrightarrow Q(1,\sigma,\tau,q) \wedge Q(1,\sigma,\vartheta,q)]\!] \Longrightarrow Q(0,\tau,\vartheta,p)$
shows $a\in 2\times A1\times A1\times A2 \Longrightarrow Q(\mathit{ftype}(a),\mathit{name1}(a),\mathit{name2}(a),\mathit{cond\_of}(a))$
⟨*proof*⟩

lemma *def_frecrel* : $\mathit{frecrel}(A) = \{z\in A\times A.\ \exists x\ y.\ z = \langle x,\ y\rangle \wedge \mathit{frecR}(x,y)\}$
⟨*proof*⟩

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*))))}
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *arity_snd_fm* [*arity*] :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*snd_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
⟨*proof*⟩

lemma *arity_snd_snd_fm* [*arity*] :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*snd_snd_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
⟨*proof*⟩

lemma *arity_ftype_fm* [*arity*] :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*ftype_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
⟨*proof*⟩

lemma *arity_name1_fm* [*arity*] :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*name1_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
⟨*proof*⟩

lemma *arity_name2_fm* [*arity*] :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*name2_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
⟨*proof*⟩

lemma *arity_cond_of_fm* [*arity*] :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*cond_of_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
⟨*proof*⟩

lemma *arity_eclose_n1_fm* [*arity*] :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*eclose_n1_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
⟨*proof*⟩

lemma *arity_eclose_n2_fm* [*arity*] :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*eclose_n2_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
⟨*proof*⟩

```
lemma arity_ecloseN_fm [arity] :
  ⟦x∈nat ; t∈nat⟧ ⟹ arity(ecloseN_fm(x,t)) = succ(x) ∪ succ(t)
  ⟨proof⟩

lemma arity_frecR_fm [arity]:
  ⟦a∈nat;b∈nat⟧ ⟹ arity(frecR_fm(a,b)) = succ(a) ∪ succ(b)
  ⟨proof⟩
end
theory Discipline_Cardinal
  imports
    Discipline_Base
    Discipline_Function
    Least
    FrecR
    Arities
    FrecR_Arities
begin

declare [[syntax_ambiguity_warning = false]]

⟨ML⟩

notation is_cardinal_fm (‹cardinal′(_′) is _›)

abbreviation
  cardinal_r :: [i,i⇒o] ⇒ i (‹|_|-›) where
  |x|^M ≡ cardinal_rel(M,x)

abbreviation
  cardinal_r_set :: [i,i]⇒i  (‹|_|-›) where
  |x|^M ≡ cardinal_rel(##M,x)

context M_trivial begin
⟨ML⟩
  ⟨proof⟩
end

⟨ML⟩
  ⟨proof⟩

⟨ML⟩

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)
  ⟨proof⟩

⟨ML⟩
```

lemma *arity_is_inj_fm* [*arity*]:
$A \in nat \Longrightarrow B \in nat \Longrightarrow I \in nat \Longrightarrow arity(is\_inj\_fm(A, B, I)) = succ(A) \cup succ(B) \cup succ(I)$
⟨*proof*⟩

⟨*ML*⟩

context *M_Perm* **begin**

⟨*ML*⟩
⟨*proof*⟩
end

⟨*ML*⟩
notation *lt_rel_fm* (‹·_ < _·›)
⟨*ML*⟩

lemma *arity_lt_rel_fm*[*arity*]: $a \in nat \Longrightarrow b \in nat \Longrightarrow arity(lt\_rel\_fm(a, b)) = succ(a) \cup succ(b)$
⟨*proof*⟩

⟨*ML*⟩
notation *is_Card_fm* (‹·*Card*'(_')·›)
⟨*ML*⟩

notation *Card_rel* (‹*Card*−'(_')›)

lemma (**in** *M_Perm*) *is_Card_iff*: $M(A) \Longrightarrow is\_Card(M, A) \longleftrightarrow Card^{M}(A)$
⟨*proof*⟩

abbreviation
Card_r_set :: [*i*,*i*]⇒*o* (‹*Card*−'(_')›) **where**
$Card^{M}(i) \equiv Card\_rel(\#\#M,i)$

⟨*ML*⟩
notation *is_InfCard_fm* (‹·*InfCard*'(_')·›)
⟨*ML*⟩

notation *InfCard_rel* (‹*InfCard*−'(_')›)

abbreviation
InfCard_r_set :: [*i*,*i*]⇒*o* (‹*InfCard*−'(_')›) **where**
$InfCard^{M}(i) \equiv InfCard\_rel(\#\#M,i)$

⟨*ML*⟩

abbreviation
cadd_r :: [*i*,*i*⇒*o*,*i*] ⇒ *i* (‹_ ⊕− _› [*66*,*1*,*66*] *65*) **where**

$A \oplus^M B \equiv cadd\_rel(M,A,B)$

context *M_basic* **begin**
⟨*ML*⟩
⟨*proof*⟩
end

⟨*ML*⟩
⟨*proof*⟩
⟨*ML*⟩

context *M_Perm* **begin**
⟨*ML*⟩
⟨*proof*⟩
end

⟨*ML*⟩

abbreviation
cmult_r :: [*i*,*i*⇒*o*,*i*] ⇒ *i* (‹_ ⊗‒ _› [*66*,*1*,*66*] *65*) **where**
$A \otimes^M B \equiv cmult\_rel(M,A,B)$

⟨*ML*⟩

declare *cartprod_iff_sats* [*iff_sats*]

⟨*ML*⟩

context *M_Perm* **begin**

⟨*ML*⟩
⟨*proof*⟩

⟨*ML*⟩
⟨*proof*⟩

end

definition
Powapply :: [*i*,*i*] ⇒ *i* **where**
Powapply(*f*,*y*) ≡ *Pow*(*f*'*y*)

⟨*ML*⟩

lemma *subset_iff_sats*[*iff_sats*]:
nth(*i*, *env*) = *x* ⟹ *nth*(*j*, *env*) = *y* ⟹ *i*∈*nat* ⟹ *j*∈*nat* ⟹

$env \in list(A) \Longrightarrow subset(\#\#A,\ x,\ y) \longleftrightarrow A,\ env \models subset\_fm(i,\ j)$
⟨proof⟩

declare *Replace_iff_sats*[*iff_sats*]

⟨ML⟩

notation *Powapply_rel* (‹*Powapply*–'(_,_')›)

context *M_basic*
begin

⟨ML⟩
⟨proof⟩

⟨ML⟩
⟨proof⟩

⟨ML⟩
⟨proof⟩

end

definition
HVfrom :: [*i*,*i*,*i*] ⇒ *i* **where**
$HVfrom(A,x,f) \equiv A \cup (\bigcup y{\in}x.\ Powapply(f,y))$

⟨ML⟩

notation *HVfrom_rel* (‹*HVfrom*–'(_,_,_')›)

locale *M_HVfrom* = *M_eclose* +
assumes
Powapply_replacement:
$M(K) \Longrightarrow strong\_replacement(M,\lambda y\ z.\ z = Powapply^{M}(f,y))$
begin

⟨ML⟩
⟨proof⟩

⟨ML⟩
⟨proof⟩

⟨ML⟩
⟨proof⟩

end

definition
Vfrom_rel :: $[i \Rightarrow o,i,i] \Rightarrow i$ (‹*Vfrom*−′(_,_′)›) **where**
$Vfrom^M(A,i) = transrec(i,\ HVfrom\_rel(M,A))$

definition
is_Vfrom :: $[i \Rightarrow o,i,i,i] \Rightarrow o$ **where**
$is\_Vfrom(M,A,i,z) \equiv is\_transrec(M,is\_HVfrom(M,A),i,z)$

locale *M_Vfrom* = *M_HVfrom* +
assumes
trepl_HVfrom : $[\![\ M(A);M(i)\ ]\!] \Longrightarrow transrec\_replacement(M,is\_HVfrom(M,A),i)$

begin

lemma *Vfrom_rel_iff* :
assumes $M(A)\ M(i)\ M(z)\ Ord(i)$
shows $is\_Vfrom(M,A,i,z) \longleftrightarrow z = Vfrom^M(A,i)$
⟨*proof*⟩

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 :: $[i \Rightarrow o,i \Rightarrow i] \Rightarrow o$ **where**
$lam\_replacement(M,b) \equiv strong\_replacement(M,\ \lambda x\ y.\ y = \langle x,\ b(x)\rangle)$

lemma *separation_univ* :
shows $separation(M,M)$
⟨*proof*⟩

context *M_basic*

begin

lemma *separation_in* :
  assumes *M(a)*
  shows *separation(M,λx . x∈a)*
⟨*proof*⟩

lemma *separation_equal* :
  shows *separation(M,λx . x=a)*
⟨*proof*⟩

lemma (**in** *M_basic*) *separation_in_rev*:
  assumes *(M)(a)*
  shows *separation(M,λx . a∈x)*
⟨*proof*⟩

lemma *lam_replacement_iff_lam_closed*:
  assumes ∀ *x*[*M*]. *M*(*b*(*x*))
  shows *lam_replacement*(*M*, *b*) ⟷ (∀ *A*[*M*]. *M*(λ*x*∈*A*. *b*(*x*)))
  ⟨*proof*⟩

lemma *lam_replacement_cong*:
  assumes *lam_replacement*(*M*,*f*) ∀ *x*[*M*]. *f*(*x*) = *g*(*x*) ∀ *x*[*M*]. *M*(*f*(*x*))
  shows *lam_replacement*(*M*,*g*)
⟨*proof*⟩

lemma *converse_subset* : *converse*(*r*) ⊆ {<*snd*(*x*),*fst*(*x*)> . *x*∈*r*}
  ⟨*proof*⟩

lemma *converse_eq_aux* :
  assumes <*0*,*0*>∈*r*
  shows *converse*(*r*) = {<*snd*(*x*),*fst*(*x*)> . *x*∈*r*}
  ⟨*proof*⟩

lemma *converse_eq_aux′* :
  assumes <*0*,*0*>∉*r*
  shows *converse*(*r*) = {<*snd*(*x*),*fst*(*x*)> . *x*∈*r*} - {<*0*,*0*>}
  ⟨*proof*⟩

lemma *diff_un* : *b*⊆*a* ⟹ (*a*-*b*) ∪ *b* = *a*
  ⟨*proof*⟩

lemma *converse_eq*: *converse*(*r*) = ({<*snd*(*x*),*fst*(*x*)> . *x*∈*r*} - {<*0*,*0*>}) ∪ (*r*∩{<*0*,*0*>})

⟨*proof*⟩

lemma *range_subset* : *range*(*r*) ⊆ {*snd*(*x*). *x*∈*r*}
  ⟨*proof*⟩

lemma *lam_replacement_imp_strong_replacement_aux*:
  assumes *lam_replacement*(M, b) $\forall x[M]. M(b(x))$
  shows *strong_replacement*$(M, \lambda x\ y.\ y = b(x))$
$\langle proof \rangle$

lemma *lam_replacement_imp_RepFun_Lam*:
  assumes *lam_replacement*(M, f) $M(A)$
  shows $M(\{y\ .\ x{\in}A\ ,\ M(y) \wedge y{=}\langle x,f(x)\rangle\})$
$\langle proof \rangle$

lemma *lam_closed_imp_closed*:
  assumes $\forall A[M]. M(\lambda x{\in}A. f(x))$
  shows $\forall x[M]. M(f(x))$
$\langle proof \rangle$

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*$(M, \lambda x.\ if\ b(x)\ then\ f(x)\ else\ g(x))$
$\langle proof \rangle$

lemma *lam_replacement_constant*: $M(b) \Longrightarrow$ *lam_replacement*$(M,\lambda\_.\ b)$
  $\langle proof \rangle$

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 $\{y\ .\ x \in A,\ P(y)\} = 0$
  $\langle proof \rangle$

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 $\{y\ .\ x \in A,\ P(x, y)\} \in Pow^M(U)$
$\langle proof \rangle$

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 $\{y\ .\ x \in \{0\},\ P(y)\} \in Pow^M(U)$
$\langle proof \rangle$

lemma *The_in_Pow_rel_Union*:
  assumes $\bigwedge b.\ P(b) \Longrightarrow b \in U$ *separation*$(M, \lambda y.\ P(y))$ $M(U)$
  shows $(THE\ i.\ P(i)) \in Pow^M(\bigcup U)$

$\langle proof \rangle$

lemma *separation_least*: *separation*$(M, \lambda y.\ Ord(y) \wedge P(y) \wedge (\forall j.\ j < y \longrightarrow \neg P(j)))$
  $\langle proof \rangle$

lemma *Least_in_Pow_rel_Union*:
  assumes $\bigwedge b.\ P(b) \Longrightarrow b \in U$
    $M(U)$
  shows $(\mu\ i.\ P(i)) \in Pow^M(\bigcup U)$
  $\langle proof \rangle$

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)
$\langle proof \rangle$

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*$(M,domain)$
$\langle proof \rangle$
lemma *lam_replacement_fst'*:
  assumes $\forall A[M].\ separation(M, \lambda y.\ \exists x{\in}A.\ y = \langle x, fst(x)\rangle)$
  shows *lam_replacement*(M,fst)
$\langle proof \rangle$

lemma *lam_replacement_restrict*:
assumes $\forall A[M].\ separation(M, \lambda y.\ \exists x{\in}A.\ y = \langle x, restrict(x,B)\rangle)$ $M(B)$
shows *lam_replacement*$(M, \lambda r\ .\ restrict(r,B))$
$\langle proof \rangle$

end

locale *M_replacement* = *M_basic* +
  assumes
    lam_replacement_domain: *lam_replacement*$(M,domain)$
    and
    lam_replacement_vimage: *lam_replacement*$(M, \lambda p.\ fst(p)$ -‘‘ $snd(p))$
    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*$(M, \lambda z.\ \exists x[M].\ z = \langle x, x\rangle)$

and
middle_separation: *separation*(M, λx. *snd*(*fst*(x))=*fst*(*snd*(x)))
and
middle_del_replacement: *strong_replacement*(M, $\lambda x\, y$. 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 : $M(a) \Longrightarrow$ *separation*(M,λx . *fst*(x)=a)
and
separation_equal_fst2 : $M(a) \Longrightarrow$ *separation*(M,λx . *fst*(*fst*(x))=a)
and
separation_equal_apply: $M(f) \Longrightarrow M(a) \Longrightarrow$ *separation*(M,λx. f`x=a)
and
separation_restrict: $M(B) \Longrightarrow \forall A[M]$. *separation*($M$, λy. $\exists x \in A$. y = ⟨x, *restrict*(x, B)⟩)
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, λx y. y = $f(x)$)
⟨*proof*⟩

lemma *Collect_middle*: {$p \in$ ($\lambda x \in A$. $f(x)$) × ($\lambda x \in$ {$f(x)$. $x \in A$}. $g(x)$) . *snd*(*fst*(p))=*fst*(*snd*(p))}
= { ⟨⟨x,$f(x)$⟩,⟨$f(x)$,$g(f(x))$⟩⟩ . $x \in A$ }
⟨*proof*⟩

lemma *RepFun_middle_del*: { ⟨*fst*(*fst*(p)),*snd*(*snd*(p))⟩ . $p \in$ { ⟨⟨x,$f(x)$⟩,⟨$f(x)$,$g(f(x))$⟩⟩ . $x \in A$ }}
= { ⟨x,$g(f(x))$⟩ . $x \in A$ }
⟨*proof*⟩

lemma *lam_replacement_imp_RepFun*:
assumes *lam_replacement*(M, f) $M(A)$
shows M({y . $x \in A$, $M(y) \wedge y$=$f(x)$})
⟨*proof*⟩

lemma *lam_replacement_product*:
assumes *lam_replacement*(M,f) *lam_replacement*(M,g)

shows *lam_replacement*$(M, \lambda x. \langle f(x),g(x)\rangle)$
⟨*proof*⟩

lemma *lam_replacement_hcomp*:
assumes *lam_replacement*(M,f) *lam_replacement*(M,g) $\forall x[M]. M(f(x))$
shows *lam_replacement*$(M, \lambda x. g(f(x)))$
⟨*proof*⟩

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*⟩

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*$(M, \lambda x. h(f(x),g(x)))$
⟨*proof*⟩

lemma *strong_replacement_separation_aux* :
assumes *strong_replacement*$(M,\lambda x y . y=f(x))$ *separation*(M,P)
shows *strong_replacement*$(M, \lambda x y . P(x) \wedge y=f(x))$
⟨*proof*⟩

lemma *lam_replacement_separation* :
assumes *lam_replacement*(M,f) *separation*(M,P)
shows *strong_replacement*$(M, \lambda x y . P(x) \wedge y=\langle x,f(x)\rangle)$
⟨*proof*⟩

lemmas *strong_replacement_separation* =
strong_replacement_separation_aux[*OF lam_replacement_imp_strong_replacement*]

lemma *lam_replacement_identity*: *lam_replacement*$(M,\lambda x. x)$
⟨*proof*⟩

lemma *lam_replacement_Un*: *lam_replacement*$(M, \lambda p. fst(p) \cup snd(p))$
⟨*proof*⟩

lemma *lam_replacement_cons*: *lam_replacement*$(M, \lambda p. cons(fst(p),snd(p)))$
⟨*proof*⟩

lemma *lam_replacement_sing*: *lam_replacement*$(M, \lambda x. \{x\})$
⟨*proof*⟩

lemmas *tag_replacement* = *lam_replacement_constant*[*unfolded lam_replacement_def*]

lemma *lam_replacement_id2*: *lam_replacement*(M, λx. $\langle x, x\rangle$)
⟨*proof*⟩

lemmas *id_replacement* = *lam_replacement_id2*[*unfolded lam_replacement_def*]

lemma *lam_replacement_apply2*:*lam_replacement*(M, λp. *fst*(p) ‘ *snd*(p))
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *map_sndI* : $\exists p \in X$. y=*snd*(p) $\Longrightarrow$ $y \in$*map_snd*(X)
⟨*proof*⟩

lemma *map_snd_closed*: $M(x)$ $\Longrightarrow$ M(*map_snd*(x))
⟨*proof*⟩

lemma *lam_replacement_imp_lam_replacement_RepFun*:
assumes *lam_replacement*(M, f) $\forall x[M]$. $M(f(x))$
separation(M, λx. (($\forall y \in$*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, λx. {$f(y)$. $y \in x$})
⟨*proof*⟩

lemma *lam_replacement_apply*:$M(S)$ $\Longrightarrow$ *lam_replacement*(M, λx. S ‘ x)
⟨*proof*⟩

lemma *apply_replacement*:$M(S)$ $\Longrightarrow$ *strong_replacement*(M, $\lambda x\ y$. $y = S$ ‘ x)
⟨*proof*⟩

lemma *lam_replacement_id_const*: $M(b)$ $\Longrightarrow$ *lam_replacement*(M, λx. $\langle x, b\rangle$)
⟨*proof*⟩

lemmas *pospend_replacement* = *lam_replacement_id_const*[*unfolded lam_replacement_def*]

lemma *lam_replacement_const_id*: $M(b)$ $\Longrightarrow$ *lam_replacement*(M, λz. $\langle b, z\rangle$)
⟨*proof*⟩

lemmas *prepend_replacement* = *lam_replacement_const_id*[*unfolded lam_replacement_def*]

lemma *lam_replacement_apply_const_id*: $M(f)$ $\Longrightarrow$ $M(z)$ $\Longrightarrow$
lam_replacement(M, λx. f ‘ $\langle z, x\rangle$)

⟨*proof*⟩

lemmas *apply_replacement2* = *lam_replacement_apply_const_id*[*unfolded lam_replacement_def*]

lemma *lam_replacement_Inl*: *lam_replacement*(M, *Inl*)
⟨*proof*⟩

lemma *lam_replacement_Inr*: *lam_replacement*(M, *Inr*)
⟨*proof*⟩

lemmas *Inl_replacement1* = *lam_replacement_Inl*[*unfolded lam_replacement_def*]

lemma *lam_replacement_Diff'*: $M(X) \Longrightarrow$ *lam_replacement*(M, $\lambda x.\ x$ - X)
⟨*proof*⟩

lemmas *Pair_diff_replacement* = *lam_replacement_Diff'*[*unfolded lam_replacement_def*]

lemma *diff_Pair_replacement*: $M(p) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ y\ .\ y=\langle x,x\text{-}\{p\}\rangle$)
⟨*proof*⟩

lemma *lam_replacement_swap*: *lam_replacement*(M, $\lambda x.\ \langle$*snd*(x),*fst*(x)$\rangle$)
⟨*proof*⟩

lemma *swap_replacement*:*strong_replacement*(M, $\lambda x\ y.\ y = \langle x, (\lambda\langle x,y\rangle.\ \langle y, x\rangle)(x)\rangle$)
⟨*proof*⟩

lemma *lam_replacement_Un_const*:$M(b) \Longrightarrow$ *lam_replacement*(M, $\lambda x.\ x \cup b$)
⟨*proof*⟩

lemmas *tag_union_replacement* = *lam_replacement_Un_const*[*unfolded lam_replacement_def*]

lemma *lam_replacement_csquare*: *lam_replacement*(M,$\lambda p.\ \langle$*fst*(p) $\cup$ *snd*(p), *fst*(p), *snd*(p)$\rangle$)
⟨*proof*⟩

lemma *csquare_lam_replacement*:*strong_replacement*(M, $\lambda x\ y.\ y = \langle x, (\lambda\langle x,y\rangle.\ \langle x \cup y, x, y\rangle)(x)\rangle$)
⟨*proof*⟩

lemma *lam_replacement_assoc*:*lam_replacement*(M,$\lambda x.\ \langle$*fst*(*fst*(x)), *snd*(*fst*(x)), *snd*(x)$\rangle$)
⟨*proof*⟩

lemma *assoc_replacement*:*strong_replacement*(M, $\lambda x\ y.\ y = \langle x, (\lambda\langle\langle x,y\rangle,z\rangle.\ \langle x, y, z\rangle)(x)\rangle$)
⟨*proof*⟩

lemma *lam_replacement_prod_fun*: $M(f) \Longrightarrow M(g) \Longrightarrow$ *lam_replacement*(M,$\lambda x.\ \langle f$ ‘ *fst*(x), g ‘ *snd*(x)$\rangle$)

⟨*proof*⟩

lemma *prod_fun_replacement*:$M(f) \Longrightarrow M(g) \Longrightarrow$
strong_replacement$(M, \lambda x\ y.\ y = \langle x, (\lambda\langle w,y\rangle.\ \langle f\ `\ w,\ g\ `\ y\rangle)(x)\rangle)$
⟨*proof*⟩

lemma *lam_replacement_vimage_sing*: *lam_replacement*$(M, \lambda p.\ fst(p)$ -`` $\{snd(p)\})$
⟨*proof*⟩

lemma *lam_replacement_vimage_sing_fun*: $M(f) \Longrightarrow$ *lam_replacement*$(M, \lambda x.$
f -`` $\{x\})$
⟨*proof*⟩

lemma *converse_apply_projs*: $\forall x[M].\ \bigcup\ (fst(x)$ -`` $\{snd(x)\}) = converse(fst(x))$
` $(snd(x))$
⟨*proof*⟩

lemma *lam_replacement_converse_app*: *lam_replacement*$(M, \lambda p.\ converse(fst(p))$
` $snd(p))$
⟨*proof*⟩

lemmas *cardinal_lib_assms4* = *lam_replacement_vimage_sing_fun*[*unfolded lam_replacement_def*]

lemma *lam_replacement_sing_const_id*:
$M(x) \Longrightarrow$ *lam_replacement*$(M, \lambda y.\ \{\langle x, y\rangle\})$
⟨*proof*⟩

lemma *tag_singleton_closed*: $M(x) \Longrightarrow M(z) \Longrightarrow M(\{\{\langle z, y\rangle\}\ .\ y \in x\})$
⟨*proof*⟩

lemma *case_closed* :
assumes $\forall x[M].\ M(f(x))\ \forall x[M].\ M(g(x))$
shows $\forall x[M].\ M(case(f,g,x))$
⟨*proof*⟩

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*$(M, \lambda x\ .\ case(f,g,x))$
⟨*proof*⟩

lemma *Pi_replacement1*: $M(x) \Longrightarrow M(y) \Longrightarrow$ *strong_replacement*$(M, \lambda ya\ z.\ ya$
$\in y \wedge z = \{\langle x, ya\rangle\})$
⟨*proof*⟩

lemma *surj_imp_inj_replacement1*:
$M(f) \Longrightarrow M(x) \Longrightarrow$ *strong_replacement*$(M, \lambda y\ z.\ y \in f$ -`` $\{x\} \wedge z = \{\langle x, y\rangle\})$
⟨*proof*⟩

lemmas *domain_replacement* = *lam_replacement_domain*[*unfolded lam_replacement_def*]

lemma *domain_replacement_simp*: *strong_replacement*(M, $\lambda x\ y.\ y$=*domain*(x))
⟨*proof*⟩

lemma *un_Pair_replacement*: $M(p) \Longrightarrow$ *strong_replacement*(M, $\lambda x\, y\ .\ y = x \cup \{p\}$)
⟨*proof*⟩

lemma *restrict_strong_replacement*: $M(A) \Longrightarrow$ *strong_replacement*(M, $\lambda x\, y.\ y$=*restrict*(x,A))
⟨*proof*⟩

lemma *diff_replacement*: $M(X) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ y.\ y = x$ - X)
⟨*proof*⟩

lemma *lam_replacement_succ*:
lam_replacement(M,λz . *succ*(z))
⟨*proof*⟩

lemma *lam_replacement_hcomp_Least*:
assumes *lam_replacement*(M, g) *lam_replacement*(M,$\lambda x.\ \mu\ i.\ x \in F(i,x)$)
$\forall\, x[M].\ M(g(x))\ \bigwedge x\ i.\ M(x) \Longrightarrow i \in F(i,\ x) \Longrightarrow M(i)$
shows *lam_replacement*(M,$\lambda x.\ \mu\ i.\ g(x) \in F(i,g(x))$)
⟨*proof*⟩

end

locale *M_replacement_extra* = *M_replacement* +
assumes
lam_replacement_minimum:*lam_replacement*(M, λp. *minimum*(*fst*(p),*snd*(p)))
and
lam_replacement_RepFun_cons:*lam_replacement*(M, λp. *RepFun*(*fst*(p), λx. $\{\langle$*snd*(p),$x\rangle\}$))
— 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) $\forall\, y[M].\ M(f(y))$
shows *lam_replacement*(M, λx. *Sigfun*(x,f))
⟨*proof*⟩

20.2 Particular instances

lemma *surj_imp_inj_replacement2*:
$M(f) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ z.\ z$ = *Sigfun*(x, $\lambda y.\ f$ -‘‘ $\{y\}$))
⟨*proof*⟩

lemma *lam_replacement_minimum_vimage*:
$M(f) \Longrightarrow M(r) \Longrightarrow$ *lam_replacement*(M, λx. *minimum*(r, f -‘‘ $\{x\}$))

⟨*proof*⟩

lemmas *surj_imp_inj_replacement4* = *lam_replacement_minimum_vimage*[*unfolded lam_replacement_def*]

lemma *lam_replacement_Pi*: $M(y) \Longrightarrow$ *lam_replacement*$(M,\ \lambda x.\ \bigcup xa\in y.\ \{\langle x,\ xa\rangle\})$
⟨*proof*⟩

lemma *Pi_replacement2*: $M(y) \Longrightarrow$ *strong_replacement*$(M,\ \lambda x\ z.\ z = (\bigcup xa\in y.\ \{\langle x,\ xa\rangle\}))$
⟨*proof*⟩

lemma *if_then_Inj_replacement*:
shows $M(A) \Longrightarrow$ *strong_replacement*$(M,\ \lambda x\ y.\ y = \langle x,$ *if* $x \in A$ *then* $Inl(x)$ *else* $Inr(x)\rangle)$
⟨*proof*⟩

lemma *lam_if_then_replacement*:
$M(b) \Longrightarrow$
$M(a) \Longrightarrow M(f) \Longrightarrow$ *strong_replacement*$(M,\ \lambda y\ ya.\ ya = \langle y,$ *if* $y = a$ *then* b *else* $f\ `\ y\rangle)$
⟨*proof*⟩

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* $g\ `\ x\rangle)$
⟨*proof*⟩

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* $b\rangle)$
⟨*proof*⟩

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* $x\rangle)$
⟨*proof*⟩

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* $z\rangle)$
⟨*proof*⟩

lemma *Inl_replacement2*:
$M(A) \Longrightarrow$
$strong\_replacement(M, \lambda x\ y.\ y = \langle x, if\ fst(x) = A\ then\ Inl(snd(x))\ else\ Inr(x)\rangle)$
⟨*proof*⟩

lemma *case_replacement1*:
$strong\_replacement(M, \lambda z\ y.\ y = \langle z,\ case(Inr,\ Inl,\ z)\rangle)$
⟨*proof*⟩

lemma *case_replacement2*:
$strong\_replacement(M, \lambda z\ y.\ y = \langle z, case(case(Inl, \lambda y.\ Inr(Inl(y))), \lambda y.\ Inr(Inr(y)), z)\rangle)$
⟨*proof*⟩

lemma *case_replacement4*:
$M(f) \Longrightarrow M(g) \Longrightarrow strong\_replacement(M, \lambda z\ y.\ y = \langle z,\ case(\lambda w.\ Inl(f\ `\ w), \lambda y.\ Inr(g\ `\ y), z)\rangle)$
⟨*proof*⟩

lemma *case_replacement5*:
$strong\_replacement(M, \lambda x\ y.\ y = \langle x, (\lambda\langle x,z\rangle.\ case(\lambda y.\ Inl(\langle y, z\rangle), \lambda y.\ Inr(\langle y, z\rangle), x))(x)\rangle)$
⟨*proof*⟩

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*: $drSR\_Y(B,D,A,x) = \{\ dr{\in}D\ .\ (\exists\, r{\in}A\ .\ restrict(r,B) = x \wedge dr{=}domain(r))\ \}$
⟨*proof*⟩

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 $lam\_replacement(M, drSR\_Y(B,D,A))$
⟨*proof*⟩

lemma *lam_if_then_apply_replacement*: $M(f) \Longrightarrow M(v) \Longrightarrow M(u) \Longrightarrow lam\_replacement(M, \lambda x.\ if\ f\ `\ x = v\ then\ f\ `\ u\ else\ f\ `\ x)$
⟨*proof*⟩

lemma *lam_if_then_apply_replacement2*: $M(f) \Longrightarrow M(m) \Longrightarrow M(y) \Longrightarrow lam\_replacement(M, \lambda z .\ if\ f\ `\ z = m\ then\ y\ else\ f\ `\ z)$
⟨*proof*⟩

lemma *lam_if_then_replacement2*: $M(A) \Longrightarrow M(f) \Longrightarrow lam\_replacement(M, \lambda x .\ if\ x \in A\ then\ f\ `\ x\ else\ x)$
⟨*proof*⟩

lemma *lam_if_then_replacement_apply*: $M(G) \Longrightarrow lam\_replacement(M, \lambda x.\ if\ M(x)\ then\ G\ `\ x\ else\ 0)$
⟨*proof*⟩

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 $lam\_replacement(M, dC\_F(A))$
⟨*proof*⟩

lemma *lam_replacement_min*: $M(f) \Longrightarrow M(r) \Longrightarrow lam\_replacement(M, \lambda x .\ minimum(r, f\ \text{-``}\ \{x\}))$
⟨*proof*⟩

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 $lam\_replacement(M, \lambda x .\ \{a \in G .\ \forall s{\in}x. \langle s, a\rangle \in Q\})$
⟨*proof*⟩

lemma *surj_imp_inj_replacement3*:
$(\bigwedge x. M(x) \Longrightarrow separation(M, \lambda y. \forall s{\in}x. \langle s, y\rangle \in Q)) \Longrightarrow M(G) \Longrightarrow M(Q) \Longrightarrow M(x) \Longrightarrow$
$strong\_replacement(M, \lambda y\ z.\ y \in \{a \in G .\ \forall s{\in}x. \langle s, a\rangle \in Q\} \wedge z = \{\langle x, y\rangle\})$
⟨*proof*⟩

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⇒o,i]=>o **where**
Finite_rel(M,A) ≡ ∃ om[M]. ∃ n[M]. *omega*(M,om) ∧ n∈om ∧ *eqpoll_rel*(M,A,n)

definition
banach_functor :: [i,i,i,i,i] ⇒ i **where**
banach_functor(X,Y,f,g,W) ≡ X - g``(Y - f``W)

definition
is_banach_functor :: [i⇒o,i,i,i,i,i,i]⇒o **where**
is_banach_functor(M,X,Y,f,g,W,b) ≡
∃ fW[M]. ∃ YfW[M]. ∃ gYfW[M]. *image*(M,f,W,fW) ∧ *setdiff*(M,Y,fW,YfW)
∧
image(M,g,YfW,gYfW) ∧ *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))
⟨*proof*⟩

lemma (**in** *M_basic*) *banach_functor_closed*:
assumes M(X) M(Y) M(f) M(g)
shows ∀ W[M]. M(*banach_functor*(X,Y,f,g,W))
⟨*proof*⟩

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*]: $M(a) \Longrightarrow M(b) \Longrightarrow M(c) \Longrightarrow M(d) \Longrightarrow M(radd(a,b,c,d))$
⟨*proof*⟩

lemma *rmult_closed*[*intro*,*simp*]: $M(a) \Longrightarrow M(b) \Longrightarrow M(c) \Longrightarrow M(d) \Longrightarrow M(rmult(a,b,c,d))$
⟨*proof*⟩

end

lemma (**in** *M_cardinals*) *is_cardinal_iff_Least*:
assumes $M(A)\ M(\kappa)$
shows *is_cardinal*$(M,A,\kappa) \longleftrightarrow \kappa = (\mu\ i.\ M(i) \wedge i \approx^{M} A)$
⟨*proof*⟩

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*$(X,Y,f,g))$
⟨*proof*⟩

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*⟩

lemma *contin_banach_functor*:
  assumes $g \in inj(Y,X)$
  shows $contin(banach\_functor(X,Y,f,g))$
  ⟨*proof*⟩

lemma *lfp_banach_functor*:
  assumes $g \in inj(Y,X)$
  shows $lfp(X,\ banach\_functor(X,Y,f,g)) =$
      $(\bigcup n\in nat.\ banach\_functor(X,Y,f,g)\hat{}n\ (0))$
  ⟨*proof*⟩

lemma *lfp_banach_functor_closed*:
  assumes $M(g)\ M(X)\ M(Y)\ M(f)\ g\in inj(Y,X)$
  shows $M(lfp(X,\ banach\_functor(X,Y,f,g)))$
⟨*proof*⟩

lemma *banach_decomposition_rel*:
  $[|\ M(f);\ M(g);\ M(X);\ M(Y);\ f \in X\text{->}Y;\ \ g \in inj(Y,X)\ |] ==>$
    $\exists XA[M].\ \exists XB[M].\ \exists YA[M].\ \exists YB[M].$
      $(XA \cap XB = 0)\ \&\ (XA \cup XB = X)\ \&$
      $(YA \cap YB = 0)\ \&\ (YA \cup YB = Y)\ \&$
      $f``XA=YA\ \&\ g``YB=XB$
  ⟨*proof*⟩

lemma *schroeder_bernstein_closed*:
  $[|\ M(f);\ M(g);\ M(X);\ M(Y);\ f \in inj(X,Y);\ \ g \in inj(Y,X)\ |] ==> \exists h[M].\ h \in$
$bij(X,Y)$
  ⟨*proof*⟩

lemma *mem_Pow_rel*: $M(r) \Longrightarrow a \in Pow\_rel(M,r) \Longrightarrow a \in Pow(r) \wedge M(a)$
  ⟨*proof*⟩

lemma *mem_bij_abs*[*simp*]: $[\![M(f);M(A);M(B)]\!] \Longrightarrow f \in bij^M(A,B) \longleftrightarrow f\in bij(A,B)$
  ⟨*proof*⟩

lemma *mem_inj_abs*[*simp*]: $[\![M(f);M(A);M(B)]\!] \Longrightarrow f \in inj^M(A,B) \longleftrightarrow f\in inj(A,B)$
  ⟨*proof*⟩

lemma *mem_surj_abs*: $[\![M(f);M(A);M(B)]\!] \Longrightarrow f \in surj^M(A,B) \longleftrightarrow f\in surj(A,B)$
  ⟨*proof*⟩

lemma *bij_imp_eqpoll_rel*:
  assumes $f \in bij(A,B)\ M(f)\ M(A)\ M(B)$
  shows $A \approx^M B$
  ⟨*proof*⟩

lemma *id_closed*: $M(A) \Longrightarrow M(id(A))$
⟨*proof*⟩

lemma *eqpoll_rel_refl*: $M(A) \Longrightarrow A \approx^M A$
⟨*proof*⟩

lemma *eqpoll_rel_sym*: $X \approx^M Y \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow Y \approx^M X$
⟨*proof*⟩

lemma *eqpoll_rel_trans* [*trans*]:
$[|X \approx^M Y; \ Y \approx^M Z; \ M(X); M(Y) ; M(Z) |] ==> X \approx^M Z$
⟨*proof*⟩

lemma *subset_imp_lepoll_rel*: $X \subseteq Y \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow X \lesssim^M Y$
⟨*proof*⟩

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*: $X \approx^M Y ==> M(X) \Longrightarrow M(Y) \Longrightarrow X \lesssim^M Y$
⟨*proof*⟩

lemma *lepoll_rel_trans* [*trans*]:
assumes
$X \lesssim^M Y \ Y \lesssim^M Z \ M(X) \ M(Y) \ M(Z)$
shows
$X \lesssim^M Z$
⟨*proof*⟩

lemma *eq_lepoll_rel_trans* [*trans*]:
assumes
$X \approx^M Y \ \ Y \lesssim^M Z \ M(X) \ M(Y) \ M(Z)$
shows
$X \lesssim^M Z$
⟨*proof*⟩

lemma *lepoll_rel_eq_trans* [*trans*]:
assumes $X \lesssim^M Y \ \ Y \approx^M Z \ M(X) \ M(Y) \ M(Z)$
shows $X \lesssim^M Z$
⟨*proof*⟩

lemma *eqpoll_relI*: $[\![X \lesssim^M Y; \ Y \lesssim^M X; \ M(X) ; \ M(Y)]\!] \Longrightarrow X \approx^M Y$
⟨*proof*⟩

lemma *eqpoll_relE*:

$[| X \approx^M Y; [| X \lesssim^M Y; Y \lesssim^M X |] ==> P ; M(X) ; M(Y) |] ==> P$
⟨*proof*⟩

lemma *eqpoll_rel_iff*: $M(X) \Longrightarrow M(Y) \Longrightarrow X \approx^M Y \longleftrightarrow X \lesssim^M Y \ \& \ Y \lesssim^M X$
⟨*proof*⟩

lemma *lepoll_rel_0_is_0*: $A \lesssim^M 0 \Longrightarrow M(A) \Longrightarrow A = 0$
⟨*proof*⟩

lemmas *empty_lepoll_relI* = *empty_subsetI* [*THEN subset_imp_lepoll_rel*, *OF nonempty*]

lemma *lepoll_rel_0_iff*: $M(A) \Longrightarrow A \lesssim^M 0 \longleftrightarrow A=0$
⟨*proof*⟩

lemma *Un_lepoll_rel_Un*:
$[| A \lesssim^M B; C \lesssim^M D; B \cap D = 0; M(A); M(B); M(C); M(D) |] ==> A \cup C \lesssim^M B \cup D$
⟨*proof*⟩

lemma *eqpoll_rel_0_is_0*: $A \approx^M 0 \Longrightarrow M(A) \Longrightarrow A = 0$
⟨*proof*⟩

lemma *eqpoll_rel_0_iff*: $M(A) \Longrightarrow A \approx^M 0 \longleftrightarrow A=0$
⟨*proof*⟩

lemma *eqpoll_rel_disjoint_Un*:
$[| A \approx^M B; C \approx^M D; A \cap C = 0; B \cap D = 0; M(A); M(B); M(C) ; M(D) |]$
$==> A \cup C \approx^M B \cup D$
⟨*proof*⟩

21.2 lesspoll_rel: contributions by Krzysztof Grabczewski

lemma *lesspoll_rel_not_refl*: $M(i) \Longrightarrow \sim (i \prec^M i)$
⟨*proof*⟩

lemma *lesspoll_rel_irrefl*: $i \prec^M i ==> M(i) \Longrightarrow P$
⟨*proof*⟩

lemma *lesspoll_rel_imp_lepoll_rel*: $[\![A \prec^M B; M(A); M(B)]\!] \Longrightarrow A \lesssim^M B$
⟨*proof*⟩

lemma *rvimage_closed* [*intro*,*simp*]:
assumes
$M(A)$ $M(f)$ $M(r)$
shows
$M(rvimage(A,f,r))$
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *lepoll_rel_iff_leqpoll_rel*: $[\![M(A);\ M(B)]\!] \Longrightarrow A \lesssim^M B \longleftrightarrow A \prec^M B \mid A \approx^M B$
⟨*proof*⟩

end

context *M_cardinals*
begin

lemma *inj_rel_is_fun_M*: $f \in inj^M(A,B) \Longrightarrow M(f) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow f \in A \to^M B$
⟨*proof*⟩
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*⟩

lemma *lesspoll_rel_trans* [*trans*]:
[| $X \prec^M Y$; $Y \prec^M Z$; *M*(*X*); *M*(*Y*) ; *M*(*Z*) |] ==> $X \prec^M Z$
⟨*proof*⟩

lemma *lesspoll_rel_trans1* [*trans*]:
[| $X \lesssim^M Y$; $Y \prec^M Z$; *M*(*X*); *M*(*Y*) ; *M*(*Z*) |] ==> $X \prec^M Z$
⟨*proof*⟩

lemma *lesspoll_rel_trans2* [*trans*]:
[| $X \prec^M Y$; $Y \lesssim^M Z$; *M*(*X*); *M*(*Y*) ; *M*(*Z*)|] ==> $X \prec^M Z$
⟨*proof*⟩

lemma *eq_lesspoll_rel_trans* [*trans*]:
[| $X \approx^M Y$; $Y \prec^M Z$; *M*(*X*); *M*(*Y*) ; *M*(*Z*) |] ==> $X \prec^M Z$
⟨*proof*⟩

lemma *lesspoll_rel_eq_trans* [*trans*]:
[| $X \prec^M Y$; $Y \approx^M Z$; *M*(*X*); *M*(*Y*) ; *M*(*Z*) |] ==> $X \prec^M Z$
⟨*proof*⟩

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*⟩
lemma *cardinal_rel_cong*: $X \approx^M Y \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow |X|^M = |Y|^M$
⟨*proof*⟩

lemma *well_ord_is_cardinal_eqpoll_rel*:
assumes *well_ord*(A,r) **shows** *is_cardinal*$(M,A,\kappa) \Longrightarrow M(A) \Longrightarrow M(\kappa) \Longrightarrow M(r) \Longrightarrow \kappa \approx^M A$
⟨*proof*⟩

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** $|A|^M \approx^M A$
⟨*proof*⟩

lemmas *Ord_cardinal_rel_eqpoll_rel* = *well_ord_Memrel*[*THEN well_ord_cardinal_rel_eqpoll_rel*]

lemma *Ord_cardinal_rel_idem*: *Ord*$(A) \Longrightarrow M(A) \Longrightarrow ||A|^M|^M = |A|^M$
⟨*proof*⟩

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*⟩

lemma *well_ord_cardinal_rel_eqpoll_rel_iff*:
[| *well_ord*(X,r); *well_ord*(Y,s); $M(X)$; $M(r)$; $M(Y)$; $M(s)$ |] ==> $|X|^M = |Y|^M \longleftrightarrow X \approx^M Y$
⟨*proof*⟩

lemma *Ord_cardinal_rel_le*: *Ord*$(i) \Longrightarrow M(i)$ ==> $|i|^M \le i$
⟨*proof*⟩

lemma *Card_rel_cardinal_rel_eq*: *Card*$^M(K)$ ==> $M(K) \Longrightarrow |K|^M = K$
⟨*proof*⟩

lemma *Card_relI*: [| *Ord*(i); !!*j*. $j<i \Longrightarrow M(j)$ ==> ~$(j \approx^M i)$; $M(i)$ |] ==> *Card*$^M(i)$
⟨*proof*⟩

lemma *Card_rel_is_Ord*: *Card*$^M(i)$ ==> $M(i) \Longrightarrow$ *Ord*(i)

⟨*proof*⟩

lemma *Card_rel_cardinal_rel_le*: $Card^M(K)$ ==> $M(K) \Longrightarrow K \le |K|^M$
⟨*proof*⟩

lemma *Ord_cardinal_rel* [*simp*,*intro*!]: $M(A) \Longrightarrow Ord(|A|^M)$
⟨*proof*⟩

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*⟩

lemma *lt_Card_rel_imp_lesspoll_rel*: [| $Card^M(a)$; $i<a$; $M(a)$; $M(i)$ |] ==> $i \prec^M a$
⟨*proof*⟩

lemma *Card_rel_0*: $Card^M(0)$
⟨*proof*⟩

lemma *Card_rel_Un*: [| $Card^M(K)$; $Card^M(L)$; $M(K)$; $M(L)$ |] ==> $Card^M(K \cup L)$
⟨*proof*⟩

lemma *Card_rel_cardinal_rel* [*iff*]: **assumes** *types*:$M(A)$ **shows** $Card^M(|A|^M)$
⟨*proof*⟩

lemma *cardinal_rel_eq_lemma*:
assumes *i*:$|i|^M \le j$ **and** *j*: $j \le i$ **and** *types*: $M(i)$ $M(j)$
shows $|j|^M = |i|^M$
⟨*proof*⟩

lemma *cardinal_rel_mono*:
assumes *ij*: $i \le j$ **and** *types*:$M(i)$ $M(j)$ **shows** $|i|^M \le |j|^M$
⟨*proof*⟩

lemma *cardinal_rel_lt_imp_lt*: [| $|i|^M < |j|^M$; $Ord(i)$; $Ord(j)$; $M(i)$; $M(j)$ |] ==> $i < j$
⟨*proof*⟩

lemma *Card_rel_lt_imp_lt*: [| $|i|^M < K$; $Ord(i)$; $Card^M(K)$; $M(i)$; $M(K)$|] ==> $i < K$
⟨*proof*⟩

lemma *Card_rel_lt_iff*: [| $Ord(i)$; $Card^M(K)$; $M(i)$; $M(K)$ |] ==> $(|i|^M < K) \longleftrightarrow (i < K)$
⟨*proof*⟩

lemma *Card_rel_le_iff*: [| $Ord(i)$; $Card^M(K)$; $M(i)$; $M(K)$ |] ==> $(K \le |i|^M) \longleftrightarrow (K \le i)$

⟨proof⟩

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 $|A|^M \le |B|^M$
⟨proof⟩

lemma *lepoll_rel_cardinal_rel_le*: [| $A \lesssim^M i$; *Ord*(i); $M(A)$; $M(i)$ |] ==> $|A|^M \le i$
⟨proof⟩

lemma *lepoll_rel_Ord_imp_eqpoll_rel*: [| $A \lesssim^M i$; *Ord*(i); $M(A)$; $M(i)$ |] ==> $|A|^M \approx^M A$
⟨proof⟩

lemma *lesspoll_rel_imp_eqpoll_rel*: [| $A \prec^M i$; *Ord*(i); $M(A)$; $M(i)$ |] ==> $|A|^M \approx^M A$
⟨proof⟩

lemma *lesspoll_cardinal_lt_rel*:
shows [| $A \prec^M i$; *Ord*(i); $M(i)$; $M(A)$ |] ==> $|A|^M < i$
⟨proof⟩

lemma *cardinal_rel_subset_Ord*: [|A<=i; *Ord*(i); $M(A)$; $M(i)$|] ==> $|A|^M \subseteq i$
⟨proof⟩
lemma *cons_lepoll_rel_consD*:
[| *cons*$(u,A) \lesssim^M$ *cons*(v,B); $u \notin A$; $v \notin B$; $M(u)$; $M(A)$; $M(v)$; $M(B)$ |] ==> $A \lesssim^M B$
⟨proof⟩

lemma *cons_eqpoll_rel_consD*: [| *cons*$(u,A) \approx^M$ *cons*(v,B); $u \notin A$; $v \notin B$; $M(u)$; $M(A)$; $M(v)$; $M(B)$ |] ==> $A \approx^M B$
⟨proof⟩

lemma *succ_lepoll_rel_succD*: *succ*$(m) \lesssim^M$ *succ*$(n) \Longrightarrow M(m) \Longrightarrow M(n)$ ==> $m \lesssim^M n$
⟨proof⟩

lemma *nat_lepoll_rel_imp_le*:
$m \in nat$ ==> $n \in nat \Longrightarrow m \lesssim^M n \Longrightarrow M(m) \Longrightarrow M(n) \Longrightarrow m \le n$
⟨proof⟩

lemma *nat_eqpoll_rel_iff*: [| $m \in nat$; $n \in nat$; $M(m)$; $M(n)$ |] ==> $m \approx^M n \longleftrightarrow m = n$
⟨proof⟩

lemma *nat_into_Card_rel*:

assumes *n*: $n \in nat$ **and** *types*: $M(n)$ **shows** $Card^M(n)$
⟨*proof*⟩

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
⟨*proof*⟩

lemma *nat_lepoll_rel_imp_ex_eqpoll_rel_n*:
[| $n \in nat$; $nat \lesssim^M X$; $M(n)$; $M(X)$|] ==> $\exists Y[M].\ Y \subseteq X$ & $n \approx^M Y$
⟨*proof*⟩

lemma *lepoll_rel_succ*: $M(i) \Longrightarrow i \lesssim^M succ(i)$
⟨*proof*⟩

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*⟩

lemma *lesspoll_rel_succ_imp_lepoll_rel*:
[| $A \prec^M succ(m)$; $m \in nat$; $M(A)$; $M(m)$ |] ==> $A \lesssim^M m$
⟨*proof*⟩

lemma *lesspoll_rel_succ_iff*: $m \in nat \Longrightarrow M(A)$ ==> $A \prec^M succ(m) \longleftrightarrow A \lesssim^M m$
⟨*proof*⟩

lemma *lepoll_rel_succ_disj*: [| $A \lesssim^M succ(m)$; $m \in nat$; $M(A)$; $M(m)$|] ==> $A \lesssim^M m$ | $A \approx^M succ(m)$
⟨*proof*⟩

lemma *lesspoll_rel_cardinal_rel_lt*: [| $A \prec^M i$; $Ord(i)$; $M(A)$; $M(i)$ |] ==> $|A|^M < i$
⟨*proof*⟩

lemma *lt_not_lepoll_rel*:
assumes *n*: $n<i$ $n \in nat$
and *types*:$M(n)$ $M(i)$ **shows** ~ $i \lesssim^M n$
⟨*proof*⟩

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 $i \approx^M n \longleftrightarrow i{=}n$
⟨*proof*⟩

lemma *Card_rel_nat*: $Card^M(nat)$
⟨*proof*⟩

lemma *nat_le_cardinal_rel*: $nat \le i \Longrightarrow M(i) ==> nat \le |i|^M$
⟨*proof*⟩

lemma *n_lesspoll_rel_nat*: $n \in nat ==> n \prec^M nat$
⟨*proof*⟩

lemma *cons_lepoll_rel_cong*:
$[| A \lesssim^M B;\ \ b \notin B;\ M(A);\ M(B);\ M(b);\ M(a)\ |] ==> cons(a,A) \lesssim^M cons(b,B)$
⟨*proof*⟩

lemma *cons_eqpoll_rel_cong*:
$[| A \approx^M B;\ \ a \notin A;\ \ b \notin B;\ \ M(A);\ M(B);\ M(a)\ ;\ M(b)\ |] ==> cons(a,A) \approx^M cons(b,B)$
⟨*proof*⟩

lemma *cons_lepoll_rel_cons_iff*:
$[| a \notin A;\ \ b \notin B;\ M(a);\ M(A);\ M(b);\ M(B)\ |] ==> cons(a,A) \lesssim^M cons(b,B) \longleftrightarrow\ A \lesssim^M B$
⟨*proof*⟩

lemma *cons_eqpoll_rel_cons_iff*:
$[| a \notin A;\ \ b \notin B;\ M(a);\ M(A);\ M(b);\ M(B)\ |] ==> cons(a,A) \approx^M cons(b,B) \longleftrightarrow\ A \approx^M B$
⟨*proof*⟩

lemma *singleton_eqpoll_rel_1*: $M(a) \Longrightarrow \{a\} \approx^M 1$
⟨*proof*⟩

lemma *cardinal_rel_singleton*: $M(a) \Longrightarrow |\{a\}|^M = 1$
⟨*proof*⟩

lemma *not_0_is_lepoll_rel_1*: $A \neq 0 ==> M(A) \Longrightarrow 1 \lesssim^M A$
⟨*proof*⟩

lemma *succ_eqpoll_rel_cong*: $A \approx^M B \Longrightarrow M(A) \Longrightarrow M(B) ==> succ(A) \approx^M succ(B)$
⟨*proof*⟩

The next result was not straightforward to port, and even a different statement was needed.

lemma *sum_bij_rel*:

[| *f* ∈ *bij*^{M}(*A*,*C*); *g* ∈ *bij*^{M}(*B*,*D*); *M*(*f*); *M*(*A*); *M*(*C*); *M*(*g*); *M*(*B*); *M*(*D*)|]
==> (λ*z*∈*A*+*B*. *case*(%*x*. *Inl*(*f*‘*x*), %*y*. *Inr*(*g*‘*y*), *z*)) ∈ *bij*^{M}(*A*+*B*, *C*+*D*)
⟨*proof*⟩

lemma *sum_bij_rel′*:
assumes *f* ∈ *bij*^{M}(*A*,*C*) *g* ∈ *bij*^{M}(*B*,*D*) *M*(*f*)
M(*A*) *M*(*C*) *M*(*g*) *M*(*B*) *M*(*D*)
shows
(λ*z*∈*A*+*B*. *case*(λ*x*. *Inl*(*f*‘*x*), λ*y*. *Inr*(*g*‘*y*), *z*)) ∈ *bij*(*A*+*B*, *C*+*D*)
M(λ*z*∈*A*+*B*. *case*(λ*x*. *Inl*(*f*‘*x*), λ*y*. *Inr*(*g*‘*y*), *z*))
⟨*proof*⟩

lemma *sum_eqpoll_rel_cong*:
assumes *A* ≈^{M} *C* *B* ≈^{M} *D* *M*(*A*) *M*(*C*) *M*(*B*) *M*(*D*)
shows *A*+*B* ≈^{M} *C*+*D*
⟨*proof*⟩

lemma *prod_bij_rel′*:
assumes *f* ∈ *bij*^{M}(*A*,*C*) *g* ∈ *bij*^{M}(*B*,*D*) *M*(*f*)
M(*A*) *M*(*C*) *M*(*g*) *M*(*B*) *M*(*D*)
shows
(λ<*x*,*y*>∈*A*∗*B*. <*f*‘*x*, *g*‘*y*>) ∈ *bij*(*A*∗*B*, *C*∗*D*)
M(λ<*x*,*y*>∈*A*∗*B*. <*f*‘*x*, *g*‘*y*>)
⟨*proof*⟩

lemma *prod_eqpoll_rel_cong*:
assumes *A* ≈^{M} *C* *B* ≈^{M} *D* *M*(*A*) *M*(*C*) *M*(*B*) *M*(*D*)
shows *A*×*B* ≈^{M} *C*×*D*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *Diff_sing_lepoll_rel*:
[| *a* ∈ *A*; *A* ≲^{M} *succ*(*n*); *M*(*a*); *M*(*A*); *M*(*n*) |] ==> *A* - {*a*} ≲^{M} *n*
⟨*proof*⟩

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*⟩

lemma *Diff_sing_eqpoll_rel*: [| *a* ∈ *A*; *A* ≈^{M} *succ*(*n*); *M*(*a*); *M*(*A*); *M*(*n*) |] ==> *A* - {*a*} ≈^{M} *n*
⟨*proof*⟩

lemma *lepoll_rel_1_is_sing*: [| $A \lesssim^M 1$; $a \in A$;$M(a)$; $M(A)$ |] ==> $A = \{a\}$
⟨*proof*⟩

lemma *Un_lepoll_rel_sum*: $M(A) \Longrightarrow M(B) \Longrightarrow A \cup B \lesssim^M A+B$
⟨*proof*⟩

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*$(X \cup Y, T)$
⟨*proof*⟩

lemma *disj_Un_eqpoll_rel_sum*: $M(A) \Longrightarrow M(B) \Longrightarrow A \cap B = 0 \Longrightarrow A \cup B \approx^M A + B$
⟨*proof*⟩

lemma *eqpoll_rel_imp_Finite_rel_iff*: $A \approx^M B$ ==> $M(A) \Longrightarrow M(B) \Longrightarrow$ *Finite_rel*$(M,A) \longleftrightarrow$ *Finite_rel*(M,B)
⟨*proof*⟩

lemma *Finite_abs*[*simp*]: **assumes** $M(A)$ **shows** *Finite_rel*$(M,A) \longleftrightarrow$ *Finite*(A)
⟨*proof*⟩

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*⟩

lemma *lesspoll_rel_nat_is_Finite_rel*:
$A \prec^M nat \Longrightarrow M(A) \Longrightarrow$ *Finite_rel*(M,A)
⟨*proof*⟩

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*⟩

lemma *succ_lepoll_rel_imp_not_empty*: $succ(x) \lesssim^M y$ ==> $M(x) \Longrightarrow M(y) \Longrightarrow y \neq 0$
⟨*proof*⟩

lemma *eqpoll_rel_succ_imp_not_empty*: $x \approx^M succ(n)$ ==> $M(x) \Longrightarrow M(n) \Longrightarrow x \neq 0$
⟨*proof*⟩

lemma *Finite_subset_closed*:
  assumes *Finite*(*B*) *B*⊆*A* *M*(*A*)
  shows *M*(*B*)
⟨*proof*⟩

lemma *Finite_Pow_abs*:
  assumes *Finite*(*A*) *M*(*A*)
  shows *Pow*(*A*) = *Pow_rel*(*M*,*A*)
  ⟨*proof*⟩

lemma *Finite_Pow_rel*:
  assumes *Finite*(*A*) *M*(*A*)
  shows *Finite*(*Pow_rel*(*M*,*A*))
  ⟨*proof*⟩

lemma *Pow_rel_0* [*simp*]: *Pow_rel*(*M*,*0*) = {*0*}
  ⟨*proof*⟩

end

end

23 Relative, Choice-less Cardinal Arithmetic

theory *CardinalArith_Relative*
  imports
    Cardinal_Relative

begin

⟨*ML*⟩

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

⟨*ML*⟩

definition
  is_csquare_lam :: [*i*⇒*o*,*i*,*i*]⇒*o* **where**
  is_csquare_lam(*M*,*K*,*l*) ≡ ∃ *K2*[*M*]. *cartprod*(*M*,*K*,*K*,*K2*) ∧
        is_lambda(*M*,*K2*,*is_csquare_lam_body*(*M*),*l*)

definition *jump_cardinal_body* :: [*i*⇒*o*,*i*] ⇒ *i* **where**

jump_cardinal_body(*M*,*X*) ≡
{*z* . *r* ∈ *Pow*^{M}(*X* × *X*), *M*(*z*) ∧ *M*(*r*) ∧ *well_ord*(*X*, *r*) ∧ *z* = *ordertype*(*X*, *r*)}

lemma (**in** *M_cardinals*) *csquare_lam_closed*[*intro*,*simp*]: *M*(*K*) ⟹ *M*(*csquare_lam*(*K*))
⟨*proof*⟩

locale *M_pre_cardinal_arith* = *M_cardinals* +
assumes
ord_iso_separation: *M*(*A*) ⟹ *M*(*r*) ⟹ *M*(*s*) ⟹
separation(*M*, λ*f*. ∀ *x*∈*A*. ∀ *y*∈*A*. ⟨*x*, *y*⟩ ∈ *r* ⟷ ⟨*f* ‘ *x*, *f* ‘ *y*⟩ ∈ *s*)
and
wfrec_pred_replacement:*M*(*A*) ⟹ *M*(*r*) ⟹
wfrec_replacement(*M*, λ*x f z*. *z* = *f* “ *Order*.*pred*(*A*, *x*, *r*), *r*)

locale *M_cardinal_arith* = *M_pre_cardinal_arith* +
assumes
ordertype_replacement :
M(*X*) ⟹ *strong_replacement*(*M*,λ *x z* . *M*(*z*) ∧ *M*(*x*) ∧ *x*∈*Pow_rel*(*M*,*X*×*X*) ∧ *well_ord*(*X*, *x*) ∧ *z*=*ordertype*(*X*,*x*))
and
strong_replacement_jc_body :
strong_replacement(*M*,λ *x z* . *M*(*z*) ∧ *M*(*x*) ∧ *z* = *jump_cardinal_body*(*M*,*x*))
and
surj_imp_inj_replacement:
M(*f*) ⟹ *M*(*x*) ⟹ *strong_replacement*(*M*, λ*y z*. *y* ∈ *f* -“ {*x*} ∧ *z* = {⟨*x*, *y*⟩})
M(*f*) ⟹ *strong_replacement*(*M*, λ*x z*. *z* = *Sigfun*(*x*, λ*y*. *f* -“ {*y*}))
M(*f*) ⟹ *strong_replacement*(*M*, λ*x y*. *y* = *f* -“ {*x*})
M(*f*) ⟹ *M*(*r*) ⟹ *strong_replacement*(*M*, λ*x y*. *y* = ⟨*x*, *minimum*(*r*, *f* -“ {*x*})⟩)

⟨*ML*⟩

lemma (**in** *M_trivial*) *rmultP_abs* [*absolut*]: ⟦ *M*(*r*); *M*(*s*); *M*(*z*) ⟧ ⟹ *is_rmultP*(*M*,*s*,*r*,*z*) ⟷
(∃ *x*′ *y*′ *x y*. *z* = ⟨⟨*x*′, *y*′⟩, *x*, *y*⟩ ∧ (⟨*x*′, *x*⟩ ∈ *r* ∨ *x*′ = *x* ∧ ⟨*y*′, *y*⟩ ∈ *s*))
⟨*proof*⟩

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*)
  ⟨*proof*⟩

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*)
  ⟨*proof*⟩

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*)>
  ⟨*proof*⟩

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*)⟩)
  ⟨*proof*⟩

lemma *csquare_lam_eq_lam*:*csquare_lam*(*K*) = (λ*z*∈*K*×*K*. <*fst*(*z*) ∪ *snd*(*z*), *fst*(*z*), *snd*(*z*)>)
⟨*proof*⟩

end

context *M_pre_cardinal_arith*
begin

lemma *csquare_rel_closed*[*intro*,*simp*]: *M*(*K*) ⟹ *M*(*csquare_rel*(*K*))
  ⟨*proof*⟩

lemma *csquare_rel_abs*[*absolut*]: ⟦ *M*(*K*); *M*(*cs*)⟧ ⟹
    is_csquare_rel(*M*,*K*,*cs*) ⟷ *cs* = *csquare_rel*(*K*)
  ⟨*proof*⟩

end

⟨*ML*⟩

abbreviation
  csucc_r :: [*i*,*i*⇒*o*] ⇒ *i* (‹'(_$^{+}$')-›) **where**
  csucc_r(*x*,*M*) ≡ *csucc_rel*(*M*,*x*)

abbreviation
  csucc_r_set :: [*i*,*i*] ⇒ *i* (‹'(_$^{+}$')-›) **where**
  csucc_r_set(*x*,*M*) ≡ *csucc_rel*(##*M*,*x*)

context *M_Perm*
begin

⟨*ML*⟩
  ⟨*proof*⟩

⟨*ML*⟩
  ⟨*proof*⟩

end

notation *csucc_rel* (‹*csucc*–'(_')›)

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*⟩

lemma *in_Card_imp_lesspoll*: $[\!|\ Card^M(K);\ b \in K;\ M(K);\ M(b)\ |\!] ==> b \prec^M K$
⟨*proof*⟩

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*⟩

lemma *cadd_rel_commute*: $M(i) \Longrightarrow M(j) \Longrightarrow i \oplus^M j = j \oplus^M i$
⟨*proof*⟩

23.1.2 Cardinal addition is associative

lemma *sum_assoc_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (A+B)+C \approx^M A+(B+C)$
  ⟨*proof*⟩

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*⟩

23.1.3 0 is the identity for addition

lemma *case_id_eq*: $x \in sum(A,B) \Longrightarrow case(\lambda z . z, \lambda z. z ,x) = snd(x)$
  ⟨*proof*⟩

lemma *lam_case_id*: $(\lambda z \in 0 + A. case(\lambda x. x, \lambda y. y, z)) = (\lambda z \in 0 + A . snd(z))$
  ⟨*proof*⟩

lemma *sum_0_eqpoll_rel*: $M(A) \Longrightarrow 0+A \approx^M A$
  ⟨*proof*⟩

lemma *cadd_rel_0* [*simp*]: $Card^M(K) \Longrightarrow M(K) \Longrightarrow 0 \oplus^M K = K$
⟨*proof*⟩

23.1.4 Addition by another cardinal

lemma *sum_lepoll_rel_self*: $M(A) \Longrightarrow M(B) \Longrightarrow A \lesssim^M A+B$
⟨*proof*⟩

lemma *cadd_rel_le_self*:
  assumes K: $Card^M(K)$ **and** L: $Ord(L)$ **and**
    types: $M(K)$ $M(L)$
  shows $K \le (K \oplus^M L)$
⟨*proof*⟩

23.1.5 Monotonicity of addition

lemma *sum_lepoll_rel_mono*:
    $[| A \lesssim^M C; B \lesssim^M D; M(A); M(B); M(C); M(D) |] ==> A + B \lesssim^M C + D$
⟨*proof*⟩

lemma *cadd_rel_le_mono*:
    $[| K' \le K; L' \le L; M(K'); M(K); M(L'); M(L) |] ==> (K' \oplus^M L') \le (K \oplus^M L)$
⟨*proof*⟩

23.1.6 Addition of finite cardinals is "ordinary" addition

lemma *sum_succ_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow succ(A)+B \approx^M succ(A+B)$
⟨*proof*⟩

lemma *cadd_succ_lemma*:
  assumes $Ord(m)$ $Ord(n)$ **and**
    types: $M(m)$ $M(n)$
  shows $succ(m) \oplus^M n = |succ(m \oplus^M n)|^M$
  ⟨*proof*⟩

lemma *nat_cadd_rel_eq_add*:
  assumes *m*: $m \in nat$ **and** [*simp*]: $n \in nat$ **shows**$m \oplus^M n = m \#+ n$
  ⟨*proof*⟩

23.2 Cardinal multiplication

23.2.1 Cardinal multiplication is commutative

lemma *prod_commute_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow A{*}B \approx^M B{*}A$
⟨*proof*⟩

lemma *cmult_rel_commute*: $M(i) \Longrightarrow M(j) \Longrightarrow i \otimes^M j = j \otimes^M i$
⟨*proof*⟩

23.2.2 Cardinal multiplication is associative

lemma *prod_assoc_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (A{*}B){*}C \approx^M A{*}(B{*}C)$
⟨*proof*⟩

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*⟩

23.2.3 Cardinal multiplication distributes over addition

lemma *sum_prod_distrib_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (A{+}B){*}C \approx^M (A{*}C){+}(B{*}C)$
⟨*proof*⟩

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*⟩

23.2.4 Multiplication by 0 yields 0

lemma *prod_0_eqpoll_rel*: $M(A) \Longrightarrow 0{*}A \approx^M 0$
⟨*proof*⟩

lemma *cmult_rel_0* [*simp*]: $M(i) \Longrightarrow 0 \otimes^M i = 0$
⟨*proof*⟩

23.2.5 1 is the identity for multiplication

lemma *prod_singleton_eqpoll_rel*: $M(x) \Longrightarrow M(A) \Longrightarrow \{x\}{*}A \approx^M A$
⟨*proof*⟩

lemma *cmult_rel_1* [*simp*]: $Card^M(K) \Longrightarrow M(K) \Longrightarrow 1 \otimes^M K = K$
⟨*proof*⟩

23.3 Some inequalities for multiplication

lemma *prod_square_lepoll_rel*: $M(A) \Longrightarrow A \lesssim^M A{*}A$
⟨*proof*⟩

lemma *cmult_rel_square_le*: $Card^M(K) \Longrightarrow M(K) \Longrightarrow K \le K \otimes^M K$
⟨*proof*⟩

23.3.1 Multiplication by a non-zero cardinal

lemma *prod_lepoll_rel_self*: $b \in B \Longrightarrow M(b) \Longrightarrow M(B) \Longrightarrow M(A) \Longrightarrow A \lesssim^M A{*}B$
⟨*proof*⟩

lemma *cmult_rel_le_self*:
$[| \; Card^M(K); \; Ord(L); \; 0<L; \; M(K);M(L) \; |] ==> K \le (K \otimes^M L)$
⟨*proof*⟩

23.3.2 Monotonicity of multiplication

lemma *prod_lepoll_rel_mono*:
$[| \; A \lesssim^M C; \; B \lesssim^M D; \; M(A); \; M(B); \; M(C); \; M(D)|] ==> A * B \lesssim^M C * D$
⟨*proof*⟩

lemma *cmult_rel_le_mono*:
$[| \; K' \le K; \; L' \le L; M(K'); M(K); M(L'); M(L) \; |] ==> (K' \otimes^M L') \le (K \otimes^M L)$
⟨*proof*⟩

23.4 Multiplication of finite cardinals is "ordinary" multiplication

lemma *prod_succ_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow succ(A){*}B \approx^M B + A{*}B$
⟨*proof*⟩

lemma *cmult_rel_succ_lemma*:
[| $Ord(m)$; $Ord(n)$; $M(m)$; $M(n)$ |] ==> $succ(m) \otimes^M n = n \oplus^M (m \otimes^M n)$
⟨*proof*⟩

lemma *nat_cmult_rel_eq_mult*: [| $m \in nat$; $n \in nat$ |] ==> $m \otimes^M n = m\#*n$
⟨*proof*⟩

lemma *cmult_rel_2*: $Card^M(n) \Longrightarrow M(n) \Longrightarrow 2 \otimes^M n = n \oplus^M n$
⟨*proof*⟩

lemma *sum_lepoll_rel_prod*:
assumes C: $2 \lesssim^M C$ **and**
types:$M(C)$ $M(B)$
shows $B+B \lesssim^M C*B$
⟨*proof*⟩

lemma *lepoll_imp_sum_lepoll_prod*: [| $A \lesssim^M B$; $2 \lesssim^M A$; $M(A)$;$M(B)$ |] ==> $A+B \lesssim^M A*B$
⟨*proof*⟩

end

23.5 Infinite Cardinals are Limit Ordinals

context *M_pre_cardinal_arith*
begin

lemma *nat_cons_lepoll_rel*: $nat \lesssim^M A \Longrightarrow M(A) \Longrightarrow M(u)$ ==> $cons(u,A) \lesssim^M A$
⟨*proof*⟩

lemma *nat_cons_eqpoll_rel*: $nat \lesssim^M A$ ==> $M(A) \Longrightarrow M(u) \Longrightarrow cons(u,A) \approx^M A$
⟨*proof*⟩

lemma *nat_succ_eqpoll_rel*: $nat \subseteq A$ ==> $M(A) \Longrightarrow succ(A) \approx^M A$
⟨*proof*⟩

lemma *InfCard_rel_nat*: $InfCard^M(nat)$
⟨*proof*⟩

lemma *InfCard_rel_is_Card_rel*: $M(K) \Longrightarrow InfCard^M(K) \Longrightarrow Card^M(K)$
⟨*proof*⟩

lemma *InfCard_rel_Un*:
[| $InfCard^M(K)$; $Card^M(L)$; $M(K)$; $M(L)$ |] ==> $InfCard^M(K \cup L)$
⟨*proof*⟩

lemma *InfCard_rel_is_Limit*: $InfCard^M(K)$ ==> $M(K) \Longrightarrow Limit(K)$
⟨*proof*⟩

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*) ⟷ *i* = *ordertype*(*A*,*r*)
⟨*proof*⟩

lemma (**in** *M_ordertype*) *ordertype_closed*[*intro*,*simp*]: ⟦ *wellordered*(*M*,*A*,*r*);*M*(*A*);*M*(*r*)⟧ ⟹ *M*(*ordertype*(*A*,*r*))
⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_trivial*) *is_transitive_iff_transitive_rel*:
M(*A*)⟹ *M*(*r*) ⟹ *transitive_rel*(*M*, *A*, *r*) ⟷ *is_transitive*(*M*,*A*, *r*)
⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_trivial*) *is_linear_iff_linear_rel*:
M(*A*)⟹ *M*(*r*) ⟹ *is_linear*(*M*,*A*, *r*) ⟷ *linear_rel*(*M*, *A*, *r*)
⟨*proof*⟩

⟨*ML*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

⟨*ML*⟩

context *M_pre_cardinal_arith*
begin

⟨*ML*⟩
  ⟨*proof*⟩

⟨*ML*⟩
  ⟨*proof*⟩

end

⟨*ML*⟩

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*)
  ⟨*proof*⟩
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*))
  ⟨*proof*⟩

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*)
  ⟨*proof*⟩

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*)
  ⟨*proof*⟩

lemma *trans_on_iff_trans*: *trans*[*A*](*r*) ⟷ *trans*(*r* ∩ *A*×*A*)
  ⟨*proof*⟩

lemma *trans_on_subset*: *trans*[*A*](*r*) ⟹ *B* ⊆ *A* ⟹ *trans*[*B*](*r*)
  ⟨*proof*⟩

lemma *relation_Int*: *relation*(*r* ∩ *B*×*B*)
  ⟨*proof*⟩

Discipline for *ordermap*

⟨*ML*⟩

context *M_pre_cardinal_arith*
begin

lemma *wfrec_on_pred_eq*:
  assumes *r* ∈ *Pow*(*A*×*A*) *M*(*A*) *M*(*r*)
  shows *wfrec*[*A*](*r*, *x*, λ*x f*. *f* ‘‘ *Order.pred*(*A*, *x*, *r*)) = *wfrec*(*r*, *x*, λ*x f*. *f* ‘‘ *Order.pred*(*A*, *x*, *r*))
⟨*proof*⟩

lemma *wfrec_on_pred_closed*:
  assumes *wf*[*A*](*r*) *trans*[*A*](*r*) *r* ∈ *Pow*(*A*×*A*) *M*(*A*) *M*(*r*) *x* ∈ *A*
  shows *M*(*wfrec*(*r*, *x*, λ*x f*. *f* ‘‘ *Order.pred*(*A*, *x*, *r*)))
⟨*proof*⟩

lemma *wfrec_on_pred_closed'*:
  assumes *wf*[*A*](*r*) *trans*[*A*](*r*) *r* ∈ *Pow*(*A*×*A*) *M*(*A*) *M*(*r*) *x* ∈ *A*
  shows *M*(*wfrec*[*A*](*r*, *x*, λ*x f*. *f* ‘‘ *Order.pred*(*A*, *x*, *r*)))
  ⟨*proof*⟩

lemma *ordermap_rel_closed'*:

assumes *wf*[*A*](*r*) *trans*[*A*](*r*) *r* ∈ *Pow*(*A*×*A*) *M*(*A*) *M*(*r*)
shows *M*(*ordermap_rel*(*M*, *A*, *r*))
⟨*proof*⟩

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*))
⟨*proof*⟩

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*⟩

end

⟨*ML*⟩

Discipline for *ordertype*

⟨*ML*⟩

context *M_pre_cardinal_arith*
begin

lemma *is_ordertype_iff*:
assumes *r* ∈ *Pow*(*A*×*A*) *wf*[*A*](*r*) *trans*[*A*](*r*)
shows *M*(*A*) ⟹ *M*(*r*) ⟹ *M*(*res*) ⟹ *is_ordertype*(*M*, *A*, *r*, *res*) ⟷ *res* = *ordertype_rel*(*M*, *A*, *r*)
⟨*proof*⟩

lemma *is_ordertype_iff'*:
assumes *r* ∈ *Pow_rel*(*M*,*A*×*A*) *well_ord*(*A*,*r*)
shows *M*(*A*) ⟹ *M*(*r*) ⟹ *M*(*res*) ⟹ *is_ordertype*(*M*, *A*, *r*, *res*) ⟷ *res* = *ordertype_rel*(*M*, *A*, *r*)
⟨*proof*⟩

lemma *is_ordertype_iff''*:
assumes *well_ord*(*A*,*r*) *r*⊆*A*×*A*
shows *M*(*A*) ⟹ *M*(*r*) ⟹ *M*(*res*) ⟹ *is_ordertype*(*M*, *A*, *r*, *res*) ⟷ *res* = *ordertype_rel*(*M*, *A*, *r*)
⟨*proof*⟩

end

⟨*ML*⟩
definition
jump_cardinal' :: *i*⇒*i* **where**
jump_cardinal'(*K*) ≡

$\bigcup X{\in}Pow(K).\ \{z.\ r \in Pow(X{*}X),\ well\_ord(X,r)\ \&\ z = ordertype(X,r)\}$

⟨*ML*⟩
definition *jump_cardinal_body′* **where**
jump_cardinal_body′(X) ≡ {z . r ∈ *Pow*(X × X), *well_ord*(X, r) ∧ z = *ordertype*(X, r)}

⟨*ML*⟩

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))
⟨*proof*⟩

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))
⟨*proof*⟩

lemma *ordertype_rel_abs*:
assumes *wellordered*(M,X,r) M(X) M(r)
shows *ordertype_rel*(M,X,r) = *ordertype*(X,r)
⟨*proof*⟩

lemma *univalent_aux1*: M(X) ⟹ *univalent*(M,*Pow_rel*(M,X×X),
λr z. M(z) ∧ M(r) ∧ r∈*Pow_rel*(M,X×X) ∧ *is_well_ord*(M, X, r) ∧ *is_ordertype*(M, X, r, z))
⟨*proof*⟩

lemma *jump_cardinal_body_eq* :
M(X) ⟹ *jump_cardinal_body*(M,X) = *jump_cardinal_body′_rel*(M,X)
⟨*proof*⟩

end

context *M_cardinal_arith*
begin
lemma *jump_cardinal_closed_aux1*:
assumes M(X)
shows
M(*jump_cardinal_body*(M,X))
⟨*proof*⟩

lemma *univalent_jc_body*: M(X) ⟹ *univalent*(M,X,λ x z . M(z) ∧ M(x) ∧ z = *jump_cardinal_body*(M,x))
⟨*proof*⟩

lemma *jump_cardinal_body_closed*:
  assumes $M(K)$
  shows $M(\{a . X \in Pow^M(K), M(a) \wedge M(X) \wedge a = jump\_cardinal\_body(M,X)\})$
  ⟨*proof*⟩

⟨*ML*⟩
  ⟨*proof*⟩

⟨*ML*⟩
⟨*proof*⟩

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*⟩

lemma *ordermap_eqpoll_pred*:
  [| *well_ord*(A,r); $x \in A$; $M(A)$;$M(r)$;$M(x)$|] ==> *ordermap*(A,r)'$x \approx^M$ *Order.pred*(A,x,r)
⟨*proof*⟩

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 $|ordermap(K \times K, csquare\_rel(K))\ ‘\ \langle x,y\rangle|^M \leq |succ(succ(x \cup y))|^M \otimes^M |succ(succ(x \cup y))|^M$
  ⟨*proof*⟩

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*⟩

lemma *InfCard_rel_csquare_eq*:
  assumes *IK*: $InfCard^M(K)$ **and**
  types: $M(K)$
  shows $K \otimes^M K = K$
⟨*proof*⟩

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 $A \times A \approx^M A$
⟨*proof*⟩

lemma *InfCard_rel_square_eqpoll*:
  assumes $InfCard^M(K)$ **and** *types*:$M(K)$ **shows** $K \times K \approx^M K$
  ⟨*proof*⟩

lemma *Inf_Card_rel_is_InfCard_rel*: [| $Card^M(i)$; ~ *Finite_rel*(*M*,*i*) ; $M(i)$ |] ==> $InfCard^M(i)$
  ⟨*proof*⟩

23.5.1 Toward's Kunen's Corollary 10.13 (1)

lemma *InfCard_rel_le_cmult_rel_eq*: [| $InfCard^M(K)$; $L \leq K$; $0<L$; $M(K)$; $M(L)$ |] ==> $K \otimes^M L = K$
⟨*proof*⟩

lemma *InfCard_rel_cmult_rel_eq*: [| $InfCard^M(K)$; $InfCard^M(L)$; $M(K)$; $M(L)$ |] ==> $K \otimes^M L = K \cup L$
⟨*proof*⟩

lemma *InfCard_rel_cdouble_eq*: $InfCard^M(K) \Longrightarrow M(K) \Longrightarrow K \oplus^M K = K$
⟨*proof*⟩

lemma *InfCard_rel_le_cadd_rel_eq*: [| $InfCard^M(K)$; $L \leq K$; $M(K)$; $M(L)$|] ==> $K \oplus^M L = K$
⟨*proof*⟩

lemma *InfCard_rel_cadd_rel_eq*: [| $InfCard^M(K)$; $InfCard^M(L)$; $M(K)$; $M(L)$ |] ==> $K \oplus^M L = K \cup L$
⟨*proof*⟩

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*⟩

lemma *ordertype_restr_eq* :
  assumes $well\_ord(X,r)$
  shows $ordertype(X, r) = ordertype(X, r \cap X{\times}X)$
  ⟨*proof*⟩

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*⟩

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*⟩

notation *jump_cardinal′_rel* (‹*jump′_cardinal′_rel*›)

lemma *Ord_jump_cardinal_rel*: $M(K) \Longrightarrow Ord(jump\_cardinal\_rel(M,K))$
⟨*proof*⟩

declare *conj_cong* [*cong del*]
— incompatible with some of the proofs of the original theory

lemma *jump_cardinal_rel_iff_old*:
    $M(i) \Longrightarrow M(K) \Longrightarrow i \in jump\_cardinal\_rel(M,K) \longleftrightarrow$
    $(\exists r[M]. \exists X[M]. r \subseteq K{*}K \,\&\, X \subseteq K \,\&\, well\_ord(X,r) \,\&\, i = ordertype(X,r))$
  ⟨*proof*⟩

lemma *K_lt_jump_cardinal_rel*: $Ord(K) ==> M(K) \Longrightarrow K < jump\_cardinal\_rel(M,K)$
⟨*proof*⟩

lemma *Card_rel_jump_cardinal_rel_lemma*:

[| *well_ord*(*X*,*r*); *r* ⊆ *K* ∗ *K*; *X* ⊆ *K*;
f ∈ *bij*(*ordertype*(*X*,*r*), *jump_cardinal_rel*(*M*,*K*));
M(*X*); *M*(*r*); *M*(*K*); *M*(*f*) |]
==> *jump_cardinal_rel*(*M*,*K*) ∈ *jump_cardinal_rel*(*M*,*K*)
⟨*proof*⟩

lemma *Card_rel_jump_cardinal_rel*: *M*(*K*) ⟹ *Card_rel*(*M*,*jump_cardinal_rel*(*M*,*K*))
⟨*proof*⟩

23.7 Basic Properties of Successor Cardinals

lemma *csucc_rel_basic*: *Ord*(*K*) ==> *M*(*K*) ⟹ *Card_rel*(*M*,*csucc_rel*(*M*,*K*))
& *K* < *csucc_rel*(*M*,*K*)
⟨*proof*⟩

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*) ⟹ *0* < *csucc_rel*(*M*,*K*)
⟨*proof*⟩

lemma *csucc_rel_le*: [| *Card_rel*(*M*,*L*); *K*<*L*; *M*(*K*); *M*(*L*) |] ==> *csucc_rel*(*M*,*K*)
≤ *L*
⟨*proof*⟩

lemma *lt_csucc_rel_iff*: [| *Ord*(*i*); *Card_rel*(*M*,*K*); *M*(*K*); *M*(*i*)|] ==> *i* <
csucc_rel(*M*,*K*) ⟷ $|i|^M$ ≤ *K*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *InfCard_rel_csucc_rel*: *InfCard_rel*(*M*,*K*) ⟹ *M*(*K*) ==> *InfCard_rel*(*M*,*csucc_rel*(*M*,*K*))
⟨*proof*⟩

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** $m + n \approx^M m$ #+ *n*
⟨*proof*⟩

lemma *Ord_nat_subset_into_Card_rel*: [| *Ord*(*i*); *i* ⊆ *nat* |] ==> $Card^M(i)$
⟨*proof*⟩

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*)))

⟨*ML*⟩

definition
  Aleph′ :: *i* => *i* **where**
  Aleph′(*a*) == *transrec*(*a*,λ*i* *r*. *HAleph*(*i*,*r*))

⟨*ML*⟩

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*)
  ⟨*proof*⟩

⟨*ML*⟩

notation *is_Aleph_fm* (‹·ℵ′(_′) *is* _·›)

lemma *is_Aleph_fm_type* [*TC*]: *a* ∈ *nat* ⟹ *c* ∈ *nat* ⟹ *is_Aleph_fm*(*a*, *c*) ∈ *formula*
  ⟨*proof*⟩

lemma *sats_is_Aleph_fm*:
  assumes *f*∈*nat* *r*∈*nat* *env* ∈ *list*(*A*) *0*∈*A* *f* < *length*(*env*) *r*< *length*(*env*)
  shows *is_Aleph*(##*A*, *nth*(*f*, *env*), *nth*(*r*, *env*)) ⟷ *A*, *env* ⊨ *is_Aleph_fm*(*f*,*r*)
  ⟨*proof*⟩

lemma *is_Aleph_iff_sats* [*iff_sats*]:
  assumes
    nth(*f*, *env*) = *fa* *nth*(*r*, *env*) = *ra* *f* < *length*(*env*) *r*< *length*(*env*)

f ∈ *nat* *r* ∈ *nat* *env* ∈ *list*(*A*) *0*∈*A*
shows *is_Aleph*(##*A*,*fa*,*ra*) ⟷ *A*, *env* ⊨ *is_Aleph_fm*(*f*,*r*)
⟨*proof*⟩

⟨*ML*⟩

context *M_cardinal_arith_jump*
begin

lemma *is_Limit_iff*:
assumes $M(a)$
shows *is_Limit*(M,a) ⟷ *Limit*(a)
⟨*proof*⟩

end

lemma *HAleph_eq_Aleph_recursive*:
Ord(i) ⟹ *HAleph*(i,r) = (*if* i = *0* *then* *nat*
else *if* ∃ j. i = *succ*(j) *then* *csucc*(r ‘ (*THE* j. i = *succ*(j))) *else* ⋃j<i.
r ‘ j)
⟨*proof*⟩

lemma *Aleph′_eq_Aleph*: *Ord*(a) ⟹ *Aleph′*(a) = *Aleph*(a)
⟨*proof*⟩

⟨*ML*⟩

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))
⟨*proof*⟩

lemma *succ_mem_Limit*: *Limit*(j) ⟹ i ∈ j ⟹ *succ*(i) ∈ j
⟨*proof*⟩

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*⟩

lemma *is_HAleph_zero*:
  assumes $M(f)$
  shows *is_HAleph*$(M,0,f,res) \longleftrightarrow res = nat$
  ⟨*proof*⟩

lemma *is_HAleph_succ*:
  assumes $M(f)$ $M(x)$ $Ord(x)$ $M(res)$
  shows *is_HAleph*$(M,succ(x),f,res) \longleftrightarrow res = csucc\_rel(M,f`(\bigcup succ(x)))$
  ⟨*proof*⟩

lemma *is_HAleph_limit*:
  assumes $M(f)$ $M(x)$ $Limit(x)$ $M(res)$
  shows *is_HAleph*$(M,x,f,res) \longleftrightarrow res = (\bigcup\{y . i\in x ,M(i) \wedge M(y) \wedge y = f`i\})$
⟨*proof*⟩

lemma *is_HAleph_iff*:
  assumes $M(a)$ $M(f)$ $M(res)$
  shows *is_HAleph*$(M, a, f, res) \longleftrightarrow res = HAleph\_rel(M, a, f)$
⟨*proof*⟩

lemma *HAleph_rel_closed* [*intro*,*simp*]:
  assumes $function(f)$ $M(a)$ $M(f)$
  shows $M(HAleph\_rel(M,a,f))$
  ⟨*proof*⟩

lemma *Aleph_rel_closed*[*intro*, *simp*]:
  assumes $Ord(a)$ $M(a)$
  shows $M(Aleph\_rel(M,a))$
⟨*proof*⟩

lemma *Aleph_rel_zero*: $\aleph_0{}^M = nat$
  ⟨*proof*⟩

lemma *Aleph_rel_succ*: $Ord(\alpha) \Longrightarrow M(\alpha) \Longrightarrow \aleph_{succ(\alpha)}{}^M = (\aleph_\alpha{}^{M+})^M$
  ⟨*proof*⟩

lemma *Aleph_rel_limit*:
  assumes $Limit(\alpha)$ $M(\alpha)$
  shows $\aleph_\alpha{}^M = \bigcup\{\aleph_j{}^M . j \in \alpha\}$
⟨*proof*⟩

lemma *is_Aleph_iff*:
  assumes $Ord(a)$ $M(a)$ $M(res)$
  shows *is_Aleph*$(M, a, res) \longleftrightarrow res = \aleph_a{}^M$
⟨*proof*⟩

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*: $Limit(l) \Longrightarrow M(l) \Longrightarrow \aleph_l{}^M = (\bigcup i{<}l.\ \aleph_i{}^M)$
⟨*proof*⟩

lemma *Ord_Aleph_rel*:
assumes $Ord(a)$
shows $M(a) \Longrightarrow Ord(\aleph_a{}^M)$
⟨*proof*⟩

lemma *Card_rel_Aleph_rel* [*simp*, *intro*]:
assumes $Ord(a)$ **and** *types*: $M(a)$ **shows** $Card^M(\aleph_a{}^M)$
⟨*proof*⟩

lemma *Aleph_rel_increasing*:
assumes *ab*: $a < b$ **and** *types*: $M(a)$ $M(b)$
shows $\aleph_a{}^M < \aleph_b{}^M$
⟨*proof*⟩

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**
$Fn(\kappa,I,J) \equiv \bigcup\{(d{\rightarrow}J)\ ..\ d \in Pow(I),\ \ d{\prec}\kappa\}$

lemma *FnI*[*intro*]:
assumes $p : d \rightarrow J$ $d \subseteq I$ $d \prec \kappa$
shows $p \in Fn(\kappa,I,J)$
⟨*proof*⟩

lemma *FnD*[*dest*]:
  assumes $p \in Fn(\kappa,I,J)$
  shows $\exists\, d.\ p : d \to J \wedge d \subseteq I \wedge d \prec \kappa$
  ⟨*proof*⟩

lemma *Fn_is_function*: $p \in Fn(\kappa,I,J) \Longrightarrow function(p)$
  ⟨*proof*⟩

lemma *Fn_csucc*:
  assumes $Ord(\kappa)$
  shows $Fn(csucc(\kappa),I,J) = \bigcup\{(d{\to}J)\ ..\ d \in Pow(I),\ d{\lesssim}\kappa\}$
  ⟨*proof*⟩

lemma *Finite_imp_lesspoll_nat*:
  assumes $Finite(A)$
  shows $A \prec nat$
  ⟨*proof*⟩

lemma *Fn_nat_eq_FiniteFun*: $Fn(nat,I,J) = I$ -||> J
  ⟨*proof*⟩

definition
  $FnleR :: i \Rightarrow i \Rightarrow o$ (**infixl** ⟨⊇⟩ *50*) **where**
  $f \supseteq g \equiv g \subseteq f$

lemma *FnleR_iff_subset* [*iff*]: $f \supseteq g \longleftrightarrow g \subseteq f$
  ⟨*proof*⟩

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 $<p,q> \in Fnle(\kappa,I,J)$
  ⟨*proof*⟩

lemma *FnleD*[*dest*]:
  assumes $<p,q> \in Fnle(\kappa,I,J)$
  shows $p \in Fn(\kappa,I,J)\ q \in Fn(\kappa,I,J)\ p \supseteq q$
  ⟨*proof*⟩

locale *cohen_data* =
  fixes $\kappa\ I\ J::i$
  assumes *zero_lt_kappa*: $0<\kappa$

begin

lemmas *zero_lesspoll_kappa* = *zero_lesspoll*[*OF zero_lt_kappa*]

end

sublocale *cohen_data* ⊆ *forcing_notion Fn*(*κ*,*I*,*J*) *Fnle*(*κ*,*I*,*J*) *0*
⟨*proof*⟩

24.1 MOVE THIS to an appropriate place

definition
antichain :: *i*⇒*i*⇒*i*⇒*o* **where**
antichain(*P*,*leq*,*A*) ≡ *A*⊆*P* ∧ (∀ *p*∈*A*. ∀ *q*∈*A*.
p≠*q* ⟶ ¬*compat_in*(*P*,*leq*,*p*,*q*))
definition
ccc :: *i* ⇒ *i* ⇒ *o* **where**
ccc(*P*,*leq*) ≡ ∀ *A*. *antichain*(*P*,*leq*,*A*) ⟶ |*A*| ≤ *nat*

24.2 Combinatorial results on Cohen posets

context *cohen_data*
begin

lemma *restrict_eq_imp_compat*:
assumes *f* ∈ *Fn*(*nat*, *I*, *J*) *g* ∈ *Fn*(*nat*, *I*, *J*) *InfCard*(*nat*)
restrict(*f*, *domain*(*f*) ∩ *domain*(*g*)) = *restrict*(*g*, *domain*(*f*) ∩ *domain*(*g*))
shows *f* ∪ *g* ∈ *Fn*(*nat*, *I*, *J*)
⟨*proof*⟩

lemma *compat_imp_Un_is_function*:
assumes *G* ⊆ *Fn*(*κ*, *I*, *J*) ⋀*p q*. *p* ∈ *G* ⟹ *q* ∈ *G* ⟹ *compat*(*p*,*q*)
shows *function*(⋃ *G*)
⟨*proof*⟩

lemma *filter_subset_notion*: *filter*(*G*) ⟹ *G* ⊆ *Fn*(*κ*, *I*, *J*)
⟨*proof*⟩

lemma *Un_filter_is_function*: *filter*(*G*) ⟹ *function*(⋃ *G*)
⟨*proof*⟩

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*] ⇒ *i* (‹_^<_› [*100*,*1*]*100*) **where**
  $B^{<\alpha} \equiv \bigcup n\in\alpha.\ (n\to B)$

lemma *seqspaceI*[*intro*]: $n\in\alpha \Longrightarrow f{:}n\to B \Longrightarrow f\in B^{<\alpha}$
  ⟨*proof*⟩

lemma *seqspaceD*[*dest*]: $f\in B^{<\alpha} \Longrightarrow \exists\, n\in\alpha.\ f{:}n\to B$
  ⟨*proof*⟩

locale *M_seqspace* = *M_trancl* + *M_replacement* +
  assumes
   seqspace_replacement: *M*(*B*) ⟹ *strong_replacement*(*M*,λ*n z*. *n*∈*nat* ∧ *is_funspace*(*M*,*n*,*B*,*z*))
begin

lemma *seqspace_closed*:
  $M(B) \Longrightarrow M(B^{<\omega})$
  ⟨*proof*⟩

end

schematic_goal *seqspace_fm_auto*:
  assumes
    i ∈ *nat j* ∈ *nat h*∈*nat env* ∈ *list*(*A*)
  shows
    (∃ *om*∈*A*. *omega*(##*A*,*om*) ∧ *nth*(*i*,*env*) ∈ *om* ∧ *is_funspace*(##*A*, *nth*(*i*,*env*),
nth(*h*,*env*), *nth*(*j*,*env*))) ⟷ (*A*, *env* ⊨ (*?sqsprp*(*i*,*j*,*h*)))
  ⟨*proof*⟩
⟨*ML*⟩

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**
  $cons\_like(f) \equiv \forall\ i{\in}domain(f)\ .\ \forall j{\in}i\ .\ fst(f`i) \neq fst(f`j)$

⟨*ML*⟩

lemma (**in** *M_seqspace*) *cons_like_abs*:
  $M(f) \Longrightarrow cons\_like(f) \longleftrightarrow cons\_like\_rel(M,f)$
  ⟨*proof*⟩

definition *FiniteFun_iso* :: $[i,i,i,i,i] \Rightarrow o$ **where**
  $FiniteFun\_iso(A,B,n,g,f) \equiv (\forall\ i{\in}n\ .\ g`i \in f) \wedge (\forall\ ab{\in}f.\ (\exists\ i{\in}n.\ g`i{=}ab))$

From a function $g \in n \to A \times B$ we obtain a finite function in A -||> B.

definition *to_FiniteFun* :: $i \Rightarrow i$ **where**
  $to\_FiniteFun(f) \equiv \{f`i.\ i{\in}domain(f)\}$

definition *FiniteFun_Repr* :: $[i,i] \Rightarrow i$ **where**
  $FiniteFun\_Repr(A,B) \equiv \{f \in (A{\times}B)^{<\omega}\ .\ cons\_like(f)\ \}$

locale *M_FiniteFun* = *M_seqspace* +
  assumes
    cons_like_separation : $separation(M,\lambda f.\ cons\_like\_rel(M,f))$
    and
    to_finiteFun_replacement: $strong\_replacement(M,\ \lambda x\ y.\ y = range(x))$
    and
    supset_separation: $separation(M,\ \lambda\ x.\ \exists\, a.\ \exists\, b.\ x = \langle a,b\rangle \wedge b \subseteq a)$
begin

lemma *fun_range_eq*: $f{\in}A{\to}B \Longrightarrow \{f`i\ .\ i{\in}domain(f)\ \} = range(f)$
  ⟨*proof*⟩

lemma *FiniteFun_fst_type*:
  assumes $h{\in}A$-||>B $p{\in}h$
  shows $fst(p){\in}domain(h)$
  ⟨*proof*⟩

lemma *FinFun_closed*:
  $M(A) \Longrightarrow M(B) \Longrightarrow M(\bigcup\{n{\to}A{\times}B\ .\ n{\in}\omega\})$
  ⟨*proof*⟩

lemma *cons_like_lt* :

assumes $n \in \omega$ $f \in succ(n) \rightarrow A \times B$ $cons\_like(f)$
shows $restrict(f,n) \in n \rightarrow A \times B$ $cons\_like(restrict(f,n))$
⟨*proof*⟩

A finite function $f \in A$ -||> B can be represented by a function $g \in n \rightarrow A \times B$, with $n = |f|$.

lemma *FiniteFun_iso_intro1*:
assumes $f \in (A$ -||> $B)$
shows $\exists\, n \in \omega$. $\exists\, g \in n \rightarrow A \times B$. $FiniteFun\_iso(A,B,n,g,f) \wedge cons\_like(g)$
⟨*proof*⟩

All the representations of $f \in A$ -||> B are equal.

lemma *FiniteFun_isoD* :
assumes $n \in \omega$ $g \in n \rightarrow A \times B$ $f \in A$-||>B $FiniteFun\_iso(A,B,n,g,f)$
shows $to\_FiniteFun(g) = f$
⟨*proof*⟩

lemma *to_FiniteFun_succ_eq* :
assumes $n \in \omega$ $f \in succ(n) \rightarrow A$
shows $to\_FiniteFun(f) = cons(f`n, to\_FiniteFun(restrict(f,n)))$
⟨*proof*⟩

If $g \in n \rightarrow A \times B$ is *cons_like*, then it is a representation of $to\_FiniteFun(g)$.

lemma *FiniteFun_iso_intro_to*:
assumes $n \in \omega$ $g \in n \rightarrow A \times B$ $cons\_like(g)$
shows $to\_FiniteFun(g) \in (A$ -||> $B) \wedge FiniteFun\_iso(A,B,n,g,to\_FiniteFun(g))$
⟨*proof*⟩

lemma *FiniteFun_iso_intro2*:
assumes $n \in \omega$ $f \in n \rightarrow A \times B$ $cons\_like(f)$
shows $\exists$ $g \in (A$ -||> $B)$. $FiniteFun\_iso(A,B,n,f,g)$
⟨*proof*⟩

lemma *FiniteFun_eq_range_Repr* :
shows $\{range(h)$. $h \in FiniteFun\_Repr(A,B)\ \} = \{to\_FiniteFun(h)$. $h \in FiniteFun\_Repr(A,B)\ \}$
⟨*proof*⟩

lemma *FiniteFun_eq_to_FiniteFun_Repr* :
shows A-||>$B = \{to\_FiniteFun(h)$. $h \in FiniteFun\_Repr(A,B)\ \}$
(**is** *?Y=?X*)
⟨*proof*⟩

lemma *FiniteFun_Repr_closed* :
assumes $M(A)$ $M(B)$
shows $M(FiniteFun\_Repr(A,B))$
⟨*proof*⟩

lemma *to_FiniteFun_closed*:
  assumes $M(A)$ $f \in A$
  shows $M(range(f))$
  ⟨*proof*⟩

lemma *To_FiniteFun_Repr_closed* :
  assumes $M(A)$ $M(B)$
  shows $M(\{range(h) . h \in FiniteFun\_Repr(A,B)\})$
  ⟨*proof*⟩

lemma *FiniteFun_closed*[*intro*,*simp*] :
  assumes $M(A)$ $M(B)$
  shows $M(A \text{ -||> } B)$
  ⟨*proof*⟩

lemma *Fnle_nat_closed*[*intro*,*simp*]:
  assumes $M(I)$ $M(J)$
  shows $M(Fnle(\omega,I,J))$
  ⟨*proof*⟩

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 $well\_ord(A,r)$ $h \in surj(A,B)$ **and**
    types:$M(A)$ $M(r)$ $M(h)$ $M(B)$
  shows $B \lesssim^M A$
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *choice_ax_well_ord*: *M*(*S*) ⟹ ∃ *r*[*M*]. *well_ord*(*S*,*r*)
⟨*proof*⟩

end

locale *M_Pi_assumptions_choice* = *M_Pi_assumptions* + *M_cardinal_AC* +
assumes
B_replacement: *strong_replacement*(*M*, *λx 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*) ⟹ *strong_replacement*(*M*, *λx y*. *y* = ⟨*x*, *minimum*(*r*, *B*(*x*))⟩)
begin

lemma *AC_M*:
assumes *a* ∈ *A* ⋀*x*. *x* ∈ *A* ⟹ ∃ *y*. *y* ∈ *B*(*x*)
shows ∃ *z*[*M*]. *z* ∈ Pi^M(*A*, *B*)
⟨*proof*⟩

lemma *AC_Pi_rel*: **assumes** ⋀*x*. *x* ∈ *A* ⟹ ∃ *y*. *y* ∈ *B*(*x*)
shows ∃ *z*[*M*]. *z* ∈ Pi^M(*A*, *B*)
⟨*proof*⟩

end

context *M_cardinal_AC*
begin

26.1 Strengthened Forms of Existing Theorems on Cardinals

lemma *cardinal_rel_eqpoll_rel*: *M*(*A*) ⟹ $|A|^M \approx^M A$
⟨*proof*⟩

lemmas *cardinal_rel_idem* = *cardinal_rel_eqpoll_rel* [*THEN cardinal_rel_cong*, *simp*]

lemma *cardinal_rel_eqE*: $|X|^M = |Y|^M$ ==> *M*(*X*) ⟹ *M*(*Y*) ⟹ $X \approx^M Y$
⟨*proof*⟩

lemma *cardinal_rel_eqpoll_rel_iff*: *M*(*X*) ⟹ *M*(*Y*) ⟹ $|X|^M = |Y|^M$ ⟷ *X*

$\approx^M$ Y
⟨*proof*⟩

lemma *cardinal_rel_disjoint_Un*:
[| $|A|^M=|B|^M$; $|C|^M=|D|^M$; $A \cap C = 0$; $B \cap D = 0$; $M(A)$; $M(B)$; $M(C)$; $M(D)$|]
==> $|A \cup C|^M = |B \cup D|^M$
⟨*proof*⟩

lemma *lepoll_rel_imp_cardinal_rel_le*: $A \lesssim^M B$ ==> $M(A) \Longrightarrow M(B) \Longrightarrow |A|^M \leq |B|^M$
⟨*proof*⟩

lemma *cadd_rel_assoc*: $[\![M(i); M(j); M(k)]\!] \Longrightarrow (i \oplus^M j) \oplus^M k = i \oplus^M (j \oplus^M k)$
⟨*proof*⟩

lemma *cmult_rel_assoc*: $[\![M(i); M(j); M(k)]\!] \Longrightarrow (i \otimes^M j) \otimes^M k = i \otimes^M (j \otimes^M k)$
⟨*proof*⟩

lemma *cadd_cmult_distrib*: $[\![M(i); M(j); M(k)]\!] \Longrightarrow (i \oplus^M j) \otimes^M k = (i \otimes^M k) \oplus^M (j \otimes^M k)$
⟨*proof*⟩

lemma *InfCard_rel_square_eq*: $InfCard^M(|A|^M) \Longrightarrow M(A) \Longrightarrow A \times A \approx^M A$
⟨*proof*⟩

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*⟩

lemma *le_Card_rel_iff*: $Card^M(K)$ ==> $M(K) \Longrightarrow M(A) \Longrightarrow |A|^M \leq K \longleftrightarrow A \lesssim^M K$
⟨*proof*⟩

lemma *cardinal_rel_0_iff_0* [*simp*]: $M(A) \Longrightarrow |A|^M = 0 \longleftrightarrow A = 0$
⟨*proof*⟩

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*⟩

lemma *cardinal_rel_le_imp_lepoll_rel*: $i \leq |A|^M$ ==> $M(i) \Longrightarrow M(A) \Longrightarrow i \lesssim^M A$
⟨*proof*⟩

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*⟩

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 $|Y|^M \leq |X|^M$
⟨*proof*⟩

end

The set-theoretic universe.

abbreviation
Universe :: $i \Rightarrow o$ (‹$\mathcal{V}$›) **where**
$\mathcal{V}(x) \equiv True$

lemma *separation_absolute*: *separation*($\mathcal{V}$, P)
⟨*proof*⟩

lemma *univalent_absolute*:
assumes *univalent*($\mathcal{V}$, A, P) $P(x, b)$ $x \in A$
shows $P(x, y) \Longrightarrow y = b$
⟨*proof*⟩

lemma *replacement_absolute*: *strong_replacement*($\mathcal{V}$, P)
⟨*proof*⟩

lemma *Union_ax_absolute*: *Union_ax*($\mathcal{V}$)
⟨*proof*⟩

lemma *upair_ax_absolute*: *upair_ax*($\mathcal{V}$)
⟨*proof*⟩

lemma *power_ax_absolute*:*power_ax*($\mathcal{V}$)
⟨*proof*⟩

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*: $M(\bigcup i{\in}K.\ X(i))$
⟨*proof*⟩

Kunen's Lemma 10.21

lemma *cardinal_rel_UN_le*:
assumes *K*: $InfCard^M(K)$
shows $(\bigwedge i.\ i{\in}K \Longrightarrow |X(i)|^M \leq K) \Longrightarrow |\bigcup i{\in}K.\ X(i)|^M \leq K$
⟨*proof*⟩

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 $Ord(\kappa)$ $M(\kappa)$

shows $(|\kappa|^{M+})^M = (\kappa^+)^M$
⟨*proof*⟩

lemma (**in** *M_cardinal_arith_jump*) *csucc_rel_le_mono*:
assumes $\kappa \le \nu$ $M(\kappa)$ $M(\nu)$
shows $(\kappa^+)^M \le (\nu^+)^M$
⟨*proof*⟩

lemma (**in** *M_cardinal_AC*) *cardinal_rel_succ_not_0*: $|A|^M = succ(n) \Longrightarrow M(A) \Longrightarrow M(n) \Longrightarrow A \neq 0$
⟨*proof*⟩

⟨*ML*⟩

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*∈*X* . *Q*(*x*)}) ⟹ *Finite*({*x*∈*X* . *M*(*x*) ∧ *Q*(*x*)})
(**is** *Finite*(*?A*) ⟹ *Finite*(*?B*))
⟨*proof*⟩

lemma *Finite_to_one_relI*[*intro*]:
assumes $f{:}X \to^M Y$ ⋀*y*. *y*∈*Y* ⟹ *Finite*({*x*∈*X* . *f*‘*x* = *y*})
and *types*:*M*(*f*) *M*(*X*) *M*(*Y*)
shows $f \in Finite\_to\_one^M(X,Y)$
⟨*proof*⟩

lemma *Finite_to_one_relI'*[*intro*]:
assumes $f{:}X \to^M Y$ ⋀*y*. *y*∈*Y* ⟹ *Finite*({*x*∈*X* . *M*(*x*) ∧ *f*‘*x* = *y*})
and *types*:*M*(*f*) *M*(*X*) *M*(*Y*)
shows $f \in Finite\_to\_one^M(X,Y)$
⟨*proof*⟩

lemma *Finite_to_one_relD*[*dest*]:
$f \in Finite\_to\_one^M(X,Y) \Longrightarrow f{:}X \to^M Y$
$f \in Finite\_to\_one^M(X,Y)$ ⟹ *y*∈*Y* ⟹ *M*(*Y*) ⟹ *Finite*({*x*∈*X* . *M*(*x*) ∧ *f*‘*x* = *y*})
⟨*proof*⟩

lemma *Diff_bij_rel*:

assumes $\forall A \in F.\ X \subseteq A$
and *types*: $M(F)$ $M(X)$ **shows** $(\lambda A \in F.\ A\text{-}X) \in bij^M(F, \{A\text{-}X.\ A \in F\})$
⟨*proof*⟩

lemma *function_space_rel_nonempty*:
assumes $b \in B$ **and** *types*: $M(B)$ $M(A)$
shows $(\lambda x \in A.\ b) : A \to^M B$
⟨*proof*⟩

lemma *mem_function_space_rel*:
assumes $f \in A \to^M y$ $M(A)$ $M(y)$
shows $f \in A \to y$
⟨*proof*⟩

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*: $f \in Pi^M(A,B) \Longrightarrow f \in Pi(A, B)$
⟨*proof*⟩

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*: $[\![f \in bij^M(A,B);\ M(A);\ M(B)]\!] \Longrightarrow f \in bij(A,B)$
⟨*proof*⟩

lemma *mem_inj_rel*: $[\![f \in inj^M(A,B);\ M(A);\ M(B)]\!] \Longrightarrow f \in inj(A,B)$
⟨*proof*⟩

lemma *mem_surj_rel*: $[\![f \in surj^M(A,B);\ M(A);\ M(B)]\!] \Longrightarrow f \in surj(A,B)$
⟨*proof*⟩

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*: ⟦$f{:}A\to^{M}B$; $f \in inj^{M}(A,B)$; $M(A)$; $M(B)$⟧ $\Longrightarrow f \in inj^{M}(A,f``A)$
⟨*proof*⟩

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*⟩

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 $f \cup g \in (A \cup C)\to^{M} (B \cup D)$
⟨*proof*⟩

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 $f \cup g \in (A \cup C)\to^{M} (B \cup D)$
⟨*proof*⟩

lemma *lepoll_relD*[*dest*]: $A \lesssim^{M} B \Longrightarrow \exists f[M].\ f \in inj^{M}(A,\ B)$
⟨*proof*⟩
lemma *lepoll_relI*[*intro*]: $f \in inj^{M}(A,\ B) \Longrightarrow M(f) \Longrightarrow A \lesssim^{M} B$
⟨*proof*⟩

lemma *eqpollD*[*dest*]: $A \approx^{M} B \Longrightarrow \exists f[M].\ f \in bij^{M}(A,\ B)$
⟨*proof*⟩
lemma *bij_rel_imp_eqpoll_rel*[*intro*]: $f \in bij^{M}(A,B) \Longrightarrow M(f) \Longrightarrow A \approx^{M} B$
⟨*proof*⟩

lemma *restrict_bij_rel*:— Unused
assumes $f \in inj^{M}(A,B)$ $C\subseteq A$
and *types*:$M(A)$ $M(B)$ $M(C)$
shows $restrict(f,C)\in bij^{M}(C,\ f``C)$
⟨*proof*⟩

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 $S \approx^{M} f `` S$
⟨*proof*⟩

end

⟨*ML*⟩

context *M_ZF_library*
begin

— MOVE THIS to an appropriate place
⟨*ML*⟩
⟨*proof*⟩

⟨*ML*⟩
⟨*proof*⟩

⟨*ML*⟩ ⟨*proof*⟩

end

⟨*ML*⟩
notation *is_cexp_fm* (‹·_$^{\uparrow}$_ *is* _·›)
⟨*ML*⟩

abbreviation
cexp_r :: [*i*,*i*,*i*⇒*o*] ⇒ *i* (‹_$^{\uparrow}$_,_›) **where**
cexp_r(*x*,*y*,*M*) ≡ *cexp_rel*(*M*,*x*,*y*)

abbreviation
cexp_r_set :: [*i*,*i*,*i*] ⇒ *i* (‹_$^{\uparrow}$_,_›) **where**
cexp_r_set(*x*,*y*,*M*) ≡ *cexp_rel*(##*M*,*x*,*y*)

context *M_ZF_library*
begin

lemma *Card_cexp*: $M(\kappa) \Longrightarrow M(\nu) \Longrightarrow Card^M(\kappa^{\uparrow\nu,M})$
⟨*proof*⟩
declare *conj_cong*[*cong*]

lemma *eq_csucc_rel_ord*:
$Ord(i) \Longrightarrow M(i) \Longrightarrow (i^+)^M = (|i|^{M+})^M$
⟨*proof*⟩

lemma *lesspoll_succ_rel*:
assumes $Ord(\kappa)$ $M(\kappa)$
shows $\kappa \lesssim^M (\kappa^+)^M$
⟨*proof*⟩

lemma *lesspoll_rel_csucc_rel*:

assumes $Ord(\kappa)$
and *types*:$M(\kappa)$ $M(d)$
shows $d \prec^M (\kappa^+)^M \longleftrightarrow d \lesssim^M \kappa$
⟨*proof*⟩

lemma *Infinite_imp_nats_lepoll*:
assumes $Infinite(X)$ $n \in \omega$
shows $n \lesssim X$
⟨*proof*⟩

lemma *nepoll_imp_nepoll_rel* :
assumes $\neg\ x \approx X$ $M(x)$ $M(X)$
shows $\neg\ (x \approx^M X)$
⟨*proof*⟩

lemma *Infinite_imp_nats_lepoll_rel*:
assumes $Infinite(X)$ $n \in \omega$
and *types*: $M(X)$
shows $n \lesssim^M X$
⟨*proof*⟩

lemma *lepoll_rel_imp_lepoll*: $A \lesssim^M B \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow A \lesssim B$
⟨*proof*⟩

lemma *zero_lesspoll_rel*: **assumes** $0<\kappa$ $M(\kappa)$ **shows** $0 \prec^M \kappa$
⟨*proof*⟩

lemma *lepoll_rel_nat_imp_Infinite*: $\omega \lesssim^M X \Longrightarrow M(X) \Longrightarrow Infinite(X)$
⟨*proof*⟩

lemma *InfCard_rel_imp_Infinite*: $InfCard^M(\kappa) \Longrightarrow M(\kappa) \Longrightarrow Infinite(\kappa)$
⟨*proof*⟩

lemma *lt_surj_rel_empty_imp_Card_rel*:
assumes $Ord(\kappa)$ $\bigwedge\alpha.\ \alpha < \kappa \Longrightarrow surj^M(\alpha,\kappa) = 0$
and *types*:$M(\kappa)$
shows $Card^M(\kappa)$
⟨*proof*⟩

end

⟨*ML*⟩

notation *mono_map_rel* (‹*mono′_map–′*(_,_,_,_′)›)

abbreviation
mono_map_r_set :: [*i*,*i*,*i*,*i*,*i*]⇒*i* (‹*mono′_map–′*(_,_,_,_′)›) **where**
$mono\_map^M(a,r,b,s) \equiv mono\_map\_rel(\#\#M,a,r,b,s)$

context *M_ZF_library*
begin

lemma *mono_map_rel_char*:
  assumes $M(a)$ $M(b)$
  shows $mono\_map^M(a,r,b,s) = \{f\in mono\_map(a,r,b,s) . M(f)\}$
  ⟨*proof*⟩

Just a sample of porting results on *mono_map*

lemma *mono_map_rel_mono*:
  assumes
    $f \in mono\_map^M(A,r,B,s)$ $B \subseteq C$
    and *types*:$M(A)$ $M(B)$ $M(C)$
  shows
    $f \in mono\_map^M(A,r,C,s)$
  ⟨*proof*⟩

lemma *nats_le_InfCard_rel*:
  assumes $n \in \omega$ $InfCard^M(\kappa)$
  shows $n \leq \kappa$
  ⟨*proof*⟩

lemma *nat_into_InfCard_rel*:
  assumes $n \in \omega$ $InfCard^M(\kappa)$
  shows $n \in \kappa$
  ⟨*proof*⟩

lemma *Finite_cardinal_rel_in_nat* [*simp*]:
  assumes $Finite(A)$ $M(A)$ **shows** $|A|^M \in \omega$
⟨*proof*⟩

lemma *Finite_cardinal_rel_eq_cardinal*:
  assumes $Finite(A)$ $M(A)$ **shows** $|A|^M = |A|$
⟨*proof*⟩

lemma *Finite_imp_cardinal_rel_cons*:
  assumes *FA*: $Finite(A)$ **and** *a*: $a\notin A$ **and** *types*:$M(A)$ $M(a)$
  shows $|cons(a,A)|^M = succ(|A|^M)$
  ⟨*proof*⟩

lemma *Finite_imp_succ_cardinal_rel_Diff*:
  assumes $Finite(A)$ $a \in A$ $M(A)$
  shows $succ(|A\text{-}\{a\}|^M) = |A|^M$
⟨*proof*⟩

lemma *InfCard_rel_Aleph_rel*:
  notes *Aleph_rel_zero*[*simp*]
  assumes $Ord(\alpha)$
    and *types*: $M(\alpha)$

shows *InfCard*^{M}($\aleph_\alpha{}^M$)
⟨*proof*⟩

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*⇒*o*,*i*,[*i*,*i*]⇒*i*,*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
lepoll_assumptions5(*M*,*A*,*F*,*S*,*fa*,*K*,*x*,*f*,*r*) ≡
strong_replacement(*M*, λ*x* *y*. *y* = ⟨*x*, μ *i*. *x* ∈ *F*(*A*, *i*), *f* ‘ (μ *i*. *x* ∈ *F*(*A*, *i*)) ‘ *x*⟩)

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*
$(M, \lambda y\ z.\ y \in inj^M(\text{if } x \in range(f) \text{ then } F(A, converse(f)\ `\ x) \text{ else } 0,K) \wedge z = \{\langle x, y\rangle\})$

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*$(M, \lambda x\ y.\ y = inj^M(\text{if } x \in range(f) \text{ then } F(A, converse(f)\ `\ x) \text{ else } 0,K))$

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*
$(M, \lambda x\ z.\ z = Sigfun(x, \lambda i.\ inj^M(\text{if } i \in range(f) \text{ then } F(A, converse(f)\ `\ i) \text{ else } 0,K)))$

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*
$(M, \lambda x\ y.\ y = \langle x, minimum(r, inj^M(\text{if } x \in range(f) \text{ then } F(A, converse(f)\ `\ x) \text{ else } 0,K))\rangle)$

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*$(H,f,i) \equiv \text{if } i \in range(f) \text{ then } H(converse(f)\ `\ i) \text{ else } 0$

definition *if_range_F_else_F* **where**
if_range_F_else_F$(H,b,f,i) \equiv \text{if } b=0 \text{ then } if\_range\_F(H,f,i) \text{ else } H(i)$

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*⟩

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 $lam\_replacement(M,\lambda x\ .\ \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),0,f,i))$
(**is** $lam\_replacement(M,\lambda x\ .\ Least(?P(x)))$)
⟨*proof*⟩

lemma (**in** *M_replacement_extra*) *lam_Least_assumption_ifM_bnot0*:
fixes F
defines $F \equiv \lambda\_\ x.\ if\ M(x)\ then\ x\ else\ 0$
assumes
separations:
$\forall A'[M].\ separation(M, \lambda y. \exists x{\in}A'.\ y = \langle x, \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i)\rangle)$
$separation(M,Ord)$
and
types:$M(A)\ M(f)$
and
$b \neq 0$
shows $lam\_replacement(M,\lambda x\ .\ \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i))$
(**is** $lam\_replacement(M,\lambda x\ .\ Least(?P(x)))$)
⟨*proof*⟩

lemma (**in** *M_replacement_extra*) *lam_Least_assumption_drSR_Y*:
fixes $F\ r'\ D$
defines $F \equiv drSR\_Y(r',D)$
assumes $\forall A'[M].\ separation(M, \lambda y. \exists x{\in}A'.\ y = \langle x, \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i)\rangle)$
$M(A)\ M(b)\ M(f)\ M(r')$
shows $lam\_replacement(M,\lambda x\ .\ \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i))$
⟨*proof*⟩

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)
 ⟨*proof*⟩

lemma *lepoll_assumptions2*:
 assumes *types*[*simp*]:$M(A)$ $M(S)$
 shows *lepoll_assumptions2*(M,A,F,S,fa,K,x,f,r)
 ⟨*proof*⟩

lemma *lepoll_assumptions3*:
 assumes *types*[*simp*]:$M(A)$
 shows *lepoll_assumptions3*(M,A,F,S,fa,K,x,f,r)
 ⟨*proof*⟩

lemma *lepoll_assumptions4*:
 assumes *types*[*simp*]:$M(A)$ $M(r)$
 shows *lepoll_assumptions4*(M,A,F,S,fa,K,x,f,r)
 ⟨*proof*⟩

lemma *lam_Least_closed* :
 assumes $M(A)$ $M(b)$ $M(f)$
 shows $\forall x[M].\ M(\mu\ i.\ x \in$ *if_range_F_else_F*$(F(A),b,f,i))$
⟨*proof*⟩

lemma *lepoll_assumptions5*:
 assumes
 types[*simp*]:$M(A)$ $M(f)$
 shows *lepoll_assumptions5*(M,A,F,S,fa,K,x,f,r)
 ⟨*proof*⟩

lemma *lepoll_assumptions6*:
 assumes *types*[*simp*]:$M(A)$ $M(S)$ $M(x)$
 shows *lepoll_assumptions6*(M,A,F,S,fa,K,x,f,r)
 ⟨*proof*⟩

lemma *lepoll_assumptions7*:
 assumes *types*[*simp*]:$M(A)$ $M(S)$ $M(x)$
 shows *lepoll_assumptions7*(M,A,F,S,fa,K,x,f,r)
 ⟨*proof*⟩

lemma *lepoll_assumptions8*:
 assumes *types*[*simp*]:$M(A)$ $M(S)$
 shows *lepoll_assumptions8*(M,A,F,S,fa,K,x,f,r)
 ⟨*proof*⟩

lemma *lepoll_assumptions9*:
assumes *types*[*simp*]:$M(A)$ $M(S)$ $M(r)$
shows *lepoll_assumptions9*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemma *lepoll_assumptions10*:
assumes *types*[*simp*]:$M(A)$ $M(f)$
shows *lepoll_assumptions10*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemma *lepoll_assumptions11*:
assumes *types*[*simp*]:$M(A)$ $M(f)$
shows *lepoll_assumptions11*$(M, A, F, S, fa, K, x, f, r)$
⟨*proof*⟩

lemma *lepoll_assumptions12*:
assumes *types*[*simp*]:$M(A)$ $M(x)$ $M(f)$
shows *lepoll_assumptions12*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemma *lepoll_assumptions13*:
assumes *types*[*simp*]:$M(A)$ $M(r)$ $M(f)$
shows *lepoll_assumptions13*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemma *lepoll_assumptions14*:
assumes *types*[*simp*]:$M(A)$ $M(f)$ $M(fa)$
shows *lepoll_assumptions14*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *lepoll_assumptions16*:
assumes *types*[*simp*]:$M(A)$ $M(f)$ $M(K)$
shows *lepoll_assumptions16*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemma *lepoll_assumptions17*:
assumes *types*[*simp*]:$M(A)$ $M(f)$ $M(K)$
shows *lepoll_assumptions17*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

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)
⟨*proof*⟩

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*]
⟨*ML*⟩

lemma *iff_sym* : $P(x,a) \longleftrightarrow a = f(x) \Longrightarrow P(x,a) \longleftrightarrow f(x) = a$
⟨*proof*⟩

lemma *subset_iff_sats* [*iff_sats*]:
[| *nth*(*i*,*env*) = *x*; *nth*(*k*,*env*) = *z*;
i ∈ *nat*; *k* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *subset*(##*A*, *x*, *z*) ⟷ *sats*(*A*, *subset_fm*(*i*,*k*), *env*)
⟨*proof*⟩

definition *radd_body* :: [*i*,*i*,*i*] ⇒ *o* **where**
$radd\_body(R,S) \equiv \lambda z.\ (\exists\, x\ y.\ z = \langle Inl(x),\ Inr(y)\rangle) \lor$
$(\exists\, x'\ x.\ z = \langle Inl(x'),\ Inl(x)\rangle \land \langle x',\ x\rangle \in R) \lor$
$(\exists\, y'\ y.\ z = \langle Inr(y'),\ Inr(y)\rangle \land \langle y',\ y\rangle \in S)$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_radd_body*:
(##*M*)(*r*) ⟹ (##*M*)(*A*) ⟹ *separation*(##*M*, *is_radd_body*(##*M*,*A*,*r*))
⟨*proof*⟩

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*)
⟨*proof*⟩

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*))
⟨*proof*⟩

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*)

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_well_ord_body*:
(##*M*)(*f*) ⟹ (##*M*)(*r*) ⟹ (##*M*)(*A*) ⟹ *separation*(##*M*, *well_ord_body*(##*M*,*A*,*f*,*r*))
⟨*proof*⟩

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*))
⟨*proof*⟩

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*)))

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_obase_body*:
(##*M*)(*r*) ⟹ (##*M*)(*A*) ⟹ *separation*(##*M*, *is_obase_body*(##*M*,*A*,*r*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_is_obase*:
(##*M*)(*f*) ⟹ (##*M*)(*r*) ⟹ (##*M*)(*A*) ⟹ *separation*
(##*M*, λ*x*. *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*))))
⟨*proof*⟩

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*)

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_obase_equals_aux*:
(##*M*)(*r*) ⟹ (##*M*)(*A*) ⟹ *separation*(##*M*, *is_obase_equals*(##*M*,*A*,*r*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_obase_equals*:
(##*M*)(*f*) ⟹ (##*M*)(*r*) ⟹ (##*M*)(*A*) ⟹ *separation*
(##*M*, λ*a*. ∃ *x*[##*M*].
∃ *g*[##*M*].
∃ *mx*[##*M*].
∃ *par*[##*M*].
ordinal(##*M*, *x*) ∧
membership(##*M*, *x*, *mx*) ∧
pred_set(##*M*, *A*, *a*, *r*, *par*) ∧ *order_isomorphism*(##*M*, *par*, *r*, *x*, *mx*, *g*))
⟨*proof*⟩

definition *id_body* :: [*i*,*i*] ⇒ *o* **where**
id_body(*A*) ≡ λ*z*. ∃ *x*∈*A*. *z* = ⟨*x*, *x*⟩
⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_id_body*:
(##*M*)(*A*) ⟹ *separation*(##*M*, *is_id_body*(##*M*,*A*))

⟨*proof*⟩

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)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_id_body*:
$(\#\#M)(A) \Longrightarrow$ *separation*
$(\#\#M,\ \lambda z.\ \exists x{\in}A.\ z = \langle x,\ x\rangle)$
⟨*proof*⟩

definition *rvimage_body* :: $[i,i,i] \Rightarrow o$ **where**
rvimage_body$(f,r) \equiv \lambda z.\ \exists x\ y.\ z = \langle x,\ y\rangle \wedge \langle f\ \text{`}\ x,\ f\ \text{`}\ y\rangle \in r$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_rvimage_body*:
$(\#\#M)(f) \Longrightarrow (\#\#M)(r) \Longrightarrow$ *separation*$(\#\#M,$ *is_rvimage_body*$(\#\#M,f,r))$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *rvimage_body_abs*:
assumes $(\#\#M)(R)$ $(\#\#M)(S)$ $(\#\#M)(x)$
shows *is_rvimage_body*$(\#\#M,R,S,x) \longleftrightarrow$ *rvimage_body*(R,S,x)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_rvimage_body*:
$(\#\#M)(f) \Longrightarrow (\#\#M)(r) \Longrightarrow$ *separation*
$(\#\#M,\ \lambda z.\ \exists x\ y.\ z = \langle x,\ y\rangle \wedge \langle f\ \text{`}\ x,\ f\ \text{`}\ y\rangle \in r)$
⟨*proof*⟩

definition *rmult_body* :: $[i,i,i] \Rightarrow o$ **where**
rmult_body$(b,d) \equiv \lambda z.\ \exists x'\ y'\ x\ y.\ z = \langle\langle x',\ y'\rangle,\ x,\ y\rangle \wedge (\langle x',\ x\rangle \in b \vee x' = x \wedge \langle y',\ y\rangle \in d)$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_rmult_body*:
$(\#\#M)(b) \Longrightarrow (\#\#M)(d) \Longrightarrow$ *separation*$(\#\#M,$ *is_rmult_body*$(\#\#M,b,d))$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *rmult_body_abs*:
assumes $(\#\#M)(b)$ $(\#\#M)(d)$ $(\#\#M)(x)$
shows *is_rmult_body*$(\#\#M,b,d,x) \longleftrightarrow$ *rmult_body*(b,d,x)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_rmult_body*:

$(\#\#M)(b) \Longrightarrow (\#\#M)(d) \Longrightarrow$ *separation*
$(\#\#M,\ \lambda z.\ \exists x'\ y'\ x\ y.\ z = \langle\langle x',\ y'\rangle,\ x,\ y\rangle \wedge (\langle x',\ x\rangle \in b \vee x' = x \wedge \langle y',\ y\rangle$
$\in d))$
⟨*proof*⟩

definition *ord_iso_body* :: $[i,i,i,i] \Rightarrow o$ **where**
ord_iso_body$(A,r,s) \equiv \lambda f.\ \forall x{\in}A.\ \forall y{\in}A.\ \langle x,\ y\rangle \in r \longleftrightarrow \langle f\ `\ x,\ f\ `\ y\rangle \in s$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_ord_iso_body*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(r) \Longrightarrow (\#\#M)(s) \Longrightarrow$ *separation*$(\#\#M,$ *is_ord_iso_body*$(\#\#M,A,r,s))$
⟨*proof*⟩

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)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_ord_iso_body*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(r) \Longrightarrow (\#\#M)(s) \Longrightarrow$ *separation*
$(\#\#M,\ \lambda f.\ \forall x{\in}A.\ \forall y{\in}A.\ \langle x,\ y\rangle \in r \longleftrightarrow \langle f\ `\ x,\ f\ `\ y\rangle \in s)$
⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_PiP_rel*:
$(\#\#M)(A) \Longrightarrow$ *separation*$(\#\#M,$ *PiP_rel*$(\#\#M,A))$
⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_injP_rel*:
$(\#\#M)(A) \Longrightarrow$ *separation*$(\#\#M,$ *injP_rel*$(\#\#M,A))$
⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_surjP_rel*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow$ *separation*$(\#\#M,$ *surjP_rel*$(\#\#M,A,B))$
⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_cons_like_rel*:
separation$(\#\#M,$ *cons_like_rel*$(\#\#M))$
⟨*proof*⟩

definition *supset_body* :: $[i] \Rightarrow o$ **where**
supset_body $\equiv \lambda\ x.\ \exists\, a.\ \exists\, b.\ x = \langle a,b\rangle \wedge b \subseteq a$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_supset_body*:
separation(##*M*, *is_supset_body*(##*M*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *supset_body_abs*:
assumes (##*M*)(*x*)
shows *is_supset_body*(##*M*,*x*) ⟷ *supset_body*(*x*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_supset_body*:
separation(##*M*, $\lambda\ x.\ \exists\, a.\ \exists\, b.\ x = \langle a,b\rangle \wedge b \subseteq a$)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_is_fst*:
(##*M*)(*a*) ⟹ *separation*(##*M*, λ*x* . *is_fst*(##*M*,*x*,*a*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_fst_equal*:
(##*M*)(*a*) ⟹ *separation*(##*M*, λ*x*. *fst*(*x*) = *a*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_is_apply*:
(##*M*)(*f*) ⟹ (##*M*)(*a*) ⟹ *separation*(##*M*, λ*x* . *is_apply*(##*M*,*f*,*x*,*a*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_equal_apply*:
(##*M*)(*f*) ⟹ (##*M*)(*a*) ⟹ *separation*(##*M*, λ*x*. *f*‘*x* = *a*)
⟨*proof*⟩

definition *id_rel* :: $[i \Rightarrow o,i] \Rightarrow o$ **where**
id_rel(*A*) $\equiv \lambda z.\ \exists\, x[A].\ z = \langle x,\ x\rangle$
⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_id_rel*:
separation(##*M*, *is_id_rel*(##*M*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *id_rel_abs*:
assumes (##*M*)(*x*)
shows *is_id_rel*(##*M*,*x*) ⟷ *id_rel*(##*M*,*x*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_id_rel*:
separation(##*M*, λ*z*. ∃*x*[##*M*]. *z* = ⟨*x*, *x*⟩)

⟨*proof*⟩

definition *sndfst_eq_fstsnd* :: [*i*] ⇒ *o* **where**
sndfst_eq_fstsnd ≡ λ*x*. *snd*(*fst*(*x*)) = *fst*(*snd*(*x*))
⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_sndfst_eq_fstsnd*:
separation(##*M*, *is_sndfst_eq_fstsnd*(##*M*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *sndfst_eq_fstsnd_abs*:
assumes (##*M*)(*x*)
shows *is_sndfst_eq_fstsnd*(##*M*,*x*) ⟷ *sndfst_eq_fstsnd*(*x*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_sndfst_eq_fstsnd*:
separation(##*M*, λ*x*. *snd*(*fst*(*x*)) = *fst*(*snd*(*x*)))
⟨*proof*⟩

definition *fstfst_eq_fstsnd* :: [*i*] ⇒ *o* **where**
fstfst_eq_fstsnd ≡ λ*x*. *fst*(*fst*(*x*)) = *fst*(*snd*(*x*))
⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_fstfst_eq_fstsnd*:
separation(##*M*, *is_fstfst_eq_fstsnd*(##*M*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *fstfst_eq_fstsnd_abs*:
assumes (##*M*)(*x*)
shows *is_fstfst_eq_fstsnd*(##*M*,*x*) ⟷ *fstfst_eq_fstsnd*(*x*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_fstfst_eq_fstsnd*:
separation(##*M*, λ*x*. *fst*(*fst*(*x*)) = *fst*(*snd*(*x*)))
⟨*proof*⟩

definition *fstfst_eq* :: [*i*,*i*] ⇒ *o* **where**
fstfst_eq(*a*) ≡ λ*x*. *fst*(*fst*(*x*)) = *a*

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_fstfst_eq*:

$(\#\#M)(a) \Longrightarrow$ *separation*$(\#\#M,$ *is_fstfst_eq*$(\#\#M,a))$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *fstfst_eq_abs*:
assumes $(\#\#M)(a)$ $(\#\#M)(x)$
shows *is_fstfst_eq*$(\#\#M,a,x) \longleftrightarrow$ *fstfst_eq*(a,x)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_fstfst_eq*:
$(\#\#M)(a) \Longrightarrow$ *separation*$(\#\#M,$ $\lambda x.$ *fst*(*fst*$(x)) = a)$
⟨*proof*⟩

definition *restrict_elem* :: $[i,i,i] \Rightarrow o$ **where**
restrict_elem$(B,A) \equiv \lambda y.\ \exists x{\in}A.\ y = \langle x,$ *restrict*$(x, B)\rangle$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_restrict_elem*:
$(\#\#M)(B) \Longrightarrow (\#\#M)(A) \Longrightarrow$ *separation*$(\#\#M,$ *is_restrict_elem*$(\#\#M,B,A))$
⟨*proof*⟩

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)
⟨*proof*⟩

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*$(x, B)\rangle)$
⟨*proof*⟩

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*$(x, B)\rangle)$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_ordinal*:
separation$(\#\#M,$ *ordinal*$(\#\#M))$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_Ord*:
separation$(\#\#M,$ *Ord*$)$
⟨*proof*⟩

definition *insnd_ballPair* :: $[i,i,i] \Rightarrow o$ **where**
insnd_ballPair$(B,A) \equiv \lambda p.\ \forall x{\in}B.\ x \in$ *snd*$(p) \longleftrightarrow (\forall s{\in}$*fst*$(p).\ \langle s, x\rangle \in A)$

$\langle ML \rangle$

lemma (**in** *M_ZF_trans*) *separation_is_insnd_ballPair*:
$(\#\#M)(B) \Longrightarrow (\#\#M)(A) \Longrightarrow separation(\#\#M, is\_insnd\_ballPair(\#\#M,B,A))$
$\langle proof \rangle$

lemma (**in** *M_ZF_trans*) *insnd_ballPair_abs*:
assumes $(\#\#M)(B)$ $(\#\#M)(A)$ $(\#\#M)(x)$
shows $is\_insnd\_ballPair(\#\#M,B,A,x) \longleftrightarrow insnd\_ballPair(B,A,x)$
$\langle proof \rangle$

lemma (**in** *M_ZF_trans*) *separation_insnd_ballPair_aux*:
$(\#\#M)(B) \Longrightarrow (\#\#M)(A) \Longrightarrow separation(\#\#M, \lambda p. \forall x\in B. x \in snd(p) \longleftrightarrow (\forall s\in fst(p). \langle s, x\rangle \in A))$
$\langle proof \rangle$

lemma (**in** *M_ZF_trans*) *separation_insnd_ballPair*:
$(\#\#M)(B) \Longrightarrow \forall A[\#\#M]. separation(\#\#M, \lambda p. \forall x\in B. x \in snd(p) \longleftrightarrow (\forall s\in fst(p). \langle s, x\rangle \in A))$
$\langle proof \rangle$

definition *bex_restrict_eq_dom* :: $[i,i,i,i] \Rightarrow o$ **where**
$bex\_restrict\_eq\_dom(B,A,x) \equiv \lambda dr. \exists r\in A . restrict(r,B) = x \wedge dr=domain(r)$

$\langle ML \rangle$

lemma (**in** *M_ZF_trans*) *separation_is_bex_restrict_eq_dom*:
$(\#\#M)(B) \Longrightarrow (\#\#M)(A) \Longrightarrow (\#\#M)(x) \Longrightarrow separation(\#\#M, is\_bex\_restrict\_eq\_dom(\#\#M,B,A,x))$
$\langle proof \rangle$
lemma (**in** *M_ZF_trans*) *bex_restrict_eq_dom_abs*:
assumes $(\#\#M)(B)$ $(\#\#M)(A)$ $(\#\#M)(x)$ $(\#\#M)(dr)$
shows $is\_bex\_restrict\_eq\_dom(\#\#M,B,A,x,dr) \longleftrightarrow bex\_restrict\_eq\_dom(B,A,x,dr)$
$\langle proof \rangle$

lemma (**in** *M_ZF_trans*) *separation_restrict_eq_dom_eq_aux*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow (\#\#M)(x) \Longrightarrow separation(\#\#M, \lambda dr. \exists r\in A . restrict(r,B) = x \wedge dr=domain(r))$
$\langle proof \rangle$

lemma (**in** *M_ZF_trans*) *separation_restrict_eq_dom_eq*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow \forall x[\#\#M]. separation(\#\#M, \lambda dr. \exists r\in A . restrict(r,B) = x \wedge dr=domain(r))$
$\langle proof \rangle$

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) ≡
∃ c[N].
∃ a[N].
(∀ x[N]. x ∈ D ⟶ x ∈ a ⟷ (∃ r[N]. ∃ b[N]. (r ∈ A ∧restriction(N, r, B, b)) ∧ b=c ∧ is_domain(N, r, x))) ∧
is_snd(N, p, a) ∧ is_fst(N, p, c)

⟨ML⟩

lemma (**in** *M_ZF_trans*) *separation_is_insnd_restrict_eq_dom*:
(##M)(B) ⟹ (##M)(A) ⟹ (##M)(D) ⟹ separation(##M, is_insnd_restrict_eq_dom(##M,B,A,D)
⟨proof⟩

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) ⟷ insnd_restrict_eq_dom(B,A,D,x)
⟨proof⟩

lemma (**in** *M_ZF_trans*) *separation_restrict_eq_dom_eq_pair_aux*:
(##M)(A) ⟹ (##M)(B) ⟹ (##M)(D) ⟹
separation(##M, λp. ∀ x∈D. x ∈ snd(p) ⟷ (∃ r∈A. restrict(r, B) = fst(p) ∧ x = domain(r)))
⟨proof⟩

lemma (**in** *M_ZF_trans*) *separation_restrict_eq_dom_eq_pair*:
(##M)(A) ⟹ (##M)(B) ⟹
∀ D[##M]. separation(##M, λp. ∀ x∈D. x ∈ snd(p) ⟷ (∃ r∈A. restrict(r, B) = fst(p) ∧ x = domain(r)))
⟨proof⟩

⟨ML⟩

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)

⟨ML⟩

definition *ifrangeF_body* :: $[i \Rightarrow o,i,i,i,i] \Rightarrow o$ **where**
ifrangeF_body(M,A,b,f) ≡ λy. ∃ x∈A. y = ⟨x,μ i. ifrFb_body(M,b,f,x,i)⟩

⟨ML⟩

lemma (**in** *M_ZF_trans*) *separation_is_ifrangeF_body*:

(##M)(A) ⟹ (##M)(r) ⟹ (##M)(s) ⟹ *separation*(##M, *is_ifrangeF_body*(##M,A,r,s))
⟨*proof*⟩

lemma (**in** *M_basic*) *is_ifrFb_body_closed*: M(r) ⟹ M(s) ⟹ *is_ifrFb_body*(M, r, s, x, i) ⟹ M(i)
⟨*proof*⟩

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*⟩

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)⟩)
⟨*proof*⟩

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)

⟨*ML*⟩

definition *ifrangeF_body2* :: [i⇒o,i,i,i,i,i] ⇒ o **where**
ifrangeF_body2(M,A,G,b,f) ≡ λy. ∃x∈A. y = ⟨x,μ i. *ifrFb_body2*(M,G,b,f,x,i)⟩

⟨*ML*⟩

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))
⟨*proof*⟩

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)
⟨*proof*⟩

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*⟩

lemma (**in** *M_ZF_trans*) *separation_ifrangeF_body2*:
(##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*. *if* (##*M*)(*a*) *then G* ' *a else 0*, *b*, *f*, *i*))⟩)
⟨*proof*⟩

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*)

⟨*ML*⟩

definition *ifrangeF_body3* :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
ifrangeF_body3(*M*,*A*,*G*,*b*,*f*) ≡ λ*y*. ∃*x*∈*A*. *y* = ⟨*x*,μ *i*. *ifrFb_body3*(*M*,*G*,*b*,*f*,*x*,*i*)⟩

⟨*ML*⟩

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*))
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

lemma (**in** *M_ZF_trans*) *separation_ifrangeF_body3*:
(##*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*. *if* (##*M*)(*a*) *then G*-''{*a*} *else 0*, *b*, *f*, *i*))⟩)
⟨*proof*⟩

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)

⟨*ML*⟩

definition *ifrangeF_body4* :: [i⇒o,i,i,i,i,i] ⇒ o **where**
ifrangeF_body4(M,A,G,b,f) ≡ λy. ∃x∈A. y = ⟨x,μ i. *ifrFb_body4*(G,b,f,x,i)⟩

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_ifrangeF_body4*:
(##M)(A) ⟹ (##M)(G) ⟹ (##M)(r) ⟹ (##M)(s) ⟹ *separation*(##M, *is_ifrangeF_body4*(##M,A,G,r,s))
⟨*proof*⟩

lemma (**in** *M_basic*) *is_ifrFb_body4_closed*: M(G) ⟹ M(r) ⟹ M(s) ⟹ *is_ifrFb_body4*(M, G, r, s, x, i) ⟹ M(i)
⟨*proof*⟩

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) ⟷ *ifrangeF_body4*(##M,A,G,r,s,x)
⟨*proof*⟩

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)⟩)
⟨*proof*⟩

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*})

⟨*ML*⟩

definition *ifrangeF_body5* :: [i⇒o,i,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)⟩

⟨*ML*⟩

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))
⟨*proof*⟩

lemma (**in** *M_basic*) *is_ifrFb_body5_closed*: $M(G) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow$ *is_ifrFb_body5*$(M, G, r, s, x, i) \Longrightarrow M(i)$
⟨*proof*⟩

definition *toplevel6_body* :: $[i,i] \Rightarrow o$ **where**
toplevel6_body$(R) \equiv \lambda x.$ *domain*$(x) = R$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_toplevel6_body*:
$(\#\#M)(A) \Longrightarrow$ *separation*$(\#\#M,$ *is_toplevel6_body*$(\#\#M,A))$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *toplevel6_body_abs*:
assumes $(\#\#M)(R)$ $(\#\#M)(x)$
shows *is_toplevel6_body*$(\#\#M,R,x) \longleftrightarrow$ *toplevel6_body*(R,x)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_toplevel6_body*:
$(\#\#M)(R) \Longrightarrow$ *separation*
$(\#\#M, \lambda x.$ *domain*$(x) = R)$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *ifrangeF_body5_abs*:
assumes $(\#\#M)(A)$ $(\#\#M)(G)$ $(\#\#M)(r)$ $(\#\#M)(s)$ $(\#\#M)(x)$
shows *is_ifrangeF_body5*$(\#\#M,A,G,r,s,x) \longleftrightarrow$ *ifrangeF_body5*$(\#\#M,A,G,r,s,x)$
⟨*proof*⟩

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*$(\lambda x. \{xa \in G . x \in xa\}, b, f, i)\rangle)$
⟨*proof*⟩

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*$(p) = i\})$

⟨*ML*⟩

definition *ifrangeF_body6* :: $[i\Rightarrow o,i,i,i,i,i] \Rightarrow o$ **where**
ifrangeF_body6$(M,A,G,b,f) \equiv \lambda y. \exists x\in A. y = \langle x,\mu\ i.$ *ifrFb_body6*$(G,b,f,x,i)\rangle$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_ifrangeF_body6*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(G) \Longrightarrow (\#\#M)(r) \Longrightarrow (\#\#M)(s) \Longrightarrow separation(\#\#M, is\_ifrangeF\_body6(\#\#M,A,G,r,s))$
⟨*proof*⟩

lemma (**in** *M_basic*) *ifrFb_body6_closed*: $M(G) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow ifrFb\_body6(G, r, s, x, i) \longleftrightarrow M(i) \wedge ifrFb\_body6(G, r, s, x, i)$
⟨*proof*⟩

lemma (**in** *M_basic*) *is_ifrFb_body6_closed*: $M(G) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow is\_ifrFb\_body6(M, G, r, s, x, i) \Longrightarrow M(i)$
⟨*proof*⟩

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*⟩

lemma (**in** *M_ZF_trans*) *separation_ifrangeF_body6*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(G) \Longrightarrow (\#\#M)(b) \Longrightarrow (\#\#M)(f) \Longrightarrow$
$separation(\#\#M,$
$\lambda y. \exists x\in A. y = \langle x, \mu\ i. x \in if\_range\_F\_else\_F(\lambda a. \{p \in G . domain(p) = a\}, b, f, i)\rangle)$
⟨*proof*⟩

definition *ifrFb_body7* **where**
$ifrFb\_body7(B,D,A,b,f,x,i) \equiv x \in$
$(if\ b = 0\ then\ if\ i \in range(f)\ then$
$\{d \in D . \exists r\in A. restrict(r, B) = converse(f)\ `\ i \wedge d = domain(r)\}\ else\ 0$
$else\ \{d \in D . \exists r\in A. restrict(r, B) = i \wedge d = domain(r)\})$

⟨*ML*⟩

definition *ifrangeF_body7* :: $[i\Rightarrow o,i,i,i,i,i,i] \Rightarrow o$ **where**
$ifrangeF\_body7(M,A,B,D,G,b,f) \equiv \lambda y. \exists x\in A. y = \langle x,\mu\ i. ifrFb\_body7(B,D,G,b,f,x,i)\rangle$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_ifrangeF_body7*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow (\#\#M)(D) \Longrightarrow (\#\#M)(G) \Longrightarrow (\#\#M)(r) \Longrightarrow (\#\#M)(s) \Longrightarrow separation(\#\#M, is\_ifrangeF\_body7(\#\#M,A,B,D,G,r,s))$

⟨*proof*⟩

lemma (**in** *M_basic*) *ifrFb_body7_closed*: $M(B) \Longrightarrow M(D) \Longrightarrow M(G) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow$
ifrFb_body7$(B,D,G, r, s, x, i) \longleftrightarrow M(i) \wedge$ *ifrFb_body7*(B,D,G, r, s, x, i)
⟨*proof*⟩

lemma (**in** *M_basic*) *is_ifrFb_body7_closed*: $M(B) \Longrightarrow M(D) \Longrightarrow M(G) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow$
is_ifrFb_body7$(M, B,D,G, r, s, x, i) \Longrightarrow M(i)$
⟨*proof*⟩

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*$(\#\#M,A,B,D,G,r,s,x) \longleftrightarrow$ *ifrangeF_body7*$(\#\#M,A,B,D,G,r,s,x)$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_ifrangeF_body7*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow (\#\#M)(D) \Longrightarrow (\#\#M)(G) \Longrightarrow (\#\#M)(b) \Longrightarrow (\#\#M)(f) \Longrightarrow$
separation$(\#\#M,$
$\lambda y.\ \exists x{\in}A.\ y = \langle x, \mu\ i.\ x \in$ *if_range_F_else_F*$($*drSR_Y*$(B, D, G), b, f, i)\rangle)$
⟨*proof*⟩

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$(\#\#M, \lambda f\ y.$ *is_range*$(\#\#M,f,y))$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_range*:
strong_replacement$(\#\#M, \lambda f\ y.\ y =$ *range*$(f))$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_is_domain*:
strong_replacement$(\#\#M, \lambda f\ y.$ *is_domain*$(\#\#M,f,y))$

⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_domain*:
strong_replacement(##*M*, λ*f y*. *y* = *domain*(*f*))
⟨*proof*⟩

Alternatively, we can use closure under lambda and get the stronger version.

lemma (**in** *M_ZF_trans*) *lam_replacement_domain* : *lam_replacement*(##*M*, *domain*)
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *lam_replacement_fst* : *lam_replacement*(##*M*, *fst*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_fst'*:
strong_replacement(##*M*, λ*f y*. *y* = *fst*(*f*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *lam_replacement_domain1* : *lam_replacement*(##*M*, *domain*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *lam_replacement_snd* : *lam_replacement*(##*M*, *snd*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_snd'*:
strong_replacement(##*M*, λ*f y*. *y* = *snd*(*f*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *lam_replacement_Union* : *lam_replacement*(##*M*, *Union*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_Union'*:
strong_replacement(##*M*, λ*f y*. *y* = *Union*(*f*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *lam_replacement_Un*:
lam_replacement(##*M*, λ*p*. *fst*(*p*) $\cup$ *snd*(*p*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *lam_replacement_image*:
  lam_replacement(##*M*, *λp. fst*(*p*) *‘‘ snd*(*p*))
  ⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *lam_replacement_Diff*:
  lam_replacement(##*M*, *λp. fst*(*p*) - *snd*(*p*))
  ⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *minimum_closed*:
  assumes *B*∈*M*
  shows *minimum*(*r*,*B*) ∈ *M*
⟨*proof*⟩

⟨*ML*⟩
definition *is_minimum′* **where**
  is_minimum′(*M*,*R*,*X*,*u*) ≡ (*M*(*u*) ∧ *u* ∈ *X* ∧ (∀ *v*[*M*]. ∃ *a*[*M*]. (*v* ∈ *X* ⟶ *v* ≠ *u* ⟶ *a* ∈ *R*) ∧ *pair*(*M*, *u*, *v*, *a*))) ∧
    (∃ *x*[*M*].
      (*M*(*x*) ∧ *x* ∈ *X* ∧ (∀ *v*[*M*]. ∃ *a*[*M*]. (*v* ∈ *X* ⟶ *v* ≠ *x* ⟶ *a* ∈ *R*) ∧ *pair*(*M*, *x*, *v*, *a*))) ∧
      (∀ *y*[*M*]. *M*(*y*) ∧ *y* ∈ *X* ∧ (∀ *v*[*M*]. ∃ *a*[*M*]. (*v* ∈ *X* ⟶ *v* ≠ *y* ⟶ *a* ∈ *R*) ∧ *pair*(*M*, *y*, *v*, *a*)) ⟶ *y* = *x*)) ∨
    ¬ (∃ *x*[*M*]. (*M*(*x*) ∧ *x* ∈ *X* ∧ (∀ *v*[*M*]. ∃ *a*[*M*]. (*v* ∈ *X* ⟶ *v* ≠ *x* ⟶ *a* ∈ *R*) ∧ *pair*(*M*, *x*, *v*, *a*))) ∧
          (∀ *y*[*M*]. *M*(*y*) ∧ *y* ∈ *X* ∧ (∀ *v*[*M*]. ∃ *a*[*M*]. (*v* ∈ *X* ⟶ *v* ≠ *y* ⟶ *a* ∈ *R*) ∧ *pair*(*M*, *y*, *v*, *a*)) ⟶ *y* = *x*)) ∧
    empty(*M*, *u*)

⟨*ML*⟩

lemma *is_minimum_eq* :
  M(*R*) ⟹ *M*(*X*) ⟹ *M*(*u*) ⟹ *is_minimum*(*M*,*R*,*X*,*u*) ⟷ *is_minimum′*(*M*,*R*,*X*,*u*)
  ⟨*proof*⟩

context *M_trivial*
begin

lemma *first_closed*:
  M(*B*) ⟹ *M*(*r*) ⟹ *first*(*u*,*r*,*B*) ⟹ *M*(*u*)
  ⟨*proof*⟩

⟨*ML*⟩
  ⟨*proof*⟩

⟨*ML*⟩

⟨*proof*⟩

end

lemma (**in** *M_ZF_trans*) *lam_replacement_minimum*:
lam_replacement(##*M*, *λp*. *minimum*(*fst*(*p*), *snd*(*p*)))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *lam_replacement_Upair*:
lam_replacement(##*M*, *λp*. *Upair*(*fst*(*p*), *snd*(*p*)))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *lam_replacement_cartprod*:
lam_replacement(##*M*, *λp*. *fst*(*p*) × *snd*(*p*))
⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *lam_replacement_vimage*:
lam_replacement(##*M*, *λp*. *fst*(*p*) -“ *snd*(*p*))
⟨*proof*⟩

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*)

⟨*ML*⟩

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*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_is_omega_funspace*:
B∈*M* ⟹ *strong_replacement*(##*M*, *is_omega_funspace*(##*M*,*B*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_omega_funspace*:
b∈*M*⟹*strong_replacement*(##*M*, *λn z*. *n*∈*ω* ∧ *is_funspace*(##*M*,*n*,*b*,*z*))
⟨*proof*⟩

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*))

⟨*ML*⟩
lemma *arity_HAleph_wfrec_repl_body*: *arity*(*HAleph_wfrec_repl_body_fm*(*2*,*0*,*1*)) = *3*
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_HAleph_wfrec_repl_body*:
B∈*M* ⟹ *strong_replacement*(##*M*, *HAleph_wfrec_repl_body*(##*M*,*B*))
⟨*proof*⟩

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*)))
⟨*proof*⟩

definition *fst2_snd2*
  where *fst2_snd2*(*x*) ≡ ⟨*fst*(*fst*(*x*)), *snd*(*snd*(*x*))⟩

⟨*ML*⟩

lemma (**in** *M_trivial*) *fst2_snd2_abs*:
  assumes *M*(*x*) *M*(*res*)
shows *is_fst2_snd2*(*M*, *x*, *res*) ⟷ *res* = *fst2_snd2*(*x*)
  ⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *replacement_is_fst2_snd2*:
 strong_replacement(##*M*, *is_fst2_snd2*(##*M*))
  ⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_fst2_snd2*: *strong_replacement*(##*M*, λ*x*
y. *y* = ⟨*fst*(*fst*(*x*)), *snd*(*snd*(*x*))⟩)
  ⟨*proof*⟩

definition *fst2_sndfst_snd2*
  where *fst2_sndfst_snd2*(*x*) ≡ ⟨*fst*(*fst*(*x*)), *snd*(*fst*(*x*)), *snd*(*snd*(*x*))⟩

⟨*ML*⟩

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*)
  ⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *replacement_is_fst2_sndfst_snd2*:
 strong_replacement(##*M*, *is_fst2_sndfst_snd2*(##*M*))
  ⟨*proof*⟩

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*))⟩)
  ⟨*proof*⟩

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*
⟨*proof*⟩

definition *RepFun_body* :: *i* ⇒ *i* ⇒ *i***where**
RepFun_body(*u*,*v*) ≡ {{⟨*v*, *x*⟩} . *x* ∈ *u*}

⟨*ML*⟩

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*)
⟨*proof*⟩

⟨*ML*⟩
lemma *arity_body_repfun*:
arity(·(·∃·*0* = *0*··) ∧ ·(·∃·*0* = *0*··) ∧ (·∃·*cons_fm*(*0*, *3*, *2*) ∧ *pair_fm*(*5*, *1*, *0*) ··)··) = *5*
⟨*proof*⟩

lemma *arity_RepFun*: *arity*(*is_RepFun_body_fm*(*0*, *1*, *2*)) = *3*
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *RepFun_SigFun_closed*: *x* ∈ *M* ⟹ *z* ∈ *M* ⟹ {{⟨*z*, *x*⟩} . *x* ∈ *x*} ∈ *M*
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_RepFun_body*:
lam_replacement(*##M*, λ*p* . {{⟨*snd*(*p*), *x*⟩} . *x* ∈ *fst*(*p*)})
⟨*proof*⟩

sublocale *M_ZF_trans* ⊆ *M_replacement_extra ##M*
⟨*proof*⟩

sublocale *M_ZF_trans* ⊆ *M_Perm ##M*
⟨*proof*⟩

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*)

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *replacement_is_order_eq_map*:
A∈*M* ⟹ *r*∈*M* ⟹ *strong_replacement*(*##M*, *order_eq_map*(*##M*,*A*,*r*))

⟨*proof*⟩

⟨*ML*⟩

definition *banach_body_iterates* **where**
banach_body_iterates$(M,X,Y,f,g,W,n,x,z) \equiv$
$\exists\, y[M].$
pair$(M, x, y, z) \wedge$
$(\exists\, fa[M].$
$(\forall\, z[M].$
$z \in fa \longleftrightarrow$
$(\exists\, xa[M].$
$\exists\, y[M].$
$\exists\, xaa[M].$
$\exists\, sx[M].$
$\exists\, r\_sx[M].$
$\exists\, f\_r\_sx[M].\ \exists\, sn[M].\ \exists\, msn[M].$ *successor*(M,n,sn)
$\wedge$
membership$(M,sn,msn) \wedge$
pair$(M, xa, y, z) \wedge$
pair$(M, xa, x, xaa) \wedge$
upair$(M, xa, xa, sx) \wedge$
pre_image$(M, msn, sx, r\_sx) \wedge$
restriction$(M, fa, r\_sx, f\_r\_sx) \wedge$
$xaa \in msn \wedge$
$($*empty*$(M, xa) \longrightarrow y = W) \wedge$
$(\forall\, m[M].$
successor$(M, m, xa) \longrightarrow$
$(\exists\, gm[M].$
is_apply$(M, f\_r\_sx, m, gm) \wedge$
is_banach_functor$(M, X, Y, f, g, gm, y))) \wedge$
$($*is_quasinat*$(M, xa) \vee$ *empty*$(M, y)))) \wedge$
$($*empty*$(M, x) \longrightarrow y = W) \wedge$
$(\forall\, m[M].$
successor$(M, m, x) \longrightarrow$
$(\exists\, gm[M].$ *is_apply*$(M, fa, m, gm) \wedge$ *is_banach_functor*$(M,$
$X, Y, f, g, gm, y))) \wedge$
$($*is_quasinat*$(M, x) \vee$ *empty*$(M, y)))$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *banach_iterates*:
assumes $X\in M$ $Y\in M$ $f\in M$ $g\in M$ $W\in M$
shows *iterates_replacement*$(\#\#M,$ *is_banach_functor*$(\#\#M,X,Y,f,g), W)$
⟨*proof*⟩

definition *banach_is_iterates_body* **where**
banach_is_iterates_body$(M,X,Y,f,g,W,n,y) \equiv \exists\, om[M].$ *omega*$(M,om) \wedge n \in$

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*].
∃ *f_r_sx*[*M*].
pair(*M*, *x*, *y*, *z*) ∧
pair(*M*, *x*, *n*, *xa*) ∧
upair(*M*, *x*, *x*, *sx*) ∧
pre_image(*M*, *msn*, *sx*, *r_sx*) ∧
restriction(*M*, *fa*, *r_sx*, *f_r_sx*) ∧
xa ∈ *msn* ∧
(*empty*(*M*, *x*) ⟶ *y* = *W*) ∧
(∀ *m*[*M*].
successor(*M*, *m*, *x*) ⟶
(∃ *gm*[*M*].
fun_apply(*M*, *f_r_sx*, *m*, *gm*) ∧
is_banach_functor(*M*, *X*, *Y*, *f*, *g*, *gm*, *y*))) ∧
(*is_quasinat*(*M*, *x*) ∨ *empty*(*M*, *y*)))) ∧
(*empty*(*M*, *n*) ⟶ *y* = *W*) ∧
(∀ *m*[*M*].
successor(*M*, *m*, *n*) ⟶
(∃ *gm*[*M*]. *fun_apply*(*M*, *fa*, *m*, *gm*) ∧ *is_banach_functor*(*M*,
X, *Y*, *f*, *g*, *gm*, *y*))) ∧
(*is_quasinat*(*M*, *n*) ∨ *empty*(*M*, *y*))))

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *banach_replacement_iterates*:
assumes *X*∈*M* *Y*∈*M* *f*∈*M* *g*∈*M* *W*∈*M*
shows *strong_replacement*(##*M*, λ*n y*. *n*∈ω ∧ *is_iterates*(##*M*,*is_banach_functor*(##*M*,*X*, *Y*, *f*, *g*),*W*,*n*,*y*))
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *lam_replacement_cardinal* : *lam_replacement*(##*M*, *cardinal_rel*(##*M*))
⟨*proof*⟩

definition *trans_apply_image* **where**
trans_apply_image(*f*) ≡ λ*a g*. *f* ‘ (*g* ‘‘ *a*)

⟨*ML*⟩

schematic_goal *arity_is_recfun_fm*[*arity*]:
p ∈ *formula* ⟹ *a* ∈ ω ⟹ *z* ∈ ω ⟹ *r* ∈ ω ⟹ *arity*(*is_recfun_fm*(*p*, *a*, *z* ,*r*)) = *?ar*
⟨*proof*⟩

schematic_goal *arity_is_wfrec_fm*[*arity*]:
p ∈ *formula* ⟹ *a* ∈ ω ⟹ *z* ∈ ω ⟹ *r* ∈ ω ⟹ *arity*(*is_wfrec_fm*(*p*, *a*, *z* ,*r*)) = *?ar*
⟨*proof*⟩
schematic_goal *arity_is_transrec_fm*[*arity*]:
p ∈ *formula* ⟹ *a* ∈ ω ⟹ *z* ∈ ω ⟹ *arity*(*is_transrec_fm*(*p*, *a*, *z*)) = *?ar*
⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_basic*) *rel2_trans_apply*:
M(*f*) ⟹ *relation2*(*M*,*is_trans_apply_image*(*M*,*f*),*trans_apply_image*(*f*))
⟨*proof*⟩

lemma (**in** *M_basic*) *apply_image_closed*:
shows *M*(*f*) ⟹ ∀ *x*[*M*]. ∀ *g*[*M*]. *function*(*g*) ⟶ *M*(*trans_apply_image*(*f*, *x*, *g*))
⟨*proof*⟩

lemma (**in** *M_basic*) *apply_image_closed′*:
shows *M*(*f*) ⟹ ∀ *x*[*M*]. ∀ *g*[*M*]. *M*(*trans_apply_image*(*f*, *x*, *g*))
⟨*proof*⟩

definition *transrec_apply_image_body* **where**
transrec_apply_image_body(*M*,*f*,*mesa*,*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*].

$\exists f\_r\_sx[M].$
$pair(M, xa, y, z) \wedge$
$pair(M, xa, x, xaa) \wedge$
$upair(M, xa, xa, sx) \wedge$
$pre\_image(M, mesa, sx, r\_sx) \wedge$
$restriction(M, fa, r\_sx, f\_r\_sx) \wedge$
$xaa \in mesa \wedge is\_trans\_apply\_image(M,$
$f, xa, f\_r\_sx, y))) \wedge$
$is\_trans\_apply\_image(M, f, x, fa, y))$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *replacement_transrec_apply_image_body* :
$(\#\#M)(f) \Longrightarrow (\#\#M)(mesa) \Longrightarrow strong\_replacement(\#\#M,transrec\_apply\_image\_body(\#\#M,f,mesa))$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *transrec_replacement_apply_image*:
assumes $(\#\#M)(f)$ $(\#\#M)(\alpha)$
shows $transrec\_replacement(\#\#M, is\_trans\_apply\_image(\#\#M, f), \alpha)$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *rec_trans_apply_image_abs*:
assumes $(\#\#M)(f)$ $(\#\#M)(x)$ $(\#\#M)(y)$ $Ord(x)$
shows $is\_transrec(\#\#M,is\_trans\_apply\_image(\#\#M, f),x,y) \longleftrightarrow y = transrec(x,trans\_apply\_image(f))$
⟨*proof*⟩

definition *is_trans_apply_image_body* **where**
$is\_trans\_apply\_image\_body(M,f,\beta,a,w) \equiv \exists z[M]. pair(M,a,z,w) \wedge a{\in}\beta \wedge (\exists sa[M].$
$\exists esa[M].$
$\exists mesa[M].$
$upair(M, a, a, sa) \wedge$
$is\_eclose(M, sa, esa) \wedge$
$membership(M, esa, mesa) \wedge$
$(\exists fa[M].$
$(\forall z[M].$
$z \in fa \longleftrightarrow$
$(\exists x[M].$
$\exists y[M].$
$\exists xa[M].$
$\exists sx[M].$
$\exists r\_sx[M].$
$\exists f\_r\_sx[M].$
$pair(M, x, y, z) \wedge$
$pair(M, x, a, xa) \wedge$
$upair(M, x, x, sx) \wedge$
$pre\_image(M, mesa, sx, r\_sx) \wedge$
$restriction(M, fa, r\_sx, f\_r\_sx) \wedge$
$xa \in mesa \wedge is\_trans\_apply\_image(M, f,$
$x, f\_r\_sx, y))) \wedge$

is_trans_apply_image(*M*, *f*, *a*, *fa*, *z*)))

⟨*ML*⟩
⟨*proof*⟩

⟨*ML*⟩

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*)))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *trans_apply_abs*:
(##*M*)(*f*) ⟹ (##*M*)(*β*) ⟹ *Ord*(*β*) ⟹ (##*M*)(*x*) ⟹ (##*M*)(*z*) ⟹
(*x*∈*β* ∧ *z* = ⟨*x*, *transrec*(*x*, *λa g*. *f* ‘ (*g* ‘‘ *a*)) ⟩) ⟷
(∃ *y*[##*M*]. *pair*(##*M*,*x*,*y*,*z*) ∧ *x*∈*β* ∧ (*is_transrec*(##*M*,*is_trans_apply_image*(##*M*, *f*),*x*,*y*)))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_trans_apply_image*:
(##*M*)(*f*) ⟹ (##*M*)(*β*) ⟹ *Ord*(*β*) ⟹
strong_replacement(##*M*, *λx y*. *x*∈*β* ∧ *y* = ⟨*x*, *transrec*(*x*, *λa g*. *f* ‘ (*g* ‘‘ *a*))⟩)
⟨*proof*⟩

definition *abs_apply_pair* **where**
abs_apply_pair(*A*,*f*,*x*) ≡ ⟨*x*, *λn*∈*A*. *f* ‘ ⟨*x*, *n*⟩⟩

⟨*ML*⟩

lemma (**in** *M_basic*) *abs_apply_pair_rel*:
assumes *M*(*A*) *M*(*f*) *M*(*x*)
shows *Relation1*(*M*,*A*,*λn a*. ∃ *b*[*M*]. *is_apply*(*M*, *f*, *b*, *a*) ∧ *pair*(*M*, *x*, *n*, *b*), *λn*. *f* ‘ ⟨*x*, *n*⟩)
⟨*proof*⟩

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*) ⟷ *res* = *abs_apply_pair*(*A*,*f*,*x*)
⟨*proof*⟩

⟨*ML*⟩

lemma *arity_is_abs_aux*: *arity*((·∃ ··*7*‘*0* *is* *1*· ∧ *pair_fm*(*5*, *2*, *0*) ··)) = *7*
⟨*proof*⟩

lemma *arity_is_abs_apply_pair_fm* :
shows *arity*(*is_abs_apply_pair_fm*(*3*, *2*, *0*, *1*)) = *4*
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_is_abs_apply_pair*:
assumes $A \in M$ $f \in M$
shows *strong_replacement*(##*M*, *is_abs_apply_pair*(##*M*,*A*,*f*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_abs_apply_pair*:
(##*M*)(*A*) $\Longrightarrow$ (##*M*)(*f*) $\Longrightarrow$ *strong_replacement*(##*M*, $\lambda x\ y.\ y = \langle x, \lambda n \in A.\ f\ `\ \langle x, n\rangle\rangle$)
⟨*proof*⟩

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*⟩

end

locale *ctm_separative* = *forcing_data* + *separative_notion*
begin

lemma *generic_not_in_M*: **assumes** *M_generic*(*G*) **shows** $G \notin M$
⟨*proof*⟩

theorem *proper_extension*: **assumes** *M_generic*(*G*) **shows** $M \neq M[G]$
⟨*proof*⟩

end

end

30 A poset of successions

theory *Succession_Poset*
  imports
    Replacement_Instances
    Proper_Extension
    FiniteFun_Relative

begin

sublocale *M_ZF_trans* ⊆ *M_seqspace* ##*M*
  ⟨*proof*⟩

definition *seq_upd* :: *i* ⇒ *i* ⇒ *i* **where**
  $seq\_upd(f,a) \equiv \lambda\ j \in succ(domain(f))\ .\ if\ j < domain(f)\ then\ f`j\ else\ a$

lemma *seq_upd_succ_type* :
  assumes $n \in nat$ $f \in n \to A$ $a \in A$
  shows $seq\_upd(f,a) \in succ(n) \to A$
⟨*proof*⟩

lemma *seq_upd_type* :
  assumes $f \in A^{<\omega}$ $a \in A$
  shows $seq\_upd(f,a) \in A^{<\omega}$
⟨*proof*⟩

lemma *seq_upd_apply_domain* [*simp*]:
  assumes $f{:}n \to A$ $n \in nat$
  shows $seq\_upd(f,a)`n = a$
  ⟨*proof*⟩

lemma *zero_in_seqspace* :
  shows $0 \in A^{<\omega}$
  ⟨*proof*⟩

definition
  seqleR :: *i* ⇒ *i* ⇒ *o* **where**
  $seqleR(f,g) \equiv g \subseteq f$

definition
  seqlerel :: *i* ⇒ *i* **where**
  $seqlerel(A) \equiv Rrel(\lambda x\ y.\ y \subseteq x, A^{<\omega})$

definition
  seqle :: *i* **where**
  $seqle \equiv seqlerel(2)$

lemma *seqleI*[*intro*!]:
  $\langle f,g\rangle \in 2^{<\omega} \times 2^{<\omega} \Longrightarrow g \subseteq f \Longrightarrow \langle f,g\rangle \in seqle$

⟨proof⟩

lemma *seqleD*[*dest*!]:
$z \in seqle \Longrightarrow \exists x\ y.\ \langle x,y\rangle \in 2^{<\omega}\times 2^{<\omega} \wedge y \subseteq x \wedge z = \langle x,y\rangle$
⟨proof⟩

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⟩

lemma *preorder_on_seqle*: $preorder\_on(2^{<\omega},seqle)$
⟨proof⟩

lemma *zero_seqle_max*: $x\in 2^{<\omega} \Longrightarrow \langle x,0\rangle \in seqle$
⟨proof⟩

interpretation *sp*:*forcing_notion* $2^{<\omega}$ *seqle 0*
⟨proof⟩

notation *sp.Leq* (**infixl** $\preceq s$ *50*)
notation *sp.Incompatible* (**infixl** $\bot s$ *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⟩

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* :
$fg \in nat \Longrightarrow seqleR\_fm(fg) \in formula$
⟨proof⟩

lemma *arity_seqleR_fm* :
$fg \in nat \Longrightarrow arity(seqleR\_fm(fg)) = succ(fg)$
⟨proof⟩

lemma (**in** *M_basic*) *seqleR_abs*:
assumes $M(f)$ $M(g)$
shows $seqleR(f,g) \longleftrightarrow is\_seqleR(M,f,g)$
⟨proof⟩

definition
$relP :: [i\Rightarrow o,[i\Rightarrow o,i,i]\Rightarrow o,i] \Rightarrow o$ **where**

relP(*M*,*r*,*xy*) ≡ (∃ *x*[*M*]. ∃ *y*[*M*]. *pair*(*M*,*x*,*y*,*xy*) ∧ *r*(*M*,*x*,*y*))

lemma (**in** *M_ctm*) *seqleR_fm_sats* :
assumes *fg*∈*nat* *env*∈*list*(*M*)
shows *sats*(*M*,*seqleR_fm*(*fg*),*env*) ⟷ *relP*(##*M*,*is_seqleR*,*nth*(*fg*, *env*))
⟨*proof*⟩

lemma (**in** *M_basic*) *is_related_abs* :
assumes ⋀ *f g* . *M*(*f*) ⟹ *M*(*g*) ⟹ *rel*(*f*,*g*) ⟷ *is_rel*(*M*,*f*,*g*)
shows ⋀*z* . *M*(*z*) ⟹ *relP*(*M*,*is_rel*,*z*) ⟷ (∃ *x y*. *z* = ⟨*x*,*y*⟩ ∧ *rel*(*x*,*y*))
⟨*proof*⟩

definition
is_RRel :: [*i*⇒*o*,[*i*⇒*o*,*i*,*i*]⇒*o*,*i*,*i*] ⇒ *o* **where**
is_RRel(*M*,*is_r*,*A*,*r*) ≡ ∃ *A2*[*M*]. *cartprod*(*M*,*A*,*A*,*A2*) ∧ *is_Collect*(*M*,*A2*, *relP*(*M*,*is_r*),*r*)

lemma (**in** *M_basic*) *is_Rrel_abs* :
assumes *M*(*A*) *M*(*r*)
⋀ *f g* . *M*(*f*) ⟹ *M*(*g*) ⟹ *rel*(*f*,*g*) ⟷ *is_rel*(*M*,*f*,*g*)
shows *is_RRel*(*M*,*is_rel*,*A*,*r*) ⟷ *r* = *Rrel*(*rel*,*A*)
⟨*proof*⟩

definition
is_seqlerel :: [*i*⇒*o*,*i*,*i*] ⇒ *o* **where**
is_seqlerel(*M*,*A*,*r*) ≡ *is_RRel*(*M*,*is_seqleR*,*A*,*r*)

lemma (**in** *M_basic*) *seqlerel_abs* :
assumes *M*(*A*) *M*(*r*)
shows *is_seqlerel*(*M*,*A*,*r*) ⟷ *r* = *Rrel*(*seqleR*,*A*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

lemma *seqle_in_M*: *seqle* ∈ *M*
⟨*proof*⟩

30.1 Cohen extension is proper

interpretation *ctm_separative* $2^{<\omega}$ *seqle 0*
⟨*proof*⟩

lemma *cohen_extension_is_proper*: ∃ *G*. *M_generic*(*G*) ∧ *M* ≠ $M^{2^{<\omega}}[G]$
⟨*proof*⟩

end

end

31 The ZFC axioms, internalized

theory *Internal_ZFC_Axioms*
imports
Forcing_Data

begin

schematic_goal *ZF_union_auto*:
Union_ax(##*A*) ⟷ (*A*, [] ⊨ *?zfunion*)
⟨*proof*⟩

⟨*ML*⟩
notation *ZF_union_fm* (‹·*Union Ax*·›)

schematic_goal *ZF_power_auto*:
power_ax(##*A*) ⟷ (*A*, [] ⊨ *?zfpow*)
⟨*proof*⟩

⟨*ML*⟩
notation *ZF_power_fm* (‹·*Powerset Ax*·›)

schematic_goal *ZF_pairing_auto*:
upair_ax(##*A*) ⟷ (*A*, [] ⊨ *?zfpair*)
⟨*proof*⟩

⟨*ML*⟩
notation *ZF_pairing_fm* (‹·*Pairing*·›)

schematic_goal *ZF_foundation_auto*:
foundation_ax(##*A*) ⟷ (*A*, [] ⊨ *?zffound*)
⟨*proof*⟩

⟨*ML*⟩
notation *ZF_foundation_fm* (‹·*Foundation*·›)

schematic_goal *ZF_extensionality_auto*:
extensionality(##*A*) ⟷ (*A*, [] ⊨ *?zfext*)
⟨*proof*⟩

⟨*ML*⟩
notation *ZF_extensionality_fm* (‹·*Extensionality*·›)

schematic_goal *ZF_infinity_auto*:
infinity_ax(##*A*) ⟷ (*A*, [] ⊨ (*?φ*(*i*,*j*,*h*)))
⟨*proof*⟩

⟨*ML*⟩
notation *ZF_infinity_fm* (‹·*Infinity*·›)

schematic_goal *ZF_choice_auto*:
choice_ax(##*A*) ⟷ (*A*, [] ⊨ (*?φ*(*i*,*j*,*h*)))
⟨*proof*⟩

⟨*ML*⟩
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*
⟨*proof*⟩

31.1 The Axiom of Separation, internalized

lemma *iterates_Forall_type* [*TC*]:
⟦ *n* ∈ *nat*; *p* ∈ *formula* ⟧ ⟹ *Forall*^*n*(*p*) ∈ *formula*
⟨*proof*⟩

lemma *last_init_eq* :

assumes $l \in list(A)$ $length(l) = succ(n)$
shows $\exists\ a{\in}A.\ \exists\, l'{\in}list(A).\ l = l'@[a]$
⟨*proof*⟩

lemma *take_drop_eq* :
assumes $l{\in}list(M)$
shows $\bigwedge\ n\ .\ n < succ(length(l)) \Longrightarrow l = take(n,l)\ @\ drop(n,l)$
⟨*proof*⟩

lemma *list_split* :
assumes $n \le succ(length(rest))$ $rest \in list(M)$
shows $\exists\, re{\in}list(M).\ \exists\, st{\in}list(M).\ rest = re\ @\ st \wedge length(re) = pred(n)$
⟨*proof*⟩

lemma *sats_nForall*:
assumes
$\varphi \in formula$
shows
$n{\in}nat \Longrightarrow ms \in list(M) \Longrightarrow$
$(M,\ ms \models (Forall\widehat{\ }n(\varphi))) \longleftrightarrow$
$(\forall\, rest \in list(M).\ length(rest) = n \longrightarrow M,\ rest\ @\ ms \models \varphi)$
⟨*proof*⟩

definition
sep_body_fm :: $i \Rightarrow i$ **where**
$sep\_body\_fm(p) \equiv (\cdot\forall(\cdot\exists(\cdot\forall\cdot\cdot 0 \in 1\cdot \leftrightarrow \cdot\cdot 0 \in 2\cdot \wedge incr\_bv1\widehat{\ }2\ (p)\ \cdot\cdot\cdot)\cdot)\cdot)$

lemma *sep_body_fm_type* [*TC*]: $p \in formula \Longrightarrow sep\_body\_fm(p) \in formula$
⟨*proof*⟩

lemma *sats_sep_body_fm*:
assumes
$\varphi \in formula$ $ms{\in}list(M)$ $rest{\in}list(M)$
shows
$(M,\ rest\ @\ ms \models sep\_body\_fm(\varphi)) \longleftrightarrow$
$separation(\#\#M,\lambda x.\ M,\ [x]\ @\ rest\ @\ ms \models \varphi)$
⟨*proof*⟩

definition
ZF_separation_fm :: $i \Rightarrow i$ (‹·Separation′(_′)·›) **where**
$ZF\_separation\_fm(p) \equiv Forall\widehat{\ }(pred(arity(p)))(sep\_body\_fm(p))$

lemma *ZF_separation_fm_type* [*TC*]: $p \in formula \Longrightarrow ZF\_separation\_fm(p) \in formula$
⟨*proof*⟩

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*⟩

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
$univalent(\#\#A,B,Q) \longleftrightarrow A,env \models ?ufm(i)$
⟨*proof*⟩

⟨*ML*⟩

lemma *univalent_fm_type* [*TC*]: $q1\in formula \Longrightarrow q2\in formula \Longrightarrow i\in nat \Longrightarrow$
$univalent\_fm(q2,q1,i) \in formula$
⟨*proof*⟩

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
$(A,env \models univalent\_fm(Q1\_fm,Q2\_fm,i)) \longleftrightarrow univalent(\#\#A,B,Q)$
⟨*proof*⟩

definition
$swap\_vars :: i\Rightarrow i$ **where**
$swap\_vars(\varphi) \equiv$
$Exists(Exists(And(Equal(0,3),And(Equal(1,2),iterates(\lambda p.\ incr\_bv(p)`2\ ,\ 2,$
$\varphi)))))$

lemma *swap_vars_type*[*TC*] :
$\varphi\in formula \Longrightarrow swap\_vars(\varphi) \in formula$
⟨*proof*⟩

lemma *sats_swap_vars* :
  $[x,y]$ @ *env* ∈ *list*(*M*) ⟹ φ∈*formula* ⟹
    (*M*, $[x,y]$ @ *env* ⊨ *swap_vars*(φ)) ⟷ *M*,$[y,x]$ @ *env* ⊨ φ
  ⟨*proof*⟩

definition
  univalent_Q1 :: *i* ⇒ *i* **where**
  univalent_Q1(φ) ≡ *incr_bv1*(*swap_vars*(φ))

definition
  univalent_Q2 :: *i* ⇒ *i* **where**
  univalent_Q2(φ) ≡ *incr_bv*(*swap_vars*(φ))‘*0*

lemma *univalent_Qs_type* [*TC*]:
  assumes φ∈*formula*
  shows *univalent_Q1*(φ) ∈ *formula* *univalent_Q2*(φ) ∈ *formula*
  ⟨*proof*⟩

lemma *sats_univalent_fm_assm*:
  assumes
    x ∈ *A* *y* ∈ *A* *z*∈*A* *env*∈ *list*(*A*) φ ∈ *formula*
  shows
  (*A*, ($[x,z]$ @ *env*) ⊨ φ) ⟷ (*A*, *Cons*(*z*,*Cons*(*y*,*Cons*(*x*,*env*))) ⊨ (*univalent_Q1*(φ)))
  (*A*, ($[x,y]$ @ *env*) ⊨ φ) ⟷ (*A*, *Cons*(*z*,*Cons*(*y*,*Cons*(*x*,*env*))) ⊨ (*univalent_Q2*(φ)))
  ⟨*proof*⟩

definition
  rep_body_fm :: *i* ⇒ *i* **where**
  rep_body_fm(*p*) ≡ *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* ∈ *formula* ⟹ *rep_body_fm*(*p*) ∈ *formula*
  ⟨*proof*⟩

lemmas *ZF_replacement_simps* = *formula_add_params1*[*of* φ *2* _ *M* [_,_]]
  sats_incr_bv_iff[*of* _ _ *M* _ []] — simplifies iterates of λ*x*. *incr_bv*(*x*) ‘ *0*
  sats_incr_bv_iff[*of* _ _ *M* _ [_,_]]— simplifies λ*x*. *incr_bv*(*x*) ‘ *2*
  sats_incr_bv1_iff[*of* _ _ *M*] *sats_swap_vars* **for** φ *M*

lemma *sats_rep_body_fm*:
  assumes
    φ ∈ *formula* *ms*∈*list*(*M*) *rest*∈*list*(*M*)
  shows
    (*M*, *rest* @ *ms* ⊨ *rep_body_fm*(φ)) ⟷
     strong_replacement(##*M*,λ*x y*. *M*, $[x,y]$ @ *rest* @ *ms* ⊨ φ)
  ⟨*proof*⟩

definition
ZF_replacement_fm :: *i* ⇒ *i* (‹·*Replacement*′(_′)·›) **where**
ZF_replacement_fm(*p*) ≡ *Forall*^(*pred*(*pred*(*arity*(*p*))))(*rep_body_fm*(*p*))

lemma *ZF_replacement_fm_type* [*TC*]: *p* ∈ *formula* ⟹ *ZF_replacement_fm*(*p*) ∈ *formula*
⟨*proof*⟩

lemma *sats_ZF_replacement_fm_iff*:
assumes
φ∈*formula*
shows
(*M*, [] ⊨ ·*Replacement*(*φ*)·)
⟷
(∀ *env*∈*list*(*M*). *arity*(*φ*) ≤ *2* #+ *length*(*env*) ⟶
strong_replacement(##*M*,λ*x y*. *M*,[*x*,*y*] @ *env* ⊨ *φ*))
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *ZF_inf_subset_formula* : *ZF_inf* ⊆ *formula*
⟨*proof*⟩

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*
⟨*proof*⟩

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 $\bigwedge\varphi.\ \varphi\in\Phi \Longrightarrow A,[] \models \varphi$
  shows $A \models \Phi$
  ⟨*proof*⟩

lemma *satTD* [*dest*] :$A \models \Phi \Longrightarrow\ \ \varphi\in\Phi \Longrightarrow A,[] \models \varphi$
  ⟨*proof*⟩

lemma *sats_ZFC_iff_sats_ZF_AC*:
  $(N \models ZFC) \longleftrightarrow (N \models ZF) \wedge (N,\ [] \models \cdot AC\cdot)$
    ⟨*proof*⟩

lemma *M_ZF_iff_M_satT*: *M_ZF*(M) ⟷ ($M \models ZF$)
⟨*proof*⟩

lemma *M_ZFC_iff_M_satT*:
  notes *iff_trans*[*trans*]
  shows *M_ZFC*(M) ⟷ ($M \models ZFC$)
⟨*proof*⟩

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 :: [$i \Rightarrow o$,i] ⇒ o **where**
  frecrelP(M,xy) ≡ (∃ x[M]. ∃ y[M]. *pair*(M,x,y,xy) ∧ *is_frecR*(M,x,y))

⟨*ML*⟩

lemma *arity_frecrelP_fm* :
  $a\in nat \Longrightarrow$ *arity*(*frecrelP_fm*(a)) = *succ*(a)
  ⟨*proof*⟩

definition
  is_frecrel :: [$i \Rightarrow o$,i,i] ⇒ o **where**
 is_frecrel(M,A,r) ≡ ∃ $A2$[M]. *cartprod*(M,A,A,$A2$) ∧ *is_Collect*(M,$A2$, *frecrelP*(M)
,r)

declare *cartprod_iff_sats* [*iff_sats*]
declare *Collect_iff_sats* [*iff_sats*]
⟨*ML*⟩

lemma *arity_frecrel_fm* :
assumes $a \in nat$ $b \in nat$
shows $arity(frecrel\_fm(a,b)) = succ(a) \cup succ(b)$
⟨*proof*⟩

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**
$x = \langle f,n1,n2,p\rangle$ $f \in 2$ $n1 \in ecloseN(z)$ $n2 \in ecloseN(z)$ $p \in P$
⟨*proof*⟩

⟨*ML*⟩

lemma *number2_iff* :
$(A)(c) \Longrightarrow number2(A,c) \longleftrightarrow (\exists\, b[A].\ \exists\, a[A].\ successor(A,\ b,\ c) \wedge successor(A,\ a,\ b) \wedge empty(A,\ a))$
⟨*proof*⟩

lemma *arity_number2_fm* :
$a \in nat \Longrightarrow arity(number2\_fm(a)) = succ(a)$
⟨*proof*⟩

⟨*ML*⟩

lemma *arity_is_names_below_fm* :
$[\![P \in nat;x \in nat;nb \in nat]\!] \Longrightarrow arity(is\_names\_below\_fm(P,x,nb)) = succ(P) \cup succ(x) \cup succ(nb)$
⟨*proof*⟩

definition
is_tuple :: $[i \Rightarrow o,i,i,i,i,i] \Rightarrow o$ **where**
$is\_tuple(M,z,t1,t2,p,t) \equiv \exists\, t1t2p[M].\ \exists\, t2p[M].\ pair(M,t2,p,t2p) \wedge pair(M,t1,t2p,t1t2p) \wedge$
$pair(M,z,t1t2p,t)$

⟨*ML*⟩

lemma *arity_is_tuple_fm* : $[\![\ z \in nat\ ;\ t1 \in nat\ ;\ t2 \in nat\ ;\ p \in nat\ ;\ tup \in nat\ ]\!] \Longrightarrow$
$arity(is\_tuple\_fm(z,t1,t2,p,tup)) = \bigcup\ \{succ(z),succ(t1),succ(t2),succ(p),succ(tup)\}$
⟨*proof*⟩

32.2 Definition of *forces* for equality and membership

definition

eq_case :: [*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
eq_case(*t1*,*t2*,*p*,*P*,*leq*,*f*) ≡ ∀ *s*. *s*∈*domain*(*t1*) ∪ *domain*(*t2*) ⟶
(∀ *q*. *q*∈*P* ∧ ⟨*q*,*p*⟩∈*leq* ⟶ (*f*'⟨*1*,*s*,*t1*,*q*⟩=*1* ⟷ *f*'⟨*1*,*s*,*t2*,*q*⟩ =*1*))

⟨*ML*⟩

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*) ∪ *succ*(*n2*) ∪ *succ*(*p*) ∪ *succ*(*P*) ∪ *succ*(*leq*) ∪ *succ*(*f*)
⟨*proof*⟩

definition
mem_case :: [*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
mem_case(*t1*,*t2*,*p*,*P*,*leq*,*f*) ≡ ∀ *v*∈*P*. ⟨*v*,*p*⟩∈*leq* ⟶
(∃ *q*. ∃ *s*. ∃ *r*. *r*∈*P* ∧ *q*∈*P* ∧ ⟨*q*,*v*⟩∈*leq* ∧ ⟨*s*,*r*⟩ ∈ *t2* ∧ ⟨*q*,*r*⟩∈*leq* ∧ *f*'⟨*0*,*t1*,*s*,*q*⟩ = *1*)

⟨*ML*⟩

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*)
⟨*proof*⟩

definition
Hfrc :: [*i*,*i*,*i*,*i*] ⇒ *o* **where**
Hfrc(*P*,*leq*,*fnnc*,*f*) ≡ ∃ *ft*. ∃ *n1*. ∃ *n2*. ∃ *c*. *c*∈*P* ∧ *fnnc* = ⟨*ft*,*n1*,*n2*,*c*⟩ ∧
(*ft* = *0* ∧ *eq_case*(*n1*,*n2*,*c*,*P*,*leq*,*f*)
∨ *ft* = *1* ∧ *mem_case*(*n1*,*n2*,*c*,*P*,*leq*,*f*))

⟨*ML*⟩

lemma *arity_Hfrc_fm* :
assumes
P∈*nat* *leq*∈*nat* *fnnc*∈*nat* *f*∈*nat*
shows
arity(*Hfrc_fm*(*P*,*leq*,*fnnc*,*f*)) = *succ*(*P*) ∪ *succ*(*leq*) ∪ *succ*(*fnnc*) ∪ *succ*(*f*)
⟨*proof*⟩

definition
is_Hfrc_at :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_Hfrc_at(*M*,*P*,*leq*,*fnnc*,*f*,*z*) ≡

(*empty*(*M*,*z*) ∧ ¬ *is_Hfrc*(*M*,*P*,*leq*,*fnnc*,*f*))
∨ (*number1*(*M*,*z*) ∧ *is_Hfrc*(*M*,*P*,*leq*,*fnnc*,*f*))

⟨*ML*⟩

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*) ∪ *succ*(*leq*) ∪ *succ*(*fnnc*) ∪ *succ*(*f*) ∪ *succ*(*z*)
⟨*proof*⟩

32.3 The well-founded relation *forcerel*

definition
forcerel :: *i* ⇒ *i* ⇒ *i* **where**
forcerel(*P*,*x*) ≡ *frecrel*(*names_below*(*P*,*x*))^+

definition
is_forcerel :: [*i*⇒*o*,*i*,*i*,*i*] ⇒ *o* **where**
is_forcerel(*M*,*P*,*x*,*z*) ≡ ∃ *r*[*M*]. ∃ *nb*[*M*]. *tran_closure*(*M*,*r*,*z*) ∧
(*is_names_below*(*M*,*P*,*x*,*nb*) ∧ *is_frecrel*(*M*,*nb*,*r*))

definition
forcerel_fm :: *i*⇒ *i* ⇒ *i* ⇒ *i* **where**
forcerel_fm(*p*,*x*,*z*) ≡ *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*) ∪ *succ*(*x*) ∪ *succ*(*z*)
⟨*proof*⟩

lemma *forcerel_fm_type*[*TC*]:
⟦*p*∈*nat*;*x*∈*nat*;*z*∈*nat*⟧ ⟹ *forcerel_fm*(*p*,*x*,*z*)∈*formula*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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*)
  ⟨*proof*⟩

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*)
  ⟨*proof*⟩

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*))
  ⟨*proof*⟩

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*))
  ⟨*proof*⟩

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*))
  ⟨*proof*⟩

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*))
  ⟨*proof*⟩

context *forcing_data*
begin

lemma *ftype_abs*:

$[\![x\in M;\ y\in M\ ]\!] \Longrightarrow$ *is_ftype*$(\#\#M,x,y) \longleftrightarrow y = $ *ftype*(x)
⟨*proof*⟩

lemma *name1_abs*:
$[\![x\in M;\ y\in M\ ]\!] \Longrightarrow$ *is_name1*$(\#\#M,x,y) \longleftrightarrow y = $ *name1*(x)
⟨*proof*⟩

lemma *snd_snd_abs*:
$[\![x\in M;\ y\in M\ ]\!] \Longrightarrow$ *is_snd_snd*$(\#\#M,x,y) \longleftrightarrow y = $ *snd*$($*snd*$(x))$
⟨*proof*⟩

lemma *name2_abs*:
$[\![x\in M;\ y\in M\ ]\!] \Longrightarrow$ *is_name2*$(\#\#M,x,y) \longleftrightarrow y = $ *name2*(x)
⟨*proof*⟩

lemma *cond_of_abs*:
$[\![x\in M;\ y\in M\ ]\!] \Longrightarrow$ *is_cond_of*$(\#\#M,x,y) \longleftrightarrow y = $ *cond_of*(x)
⟨*proof*⟩

lemma *tuple_abs*:
$[\![z\in M;t1\in M;t2\in M;p\in M;t\in M]\!] \Longrightarrow$
is_tuple$(\#\#M,z,t1,t2,p,t) \longleftrightarrow t = \langle z,t1,t2,p\rangle$
⟨*proof*⟩

lemmas *components_abs* = *ftype_abs name1_abs name2_abs cond_of_abs tuple_abs*

lemma *oneN_in_M*[*simp*]: $1\in M$
⟨*proof*⟩

lemma *twoN_in_M* : $2\in M$
⟨*proof*⟩

lemma *comp_in_M*:
$p \preceq q \Longrightarrow p\in M$
$p \preceq q \Longrightarrow q\in M$
⟨*proof*⟩

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*⟩

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*⟩

lemma *Hfrc_abs*:
⟦*fnnc*∈*M*; *f*∈*M*⟧ ⟹
is_Hfrc(##*M*,*P*,*leq*,*fnnc*,*f*) ⟷ *Hfrc*(*P*,*leq*,*fnnc*,*f*)
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *components_closed* :
x∈*M* ⟹ (##*M*)(*ftype*(*x*))
x∈*M* ⟹ (##*M*)(*name1*(*x*))
x∈*M* ⟹ (##*M*)(*name2*(*x*))
x∈*M* ⟹ (##*M*)(*cond_of*(*x*))
⟨*proof*⟩

lemma *ecloseN_closed*:
(##*M*)(*A*) ⟹ (##*M*)(*ecloseN*(*A*))
(##*M*)(*A*) ⟹ (##*M*)(*eclose_n*(*name1*,*A*))
(##*M*)(*A*) ⟹ (##*M*)(*eclose_n*(*name2*,*A*))
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *ecloseN_abs* :
⟦*x*∈*M*;*ec*∈*M*⟧ ⟹ *is_ecloseN*(##*M*,*x*,*ec*) ⟷ *ec* = *ecloseN*(*x*)
⟨*proof*⟩

lemma *frecR_abs* :
x∈*M* ⟹ *y*∈*M* ⟹ *frecR*(*x*,*y*) ⟷ *is_frecR*(##*M*,*x*,*y*)
⟨*proof*⟩

lemma *frecrelP_abs* :
z∈*M* ⟹ *frecrelP*(##*M*,*z*) ⟷ (∃ *x* *y*. *z* = ⟨*x*,*y*⟩ ∧ *frecR*(*x*,*y*))
⟨*proof*⟩

lemma *frecrel_abs*:

assumes
$A{\in}M$ $r{\in}M$
shows
is_frecrel$(\#\#M,A,r) \longleftrightarrow r = $ *frecrel*(A)
⟨*proof*⟩

lemma *frecrel_closed*:
assumes
$x{\in}M$
shows
frecrel$(x){\in}M$
⟨*proof*⟩

lemma *field_frecrel* : *field*(*frecrel*(*names_below*(P,x))) $\subseteq$ *names_below*(P,x)
⟨*proof*⟩

lemma *forcerelD* : $uv \in$ *forcerel*$(P,x) \Longrightarrow uv{\in}$ *names_below*$(P,x) \times$ *names_below*(P,x)
⟨*proof*⟩

lemma *wf_forcerel* :
wf(*forcerel*(P,x))
⟨*proof*⟩

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
⟨*proof*⟩

lemma *names_belowI* :
assumes *frecR*$(\langle ft,n1,n2,p\rangle,\langle a,b,c,d\rangle)$ $p{\in}P$
shows $\langle ft,n1,n2,p\rangle \in$ *names_below*$(P,\langle a,b,c,d\rangle)$ (**is** *?x* $\in$ *names_below*(_,*?y*))
⟨*proof*⟩

lemma *names_below_tr* :
assumes $x{\in}$ *names_below*(P,y)
$y{\in}$ *names_below*(P,z)
shows $x{\in}$ *names_below*(P,z)
⟨*proof*⟩

lemma *arg_into_names_below2* :
assumes $\langle x,y\rangle \in$ *frecrel*(*names_below*(P,z))
shows $x \in$ *names_below*(P,y)
⟨*proof*⟩

lemma *arg_into_names_below* :
assumes $\langle x,y\rangle \in$ *frecrel*(*names_below*(P,z))
shows $x \in$ *names_below*(P,x)
⟨*proof*⟩

lemma *forcerel_arg_into_names_below* :
assumes $\langle x,y\rangle \in$ *forcerel*(P,z)
shows $x \in$ *names_below*(P,x)
⟨*proof*⟩

lemma *names_below_mono* :
assumes $\langle x,y\rangle \in$ *frecrel*(*names_below*(P,z))
shows *names_below*$(P,x) \subseteq$ *names_below*(P,y)
⟨*proof*⟩

lemma *frecrel_mono* :
assumes $\langle x,y\rangle \in$ *frecrel*(*names_below*(P,z))
shows *frecrel*(*names_below*(P,x)) $\subseteq$ *frecrel*(*names_below*(P,y))
⟨*proof*⟩

lemma *forcerel_mono2* :
assumes $\langle x,y\rangle \in$ *frecrel*(*names_below*(P,z))
shows *forcerel*$(P,x) \subseteq$ *forcerel*(P,y)
⟨*proof*⟩

lemma *forcerel_mono_aux* :
assumes $\langle x,y\rangle \in$ *frecrel*(*names_below*(P, w))^+
shows *forcerel*$(P,x) \subseteq$ *forcerel*(P,y)
⟨*proof*⟩

lemma *forcerel_mono* :
assumes $\langle x,y\rangle \in$ *forcerel*(P,z)
shows *forcerel*$(P,x) \subseteq$ *forcerel*(P,y)
⟨*proof*⟩

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))
⟨*proof*⟩

lemma *forcerel_eq* :
assumes $\langle z,x\rangle \in$ *forcerel*(P,x)
shows *forcerel*(P,z) = *forcerel*$(P,x) \cap$ *names_below*$(P,z)\times$*names_below*(P,z)
⟨*proof*⟩

lemma *forcerel_below_aux* :
assumes $\langle z,x\rangle \in$ *forcerel*(P,x) $\langle u,z\rangle \in$ *forcerel*(P,x)
shows $u \in$ *names_below*(P,z)
⟨*proof*⟩

lemma *forcerel_below* :
assumes $\langle z,x\rangle \in$ *forcerel*(P,x)
shows *forcerel*(P,x) -‘‘ $\{z\} \subseteq$ *names_below*(P,z)
⟨*proof*⟩

lemma *relation_forcerel* :
  shows *relation*(*forcerel*(*P*,*z*)) *trans*(*forcerel*(*P*,*z*))
  ⟨*proof*⟩

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*))^+)-``{*y*})))
  ⟨*proof*⟩

lemma *frc_at_trancl*: *frc_at*(*P*,*leq*,*z*) = *wfrec*(*forcerel*(*P*,*z*),*z*,λ*x f*. *bool_of_o*(*Hfrc*(*P*,*leq*,*x*,*f*)))
  ⟨*proof*⟩

lemma *forcerelI1* :
  assumes *n1* ∈ *domain*(*b*) ∨ *n1* ∈ *domain*(*c*) *p*∈*P* *d*∈*P*
  shows ⟨⟨*1*, *n1*, *b*, *p*⟩, ⟨*0*,*b*,*c*,*d*⟩⟩∈ *forcerel*(*P*,⟨*0*,*b*,*c*,*d*⟩)
⟨*proof*⟩

lemma *forcerelI2* :
  assumes *n1* ∈ *domain*(*b*) ∨ *n1* ∈ *domain*(*c*) *p*∈*P* *d*∈*P*
  shows ⟨⟨*1*, *n1*, *c*, *p*⟩, ⟨*0*,*b*,*c*,*d*⟩⟩∈ *forcerel*(*P*,⟨*0*,*b*,*c*,*d*⟩)
⟨*proof*⟩

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*⟩

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*⟩

32.5 Recursive expression of *frc_at*

lemma *def_frc_at* :
  assumes *p*∈*P*
  shows
    frc_at(*P*,*leq*,⟨*ft*,*n1*,*n2*,*p*⟩) =
   bool_of_o(*p* ∈*P* ∧
  (*ft* = *0* ∧ (∀ *s*. *s*∈*domain*(*n1*) ∪ *domain*(*n2*) ⟶
     (∀ *q*. *q*∈*P* ∧ *q* ≼ *p* ⟶ (*frc_at*(*P*,*leq*,⟨*1*,*s*,*n1*,*q*⟩) =*1* ⟷ *frc_at*(*P*,*leq*,⟨*1*,*s*,*n2*,*q*⟩)
=*1*)))
   ∨ *ft* = *1* ∧ (∀ *v*∈*P*. *v* ≼ *p* ⟶

$(\exists\, q.\ \exists\, s.\ \exists\, r.\ r{\in}P \wedge q{\in}P \wedge q \preceq v \wedge \langle s,r\rangle \in n2 \wedge q \preceq r \wedge\ frc\_at(P,leq,\langle 0,n1,s,q\rangle)$
$= 1))))$
$\langle proof\rangle$

32.6 Absoluteness of *frc_at*

lemma *forcerel_in_M* :
assumes
$x{\in}M$
shows
$forcerel(P,x){\in}M$
$\langle proof\rangle$

lemma *relation2_Hfrc_at_abs*:
$relation2(\#\#M,is\_Hfrc\_at(\#\#M,P,leq),\lambda x\ f.\ bool\_of\_o(Hfrc(P,leq,x,f)))$
$\langle proof\rangle$

lemma *Hfrc_at_closed* :
$\forall\, x{\in}M.\ \forall\, g{\in}M.\ function(g) \longrightarrow bool\_of\_o(Hfrc(P,leq,x,g)){\in}M$
$\langle proof\rangle$

lemma *wfrec_Hfrc_at* :
assumes
$X{\in}M$
shows
$wfrec\_replacement(\#\#M,is\_Hfrc\_at(\#\#M,P,leq),forcerel(P,X))$
$\langle proof\rangle$

lemma *names_below_abs* :
$[\![Q{\in}M;x{\in}M;nb{\in}M]\!] \Longrightarrow is\_names\_below(\#\#M,Q,x,nb) \longleftrightarrow nb = names\_below(Q,x)$
$\langle proof\rangle$

lemma *names_below_closed*:
$[\![Q{\in}M;x{\in}M]\!] \Longrightarrow names\_below(Q,x) \in M$
$\langle proof\rangle$

lemma *names_below_productE* :
assumes $Q \in M\ x \in M$
$\bigwedge A1\ A2\ A3\ A4.\ A1 \in M \Longrightarrow A2 \in M \Longrightarrow A3 \in M \Longrightarrow A4 \in M \Longrightarrow R(A1$
$\times A2 \times A3 \times A4)$
shows $R(names\_below(Q,x))$
$\langle proof\rangle$

lemma *forcerel_abs* :
$[\![x{\in}M;z{\in}M]\!] \Longrightarrow is\_forcerel(\#\#M,P,x,z) \longleftrightarrow z = forcerel(P,x)$
$\langle proof\rangle$

lemma *frc_at_abs*:
assumes $fnnc{\in}M\ z{\in}M$

shows $is\_frc\_at(\#\#M,P,leq,fnnc,z) \longleftrightarrow z = frc\_at(P,leq,fnnc)$
$\langle proof \rangle$

lemma *forces_eq'_abs* :
$[\![p\in M ; t1\in M ; t2\in M]\!] \Longrightarrow is\_forces\_eq'(\#\#M,P,leq,p,t1,t2) \longleftrightarrow forces\_eq'(P,leq,p,t1,t2)$
$\langle proof \rangle$

lemma *forces_mem'_abs* :
$[\![p\in M ; t1\in M ; t2\in M]\!] \Longrightarrow is\_forces\_mem'(\#\#M,P,leq,p,t1,t2) \longleftrightarrow forces\_mem'(P,leq,p,t1,t2)$
$\langle proof \rangle$

lemma *forces_neq'_abs* :
assumes
$p\in M$ $t1\in M$ $t2\in M$
shows
$is\_forces\_neq'(\#\#M,P,leq,p,t1,t2) \longleftrightarrow forces\_neq'(P,leq,p,t1,t2)$
$\langle proof \rangle$

lemma *forces_nmem'_abs* :
assumes
$p\in M$ $t1\in M$ $t2\in M$
shows
$is\_forces\_nmem'(\#\#M,P,leq,p,t1,t2) \longleftrightarrow forces\_nmem'(P,leq,p,t1,t2)$
$\langle proof \rangle$

end

32.7 Forcing for general formulas

definition
ren_forces_nand :: $i\Rightarrow i$ **where**
$ren\_forces\_nand(\varphi) \equiv Exists(And(Equal(0,1),iterates(\lambda p.\ incr\_bv(p)`1\ ,\ 2,\ \varphi)))$

lemma *ren_forces_nand_type*[*TC*] :
$\varphi\in formula \Longrightarrow ren\_forces\_nand(\varphi) \in formula$
$\langle proof \rangle$

lemma *arity_ren_forces_nand* :
assumes $\varphi\in formula$
shows $arity(ren\_forces\_nand(\varphi)) \leq succ(arity(\varphi))$
$\langle proof \rangle$

lemma *sats_ren_forces_nand*:
$[q,P,leq,o,p]$ @ $env \in list(M) \Longrightarrow \varphi\in formula \Longrightarrow$
$sats(M,\ ren\_forces\_nand(\varphi),[q,p,P,leq,o]$ @ $env) \longleftrightarrow sats(M,\ \varphi,[q,P,leq,o]$ @ $env)$
$\langle proof \rangle$

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*(λ*p*. *incr_bv*(*p*)‘*5* , *5*, φ)))))))))))

lemma *arity_ren_forces_all* :
assumes φ∈*formula*
shows *arity*(*ren_forces_forall*(φ)) = *5* ∪ *arity*(φ)
⟨*proof*⟩

lemma *ren_forces_forall_type*[*TC*] :
φ∈*formula* ⟹ *ren_forces_forall*(φ) ∈*formula*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *leq_fm_type*[*TC*] :
⟦*leq*∈*nat*;*q*∈*nat*;*p*∈*nat*⟧ ⟹ *leq_fm*(*leq*,*q*,*p*)∈*formula*
⟨*proof*⟩

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*))
⟨*proof*⟩

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
  ⟨proof⟩
```

```
lemma forces_type[TC] : φ∈formula ⟹ forces(φ) ∈ formula
  ⟨proof⟩
```

```
context forcing_data
begin
```

32.8 Forcing for atomic formulas in context

```
definition
  forces_eq :: [i,i,i] ⇒ o (‹_ forces_a '(_ = _')› [36,1,1] 60) where
  forces_eq ≡ forces_eq'(P,leq)
```

```
definition
  forces_mem :: [i,i,i] ⇒ o (‹_ forces_a '(_ ∈ _')› [36,1,1] 60) where
  forces_mem ≡ forces_mem'(P,leq)
```

```
definition
  is_forces_eq :: [i,i,i] ⇒ o where
  is_forces_eq ≡ is_forces_eq'(##M,P,leq)
```

```
definition
  is_forces_mem :: [i,i,i] ⇒ o where
  is_forces_mem ≡ is_forces_mem'(##M,P,leq)
```

```
lemma def_forces_eq: p∈P ⟹ p forces_a (t1 = t2) ⟷
      (∀ s∈domain(t1) ∪ domain(t2). ∀ q. q∈P ∧ q ≼ p ⟶
      (q forces_a (s ∈ t1) ⟷ q forces_a (s ∈ t2)))
  ⟨proof⟩
```

lemma *def_forces_mem*: $p \in P \Longrightarrow p\ forces_a\ (t1 \in t2) \longleftrightarrow$
$(\forall v \in P.\ v \preceq p \longrightarrow$
$(\exists q.\ \exists s.\ \exists r.\ r \in P \wedge q \in P \wedge q \preceq v \wedge \langle s,r\rangle \in t2 \wedge q \preceq r \wedge q\ forces_a\ (t1 = s)))$
⟨*proof*⟩

lemma *forces_eq_abs* :
$[\![p \in M\ ;\ t1 \in M\ ;\ t2 \in M]\!] \Longrightarrow is\_forces\_eq(p,t1,t2) \longleftrightarrow p\ forces_a\ (t1 = t2)$
⟨*proof*⟩

lemma *forces_mem_abs* :
$[\![p \in M\ ;\ t1 \in M\ ;\ t2 \in M]\!] \Longrightarrow is\_forces\_mem(p,t1,t2) \longleftrightarrow p\ forces_a\ (t1 \in t2)$
⟨*proof*⟩

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 $sats(M,forces\_eq\_fm(p,l,q,t1,t2),env) \longleftrightarrow$
$is\_forces\_eq(nth(q,env),nth(t1,env),nth(t2,env))$
⟨*proof*⟩

lemma *sats_forces_mem_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 $sats(M,forces\_mem\_fm(p,l,q,t1,t2),env) \longleftrightarrow$
$is\_forces\_mem(nth(q,env),nth(t1,env),nth(t2,env))$
⟨*proof*⟩

definition
$forces\_neq :: [i,i,i] \Rightarrow o$ (‹_ $forces_a$ '(_ ≠ _')› [36,1,1] 60) **where**
$p\ forces_a\ (t1 \neq t2) \equiv \neg (\exists q \in P.\ q \preceq p \wedge q\ forces_a\ (t1 = t2))$

definition
$forces\_nmem :: [i,i,i] \Rightarrow o$ (‹_ $forces_a$ '(_ ∉ _')› [36,1,1] 60) **where**
$p\ forces_a\ (t1 \notin t2) \equiv \neg (\exists q \in P.\ q \preceq p \wedge q\ forces_a\ (t1 \in t2))$

lemma *forces_neq* :
$p\ forces_a\ (t1 \neq t2) \longleftrightarrow forces\_neq'(P,leq,p,t1,t2)$
⟨*proof*⟩

lemma *forces_nmem* :
$p\ forces_a\ (t1 \notin t2) \longleftrightarrow forces\_nmem'(P,leq,p,t1,t2)$
⟨*proof*⟩

abbreviation *Forces* :: $[i,\ i,\ i] \Rightarrow o$ (_ ⊩ _ _ [36,36,36] 60) **where**
$p \Vdash \varphi\ env\ \ \equiv\ \ M,\ ([p,P,leq,one]\ @\ env) \models forces(\varphi)$

lemma *sats_forces_Member* :

assumes $x\in nat$ $y\in nat$ $env\in list(M)$
$nth(x,env)=xx$ $nth(y,env)=yy$ $q\in M$
shows $q \Vdash \cdot x \in y \cdot\ env \longleftrightarrow q \in P \wedge is\_forces\_mem(q,\ xx,\ yy)$
⟨*proof*⟩

lemma *sats_forces_Equal* :
assumes $x\in nat$ $y\in nat$ $env\in list(M)$
$nth(x,env)=xx$ $nth(y,env)=yy$ $q\in M$
shows $q \Vdash \cdot x = y \cdot\ env \longleftrightarrow q \in P \wedge is\_forces\_eq(q,\ xx,\ yy)$
⟨*proof*⟩

lemma *sats_forces_Nand* :
assumes $\varphi\in formula$ $\psi\in formula$ $env\in list(M)$ $p\in M$
shows $p \Vdash \cdot\neg(\varphi \wedge \psi)\cdot\ env \longleftrightarrow$
$p\in P \wedge \neg(\exists q\in M.\ q\in P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
$(M,[q,P,leq,one]@env \models forces'(\varphi)) \wedge (M,[q,P,leq,one]@env \models forces'(\psi)))$
⟨*proof*⟩

lemma *sats_forces_Neg* :
assumes $\varphi\in formula$ $env\in list(M)$ $p\in M$
shows $p \Vdash \cdot\neg\varphi\cdot\ env \longleftrightarrow$
$(p\in P \wedge \neg(\exists q\in M.\ q\in P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
$(M,\ [q,\ P,\ leq,\ one]\ @\ env \models forces'(\varphi))))$
⟨*proof*⟩

lemma *sats_forces_Forall* :
assumes $\varphi\in formula$ $env\in list(M)$ $p\in M$
shows $p \Vdash (\cdot\forall \varphi\cdot)\ env \longleftrightarrow p \in P \wedge (\forall x\in M.\ M,[p,P,leq,one,x]\ @\ env \models forces'(\varphi))$
⟨*proof*⟩

end

32.9 The arity of *forces*

lemma *arity_forces_at*:
assumes $x \in nat$ $y \in nat$
shows $arity(forces(Member(x,\ y))) = (succ(x) \cup succ(y))\ \#+\ 4$
$arity(forces(Equal(x,\ y))) = (succ(x) \cup succ(y))\ \#+\ 4$
⟨*proof*⟩

lemma *arity_forces'*:
assumes $\varphi\in formula$
shows $arity(forces'(\varphi)) \leq arity(\varphi)\ \#+\ 4$
⟨*proof*⟩

lemma *arity_forces* :
assumes $\varphi\in formula$
shows $arity(forces(\varphi)) \leq 4\#+arity(\varphi)$
⟨*proof*⟩

lemma *arity_forces_le* :
  assumes φ∈*formula* *n*∈*nat* *arity*(φ) ≤ *n*
  shows *arity*(*forces*(φ)) ≤ *4*#+*n*
  ⟨*proof*⟩

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: φ∈*formula* **and**
    far: *arity*(φ)≤*length*(*env*) **and**
    envty: *env*∈*list*(*M*)
  shows
    {*p*∈*P* . *p* ⊩ φ *env*} ∈ *M*
⟨*proof*⟩

lemma *forces_mem_iff_dense_below*: *p*∈*P* ⟹ *p* $forces_a$ (*t1* ∈ *t2*) ⟷ *dense_below*(
    {*q*∈*P*. ∃ *s*. ∃ *r*. *r*∈*P* ∧ ⟨*s*,*r*⟩ ∈ *t2* ∧ *q*≼*r* ∧ *q* $forces_a$ (*t1* = *s*)}
    ,*p*)
  ⟨*proof*⟩

33.2 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_eq*:
  assumes *p*∈*P* *r*∈*P* *r*≼*p* *p* $forces_a$ (*t1* = *t2*)
  shows *r* $forces_a$ (*t1* = *t2*)
  ⟨*proof*⟩

33.3 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_mem*:
  assumes *p*∈*P* *r*∈*P* *r*≼*p* *p* $forces_a$ (*t1* ∈ *t2*)
  shows *r* $forces_a$ (*t1* ∈ *t2*)
  ⟨*proof*⟩

33.4 Kunen 2013, Lemma IV.2.37(b)

lemma *density_mem*:
  assumes $p \in P$
  shows p $forces_a$ $(t1 \in t2)$ $\longleftrightarrow$ $dense\_below(\{q \in P.\ q\ forces_a\ (t1 \in t2)\},p)$
$\langle proof \rangle$

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
    $dense\_below(\{r \in P.\ r\ forces_a\ (s \in t2)\},q)$
$\langle proof \rangle$

lemma *density_eq*:
  assumes $p \in P$
  shows p $forces_a$ $(t1 = t2)$ $\longleftrightarrow$ $dense\_below(\{q \in P.\ q\ forces_a\ (t1 = t2)\},p)$
$\langle proof \rangle$

33.5 Kunen 2013, Lemma IV.2.38

lemma *not_forces_neq*:
  assumes $p \in P$
  shows $p\ forces_a\ (t1 = t2) \longleftrightarrow \neg\ (\exists\ q \in P.\ q \preceq p \wedge q\ forces_a\ (t1 \neq t2))$
  $\langle proof \rangle$

lemma *not_forces_nmem*:
  assumes $p \in P$
  shows $p\ forces_a\ (t1 \in t2) \longleftrightarrow \neg\ (\exists\ q \in P.\ q \preceq p \wedge q\ forces_a\ (t1 \notin t2))$
  $\langle proof \rangle$

lemma *sats_forces_Nand'*:
  assumes
    $p \in P$ $\varphi \in formula$ $\psi \in formula$ $env \in list(M)$
  shows
    $(M,\ [p,P,leq,one]\ @\ env \models forces(Nand(\varphi,\psi))) \longleftrightarrow$
     $\neg(\exists\ q \in M.\ q \in P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
          $(M,\ [q,P,leq,one]\ @\ env \models forces(\varphi)) \wedge$
          $(M,\ [q,P,leq,one]\ @\ env \models forces(\psi)))$
  $\langle proof \rangle$

lemma *sats_forces_Neg'*:
assumes
$p \in P$ $env \in list(M)$ $\varphi \in formula$
shows
$(M,\ [p,P,leq,one]\ @\ env \models forces(Neg(\varphi))) \longleftrightarrow$
$\neg(\exists\, q \in M.\ q \in P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
$(M,\ [q,P,leq,one]@env \models forces(\varphi)))$
⟨*proof*⟩

lemma *sats_forces_Forall'*:
assumes
$p \in P$ $env \in list(M)$ $\varphi \in formula$
shows
$(M,[p,P,leq,one]\ @\ env \models forces(Forall(\varphi))) \longleftrightarrow$
$(\forall\, x \in M.\quad M,\ [p,P,leq,one,x]\ @\ env \models forces(\varphi))$
⟨*proof*⟩

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
$(p \Vdash Equal(n,m)\ env) \longleftrightarrow p\ forces_a\ (t1 = t2)$
⟨*proof*⟩

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
$(p \Vdash Member(n,m)\ env) \longleftrightarrow p\ forces_a\ (t1 \in t2)$
⟨*proof*⟩

lemma *Forces_Neg*:
assumes
$p \in P$ $env \in list(M)$ $\varphi \in formula$
shows
$(p \Vdash Neg(\varphi)\ env) \longleftrightarrow \neg(\exists\, q \in M.\ q \in P \wedge q \preceq p \wedge (q \Vdash \varphi\ env))$
⟨*proof*⟩

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
$(p \Vdash Nand(\varphi,\psi)\ env) \longleftrightarrow \neg(\exists\, q \in M.\ q \in P \wedge q \preceq p \wedge (q \Vdash \varphi\ env) \wedge (q \Vdash \psi\ env))$
⟨*proof*⟩

lemma *Forces_And_aux*:

assumes
$p \in P$ *env* $\in$ *list*(M) $\varphi \in$*formula* $\psi \in$*formula*
shows
$p \Vdash$ *And*(φ,ψ) *env* $\longleftrightarrow$
$(\forall q \in M.\ q \in P \wedge q \preceq p \longrightarrow (\exists r \in M.\ r \in P \wedge r \preceq q \wedge (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)))$
⟨*proof*⟩

lemma *Forces_And_iff_dense_below*:
assumes
$p \in P$ *env* $\in$ *list*(M) $\varphi \in$*formula* $\psi \in$*formula*
shows
$(p \Vdash And(\varphi,\psi)\ env) \longleftrightarrow dense\_below(\{r \in P.\ (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)\ \},p)$
⟨*proof*⟩

lemma *Forces_Forall*:
assumes
$p \in P$ *env* $\in$ *list*(M) $\varphi \in$*formula*
shows
$(p \Vdash Forall(\varphi)\ env) \longleftrightarrow (\forall x \in M.\ (p \Vdash \varphi\ ([x]\ @\ env)))$
⟨*proof*⟩

bundle *some_rules* = *elem_of_val_pair* [*dest*] *SepReplace_iff* [*simp del*] *SepReplace_iff*[*iff*]

context
includes *some_rules*
begin

lemma *elem_of_valI*: $\exists \vartheta.\ \exists p \in P.\ p \in G \wedge \langle \vartheta,p \rangle \in \pi \wedge val(P,G,\vartheta) = x \Longrightarrow x \in val(P,G,\pi)$
⟨*proof*⟩

lemma *GenExtD*: $x \in M[G] \longleftrightarrow (\exists \tau \in M.\ x = val(P,G,\tau))$
⟨*proof*⟩

lemma *left_in_M* : $tau \in M \Longrightarrow \langle a,b \rangle \in tau \Longrightarrow a \in M$
⟨*proof*⟩

33.8 Kunen 2013, Lemma IV.2.29

lemma *generic_inter_dense_below*:
assumes $D \in M$ *M_generic*(G) *dense_below*(D,p) $p \in G$
shows $D \cap G \neq 0$
⟨*proof*⟩

33.9 Auxiliary results for Lemma IV.2.40(a)

lemma *IV240a_mem_Collect*:
assumes
$\pi \in M$ $\tau \in M$
shows

$\{q{\in}P.\ \exists\sigma.\ \exists r.\ r{\in}P \wedge \langle\sigma,r\rangle \in \tau \wedge q{\preceq}r \wedge q\ forces_a\ (\pi = \sigma)\}{\in}M$
⟨*proof*⟩

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*⟩

lemma *refl_forces_eq*:$p{\in}P \Longrightarrow p\ forces_a\ (x = x)$
⟨*proof*⟩

lemma *forces_memI*: $\langle\sigma,r\rangle{\in}\tau \Longrightarrow p{\in}P \Longrightarrow r{\in}P \Longrightarrow p{\preceq}r \Longrightarrow p\ forces_a\ (\sigma \in \tau)$
⟨*proof*⟩

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*⟩

lemma *IV240a_eq_2nd_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
$val(P,G,\vartheta) \subseteq val(P,G,\tau)$
⟨*proof*⟩

lemma *IV240a_eq*:
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
$val(P,G,\tau) = val(P,G,\vartheta)$
⟨*proof*⟩

33.10 Induction on names

lemma *core_induction*:
assumes
$\bigwedge\tau\ \vartheta\ p.\ p \in P \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![q{\in}P\ ;\ \sigma{\in}domain(\vartheta)]\!] \Longrightarrow Q(0,\tau,\sigma,q)]\!] \Longrightarrow Q(1,\tau,\vartheta,p)$
$\bigwedge\tau\ \vartheta\ p.\ p \in P \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![q{\in}P\ ;\ \sigma{\in}domain(\tau) \cup domain(\vartheta)]\!] \Longrightarrow Q(1,\sigma,\tau,q) \wedge Q(1,\sigma,\vartheta,q)]\!] \Longrightarrow Q(0,\tau,\vartheta,p)$
$ft \in 2$ $p \in P$
shows
$Q(ft,\tau,\vartheta,p)$
⟨*proof*⟩

lemma *forces_induction_with_conds*:
assumes
$\bigwedge\tau\ \vartheta\ p.\ p \in P \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![q{\in}P\ ;\ \sigma{\in}domain(\vartheta)]\!] \Longrightarrow Q(q,\tau,\sigma)]\!] \Longrightarrow R(p,\tau,\vartheta)$
$\bigwedge\tau\ \vartheta\ p.\ p \in P \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![q{\in}P\ ;\ \sigma{\in}domain(\tau) \cup domain(\vartheta)]\!] \Longrightarrow R(q,\sigma,\tau) \wedge R(q,\sigma,\vartheta)]\!] \Longrightarrow Q(p,\tau,\vartheta)$
$p \in P$
shows
$Q(p,\tau,\vartheta) \wedge R(p,\tau,\vartheta)$
⟨*proof*⟩

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*⟩

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** $?Q(\tau,\vartheta) \wedge\ ?R(\tau,\vartheta)$)
⟨*proof*⟩

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*⟩

end

lemma *Collect_forces_eq_in_M*:
assumes $\tau \in M$ $\vartheta \in M$
shows $\{p \in P.\ p\ forces_a\ (\tau = \vartheta)\} \in M$
⟨*proof*⟩

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*⟩

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*⟩

lemma *IV240b*:
assumes
$M\_generic(G)$
shows
$(\tau \in M \longrightarrow \vartheta \in M \longrightarrow val(P,G,\tau) = val(P,G,\vartheta) \longrightarrow (\exists\, p \in G.\ p\ forces_a\ (\tau = \vartheta))) \wedge$
$(\tau \in M \longrightarrow \vartheta \in M \longrightarrow val(P,G,\tau) \in val(P,G,\vartheta) \longrightarrow (\exists\, p \in G.\ p\ forces_a\ (\tau \in \vartheta)))$
(**is** $?Q(\tau,\vartheta) \wedge\ ?R(\tau,\vartheta)$)
⟨*proof*⟩

lemma *map_val_in_MG*:
assumes
$env \in list(M)$
shows
$map(val(P,G),env) \in list(M[G])$
⟨*proof*⟩

lemma *truth_lemma_mem*:
assumes
$env \in list(M)$ $M\_generic(G)$
$n \in nat$ $m \in nat$ $n < length(env)$ $m < length(env)$
shows
$(\exists p \in G.\ p \Vdash Member(n,m)\ env) \longleftrightarrow M[G], map(val(P,G),env) \models Member(n,m)$
⟨*proof*⟩

lemma *truth_lemma_eq*:
assumes
$env \in list(M)$ $M\_generic(G)$
$n \in nat$ $m \in nat$ $n < length(env)$ $m < length(env)$
shows
$(\exists p \in G.\ p \Vdash Equal(n,m)\ env) \longleftrightarrow M[G], map(val(P,G),env) \models Equal(n,m)$
⟨*proof*⟩

lemma *arities_at_aux*:
assumes
$n \in nat$ $m \in nat$ $env \in list(M)$ $succ(n) \cup succ(m) \leq length(env)$
shows
$n < length(env)$ $m < length(env)$
⟨*proof*⟩

33.13 The Strenghtening Lemma

lemma *strengthening_lemma*:
assumes
$p \in P$ $\varphi \in formula$ $r \in P$ $r \preceq p$
$env \in list(M)$ $arity(\varphi) \leq length(env)$
shows
$p \Vdash \varphi\ env \Longrightarrow r \Vdash \varphi\ env$
⟨*proof*⟩

33.14 The Density Lemma

lemma *arity_Nand_le*:
assumes $\varphi \in formula$ $\psi \in formula$ $arity(Nand(\varphi, \psi)) \leq length(env)$ $env \in list(A)$
shows $arity(\varphi) \leq length(env)$ $arity(\psi) \leq length(env)$
⟨*proof*⟩

lemma *dense_below_imp_forces*:
assumes
$p \in P$ $\varphi \in formula$

env∈*list*(*M*) *arity*(φ)≤*length*(*env*)
shows
dense_below({*q*∈*P*. (*q* ⊩ φ *env*)},*p*) ⟹ (*p* ⊩ φ *env*)
⟨*proof*⟩

lemma *density_lemma*:
assumes
p∈*P* φ∈*formula* *env*∈*list*(*M*) *arity*(φ)≤*length*(*env*)
shows
p ⊩ φ *env* ⟷ *dense_below*({*q*∈*P*. (*q* ⊩ φ *env*)},*p*)
⟨*proof*⟩

33.15 The Truth Lemma

lemma *Forces_And*:
assumes
p∈*P* *env* ∈ *list*(*M*) φ∈*formula* ψ∈*formula*
arity(φ) ≤ *length*(*env*) *arity*(ψ) ≤ *length*(*env*)
shows
p ⊩ *And*(φ,ψ) *env* ⟷ (*p* ⊩ φ *env*) ∧ (*p* ⊩ ψ *env*)
⟨*proof*⟩

lemma *Forces_Nand_alt*:
assumes
p∈*P* *env* ∈ *list*(*M*) φ∈*formula* ψ∈*formula*
arity(φ) ≤ *length*(*env*) *arity*(ψ) ≤ *length*(*env*)
shows
(*p* ⊩ *Nand*(φ,ψ) *env*) ⟷ (*p* ⊩ *Neg*(*And*(φ,ψ)) *env*)
⟨*proof*⟩

lemma *truth_lemma_Neg*:
assumes
φ∈*formula* *M_generic*(*G*) *env*∈*list*(*M*) *arity*(φ)≤*length*(*env*) **and**
IH: (∃ *p*∈*G*. *p* ⊩ φ *env*) ⟷ *M*[*G*], *map*(*val*(*P*,*G*),*env*) ⊨ φ
shows
(∃ *p*∈*G*. *p* ⊩ *Neg*(φ) *env*) ⟷ *M*[*G*], *map*(*val*(*P*,*G*),*env*) ⊨ *Neg*(φ)
⟨*proof*⟩

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)
⟨*proof*⟩

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(λ*p*. *incr_bv*(*p*)‘*5* , *6*, φ)))))))))))

lemma *ren_truth_lemma_type*[*TC*] :
φ∈*formula* ⟹ *ren_truth_lemma*(φ) ∈*formula*
⟨*proof*⟩

lemma *arity_ren_truth* :
assumes φ∈*formula*
shows *arity*(*ren_truth_lemma*(φ)) ≤ *6* ∪ *succ*(*arity*(φ))
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *truth_lemma'* :
assumes
φ∈*formula* *env*∈*list*(*M*) *arity*(φ) ≤ *succ*(*length*(*env*))
shows
separation(##*M*,λ*d*. ∃ *b*∈*M*. ∀ *q*∈*P*. *q*≼*d* ⟶ ¬(*q* ⊩ φ ([*b*]@*env*)))
⟨*proof*⟩

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
⟨*proof*⟩

33.16 The “Definition of forcing”

lemma *definition_of_forcing*:
assumes
p∈*P* φ∈*formula* *env*∈*list*(*M*) *arity*(φ)≤*length*(*env*)
shows
(*p* ⊩ φ *env*) ⟷
(∀ *G*. *M_generic*(*G*) ∧ *p*∈*G* ⟶ *M*[*G*], *map*(*val*(*P*,*G*),*env*) ⊨ φ)
⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *nth_concat2* : *env*∈ *list*(*A*) ⟹ *nth*(*length*(*env*),*env* @ [*p*,*t*]) = *p*
  ⟨*proof*⟩

lemma *nth_concat3* : *env*∈ *list*(*A*) ⟹ *u* = *nth*(*succ*(*length*(*env*)), *env* @ [*pi*, *u*])
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *weak_equal* :
  assumes *n*∈*nat* *m*∈*nat*
  shows *weak*(*n*,*m*) = (*m*#+*n*) - *m*
⟨*proof*⟩

lemma *weak_zero*:
  shows *weak*(*0*,*n*) = *0*
  ⟨*proof*⟩

lemma *weakening_diff* :

assumes *n* ∈ *nat*
shows *weak*(*n*,*7*) - *weak*(*n*,*5*) ⊆ {*5*#+*n*, *6*#+*n*}
⟨*proof*⟩

lemma *in_add_del* :
assumes *x*∈*j*#+*n* *n*∈*nat* *j*∈*nat*
shows *x* < *j* ∨ *x* ∈ *weak*(*n*,*j*)
⟨*proof*⟩

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*⟩

lemma *sep_env_type* :
assumes *n* ∈ *nat*
shows *sep_env*(*n*) : (*5*#+*n*)-*5* → (*7*#+*n*)-*7*
⟨*proof*⟩

lemma *sep_var_fin_type* :
assumes *n* ∈ *nat*
shows *sep_var*(*n*) : *7*#+*n* -||> *7*#+*n*
⟨*proof*⟩

lemma *sep_var_domain* :
assumes *n* ∈ *nat*
shows *domain*(*sep_var*(*n*)) = *7*#+*n* - *weak*(*n*,*5*)
⟨*proof*⟩

lemma *sep_var_type* :
assumes *n* ∈ *nat*
shows *sep_var*(*n*) : (*7*#+*n*)-*weak*(*n*,*5*) → *7*#+*n*
⟨*proof*⟩

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*])
⟨*proof*⟩

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*⟩

lemma *rensep_type* :
assumes *n*∈*nat*
shows *rensep*(*n*) ∈ *7#+n* → *7#+n*
⟨*proof*⟩

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*⟩

definition *sep_ren* :: [*i*,*i*] ⇒ *i* **where**
sep_ren(*n*,*φ*) ≡ *ren*(*φ*)‘(*7#+n*)‘(*7#+n*)‘*rensep*(*n*)

lemma *arity_rensep*: **assumes** *φ*∈*formula* *env* ∈ *list*(*M*)
arity(*φ*) ≤ *7#+length*(*env*)
shows *arity*(*sep_ren*(*length*(*env*),*φ*)) ≤ *7#+length*(*env*)
⟨*proof*⟩

lemma *type_rensep* [*TC*]:
assumes *φ*∈*formula* *env*∈*list*(*M*)
shows *sep_ren*(*length*(*env*),*φ*) ∈ *formula*
⟨*proof*⟩

lemma *sepren_action*:
assumes *arity*(*φ*) ≤ *7* #+ *length*(*env*)
[*t*,*p*,*u*,*P*,*leq*,*o*,*pi*] ∈ *list*(*M*)
env∈*list*(*M*)
φ∈*formula*
shows *sats*(*M*, *sep_ren*(*length*(*env*),*φ*),[*t*,*p*,*u*,*P*,*leq*,*o*,*pi*] @ *env*) ⟷ *sats*(*M*, *φ*,[*p*,*P*,*leq*,*o*,*t*] @ *env* @ [*pi*,*u*])
⟨*proof*⟩

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 $env \in list(M[G])$
  shows $\exists\, nenv \in list(M).\ env = map(val(P,G),nenv)$
  ⟨*proof*⟩

lemma *Collect_sats_in_MG* :
  assumes
    $c \in M[G]$
    $\varphi \in formula$ $env \in list(M[G])$ $arity(\varphi) \le 1 \ \#+\ length(env)$
  shows
    $\{x \in c.\ (M[G],\ [x]\ @\ env \models \varphi)\} \in M[G]$
⟨*proof*⟩

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*⟩

end

end

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

theory *Pairing_Axiom* **imports** *Names* **begin**

context *forcing_data*
begin

lemma *val_Upair* :
  $one \in G \Longrightarrow val(P,G,\{\langle\tau,one\rangle,\langle\varrho,one\rangle\}) = \{val(P,G,\tau),val(P,G,\varrho)\}$
  ⟨*proof*⟩

lemma *pairing_in_MG* :
  assumes $M\_generic(G)$
  shows $upair\_ax(\#\#M[G])$
⟨*proof*⟩

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**
$Union\_name\_body(P',leq',\tau,\vartheta p) \equiv (\exists\ \sigma[\#\#M].$
$\exists\ q[\#\#M].\ (q\in\ P' \wedge (\langle\sigma,q\rangle \in \tau \wedge$
$(\exists\ r[\#\#M].r\in P' \wedge (\langle fst(\vartheta p),r\rangle \in \sigma \wedge \langle snd(\vartheta p),r\rangle \in leq' \wedge \langle snd(\vartheta p),q\rangle$
$\in leq')))))$

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*
⟨*proof*⟩

lemma *arity_Union_name_fm* :
arity(*Union_name_fm*) = *4*
⟨*proof*⟩

lemma *sats_Union_name_fm* :
$[\![\ env \in list(M);\ P' \in M\ ;\ p \in M\ ;\ \vartheta \in M\ ;\ \tau \in M\ ;\ leq' \in M\ ]\!] \Longrightarrow$
$sats(M,Union\_name\_fm,[\langle\vartheta,p\rangle,\tau,leq',P']@env) \longleftrightarrow$
$Union\_name\_body(P',leq',\tau,\langle\vartheta,p\rangle)$
⟨*proof*⟩

definition *Union_name* :: $i \Rightarrow i$ **where**
$Union\_name(\tau) \equiv$
$\{u \in domain(\bigcup (domain(\tau))) \times P\ .\ Union\_name\_body(P,leq,\tau,u)\}$

lemma *Union_name_M* : **assumes** $\tau \in M$
shows $Union\_name(\tau) \in M$

⟨*proof*⟩

lemma *Union_MG_Eq* :
  assumes $a \in M[G]$ **and** $a = val(P,G,\tau)$ **and** $filter(G)$ **and** $\tau \in M$
  shows $\bigcup\ a = val(P,G,Union\_name(\tau))$
⟨*proof*⟩

lemma *union_in_MG* : **assumes** $filter(G)$
  shows $Union\_ax(\#\#M[G])$
⟨*proof*⟩

theorem *Union_MG* : $M\_generic(G) \Longrightarrow Union\_ax(\#\#M[G])$
  ⟨*proof*⟩

end
end

38 The Powerset Axiom in $M[G]$

theory *Powerset_Axiom*
  imports *Renaming_Auto Separation_Axiom Pairing_Axiom Union_Axiom*
begin

⟨*ML*⟩

lemma *Collect_inter_Transset*:
  assumes
    $Transset(M)\ b \in M$
  shows
    $\{x{\in}b\ .\ P(x)\} = \{x{\in}b\ .\ P(x)\} \cap M$
  ⟨*proof*⟩

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*⟩

lemma *sats_fst_snd_in_M*:
  assumes
    $A{\in}M\ B{\in}M\ \varphi \in formula\ p{\in}M\ l{\in}M\ o{\in}M\ \chi{\in}M$
    $arity(\varphi) \leq 6$
  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*⟩

lemma *Pow_inter_MG*:

assumes
$a \in M[G]$
shows
$Pow(a) \cap M[G] \in M[G]$
⟨*proof*⟩
end

context *G_generic* **begin**

interpretation *mgtriv*: *M_trivial* $\#\#M[G]$
⟨*proof*⟩

theorem *power_in_MG* : *power_ax*$(\#\#(M[G]))$
⟨*proof*⟩

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*⟩

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*]))
⟨*proof*⟩

lemma *foundation_ax*(##(*M*[*G*]))
⟨*proof*⟩

end
end

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

theory *Replacement_Axiom*
imports
Least Relative_Univ Separation_Axiom Renaming_Auto
begin

⟨*ML*⟩

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*
⟨*proof*⟩

lemma *arity_renrep*:
assumes *φ*∈*formula arity*(*φ*)≤ *6*#+*length*(*env*) *env* ∈ *list*(*M*)
shows *arity*(*renrep*(*φ*,*env*)) ≤ *8*#+*length*(*env*)
⟨*proof*⟩

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*)
⟨*proof*⟩

⟨*ML*⟩

definition *renpbdy_fn* :: *i* ⇒ *i* **where**
renpbdy_fn(*env*) ≡ *rsum*(*renpbdy1_fn*,*id*(*length*(*env*)),*6*,*7*,*length*(*env*))

definition
renpbdy :: $[i,i] \Rightarrow i$ **where**
$renpbdy(\varphi,env) = ren(\varphi)`(6\#+length(env))`(7\#+length(env))`renpbdy\_fn(env)$

lemma
renpbdy_type [*TC*]: $\varphi \in formula \Longrightarrow env \in list(M) \Longrightarrow renpbdy(\varphi,env) \in formula$
⟨*proof*⟩

lemma *arity_renpbdy*: $\varphi \in formula \Longrightarrow arity(\varphi) \leq 6 \#+ length(env) \Longrightarrow env \in list(M) \Longrightarrow arity(renpbdy(\varphi,env)) \leq 7 \#+ length(env)$
⟨*proof*⟩

lemma
sats_renpbdy: $arity(\varphi) \leq 6 \#+ length(nenv) \Longrightarrow [\varrho,p,x,\alpha,P,leq,o,\pi] @ nenv \in list(M) \Longrightarrow \varphi \in formula \Longrightarrow$
$sats(M, \varphi, [\varrho,p,\alpha,P,leq,o] @ nenv) \longleftrightarrow sats(M, renpbdy(\varphi,nenv), [\varrho,p,x,\alpha,P,leq,o] @ nenv)$
⟨*proof*⟩

⟨*ML*⟩

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**
$renbody(\varphi,env) = ren(\varphi)`(5\#+length(env))`(6\#+length(env))`renbody\_fn(env)$

lemma
renbody_type [*TC*]: $\varphi \in formula \Longrightarrow env \in list(M) \Longrightarrow renbody(\varphi,env) \in formula$
⟨*proof*⟩

lemma *arity_renbody*: $\varphi \in formula \Longrightarrow arity(\varphi) \leq 5 \#+ length(env) \Longrightarrow env \in list(M) \Longrightarrow$
$arity(renbody(\varphi,env)) \leq 6 \#+ length(env)$
⟨*proof*⟩

lemma
sats_renbody: $arity(\varphi) \leq 5 \#+ length(nenv) \Longrightarrow [\alpha,x,m,P,leq,o] @ nenv \in list(M) \Longrightarrow \varphi \in formula \Longrightarrow$
$sats(M, \varphi, [x,\alpha,P,leq,o] @ nenv) \longleftrightarrow sats(M, renbody(\varphi,nenv), [\alpha,x,m,P,leq,o] @ nenv)$
⟨*proof*⟩

context *G_generic*
begin

lemma *pow_inter_M*:
assumes
$x \in M$ $y \in M$
shows
$powerset(\#\#M,x,y) \longleftrightarrow y = Pow(x) \cap M$
⟨*proof*⟩

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
$(\exists \tau \in M. \exists V \in M. is\_Vset(\#\#M,\alpha,V) \wedge \tau \in V \wedge sats(M,forces(\varphi),[p,P,leq,one,\varrho,\tau]$ @ $nenv))$
$\longleftrightarrow sats(M,?prebody\_fm,[\varrho,p,\alpha,P,leq,one]$ @ $nenv)$
⟨*proof*⟩

⟨*ML*⟩

lemma *prebody_fm_type* [*TC*]:
assumes $\varphi \in formula$
$env \in list(M)$
shows $prebody\_fm(\varphi,env) \in formula$
⟨*proof*⟩

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
$[P,leq,one,p,\varrho]$ @ $nenv \in list(M)$ $\varphi \in formula$ $\alpha \in M$ $arity(\varphi) \le 2 \#+ length(nenv)$
shows
$sats(M,prebody\_fm(\varphi,nenv),[\varrho,p,\alpha,P,leq,one]$ @ $nenv) \longleftrightarrow$
$(\exists \tau \in M. \exists V \in M. is\_Vset(\#\#M,\alpha,V) \wedge \tau \in V \wedge sats(M,forces(\varphi),[p,P,leq,one,\varrho,\tau]$ @ $nenv))$
⟨*proof*⟩

lemma *arity_prebody_fm*:
assumes
$\varphi \in formula$ $\alpha \in M$ $env \in list(M)$ $arity(\varphi) \le 2 \#+ length(env)$
shows
$arity(prebody\_fm(\varphi,env)) \le 6 \#+ length(env)$
⟨*proof*⟩

definition
body_fm' :: $[i,i]\Rightarrow i$ **where**
body_fm'$(\varphi,env) \equiv$ *Exists*(*Exists*(*And*(*pair_fm*(*0*,*1*,*2*),*renpbdy*(*prebody_fm*(φ,*env*),*env*))))

lemma *body_fm'_type*[*TC*]: $\varphi\in$*formula* $\Longrightarrow$ *env*$\in$*list*(*M*) $\Longrightarrow$ *body_fm'*(φ,*env*)$\in$*formula*
⟨*proof*⟩

lemma *arity_body_fm'*:
assumes
$\varphi\in$*formula* $\alpha\in M$ *env*$\in$*list*(*M*) *arity*(φ) $\leq$ *2* #+ *length*(*env*)
shows
arity(*body_fm'*(φ,*env*))$\leq$*5* #+ *length*(*env*)
⟨*proof*⟩

lemma *sats_body_fm'*:
assumes
$\exists$ *t p*. *x*=⟨*t*,*p*⟩ *x*$\in$*M* [α,*P*,*leq*,*one*,*p*,ϱ] @ *nenv* $\in$*list*(*M*) $\varphi\in$*formula* *arity*(φ) $\leq$ *2*
#+ *length*(*nenv*)
shows
sats(*M*,*body_fm'*(φ,*nenv*),[*x*,α,*P*,*leq*,*one*] @ *nenv*) $\longleftrightarrow$
sats(*M*,*renpbdy*(*prebody_fm*(φ,*nenv*),*nenv*),[*fst*(*x*),*snd*(*x*),*x*,α,*P*,*leq*,*one*] @ *nenv*)
⟨*proof*⟩

definition
body_fm :: $[i,i]\Rightarrow i$ **where**
body_fm(φ,*env*) $\equiv$ *renbody*(*body_fm'*(φ,*env*),*env*)

lemma *body_fm_type* [*TC*]: *env*$\in$*list*(*M*) $\Longrightarrow$ $\varphi\in$*formula* $\Longrightarrow$ *body_fm*(φ,*env*)$\in$*formula*
⟨*proof*⟩

lemma *sats_body_fm*:
assumes
$\exists$ *t p*. *x*=⟨*t*,*p*⟩ [α,*x*,*m*,*P*,*leq*,*one*] @ *nenv* $\in$*list*(*M*)
$\varphi\in$*formula* *arity*(φ) $\leq$ *2* #+ *length*(*nenv*)
shows
sats(*M*,*body_fm*(φ,*nenv*),[α,*x*,*m*,*P*,*leq*,*one*] @ *nenv*) $\longleftrightarrow$
sats(*M*,*renpbdy*(*prebody_fm*(φ,*nenv*),*nenv*),[*fst*(*x*),*snd*(*x*),*x*,α,*P*,*leq*,*one*] @ *nenv*)
⟨*proof*⟩

lemma *sats_renpbdy_prebody_fm*:
assumes
$\exists$ *t p*. *x*=⟨*t*,*p*⟩ *x*$\in$*M* [α,*m*,*P*,*leq*,*one*] @ *nenv* $\in$*list*(*M*)
$\varphi\in$*formula* *arity*(φ) $\leq$ *2* #+ *length*(*nenv*)
shows
sats(*M*,*renpbdy*(*prebody_fm*(φ,*nenv*),*nenv*),[*fst*(*x*),*snd*(*x*),*x*,α,*P*,*leq*,*one*] @ *nenv*)
$\longleftrightarrow$
sats(*M*,*prebody_fm*(φ,*nenv*),[*fst*(*x*),*snd*(*x*),α,*P*,*leq*,*one*] @ *nenv*)
⟨*proof*⟩

lemma *body_lemma*:
  assumes
    $\exists$ *t p. x*=$\langle$*t*,*p*$\rangle$ *x*$\in$*M* [*x*,α,*m*,*P*,*leq*,*one*] @ *nenv* $\in$*list*(*M*)
    $\varphi$$\in$*formula* *arity*($\varphi$) $\leq$ *2* #+ *length*(*nenv*)
  shows
    sats(*M*,*body_fm*(φ,*nenv*),[α,*x*,*m*,*P*,*leq*,*one*] @ *nenv*) $\longleftrightarrow$
  ($\exists$ $\tau$$\in$*M*. $\exists$ *V*$\in$*M*. *is_Vset*(λ*a*. (##*M*)(*a*),α,*V*) $\wedge$ τ $\in$ *V* $\wedge$ (*snd*(*x*) $\Vdash$ φ ([*fst*(*x*),τ]@*nenv*)))
  $\langle$*proof*$\rangle$

lemma *Replace_sats_in_MG*:
  assumes
    c$\in$*M*[*G*] *env* $\in$ *list*(*M*[*G*])
    φ $\in$ *formula* *arity*(φ) $\leq$ *2* #+ *length*(*env*)
    univalent(##*M*[*G*], *c*, λ*x v*. (*M*[*G*] , [*x*,*v*]@*env* $\models$ φ))
  shows
    {*v*. *x*$\in$*c*, *v*$\in$*M*[*G*] $\wedge$ (*M*[*G*] , [*x*,*v*]@*env* $\models$ φ)} $\in$ *M*[*G*]
$\langle$*proof*$\rangle$

theorem *strong_replacement_in_MG*:
  assumes
    $\varphi$$\in$*formula* **and** *arity*(φ) $\leq$ *2* #+ *length*(*env*) *env* $\in$ *list*(*M*[*G*])
  shows
    strong_replacement(##*M*[*G*],λ*x v*. *sats*(*M*[*G*],φ,[*x*,*v*] @ *env*))
$\langle$*proof*$\rangle$

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*]
  $\langle$*proof*$\rangle$

lemma *infinity_in_MG* : *infinity_ax*(##*M*[*G*])
$\langle$*proof*$\rangle$

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)*
  ⟨*proof*⟩

lemma *range_restrict_vimage*:
  assumes *function(f)*
  shows *range(restrict(f,f-‘‘a))* ⊆ *a*
⟨*proof*⟩

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*⟩

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)*
  ⟨*proof*⟩

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*}
  ⟨*proof*⟩

⟨*ML*⟩

lemma *upair_name_abs* :

assumes *x*∈*M* *y*∈*M* *z*∈*M*
shows *is_upair_name*(##*M*,*x*,*y*,*z*) ⟷ *z* = *upair_name*(*x*,*y*)
⟨*proof*⟩

definition
opair_name :: *i* ⇒ *i* ⇒ *i* **where**
opair_name(τ,ϱ) ≡ *upair_name*(*upair_name*(τ,τ),*upair_name*(τ,ϱ))

⟨*ML*⟩

lemma *upair_name_closed* :
⟦ *x*∈*M*; *y*∈*M* ⟧ ⟹ *upair_name*(*x*,*y*)∈*M*
⟨*proof*⟩

definition
upair_name_fm :: [*i*,*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*
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *opair_name_abs* :
assumes *x*∈*M* *y*∈*M* *z*∈*M*
shows *is_opair_name*(##*M*,*x*,*y*,*z*) ⟷ *z* = *opair_name*(*x*,*y*)
⟨*proof*⟩

lemma *opair_name_closed* :
⟦ *x*∈*M*; *y*∈*M* ⟧ ⟹ *opair_name*(*x*,*y*)∈*M*
⟨*proof*⟩

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*
⟨*proof*⟩

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))
⟨*proof*⟩

lemma *val_upair_name* : *val(P,G,upair_name(τ,ϱ))* = {*val(P,G,τ),val(P,G,ϱ)*}
⟨*proof*⟩

lemma *val_opair_name* : *val(P,G,opair_name(τ,ϱ))* = ⟨*val(P,G,τ),val(P,G,ϱ)*⟩
⟨*proof*⟩

lemma *val_RepFun_one*: *val(P,G,{⟨f(x),one⟩ . x∈a})* = {*val(P,G,f(x)) . x∈a*}
⟨*proof*⟩

43.1 $M[G]$ is a transitive model of ZF

interpretation *mgzf*: *M_ZF_trans M[G]*
⟨*proof*⟩

definition
is_opname_check :: [*i,i,i*] ⇒ *o* **where**
is_opname_check(s,x,y) ≡ ∃ *chx*∈*M*. ∃ *sx*∈*M*. *is_check(x,chx)* ∧ *fun_apply(##M,s,x,sx)*
∧
is_opair_name(##M,chx,sx,y)

definition
opname_check_fm :: [*i,i,i,i*] ⇒ *i* **where**
opname_check_fm(s,x,y,o) ≡ *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*∈*nat*;*x*∈*nat*;*y*∈*nat*;*o*∈*nat*⟧ ⟹ *opname_check_fm(s,x,y,o)*∈*formula*
⟨*proof*⟩

lemma *sats_opname_check_fm*:
assumes *x*∈*nat y*∈*nat z*∈*nat o*∈*nat env*∈*list(M) nth(o,env)=one*
y<length(env)
shows
sats(M,opname_check_fm(x,y,z,o),env) ⟷ is_opname_check(nth(x,env),nth(y,env),nth(z,env))
⟨*proof*⟩

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)*
⟨*proof*⟩

lemma *repl_opname_check* :

assumes
A∈*M* *f*∈*M*
shows
{*opair_name*(*check*(*x*),*f*‘*x*). *x*∈*A*}∈*M*
⟨*proof*⟩

theorem *choice_in_MG*:
assumes *choice_ax*(##*M*)
shows *choice_ax*(##*M*[*G*])
⟨*proof*⟩

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*⟩

lemma *Ord_MG_iff*:
assumes *Ord*(*α*)
shows *α* ∈ *M* ⟷ *α* ∈ *M*[*G*]
⟨*proof*⟩

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*⟩

lemma (**in** *G_generic*) *MG_eqpoll_nat*: $M[G] \approx \omega$
⟨*proof*⟩

45.2 The main result

theorem *extensions_of_ctms*:
  assumes
    $M \approx \omega$ $Transset(M)$ $M \models ZF$
  shows
    $\exists N.$
      $M \subseteq N \wedge N \approx \omega \wedge Transset(N) \wedge N \models ZF \wedge M \neq N \wedge$
      $(\forall \alpha.\ Ord(\alpha) \longrightarrow (\alpha \in M \longleftrightarrow \alpha \in N)) \wedge$
      $((M, [] \models \cdot AC\cdot) \longrightarrow N \models ZFC)$
⟨*proof*⟩

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 $|\{S`a\ .\ a \in A\}|^M \le |A|^M$
⟨*proof*⟩

lemma *subset_imp_le_cardinal_rel*: $A \subseteq B \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow |A|^M \le |B|^M$
  ⟨*proof*⟩

lemma *lt_cardinal_rel_imp_not_subset*: $|A|^M < |B|^M \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow \neg\ B \subseteq A$

⟨*proof*⟩

lemma *cardinal_rel_lt_csucc_rel_iff*:
Card_rel$(M,K) \Longrightarrow M(K) \Longrightarrow M(K') \Longrightarrow |K'|^M < (K^+)^M \longleftrightarrow |K'|^M \leq K$
⟨*proof*⟩

lemmas *inj_rel_is_fun* = *inj_is_fun*[*OF mem_inj_rel*]

lemma *inj_rel_bij_rel_range*: $f \in inj^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow f \in bij^M(A,range(f))$
⟨*proof*⟩

lemma *bij_rel_is_inj_rel*: $f \in bij^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow f \in inj^M(A,B)$
⟨*proof*⟩

lemma *inj_rel_weaken_type*: [| $f \in inj^M(A,B)$; $B \subseteq D$; $M(A)$; $M(B)$; $M(D)$ |] ==> $f \in inj^M(A,D)$
⟨*proof*⟩

lemma *bij_rel_converse_bij_rel* [*TC*]: $f \in bij^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B)$ ==> *converse*(f): $bij^M(B,A)$
⟨*proof*⟩

lemma *bij_rel_is_fun_rel*: $f \in bij^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow f \in A \rightarrow^M B$
⟨*proof*⟩

lemmas *bij_rel_is_fun* = *bij_rel_is_fun_rel*[*THEN mem_function_space_rel*]

lemma *comp_bij_rel*:
$g \in bij^M(A,B) \Longrightarrow f \in bij^M(B,C) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (f\ O\ g) \in bij^M(A,C)$
⟨*proof*⟩

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*⟩

end

locale *M_cardinal_UN_nat* = *M_cardinal_UN* _ ω *X* **for** *X*
begin
lemma *cardinal_rel_UN_le_nat*:
assumes $\bigwedge i.\ i \in \omega \Longrightarrow |X(i)|^M \leq \omega$
shows $|\bigcup i \in \omega.\ X(i)|^M \leq \omega$
⟨*proof*⟩

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 $Y(k) \equiv$ *if k∈range(f) then X(converse(f)‘k) else 0*
assumes $InfCard^M(K)$ $\bigwedge i.\ i{\in}J \Longrightarrow |X(i)|^M \le K$
shows $|\bigcup i{\in}J.\ X(i)|^M \le K$
⟨*proof*⟩

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 $InfCard^M(K)$ $J \lesssim^M K$ $\bigwedge i.\ i{\in}J \Longrightarrow |X(i)|^M \le K$
$M(K)$
shows $|\bigcup i{\in}J.\ X(i)|^M \le K$
⟨*proof*⟩

end

context *M_library*
begin

lemma *cardinal_rel_lt_csucc_rel_iff'*:
includes *Ord_dests*
assumes *Card_rel*(*M*,*κ*)
and *types*:$M(\kappa)$ $M(X)$
shows $\kappa < |X|^M \longleftrightarrow (\kappa^+)^M \le |X|^M$
⟨*proof*⟩

lemma *lepoll_rel_imp_subset_bij_rel*:
assumes $M(X)$ $M(Y)$
shows $X \lesssim^M Y \longleftrightarrow (\exists Z[M].\ Z \subseteq Y \wedge Z \approx^M X)$
⟨*proof*⟩

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*:

$Card\_rel(M,\kappa) \Longrightarrow M(\kappa) \Longrightarrow M(X) \Longrightarrow |X|^M = \kappa \longleftrightarrow X \approx^M \kappa$
⟨*proof*⟩

lemma *lepoll_rel_imp_lepoll_rel_cardinal_rel*:
assumes$X \lesssim^M Y$ $M(X)$ $M(Y)$
shows $X \lesssim^M |Y|^M$
⟨*proof*⟩

lemma *lepoll_rel_Un*:
assumes $InfCard\_rel(M,\kappa)$ $A \lesssim^M \kappa$ $B \lesssim^M \kappa$ $M(A)$ $M(B)$ $M(\kappa)$
shows $A \cup B \lesssim^M \kappa$
⟨*proof*⟩

lemma *cardinal_rel_Un_le*:
assumes $InfCard\_rel(M,\kappa)$ $|A|^M \leq \kappa$ $|B|^M \leq \kappa$ $M(\kappa)$ $M(A)$ $M(B)$
shows $|A \cup B|^M \leq \kappa$
⟨*proof*⟩

lemma *eqpoll_rel_imp_Finite*: $A \approx^M B \Longrightarrow Finite(A) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow Finite(B)$
⟨*proof*⟩

lemma *eqpoll_rel_imp_Finite_iff*: $A \approx^M B \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow Finite(A) \longleftrightarrow Finite(B)$
⟨*proof*⟩

lemma *Finite_cardinal_rel_iff'*: $M(i) \Longrightarrow Finite(|i|^M) \longleftrightarrow Finite(i)$
⟨*proof*⟩

lemma *cardinal_rel_subset_of_Card_rel*:
assumes $Card\_rel(M,\gamma)$ $a \subseteq \gamma$ $M(a)$ $M(\gamma)$
shows $|a|^M < \gamma \vee |a|^M = \gamma$
⟨*proof*⟩

lemma *cardinal_rel_cases*:
includes *Ord_dests*
assumes $M(\gamma)$ $M(X)$
shows $Card\_rel(M,\gamma) \Longrightarrow |X|^M < \gamma \longleftrightarrow \neg\ |X|^M \geq \gamma$
⟨*proof*⟩

end

46.2 Countable and uncountable sets

definition
countable :: $i \Rightarrow o$ **where**
$countable(X) \equiv X \lesssim \omega$

⟨*ML*⟩

context *M_library*
begin

lemma *countableI*[*intro*]: $X \lesssim^M \omega \Longrightarrow$ *countable_rel*(M,X)
⟨*proof*⟩

lemma *countableD*[*dest*]: *countable_rel*(M,X) $\Longrightarrow X \lesssim^M \omega$
⟨*proof*⟩

lemma *countable_rel_iff_cardinal_rel_le_nat*: $M(X) \Longrightarrow$ *countable_rel*(M,X) $\longleftrightarrow |X|^M \leq \omega$
⟨*proof*⟩

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)
⟨*proof*⟩
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)
⟨*proof*⟩

lemma *Finite_imp_countable_rel*: *Finite_rel*(M,X) $\Longrightarrow M(X) \Longrightarrow$ *countable_rel*(M,X)
⟨*proof*⟩

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,$\bigcup i \in J.\ X(i)$)
⟨*proof*⟩

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)⟩)
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)⟩)
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*⟩

end

abbreviation
uncountable_rel :: $[i \Rightarrow o,i] \Rightarrow o$ **where**
uncountable_rel$(M,X) \equiv \neg$ *countable_rel*(M,X)

context *M_cardinal_library*
begin

lemma *uncountable_rel_iff_nat_lt_cardinal_rel*:
$M(X) \Longrightarrow$ *uncountable_rel*$(M,X) \longleftrightarrow \omega < |X|^M$
⟨*proof*⟩

lemma *uncountable_rel_not_empty*: *uncountable_rel*$(M,X) \Longrightarrow X \neq 0$
⟨*proof*⟩

lemma *uncountable_rel_imp_Infinite*: *uncountable_rel*$(M,X) \Longrightarrow M(X) \Longrightarrow$ *Infinite*(X)
⟨*proof*⟩

lemma *uncountable_rel_not_subset_countable_rel*:
assumes *countable_rel*(M,X) *uncountable_rel*(M,Y) $M(X)$ $M(Y)$
shows $\neg\ (Y \subseteq X)$
⟨*proof*⟩

46.3 Results on Aleph_rels

lemma *nat_lt_Aleph_rel1*: $\omega < \aleph_1{}^M$
⟨*proof*⟩

lemma *zero_lt_Aleph_rel1*: $0 < \aleph_1{}^M$
⟨*proof*⟩

lemma *le_Aleph_rel1_nat*: $M(k) \Longrightarrow Card\_rel(M,k) \Longrightarrow k{<}\aleph_1{}^M \Longrightarrow k \le \omega$
⟨*proof*⟩

lemma *lesspoll_rel_Aleph_rel_plus_one*:
assumes $Ord(\alpha)$
and *types*:$M(\alpha)$ $M(d)$
shows $d \prec^M \aleph_{succ(\alpha)}{}^M \longleftrightarrow d \lesssim^M \aleph_\alpha{}^M$
⟨*proof*⟩

lemma *cardinal_rel_Aleph_rel* [*simp*]: $Ord(\alpha) \Longrightarrow M(\alpha) \Longrightarrow |\aleph_\alpha{}^M|^M = \aleph_\alpha{}^M$
⟨*proof*⟩
lemma *Aleph_rel_lesspoll_rel_increasing*:
includes *Aleph_rel_intros*
assumes $M(b)$ $M(a)$
shows $a < b \Longrightarrow \aleph_a{}^M \prec^M \aleph_b{}^M$
⟨*proof*⟩

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*⟩

lemma *UN_if_zero*: $M(K) \Longrightarrow (\bigcup x{\in}K.\ if\ M(x)\ then\ G\ `\ x\ else\ 0) = (\bigcup x{\in}K.\ G\ `\ x)$
⟨*proof*⟩

lemma *mem_F_bound1*:
fixes F G
defines $F \equiv \lambda\_\ x.\ if\ M(x)\ then\ G`x\ else\ 0$
shows $x{\in}F(A,c) \Longrightarrow c \in (range(f) \cup domain(G)\ )$
⟨*proof*⟩

lemma *lt_Aleph_rel_imp_cardinal_rel_UN_le_nat*: $function(G) \Longrightarrow domain(G) \lesssim^M \omega \Longrightarrow$
$\forall n{\in}domain(G).\ |G`n|^M{<}\aleph_1{}^M \Longrightarrow M(G) \Longrightarrow |\bigcup n{\in}domain(G).\ G`n|^M{\le}\omega$
⟨*proof*⟩

lemma *Aleph_rel1_eq_cardinal_rel_vimage*: $f{:}\aleph_1{}^M{\to}^M\omega \Longrightarrow \exists n{\in}\omega.\ |f\text{-}``\{n\}|^M = \aleph_1{}^M$
⟨*proof*⟩

lemma *eqpoll_rel_Aleph_rel1_cardinal_rel_vimage*:
assumes $Z \approx^M (\aleph_1{}^M)$ $f \in Z \to^M \omega$ $M(Z)$
shows $\exists\, n \in \omega.\ |f\text{-}``\{n\}|^M = \aleph_1{}^M$
⟨*proof*⟩

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*$(f,\alpha) = f`(\{$*rec_constr*$(f,\beta).\ \beta\in\alpha\})$
⟨*proof*⟩

lemma *rec_constr_type*:
assumes f:*Pow_rel*$(M,G)\to^M G$ *Ord*(α) $M(G)$
shows $M(\alpha) \Longrightarrow$ *rec_constr*$(f,\alpha) \in G$
⟨*proof*⟩

lemma *rec_constr_closed* :
assumes f:*Pow_rel*$(M,G)\to^M G$ *Ord*(α) $M(G)$ $M(\alpha)$
shows $M($*rec_constr*$(f,\alpha))$
⟨*proof*⟩

lemma *lambda_rec_constr_closed* :
assumes *Ord*(γ) $M(\gamma)$ $M(f)$ f:*Pow_rel*$(M,G)\to^M G$ $M(G)$
shows $M(\lambda\alpha\in\gamma$. *rec_constr*$(f,\alpha))$
⟨*proof*⟩

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
$\bigwedge Z.\ |Z|^M < \gamma \Longrightarrow Z \subseteq G \Longrightarrow M(Z) \Longrightarrow \exists\, a\in G.\ \forall\, s\in Z.\ <s,a>\in Q$ $b\in G$
Card_rel(M,γ)
$M(G)$ $M(Q)$ $M(\gamma)$
shows
$\exists\, S[M].\ S : \gamma \to^M G \wedge (\forall\, \alpha \in \gamma.\ \forall\, \beta \in \gamma.\ \ \alpha<\beta \longrightarrow <S`\alpha,S`\beta>\in Q)$
⟨*proof*⟩

The following basic result can, in turn, be proved by a bounded-cardinal_rel selection.

lemma *Infinite_iff_lepoll_rel_nat*: $M(Z) \Longrightarrow$ *Infinite*$(Z) \longleftrightarrow \omega \lesssim^{M} Z$
⟨*proof*⟩

lemma *Infinite_InfCard_rel_cardinal_rel*: *Infinite*$(Z) \Longrightarrow M(Z) \Longrightarrow$ *InfCard_rel*$(M,|Z|^{M})$
⟨*proof*⟩

lemma (**in** *M_trans*) *mem_F_bound2*:
fixes F A
defines $F \equiv \lambda\_\ x.$ *if* $M(x)$ *then* A-‘‘$\{x\}$ *else* 0
shows $x{\in}F(A,c) \Longrightarrow c \in$ (*range*$(f) \cup$ *range*(A))
⟨*proof*⟩

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*⟩

lemma *cardinal_rel_map_Un*:
assumes *Infinite*(X) *Finite*(b) $M(X)$ $M(b)$
shows $|\{a \cup b\ .\ a \in X\}|^{M} = |X|^{M}$
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *delta_systemD*[*dest*]:
delta_system$(D) \Longrightarrow \exists\, r.\ \forall\, A{\in}D.\ \forall\, B{\in}D.\ A \neq B \longrightarrow A \cap B = r$

⟨*proof*⟩

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*⟩

⟨*ML*⟩

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 $x\in F(G,c) \Longrightarrow c \in (range(f) \cup \bigcup G)$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma *Finite_imp_lesspoll_rel_nat*:
assumes $Finite(A)$ $M(A)$
shows $A \prec^M \omega$
⟨*proof*⟩

lemma *InfCard_rel_lesspoll_rel_Un*:
includes *Ord_dests*
assumes $InfCard\_rel(M,\kappa)$ $A \prec^M \kappa$ $B \prec^M \kappa$

and *types*: $M(\kappa)$ $M(A)$ $M(B)$
shows $A \cup B \prec^M \kappa$
⟨*proof*⟩

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*)
⟨*proof*⟩

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
    $ccc^M(P,leq) \longleftrightarrow (\forall A[M].\ antichain^M(P,leq,A) \longrightarrow |A|^M \leq \omega)$
  ⟨*proof*⟩

end

context *M_add_reals*
begin

lemma *lam_replacement_drSR_Y*: $M(A) \Longrightarrow M(D) \Longrightarrow M(r') \Longrightarrow lam\_replacement(M, drSR\_Y(r',D,A))$
  ⟨*proof*⟩

lemma (**in** *M_trans*) *mem_F_bound3*:
  fixes *F A*
  defines $F \equiv dC\_F$
  shows $x \in F(A,c) \Longrightarrow c \in (range(f) \cup \{domain(x).\ x \in A\})$
  ⟨*proof*⟩

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*⟩

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*)))
⟨*proof*⟩

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*)

⟨*ML*⟩
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*(*succ*(*r*))))))))), *0*) ··)

lemma *type_omap_wfrec_body_fm* :*A*∈*nat* ⟹ *r*∈*nat* ⟹ *omap_wfrec_body*(*A*,*r*)∈*formula*
⟨*proof*⟩

lemma *arity_aux* : *A*∈*nat* ⟹ *r*∈*nat* ⟹ *arity*(*omap_wfrec_body*(*A*,*r*)) = (*9*#+*A*) ∪ (*9*#+*r*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *arity_isordermap*: *A*∈*nat* ⟹ *r*∈*nat* ⟹*d*∈*nat*⟹
arity(*is_ordermap_fm*(*A*,*r*,*d*)) = *succ*(*d*) ∪ (*succ*(*A*) ∪ *succ*(*r*))
⟨*proof*⟩

lemma *arity_is_ordertype*: *A*∈*nat* ⟹ *r*∈*nat* ⟹*d*∈*nat*⟹
arity(*is_ordertype_fm*(*A*,*r*,*d*)) = *succ*(*d*) ∪ (*succ*(*A*) ∪ *succ*(*r*))
⟨*proof*⟩

⟨*ML*⟩

lemma *arity_is_order_body*: *arity*(*is_order_body_fm*(*2*,*0*,*1*)) = *3*
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_is_order_body*:
X∈*M* ⟹ *strong_replacement*(##*M*, *is_order_body*(##*M*,*X*))
⟨*proof*⟩

lemma (**in** *M_pre_cardinal_arith*) *is_order_body_abs* :

$M(X) \Longrightarrow M(x) \Longrightarrow M(z) \Longrightarrow is\_order\_body(M, X, x, z) \longleftrightarrow$
$M(z) \wedge M(x) \wedge x{\in}Pow\_rel(M,X{\times}X) \wedge well\_ord(X, x) \wedge z = ordertype(X, x)$
⟨*proof*⟩

definition *H_order_pred* **where**
$H\_order\_pred(A,r) \equiv \lambda x\, f\, .\, f \text{ `` } Order.pred(A, x, r)$

⟨*ML*⟩

lemma (**in** *M_basic*) *H_order_pred_abs* :
$M(A) \Longrightarrow M(r) \Longrightarrow M(x) \Longrightarrow M(f) \Longrightarrow M(z) \Longrightarrow$
$is\_H\_order\_pred(M,A,r,x,f,z) \longleftrightarrow z = H\_order\_pred(A,r,x,f)$
⟨*proof*⟩

⟨*ML*⟩

definition *order_pred_wfrec_body* **where**
$order\_pred\_wfrec\_body(M,A,r,z,x) \equiv \exists\, y[M].$
$pair(M, x, y, z) \wedge$
$(\exists\, f[M].$
$(\forall\, z[M].$
$z \in f \longleftrightarrow$
$(\exists\, xa[M].$
$\exists\, y[M].$
$\exists\, xaa[M].$
$\exists\, sx[M].$
$\exists\, r\_sx[M].$
$\exists\, f\_r\_sx[M].$
$pair(M, xa, y, z) \wedge$
$pair(M, xa, x, xaa) \wedge$
$upair(M, xa, xa, sx) \wedge$
$pre\_image(M, r, sx, r\_sx) \wedge$
$restriction(M, f, r\_sx, f\_r\_sx) \wedge$
$xaa \in r \wedge (\exists\, a[M].\ image(M, f\_r\_sx, a, y) \wedge$
$pred\_set(M, A, xa, r, a)))) \wedge$
$(\exists\, a[M].\ image(M, f, a, y) \wedge pred\_set(M, A, x, r, a)))$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *wfrec_replacement_order_pred*:
$A{\in}M \Longrightarrow r{\in}M \Longrightarrow wfrec\_replacement(\#\#M, \lambda x\, g\, z.\ is\_H\_order\_pred(\#\#M,A,r,x,g,z)$
$, r)$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *wfrec_replacement_order_pred'*:
$A{\in}M \Longrightarrow r{\in}M \Longrightarrow wfrec\_replacement(\#\#M, \lambda x\, g\, z.\ z = H\_order\_pred(A,r,x,g)$
$, r)$

⟨*proof*⟩

sublocale *M_ZF_trans* ⊆ *M_pre_cardinal_arith* *##M*
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_ordertype*:
$X{\in}M \Longrightarrow$ *strong_replacement*(*##M*, $\lambda x\ z.\ z \in M \wedge x \in M \wedge x \in Pow^M(X \times X) \wedge well\_ord(X, x) \wedge z = ordertype(X, x)$)
⟨*proof*⟩

lemma *arity_is_jump_cardinal_body*: *arity*(*is_jump_cardinal_body′_fm*(*0*,*1*)) = *2*
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_is_jump_cardinal_body*:
strong_replacement(*##M*, *is_jump_cardinal_body′*(*##M*))
⟨*proof*⟩

lemma (**in** *M_pre_cardinal_arith*) *univalent_aux2*: $M(X) \Longrightarrow$ *univalent*(*M*,*Pow_rel*(*M*,*X*×*X*), $\lambda r\ z.\ M(z) \wedge M(r) \wedge$ *is_well_ord*(*M*, *X*, *r*) ∧ *is_ordertype*(*M*, *X*, *r*, *z*))
⟨*proof*⟩

lemma (**in** *M_pre_cardinal_arith*) *is_jump_cardinal_body_abs* :
$M(X) \Longrightarrow M(c) \Longrightarrow$ *is_jump_cardinal_body′*(*M*, *X*, *c*) ⟷ *c* = *jump_cardinal_body′_rel*(*M*,*X*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_jump_cardinal_body*:
strong_replacement(*##M*, $\lambda x\ z.\ z \in M \wedge x \in M \wedge z =$ *jump_cardinal_body*(*##M*, *x*))
⟨*proof*⟩

sublocale *M_ZF_trans* ⊆ *M_pre_aleph* *##M*
⟨*proof*⟩

⟨*ML*⟩
lemma *arity_is_HAleph_fm*: *arity*(*is_HAleph_fm*(*2*, *1*, *0*)) = *3*
⟨*proof*⟩

lemma *arity_is_Aleph*: *arity*(*is_Aleph_fm*(*0*, *1*)) = *2*
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_is_aleph*:
strong_replacement(*##M*, $\lambda x\ y.$ *Ord*(*x*) ∧ *is_Aleph*(*##M*,*x*,*y*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *replacement_aleph_rel*:
shows *strong_replacement*(*##M*, $\lambda x\ y.\ Ord(x) \wedge y = \aleph_x{}^M$)
⟨*proof*⟩

sublocale *M_ZF_trans* ⊆ *M_aleph* ##*M*
⟨*proof*⟩

sublocale *M_ZF_trans* ⊆ *M_FiniteFun* ##*M*
⟨*proof*⟩

sublocale *M_ZFC_trans* ⊆ *M_AC* ##*M*
⟨*proof*⟩

sublocale *M_ZFC_trans* ⊆ *M_cardinal_AC* ##*M* ⟨*proof*⟩

definition *toplevel1_body* :: [*i*,*i*,*i*] ⇒ *o* **where**
toplevel1_body(*Q*,*x*) ≡ λ*a*. ∀ *s*∈*x*. ⟨*s*, *a*⟩ ∈ *Q*

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_toplevel1_body*:
(##*M*)(*A*) ⟹ (##*M*)(*B*) ⟹ *separation*(##*M*, *is_toplevel1_body*(##*M*,*A*,*B*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *toplevel1_body_abs*:
assumes (##*M*)(*A*) (##*M*)(*B*) (##*M*)(*x*)
shows *is_toplevel1_body*(##*M*,*A*,*B*,*x*) ⟷ *toplevel1_body*(*A*,*B*,*x*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_toplevel1_body*:
(##*M*)(*Q*) ⟹ (##*M*)(*x*) ⟹ *separation*(##*M*, λ*a*. ∀ *s*∈*x*. ⟨*s*, *a*⟩ ∈ *Q*)
⟨*proof*⟩

definition *toplevel2_body* :: [*i*,*i*] ⇒ *o* **where**
toplevel2_body(*x*) ≡ λ*a*. |*a*| < *x*

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_toplevel2_body*:
(##*M*)(*A*) ⟹ *separation*(##*M*, *is_toplevel2_body*(##*M*,*A*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *toplevel2_body_abs*:
assumes (##*M*)(*A*) (##*M*)(*x*)
shows *is_toplevel2_body*(##*M*,*A*,*x*) ⟷ *toplevel2_body_rel*(##*M*,*A*,*x*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_toplevel2_body*:
(##*M*)(*x*) ⟹ *separation*(##*M*, λ*a*. $|a|^M$ < *x*)

⟨*proof*⟩

definition *toplevel3_body* :: $i \Rightarrow o$ **where**
toplevel3_body $\equiv$ $\lambda x.\ \exists\, a\ b.\ x = \langle a,\ b\rangle \wedge a \neq b$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_toplevel3_body*:
separation(##*M*, *is_toplevel3_body*(##*M*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *toplevel3_body_abs*:
assumes (##*M*)(*x*)
shows *is_toplevel3_body*(##*M*,*x*) $\longleftrightarrow$ *toplevel3_body*(*x*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_toplevel3_body*:
separation(##*M*, $\lambda x\ .\ \exists\, a\ b.\ x = \langle a,\ b\rangle \wedge a \neq b$)
⟨*proof*⟩

definition *toplevel4_body* :: $[i,i] \Rightarrow o$ **where**
toplevel4_body(*R*) $\equiv \lambda z.\ \exists\, a\ b.\ z = \langle a,\ b\rangle \wedge a \cap b = R$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_toplevel4_body*:
(##*M*)(*A*) $\Longrightarrow$ *separation*(##*M*, *is_toplevel4_body*(##*M*,*A*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *toplevel4_body_abs*:
assumes (##*M*)(*R*) (##*M*)(*x*)
shows *is_toplevel4_body*(##*M*,*R*,*x*) $\longleftrightarrow$ *toplevel4_body*(*R*,*x*)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_toplevel4_body*:
(##*M*)(*R*) $\Longrightarrow$ *separation*
(##*M*, $\lambda x.\ \exists\, a\ b.\ x = \langle a,\ b\rangle \wedge a \cap b = R$)
⟨*proof*⟩

definition *toplevel5_body* :: $[i,i] \Rightarrow o$ **where**
toplevel5_body(*R*) $\equiv \lambda x.$ *domain*(*x*) $\in R$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_toplevel5_body*:
(##*M*)(*A*) $\Longrightarrow$ *separation*(##*M*, *is_toplevel5_body*(##*M*,*A*))

⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *toplevel5_body_abs*:
assumes $(\#\#M)(R)$ $(\#\#M)(x)$
shows *is_toplevel5_body*$(\#\#M,R,x) \longleftrightarrow$ *toplevel5_body*(R,x)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_toplevel5_body*:
$(\#\#M)(R) \Longrightarrow$ *separation*
$(\#\#M,\ \lambda x.\ domain(x) \in R)$
⟨*proof*⟩

definition *toplevel7_body* :: $[i,i,i] \Rightarrow o$ **where**
toplevel7_body$(Q,x) \equiv \lambda a.\ restrict(a,\ Q) = x$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_toplevel7_body*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow$ *separation*$(\#\#M,$ *is_toplevel7_body*$(\#\#M,A,B))$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *toplevel7_body_abs*:
assumes $(\#\#M)(A)$ $(\#\#M)(B)$ $(\#\#M)(x)$
shows *is_toplevel7_body*$(\#\#M,A,B,x) \longleftrightarrow$ *toplevel7_body*(A,B,x)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_toplevel7_body*:
$(\#\#M)(Q) \Longrightarrow (\#\#M)(x) \Longrightarrow$ *separation*$(\#\#M,\ \lambda a.\ restrict(a,\ Q) = x)$
⟨*proof*⟩

definition *toplevel8_body* :: $[i,i] \Rightarrow o$ **where**
toplevel8_body$(R) \equiv \lambda z.\ R \in domain(z)$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_toplevel8_body*:
$(\#\#M)(A) \Longrightarrow$ *separation*$(\#\#M,$ *is_toplevel8_body*$(\#\#M,A))$
⟨*proof*⟩

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)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_toplevel8_body*:
$(\#\#M)(R) \Longrightarrow$ *separation*
$(\#\#M,\ \lambda z.\ R \in domain(z))$

⟨*proof*⟩

definition *toplevel9_body* :: $[i,i,i] \Rightarrow o$ **where**
toplevel9_body(Q,x) $\equiv \lambda z.\ \exists n\in\omega.\ \langle\langle Q,\ n\rangle,\ 1\rangle \in z \wedge \langle\langle x,\ n\rangle,\ 0\rangle \in z$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_toplevel9_body*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow$ *separation*(##M, *is_toplevel9_body*(##M,A,B))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *toplevel9_body_abs*:
assumes $(\#\#M)(A)$ $(\#\#M)(B)$ $(\#\#M)(x)$
shows *is_toplevel9_body*(##M,A,B,x) $\longleftrightarrow$ *toplevel9_body*(A,B,x)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_toplevel9_body*:
$(\#\#M)(Q) \Longrightarrow (\#\#M)(x) \Longrightarrow$ *separation*(##M, $\lambda z.\ \exists n\in\omega.\ \langle\langle Q,\ n\rangle,\ 1\rangle \in z \wedge \langle\langle x,\ n\rangle,\ 0\rangle \in z$)
⟨*proof*⟩

definition *toplevel10_body* :: $[i,i,i] \Rightarrow o$ **where**
toplevel10_body(A,r) $\equiv$ $\lambda y.\ \exists x\in A.\ y = \langle x,$ *restrict*$(x,\ r)\rangle$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_toplevel10_body*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(r) \Longrightarrow$ *separation*(##M, *is_toplevel10_body*(##M,A,r))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *toplevel10_body_abs*:
assumes $(\#\#M)(A)$ $(\#\#M)(r)$ $(\#\#M)(x)$
shows *is_toplevel10_body*(##M,A,r,x) $\longleftrightarrow$ *toplevel10_body*(A,r,x)
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_toplevel10_body*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(r) \Longrightarrow$ *separation*(##M, $\lambda y.\ \exists x\in A.\ y = \langle x,$ *restrict*$(x,\ r)\rangle$)
⟨*proof*⟩

definition *toplevel11_body* :: $[i,i] \Rightarrow o$ **where**
toplevel11_body(A) $\equiv$ $\lambda p.\ (\forall x\in A.\ (x \in$ *snd*$(p) \longleftrightarrow$ *domain*$(x) =$ *fst*$(p)))$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_toplevel11_body*:
$(\#\#M)(A) \Longrightarrow separation(\#\#M,\ is\_toplevel11\_body(\#\#M,A))$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *toplevel11_body_abs*:
assumes $(\#\#M)(A)$ $(\#\#M)(x)$
shows $is\_toplevel11\_body(\#\#M,A,x) \longleftrightarrow toplevel11\_body(A,x)$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_toplevel11_body*:
$(\#\#M)(A) \Longrightarrow separation(\#\#M,\ \lambda p.\ \forall x{\in}A.\ x \in snd(p) \longleftrightarrow domain(x) = fst(p))$
⟨*proof*⟩

definition *toplevel12_body*
where $toplevel12\_body(G,p) \equiv \forall x{\in}G.\ x \in snd(p) \longleftrightarrow fst(p) \in x$

⟨*ML*⟩

lemma (**in** *M_ZF_trans*) *separation_is_toplevel12_body*:
$(\#\#M)(G) \Longrightarrow separation(\#\#M,\ is\_toplevel12\_body(\#\#M,G))$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *toplevel12_body_abs*:
assumes $(\#\#M)(G)$ $(\#\#M)(x)$
shows $is\_toplevel12\_body(\#\#M,G,x) \longleftrightarrow toplevel12\_body(G,x)$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *separation_toplevel12_body*:
$(\#\#M)(G) \Longrightarrow separation$
$(\#\#M,\ \lambda p.\ \ \forall x{\in}G.\ x \in snd(p) \longleftrightarrow fst(p) \in x)$
⟨*proof*⟩

end
theory *Cardinal_Preservation*
imports
Cohen_Posets_Relative
Forcing_Main
ZF_Trans_Interpretations
begin

context *forcing_notion*
begin

definition
antichain :: $i \Rightarrow o$ **where**
$antichain(A) \equiv A{\subseteq}P \land (\forall p{\in}A.\ \forall q{\in}A.\ p \neq q \longrightarrow p \perp q)$

definition

ccc :: *o* **where**
ccc ≡ ∀ *A*. *antichain*(*A*) ⟶ |*A*| ≤ *ω*

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*)
⟨*proof*⟩

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* ⟨*proof*⟩

context *forcing_data*
begin

lemma *antichain_abs″* [*absolut*]: *A*∈*M* ⟹ *antichain_r′*(*A*) ⟷ *antichain*(*A*)
⟨*proof*⟩

end

lemma (**in** *forcing_notion*) *Incompatible_imp_not_eq*: ⟦ *p* ⊥ *q*; *p*∈*P*; *q*∈*P* ⟧⟹ *p* ≠ *q*
⟨*proof*⟩

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*⟩

notation (**in** *forcing_data*) *check* (‹_^{v}› [*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*]
⟨*proof*⟩

sublocale *G_generic_AC* ⊆ *ext*:*M_ZFC_trans M*[*G*]
⟨*proof*⟩

lemma (**in** *forcing_data*) *forces_neq_apply_imp_incompatible*:
assumes
$p \Vdash \cdot 0`1\ is\ 2\cdot\ [f,a,b^v]$
$q \Vdash \cdot 0`1\ is\ 2\cdot\ [f,a,b'^v]$
$b \neq b'$
— 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*⟩
include *G_generic_lemmas*
— FIXME: make a locale containg two *M_ZF_trans* instances, one for *M* and one for *M*[*G*]
⟨*proof*⟩

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 $[\![(\#\#M[G])(?f); (\#\#M[G])(?a)]\!] \Longrightarrow (\#\#M[G])(?f \text{ ` } ?a)$ 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*⟩

abbreviation
fm_leq :: $[i,i,i] \Rightarrow i$ (‹_≼-_›) **where**
$fm\_leq(A,l,B) \equiv leq\_fm(l,A,B)$

⟨*ML*⟩

lemma *ccc_fun_closed_lemma_aux*:
  assumes *f_dot*∈*M p*∈*M a*∈*M b*∈*M*
  shows $\{q \in P . q \preceq p \wedge (M, [q, P, leq, one, f\_dot, a^v, b^v] \models forces(\cdot 0`1\ is\ 2\cdot$
$))\} \in M$
⟨*proof*⟩

lemma *ccc_fun_closed_lemma_aux2*:
  assumes *B*∈*M f_dot*∈*M p*∈*M a*∈*M*
  shows $(\#\#M)(\lambda b{\in}B. \{q \in P . q \preceq p \wedge (M, [q, P, leq, one, f\_dot, a^v, b^v] \models$
$forces(\cdot 0`1\ is\ 2\cdot\ ))\})$
⟨*proof*⟩

lemma *ccc_fun_closed_lemma*:
  assumes *A*∈*M B*∈*M f_dot*∈*M p*∈*M*
  shows $(\lambda a{\in}A. \{b{\in}B. \exists q{\in}P. q \preceq p \wedge (q \Vdash \cdot 0`1\ is\ 2\cdot\ [f\_dot, a^v, b^v])\}) \in M$
⟨*proof*⟩
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
    $\exists F{\in}M. F : A \to Pow(B) \wedge (\forall a{\in}A. f`a \in F`a \wedge |F`a|^M \leq \omega)$
⟨*proof*⟩

end

end

end
theory *Not_CH*
  imports
    Cardinal_Preservation
begin

definition
  Add_subs :: [*i*,*i*] ⇒ *i* **where**
  $Add\_subs(\kappa,\alpha) \equiv Fn(\omega,\kappa\times\alpha,2)$

locale *M_master* = *M_cohen* +
  assumes
  domain_separation: $M(x) \Longrightarrow separation(M, \lambda z. x \in domain(z))$
  and
  inj_dense_separation: $M(x) \Longrightarrow M(w) \Longrightarrow$
    $separation(M, \lambda z. \exists n{\in}\omega.\ \ \langle\langle w, n\rangle, 1\rangle \in z \wedge \langle\langle x, n\rangle, 0\rangle \in z)$
  and
  lam_apply_replacement: $M(A) \Longrightarrow M(f) \Longrightarrow$
      $strong\_replacement(M, \lambda x\ y. y = \langle x, \lambda n{\in}A. f\ `\ \langle x, n\rangle\rangle)$
  and

UN_lepoll_assumptions:
$M(A) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow M(A') \Longrightarrow$ *separation*$(M, \lambda y. \exists x\in A'. y = \langle x, \mu i. x\in$ *if_range_F_else_F*$((`)(A), b, f, i)\rangle)$

begin

lemma (**in** *M_FiniteFun*) *Fn_nat_closed*:
assumes $M(A)$ $M(B)$ **shows** $M(Fn(\omega,A,B))$
⟨*proof*⟩

lemma *Aleph_rel2_closed*[*intro*,*simp*]: $M(\aleph_2{}^M)$
⟨*proof*⟩

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* ⊆ *M_N_Perm*
⟨*proof*⟩

context *M_master_sub*
begin

lemma *cardinal_rel_le_cardinal_rel*: $M(X) \Longrightarrow |X|^N \le |X|^M$
⟨*proof*⟩

lemma *Aleph_rel_sub_closed*: $Ord(\alpha) \Longrightarrow M(\alpha) \Longrightarrow N(\aleph_\alpha{}^M)$
⟨*proof*⟩

lemma *Card_rel_imp_Card_rel*: $M(\kappa) \Longrightarrow Card^N(\kappa) \Longrightarrow Card^M(\kappa)$
⟨*proof*⟩

lemma *csucc_rel_le_csucc_rel*:
assumes $Ord(\kappa)$ $M(\kappa)$
shows $(\kappa^+)^M \le (\kappa^+)^N$
⟨*proof*⟩

lemma *Aleph_rel_le_Aleph_rel*: $Ord(\alpha) \Longrightarrow M(\alpha) \Longrightarrow \aleph_\alpha{}^M \le \aleph_\alpha{}^N$
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

end

sublocale *G_generic_AC* ⊆ *M_master_sub* ##*M* ##(*M*[*G*])
⟨*proof*⟩

lemma (**in** *M_trans*) *mem_F_bound4*:
fixes *F A*
defines *F* ≡ (‘)
shows *x*∈*F*(*A*,*c*) ⟹ *c* ∈ (*range*(*f*) ∪ *domain*(*A*))
⟨*proof*⟩

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*))
⟨*proof*⟩

context *G_generic_AC* **begin**

context
includes *G_generic_lemmas*
begin

lemma *G_in_MG*: *G* ∈ *M*[*G*]
⟨*proof*⟩

lemma *ccc_preserves_Aleph_succ*:

assumes $ccc^M(P,leq)$ $Ord(z)$ $z \in M$
shows $Card^{M[G]}(\aleph_{succ(z)}{}^M)$
⟨*proof*⟩

end

end

context *M_ctm*
begin

abbreviation
Add :: *i* **where**
$Add \equiv Fn(\omega,\ \aleph_2{}^M \times \omega,\ 2)$

end

locale *add_generic* = *G_generic_AC* $Fn(\omega,\ \aleph_2{}^{\#\#M} \times \omega,\ 2)$ $Fnle(\omega,\ \aleph_2{}^{\#\#M} \times \omega,\ 2)$ *0*

sublocale *add_generic* ⊆ *cohen_data* ω $\aleph_2{}^M \times \omega$ *2* ⟨*proof*⟩

context *add_generic*
begin

notation *Leq* (**infixl** ≼ *50*)
notation *Incompatible* (**infixl** ⊥ *50*)
notation *GenExt_at_P* (_[_] [*71*,*1*])

lemma *Add_subs_preserves_Aleph_succ*: $Ord(z) \Longrightarrow z{\in}M \Longrightarrow Card^{M[G]}(\aleph_{succ(z)}{}^M)$
⟨*proof*⟩

lemma *Aleph_rel_nats_MG_eq_Aleph_rel_nats_M*:
includes *G_generic_lemmas*
assumes $z \in \omega$
shows $\aleph_z{}^{M[G]} = \aleph_z{}^M$
⟨*proof*⟩

abbreviation
f_G :: *i* ($\langle f_G \rangle$) **where**
$f_G \equiv \bigcup G$

abbreviation
dom_dense :: *i*⇒*i* **where**
$dom\_dense(x) \equiv \{\ p{\in}Add\ .\ x \in domain(p)\ \}$

— FIXME write general versions of this for $Fn(\omega,\ I,\ J)$ 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*⟩

lemma *dom_dense_closed*[*intro*,*simp*]: $x \in \aleph_2{}^M \times \omega \Longrightarrow$ *dom_dense*(x) $\in M$
⟨*proof*⟩

lemma *domain_f_G*: **assumes** $x \in \aleph_2{}^M$ $y \in \omega$
shows $\langle x,\ y\rangle \in$ *domain*(f_G)
⟨*proof*⟩
lemma *Fn_nat_subset_Pow*: *Fn*(ω,I,J) $\subseteq$ *Pow*($I\times J$)
⟨*proof*⟩

lemma *f_G_funtype*:
includes *G_generic_lemmas*
shows $f_G : \aleph_2{}^M \times \omega \to 2$
⟨*proof*⟩

abbreviation
inj_dense :: $i \Rightarrow i \Rightarrow i$ **where**
inj_dense(w,x) $\equiv$
{ $p \in$*Add* . ($\exists\, n\in\omega$. <<w,n>,1> $\in p \wedge$ <<x,n>,0> $\in p$) }

— 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 $w \in \aleph_2{}^M$ $x \in \aleph_2{}^M$ $w \neq x$
shows *dense*(*inj_dense*(w,x))
⟨*proof*⟩

lemma *inj_dense_closed*[*intro*,*simp*]:
$w \in \aleph_2{}^M \Longrightarrow x \in \aleph_2{}^M \Longrightarrow$ *inj_dense*(w,x) $\in M$
⟨*proof*⟩

lemma *Aleph_rel2_new_reals*:
assumes $w \in \aleph_2{}^M$ $x \in \aleph_2{}^M$ $w \neq x$
shows ($\lambda n\in\omega$. f_G ‘ $\langle w,\ n\rangle$) $\neq$ ($\lambda n\in\omega$. f_G ‘ $\langle x,\ n\rangle$)
⟨*proof*⟩

definition
h_G :: *i* (‹h_G›) **where**
$h_G \equiv \lambda\alpha\in\aleph_2{}^M.\ \lambda n\in\omega.\ f_G$‘<$\alpha$,$n$>

lemma *h_G_in_MG*[*simp*]:
includes *G_generic_lemmas*
shows $h_G \in M[G]$
⟨*proof*⟩

lemma *h_G_inj_Aleph_rel2_reals*: $h_G \in inj^{M[G]}(\aleph_2{}^M,\ \omega \to^{M[G]} 2)$
⟨*proof*⟩

lemma *Aleph2_extension_le_continuum_rel*:
  includes *G_generic_lemmas*
  shows $\aleph_2^{M[G]} \leq 2^{\uparrow \aleph_0^{M[G]}, M[G]}$
⟨*proof*⟩

lemma *Aleph_rel_lt_continuum_rel*: $\aleph_1^{M[G]} < 2^{\uparrow \aleph_0^{M[G]}, M[G]}$
  ⟨*proof*⟩

corollary *not_CH*: $\aleph_1^{M[G]} \neq 2^{\uparrow \aleph_0^{M[G]}, M[G]}$
  ⟨*proof*⟩

end

definition
  ContHyp :: *o* **where**
  $ContHyp \equiv \aleph_1 = 2^{\uparrow \aleph_0}$

⟨*ML*⟩
notation *ContHyp_rel* (‹*CH*–›)
⟨*ML*⟩

context *M_master*
begin

⟨*ML*⟩
  ⟨*proof*⟩

end

⟨*ML*⟩

notation *is_ContHyp_fm* (‹·*CH*·›)

theorem *ctm_of_not_CH*:
  assumes
    $M \approx \omega$ $Transset(M)$ $M \models ZFC$
  shows
    $\exists N.$
      $M \subseteq N \wedge N \approx \omega \wedge Transset(N) \wedge N \models ZFC \cup \{\cdot\neg\cdot CH\cdot\cdot\} \wedge$
      $(\forall \alpha.\ Ord(\alpha) \longrightarrow (\alpha \in M \longleftrightarrow \alpha \in N))$
⟨*proof*⟩

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* $\mathcal{V}$
⟨*proof*⟩

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* $\mathcal{V}$
⟨*proof*⟩

interpretation *V*:*M_eclose* $\mathcal{V}$
⟨*proof*⟩

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* $\mathcal{V}$
⟨*proof*⟩

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* $\mathcal{V}$
⟨*proof*⟩

lemma *choice_ax_Universe*: *choice_ax*($\mathcal{V}$)
⟨*proof*⟩

interpretation *V*:*M_master* $\mathcal{V}$
⟨*proof*⟩

named_theorems *V_simps*

— To work systematically, ASCII versions of "_absolute" theorems as those below are preferable.

lemma *eqpoll_rel_absolute*[*V_simps*]: $x \approx^{\mathcal{V}} y \longleftrightarrow x \approx y$
⟨*proof*⟩

lemma *cardinal_rel_absolute*[*V_simps*]: $|x|^{\mathcal{V}} = |x|$
⟨*proof*⟩

lemma *Card_rel_absolute*[*V_simps*]:$Card^{\mathcal{V}}(a) \longleftrightarrow Card(a)$
⟨*proof*⟩

lemma *csucc_rel_absolute*[*V_simps*]:$(a^{+})^{\mathcal{V}} = a^{+}$
⟨*proof*⟩

lemma *function_space_rel_absolute*[*V_simps*]:$x \to^{\mathcal{V}} y = x \to y$
⟨*proof*⟩

lemma *cexp_rel_absolute*[*V_simps*]:$x^{\uparrow y,\mathcal{V}} = x^{\uparrow y}$
⟨*proof*⟩

lemma *HAleph_rel_absolute*[*V_simps*]:$HAleph\_rel(\mathcal{V},a,b) = HAleph(a,b)$
⟨*proof*⟩

lemma *Aleph_rel_absolute*[*V_simps*]: $Ord(x) \Longrightarrow \aleph_x{}^{\mathcal{V}} = \aleph_x$
⟨*proof*⟩

Example of absolute lemmas obtained from the relative versions. Note the *only* declarations

lemma *Ord_cardinal_idem'*: $Ord(A) \Longrightarrow ||A|| = |A|$
⟨*proof*⟩

lemma *Aleph_succ'*: $Ord(\alpha) \Longrightarrow \aleph_{succ(\alpha)} = \aleph_\alpha{}^{+}$
⟨*proof*⟩

These two results are new, first obtained in relative form (not ported).

lemma *csucc_cardinal*:
assumes $Ord(\kappa)$ **shows** $|\kappa|^{+} = \kappa^{+}$
⟨*proof*⟩

lemma *csucc_le_mono*:
assumes $\kappa \le \nu$ **shows** $\kappa^{+} \le \nu^{+}$
⟨*proof*⟩

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*⟩

The “relationalized” *CH* with respect to $\mathcal{V}$ corresponds to the real *CH*.

lemma *is_ContHyp_iff_CH*: *is_ContHyp*($\mathcal{V}$) ⟷ *ContHyp*
⟨*proof*⟩

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'*: $\forall x.\ x \notin 0$ ⟨*proof*⟩
lemma *succ_def'*: $succ(i) = i \cup \{i\}$ ⟨*proof*⟩

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*

$length([]) = 0$
$length(Cons(a, l)) = succ(length(l))$
$[] \mathbin{@} ys = ys$
$Cons(a, l) \mathbin{@} ys = Cons(a, l \mathbin{@} ys)$
$nth(0, Cons(a, l)) = a$
$n \in \omega \Longrightarrow nth(succ(n), Cons(a, l)) = nth(n, l)$

We have the usual Haskell-like notation for iterated applications of *Cons*:

lemma *Cons_app*: $[a,b,c] = Cons(a,Cons(b,Cons(c,[])))$ ⟨*proof*⟩

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 $\forall x[M]. P(x) \equiv \forall x. M(x) \longrightarrow P(x)$
$\exists x[M]. P(x) \equiv \exists x. M(x) \wedge P(x)$
⟨*proof*⟩

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*

$is\_function(M, r) \equiv$
$\forall x[M].$
$\quad \forall y[M].$
$\quad\quad \forall y'[M].$
$\quad\quad\quad \forall p[M].$
$\quad\quad\quad\quad \forall p'[M].$
$\quad\quad\quad\quad\quad pair(M, x, y, p) \longrightarrow$
$\quad\quad\quad\quad\quad pair(M, x, y', p') \longrightarrow p \in r \longrightarrow p' \in r \longrightarrow y = y'$

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*

$is\_domain(M, r, z) \equiv$
$\forall x[M]. x \in z \longleftrightarrow (\exists w[M]. w \in r \wedge (\exists y[M]. pair(M, x, y, w)))$

thm *typed_function_def*

$typed\_function(M, A, B, r) \equiv$
$is\_function(M, r) \wedge$
$is\_relation(M, r) \wedge$
$is\_domain(M, r, A) \wedge$
$(\forall u[M]. u \in r \longrightarrow (\forall x[M]. \forall y[M]. pair(M, x, y, u) \longrightarrow y \in B))$

thm *is_function_space_def[unfolded is_funspace_def]*
function_space_rel_def surjection_def

$is\_function\_space(M, A, B, fs) \equiv$
$M(fs) \wedge (\forall f[M]. f \in fs \longleftrightarrow typed\_function(M, A, B, f))$
$A \to^{M} B \equiv THE\ d.\ is\_function\_space(M, A, B, d)$
$surjection(M, A, B, f) \equiv$
$typed\_function(M, A, B, f) \wedge$
$(\forall y[M]. y \in B \longrightarrow (\exists x[M]. x \in A \wedge is\_apply(M, f, x, y)))$

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*

$infinity\_ax(M) \equiv$
$\exists I[M].$
$\quad (\exists z[M]. empty(M, z) \wedge z \in I) \wedge$
$\quad (\forall y[M]. y \in I \longrightarrow (\exists sy[M]. successor(M, y, sy) \wedge sy \in I))$

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 ∈-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*

$[\![M \approx \omega;\ Transset(M);\ M \models ZF]\!]$
$\Longrightarrow \exists N.\ M \subseteq N \wedge$
$N \approx \omega \wedge$
$Transset(N) \wedge$
$N \models ZF \wedge$
$M \neq N \wedge (\forall \alpha.\ Ord(\alpha) \longrightarrow \alpha \in M \longleftrightarrow \alpha \in N) \wedge ((M, [] \models \cdot AC\cdot) \longrightarrow$
$N \models ZFC)$

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** $A \approx^M B \longleftrightarrow (\exists f[M].\ f \in bij(A,B))$
  ⟨*proof*⟩

Below, μ denotes the minimum operator on the ordinals.

lemma *cardinalities_defs*:
  fixes $M::i \Rightarrow o$
  shows
    $|A|^M \equiv \mu\ i.\ M(i) \wedge i \approx^M A$
    $Card^M(\alpha) \equiv \alpha = |\alpha|^M$
    $\kappa^{\uparrow\nu,M} \equiv |\nu \rightarrow^M \kappa|^M$
    $(\kappa^+)^M \equiv \mu\ x.\ M(x) \wedge Card^M(x) \wedge \kappa < x$
    $CH^M \equiv \aleph_1{}^M = 2^{\uparrow\aleph_0{}^M,M}$
  ⟨*proof*⟩

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
    $CH^N \equiv \aleph_1{}^N = 2^{\uparrow\aleph_0{}^N,N}$
  ⟨*proof*⟩

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).