Formalization of Forcing in Isabelle/ZF

Emmanuel Gunther* Miguel Pagano* Pedro Sánchez Terraf*†

September 3, 2020

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

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

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

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

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

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

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

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

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

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

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

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

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

definition
  dense_below :: $i \Rightarrow i \Rightarrow o$ **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)
  by (*insert leq_preord*, *auto simp add*: *preorder_on_def refl_def dense_def*)

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

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

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

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

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

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

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

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

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

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

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

lemma *dense_below_cong*: $p \in P \Longrightarrow D = D' \Longrightarrow dense\_below(D,p) \longleftrightarrow dense\_below(D',p)$
by *blast*

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

lemma *dense_below_mono*: $p \in P \Longrightarrow D \subseteq D' \Longrightarrow dense\_below(D,p) \Longrightarrow dense\_below(D',p)$
by *blast*

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

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

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

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

definition
filter :: $i \Rightarrow o$ **where**
$filter(G) \equiv G \subseteq P \land increasing(G) \land (\forall p \in G.\ \forall q \in G.\ compat\_in(G,leq,p,q))$

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

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

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

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

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

definition
upclosure :: $i \Rightarrow i$ **where**

upclosure$(A) \equiv \{p{\in}P. \exists\, a{\in}A.\ a{\preceq}p\}$

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

lemma *upclosureE* [*elim*] :
  $p{\in}$*upclosure*$(A) \Longrightarrow (\bigwedge x\ a.\ x{\in}P \Longrightarrow a{\in}A \Longrightarrow a{\preceq}x \Longrightarrow R) \Longrightarrow R$
  by (*auto simp add*:*upclosure_def*)

lemma *upclosureD* [*dest*] :
  $p{\in}$*upclosure*$(A) \Longrightarrow \exists\, a{\in}A.(a{\preceq}p) \wedge p{\in}P$
  by (*simp add*:*upclosure_def*)

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

lemma *upclosure_in_P*: $A \subseteq P \Longrightarrow$ *upclosure*$(A) \subseteq P$
  using *subsetI upclosure_def* **by** *simp*

lemma *A_sub_upclosure*: $A \subseteq P \Longrightarrow A{\subseteq}$*upclosure*$(A)$
  using *subsetI leq_preord*
  unfolding *upclosure_def preorder_on_def refl_def* **by** *auto*

lemma *elem_upclosure*: $A{\subseteq}P \Longrightarrow x{\in}A \Longrightarrow x{\in}$*upclosure*$(A)$
  by (*blast dest*:*A_sub_upclosure*)

lemma *closure_compat_filter*:
  assumes $A{\subseteq}P$ $(\forall\, p{\in}A.\forall\, q{\in}A.$ *compat_in*$(A,leq,p,q))$
  shows *filter*(*upclosure*(A))
  unfolding *filter_def*
proof(*auto*)
  show *increasing*(*upclosure*(A))
    using *assms upclosure_increasing* **by** *simp*
next
  let *?UA*=*upclosure*(A)
  show *compat_in*(*upclosure*(A), *leq*, p, q) **if** $p{\in}$*?UA* $q{\in}$*?UA* **for** p q
  proof -
    from *that*
    obtain a b **where** *1*:$a{\in}A$ $b{\in}A$ $a{\preceq}p$ $b{\preceq}q$ $p{\in}P$ $q{\in}P$
      using *upclosureD*[*OF* ‹$p{\in}$*?UA*›] *upclosureD*[*OF* ‹$q{\in}$*?UA*›] **by** *auto*
    with *assms*(*2*)
    obtain d **where** $d{\in}A$ $d{\preceq}a$ $d{\preceq}b$
      unfolding *compat_in_def* **by** *auto*
    with *1*
    have $d{\preceq}p$ $d{\preceq}q$ $d{\in}$*?UA*
      using *A_sub_upclosure*[*THEN subsetD*] ‹$A{\subseteq}P$›

```
    leq_transD'[of A d a] leq_transD'[of A d b] by auto
  then
  show ?thesis unfolding compat_in_def by auto
 qed
qed

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

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

lemma decr_seq_linear:
 assumes refl(P,leq) f ∈ nat → P
  ∀ n∈nat. ⟨f ` succ(n), f ` n⟩ ∈ leq
  trans[P](leq)
 shows linear(f `` nat, leq)
proof -
 have preorder_on(P,leq)
  unfolding preorder_on_def using assms by simp
 {
  fix n m
  assume n∈nat m∈nat
  then
```

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

end

2.2 Towards Rasiowa-Sikorski Lemma (RSL)

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

begin

definition
D_generic :: *i*⇒*o* **where**
D_generic(*G*) ≡ *filter*(*G*) ∧ (∀ *n*∈*nat*.($\mathcal{D}$‘*n*)∩*G*≠*0*)

The next lemma identifies a sufficient condition for obtaining RSL.

lemma *RS_sequence_imp_rasiowa_sikorski*:
assumes
p∈*P f* : *nat*→*P f* ‘ *0* = *p*
⋀*n*. *n*∈*nat* ⟹ *f* ‘ *succ*(*n*)$\preceq$ *f* ‘ *n* ∧ *f* ‘ *succ*(*n*) ∈ $\mathcal{D}$ ‘ *n*
shows
∃ *G*. *p*∈*G* ∧ *D_generic*(*G*)
proof -
note *assms*
moreover from *this*
have *f*‘‘*nat* ⊆ *P*
by (*simp add*:*subset_fun_image*)
moreover from *calculation*
have *refl*(*f*‘‘*nat*, *leq*) ∧ *trans*[*P*](*leq*)
using *leq_preord* **unfolding** *preorder_on_def* **by** (*blast intro*:*refl_monot_domain*)

```
  moreover from calculation
  have ∀n∈nat. f ` succ(n)≼ f ` n by (simp)
  moreover from calculation
  have linear(f``nat, leq)
    using leq_preord and decr_seq_linear unfolding preorder_on_def by (blast)
  moreover from calculation
  have (∀p∈f``nat.∀q∈f``nat. compat_in(f``nat,leq,p,q))
    using chain_compat by (auto)
  ultimately
  have filter(upclosure(f``nat)) (is filter(?G))
    using closure_compat_filter by simp
  moreover
  have ∀n∈nat. 𝒟 ` n ∩ ?G ≠ 0
  proof
    fix n
    assume n∈nat
    with assms
    have f`succ(n) ∈ ?G ∧ f`succ(n) ∈ 𝒟 ` n
      using aux_RS1 by simp
    then
    show 𝒟 ` n ∩ ?G ≠ 0 by blast
  qed
  moreover from assms
  have p ∈ ?G
    using aux_RS1 by auto
  ultimately
  show ?thesis unfolding D_generic_def by auto
qed

end

lemma Pi_rangeD:
  assumes f∈Pi(A,B) b ∈ range(f)
  shows ∃a∈A. f`a = b
  using assms apply_equality[OF _ assms(1), of _ b] domain_type[OF _ assms(1)]
by auto
```

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

```
consts RS_seq :: [i,i,i,i,i,i] ⇒ i
primrec
  RS_seq(0,P,leq,p,enum,𝒟) = p
  RS_seq(succ(n),P,leq,p,enum,𝒟) =
    enum`(μ m. ⟨enum`m, RS_seq(n,P,leq,p,enum,𝒟)⟩ ∈ leq ∧ enum`m ∈ 𝒟 ` n)

context countable_generic
begin

lemma countable_RS_sequence_aux:
```

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

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

```
    fix x k
    assume x∈P k∈nat
    then
    have ∃ q∈P. q≼ x ∧ q ∈ 𝒟 ‘ k
      using seq_of_denses apply_funtype[OF countable_subs_of_P]
      unfolding dense_def by blast
  }
  with assms
  show f‘succ(n)≼ f‘n ∧ f‘succ(n) ∈ 𝒟 ‘ n f‘succ(n)∈P
    unfolding f_def using countable_RS_sequence_aux by simp_all
qed

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

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

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

end

end
```

3 A pointed version of DC

```
theory Pointed_DC imports ZF.AC

begin
```

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

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

lemma witness_into_A [TC]:
  assumes a∈A
    (∀ X . X≠0 ∧ X⊆A ⟶ s‘X∈X)
    ∀ y∈A. {x∈A. ⟨y,x⟩∈R } ≠ 0 n∈nat
  shows dc_witness(n, A, a, s, R)∈A
  using ‹n∈nat›
proof(induct n)
```

```
  case 0
  then show ?case using ‹a∈A› by simp
next
  case (succ x)
  then
  show ?case using assms by auto
qed

lemma witness_related :
  assumes a∈A
    (∀ X . X≠0 ∧ X⊆A ⟶ s‘X∈X)
    ∀ y∈A. {x∈A. ⟨y,x⟩∈R } ≠ 0 n∈nat
  shows ⟨dc_witness(n, A, a, s, R),dc_witness(succ(n), A, a, s, R)⟩∈R
proof -
  from assms
  have dc_witness(n, A, a, s, R)∈A (is ?x ∈ A)
    using witness_into_A[of _ _ s R n] by simp
  with assms
  show ?thesis by auto
qed

lemma witness_funtype:
  assumes a∈A
    (∀ X . X≠0 ∧ X⊆A ⟶ s‘X∈X)
    ∀ y∈A. {x∈A. ⟨y,x⟩∈R } ≠ 0
  shows (λn∈nat. dc_witness(n, A, a, s, R)) ∈ nat → A (is ?f ∈ _ → _)
proof -
  have ?f ∈ nat → {dc_witness(n, A, a, s, R). n∈nat} (is _ ∈ _ → ?B)
    using lam_funtype assms by simp
  then
  have ?B ⊆ A
    using witness_into_A assms by auto
  with ‹?f ∈ _›
  show ?thesis
    using fun_weaken_type
    by simp
qed

lemma witness_to_fun:  assumes a∈A
  (∀ X . X≠0 ∧ X⊆A ⟶ s‘X∈X)
  ∀ y∈A. {x∈A. ⟨y,x⟩∈R } ≠ 0
shows ∃ f ∈ nat→A. ∀ n∈nat. f‘n =dc_witness(n,A,a,s,R)
  using assms bexI[of _ λn∈nat. dc_witness(n,A,a,s,R)] witness_funtype
  by simp

theorem pointed_DC  :
  assumes (∀ x∈A. ∃ y∈A. ⟨x,y⟩∈ R)
  shows ∀ a∈A. (∃ f ∈ nat→A. f‘0 = a ∧ (∀ n ∈ nat. ⟨f‘n,f‘succ(n)⟩∈R))
proof -
```

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

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

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

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

```
    case 0
    then show ?case using F by simp
  next
    case (succ x)
    then
    have ⟨f‘x, f‘succ(x)⟩ ∈ ?R′ f‘x ∈ A×nat f‘succ(x)∈A×nat
      using F by simp_all
    then
    have snd(f‘succ(x)) = succ(snd(f‘x)) by simp
    with succ ‹f‘x∈_›
    show ?case using infer_snd[OF ‹f‘succ(_)∈_›] by auto
  qed
  have ⟨⟨?f‘n,n⟩,⟨?f‘succ(n),succ(n)⟩⟩ ∈ R if n∈nat for n
    using that 1[of succ(n)] 1[OF ‹n∈_›] F(3) by simp
  with ‹f‘0=⟨a,0⟩›
  show ?thesis using rev_bexI[OF ‹?f∈_›] by simp
qed

lemma aux_sequence_DC :
  assumes ∀x∈A. ∀n∈nat. ∃y∈A. ⟨x,y⟩ ∈ S‘n
    R={⟨⟨x,n⟩,⟨y,m⟩⟩ ∈ (A×nat)×(A×nat). ⟨x,y⟩∈S‘m }
  shows ∀ x∈A×nat . ∃y∈A. ⟨x,⟨y,succ(snd(x))⟩⟩ ∈ R
  using assms Pair_fst_snd_eq by auto

lemma aux_sequence_DC2 : ∀x∈A. ∀n∈nat. ∃y∈A. ⟨x,y⟩ ∈ S‘n ⟹
      ∀x∈A×nat. ∃y∈A. ⟨x,⟨y,succ(snd(x))⟩⟩ ∈ {⟨⟨x,n⟩,⟨y,m⟩⟩∈(A×nat)×(A×nat).
⟨x,y⟩∈S‘m }
  by auto

lemma sequence_DC:
  assumes ∀x∈A. ∀n∈nat. ∃y∈A. ⟨x,y⟩ ∈ S‘n
  shows ∀a∈A. (∃f ∈ nat→A. f‘0 = a ∧ (∀n ∈ nat. ⟨f‘n,f‘succ(n)⟩∈S‘succ(n)))
  by (rule ballI,insert assms,drule aux_sequence_DC2, drule DC_on_A_x_nat, auto)

end
```

4 The general Rasiowa-Sikorski lemma

```
theory Rasiowa_Sikorski imports Forcing_Notions Pointed_DC begin

context countable_generic
begin

lemma RS_relation:
  assumes p∈P n∈nat
  shows ∃y∈P. ⟨p,y⟩ ∈ (λm∈nat. {⟨x,y⟩∈P×P. y≼x ∧ y∈𝒟‘(pred(m))})‘n
proof -
  from seq_of_denses ‹n∈nat›
  have dense(𝒟 ‘ pred(n)) by simp
```

```
  with ‹p∈P›
  have ∃ d∈𝒟 ‘ Arith.pred(n). d≼ p
    unfolding dense_def by simp
  then obtain d where 3: d ∈ 𝒟 ‘ Arith.pred(n) ∧ d≼ p
    by blast
  from countable_subs_of_P ‹n∈nat›
  have 𝒟 ‘ Arith.pred(n) ∈ Pow(P)
    by (blast dest:apply_funtype intro:pred_type)
  then
  have 𝒟 ‘ Arith.pred(n) ⊆ P
    by (rule PowD)
  with 3
  have d ∈ P ∧ d≼ p ∧ d ∈ 𝒟 ‘ Arith.pred(n)
    by auto
  with ‹p∈P› ‹n∈nat›
  show ?thesis by auto
qed

lemma DC_imp_RS_sequence:
  assumes p∈P
  shows ∃ f. f: nat→P ∧ f ‘ 0 = p ∧
     (∀ n∈nat. f ‘ succ(n)≼ f ‘ n ∧ f ‘ succ(n) ∈ 𝒟 ‘ n)
proof -
  let ?S=(λm∈nat. {⟨x,y⟩∈P×P. y≼x ∧ y∈𝒟‘(pred(m))})
  have ∀ x∈P. ∀ n∈nat. ∃ y∈P. ⟨x,y⟩ ∈ ?S‘n
    using RS_relation by (auto)
  then
  have ∀ a∈P. (∃f ∈ nat→P. f‘0 = a ∧ (∀ n ∈ nat. ⟨f‘n,f‘succ(n)⟩∈?S‘succ(n)))
    using sequence_DC by (blast)
  with ‹p∈P›
  show ?thesis by auto
qed

theorem rasiowa_sikorski:
  p∈P ⟹ ∃ G. p∈G ∧ D_generic(G)
  using RS_sequence_imp_rasiowa_sikorski by (auto dest:DC_imp_RS_sequence)

end

end
```

5 Auxiliary results on arithmetic

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

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

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

lemma nat_succD : m ∈ nat ⟹ succ(n) ∈ succ(m) ⟹ n ∈ m
```

by (*drule_tac j=succ(m)* **in** *ltI,auto elim:ltD*)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *Un_leD2* : *Ord*(*i*)⟹ *Ord*(*j*)⟹ *Ord*(*k*)⟹ *i* ∪ *j* ≤*k* ⟹ *j* ≤ *k*
 by (*rule Un_least_lt_iff*[*THEN iffD1*[*THEN conjunct2*]],*simp_all*)

lemma *gt1* : *n* ∈ *nat* ⟹ *i* ∈ *n* ⟹ *i* ≠ *0* ⟹ *i* ≠ *1* ⟹ *1*<*i*
 by(*rule_tac n*=*i* **in** *natE*,*erule in_n_in_nat*,*auto intro*: *Ord_0_lt*)

lemma *pred_mono* : *m* ∈ *nat* ⟹ *n* ≤ *m* ⟹ *pred*(*n*) ≤ *pred*(*m*)
 by(*rule_tac n*=*n* **in** *natE*,*auto simp add*:*le_in_nat*,*erule_tac n*=*m* **in** *natE*,*auto*)

lemma *succ_mono* : *m* ∈ *nat* ⟹ *n* ≤ *m* ⟹ *succ*(*n*) ≤ *succ*(*m*)
 by *auto*

lemma *pred2_Un*:
 assumes *j* ∈ *nat m* ≤ *j n* ≤ *j*
 shows *pred*(*pred*(*m* ∪ *n*)) ≤ *pred*(*pred*(*j*))
 using *assms pred_mono*[*of j*] *le_in_nat Un_least_lt pred_mono* **by** *simp*

lemma *nat_union_abs1* :
 ⟦ *Ord*(*i*) ; *Ord*(*j*) ; *i* ≤ *j* ⟧ ⟹ *i* ∪ *j* = *j*
 by (*rule Un_absorb1*,*erule le_imp_subset*)

lemma *nat_union_abs2* :
 ⟦ *Ord*(*i*) ; *Ord*(*j*) ; *i* ≤ *j* ⟧ ⟹ *j* ∪ *i* = *j*
 by (*rule Un_absorb2*,*erule le_imp_subset*)

lemma *nat_un_max* : *Ord*(*i*) ⟹ *Ord*(*j*) ⟹ *i* ∪ *j* = *max*(*i*,*j*)
 using *max_def nat_union_abs1 not_lt_iff_le leI nat_union_abs2*
 by *auto*

lemma *nat_max_ty* : *Ord*(*i*) ⟹*Ord*(*j*) ⟹ *Ord*(*max*(*i*,*j*))
 unfolding *max_def* **by** *simp*

lemma *le_not_lt_nat* : *Ord*(*p*) ⟹ *Ord*(*q*) ⟹ ¬ *p*≤ *q* ⟹ *q* ≤ *p*
 by (*rule ltE*,*rule not_le_iff_lt*[*THEN iffD1*],*auto*,*drule ltI*[*of q p*],*auto*,*erule leI*)

lemmas *nat_simp_union* = *nat_un_max nat_max_ty max_def*

lemma *le_succ* : *x*∈*nat* ⟹ *x*≤*succ*(*x*) **by** *simp*
lemma *le_pred* : *x*∈*nat* ⟹ *pred*(*x*)≤*x*
 using *pred_le*[*OF _ _ le_succ*] *pred_succ_eq*
 by *simp*

lemma *Un_le_compat* : *o* ≤ *p* ⟹ *q* ≤ *r* ⟹ *Ord*(*o*) ⟹ *Ord*(*p*) ⟹ *Ord*(*q*) ⟹ *Ord*(*r*) ⟹ *o* ∪ *q* ≤ *p* ∪ *r*
 using *le_trans*[*of q r p*∪*r*,*OF _ Un_upper2_le*] *le_trans*[*of o p p*∪*r*,*OF _ Un_upper1_le*] *nat_simp_union*
 by *auto*

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

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

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

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

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

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

lemma *diff_cancel* :
assumes $m \in nat$ $n{\in}nat$ $m < n$
shows *m#-n* = *0*
using *assms diff_is_0_lemma leI* **by** *simp*

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

5.1 Some results in ordinal arithmetic

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

lemma *max_cong* :
assumes $x \leq y$ $Ord(y)$ $Ord(z)$ **shows** $max(x,y) \leq max(y,z)$
using *assms*

proof (*cases* $y \leq z$)
case *True*
then show *?thesis*
unfolding *max_def* **using** *assms* **by** *simp*
next
case *False*
then have $z \leq y$ **using** *assms not_le_iff_lt leI* **by** *simp*
then show *?thesis*
unfolding *max_def* **using** *assms* **by** *simp*
qed

lemma *max_commutes* :
assumes *Ord*(x) *Ord*(y)
shows *max*(x,y) = *max*(y,x)
using *assms Un_commute nat_simp_union*(*1*) *nat_simp_union*(*1*)[*symmetric*] **by** *auto*

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

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

lemma *max_D2* :
assumes $w = y \lor w = z$ $x < y$ *Ord*(x) *Ord*(w) *Ord*(y) *Ord*(z) *max*(x,w) = *max*(y,z)
shows $x<w$
proof -
from *assms*

```
  have x < z ∪ y using Un_upper2_lt[OF ‹x<y›] by simp
  then
  consider (a) x < y | (b) x < w
    using assms nat_simp_union by simp
  then show ?thesis proof (cases)
    case a
    consider (c) w = y | (d) w = z
      using assms by auto
    then show ?thesis proof (cases)
      case c
      with a show ?thesis by simp
    next
      case d
      with a
      show ?thesis
      proof (cases y <w)
        case True
        then show ?thesis using lt_trans[OF ‹x<y›] by simp
      next
        case False
        then
        have w ≤ y
          using not_lt_iff_le[OF assms(5) assms(4)] by simp
        with ‹w=z›
        have max(z,y) = y  unfolding max_def using assms by simp
        with assms
        have ... = x ∪ w using nat_simp_union max_commutes  by simp
        then show ?thesis using le_Un_iff assms by blast
      qed
    qed
  next
    case b
    then show ?thesis .
  qed
qed

lemma oadd_lt_mono2 :
  assumes  Ord(n) Ord(α) Ord(β) α < β x < n y < n 0 <n
  shows n ** α ++ x < n **β ++ y
proof -
  consider (0) β=0 | (s) γ where  Ord(γ) β = succ(γ) | (l) Limit(β)
    using Ord_cases[OF ‹Ord(β)›,of ?thesis] by force
  then show ?thesis
  proof cases
    case 0
    then show ?thesis using ‹α<β› by auto
  next
    case s
    then
```

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

6 Automatic synthesis of formulas

theory *Synthetic_Definition*
imports *ZF-Constructible.Formula*
keywords
synthesize :: *thy_decl* % *ML*
and
synthesize_notc :: *thy_decl* % *ML*
and
from_schematic

begin
ML_file‹*Utils.ml*›
ML‹
structure Formulas = *Named_Thms*
(*val name* = @{*binding fm_definitions*}
val description = *Theorems for synthetising formulas*.)
›

setup‹*Formulas.setup*›

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

fun pair f g x = (f x, g x)

fun prove_tc_form goal thms ctxt =
  Goal.prove ctxt [] [] goal
     (fn _ => rewrite_goal_tac ctxt thms 1
             THEN TypeCheck.typecheck_tac ctxt)

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

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

fun synth_thm_sats def_name term lhs set env hyps vars vs pos thm_auto lthy =
let val (_,tm,ctxt1) = Utils.thm_concl_tm lthy term
     val (thm_refs,ctxt2) = Variable.import true [Proof_Context.get_thm lthy term]
ctxt1 |>> #2
    val vs' = map (Thm.term_of o #2) vs
    val vars' = map (Thm.term_of o #2) vars
    val r_tm = tm |> Utils.dest_lhs_def |> fold (op $`) vs'
    val sats = @{const apply} $ (@{const satisfies} $ set $ r_tm) $ env
    val rhs = @{const IFOL.eq(i)} $ sats $ (@{const succ} $ @{const zero})
    val concl = @{const IFOL.iff} $ lhs $ rhs
    val g_iff = Logic.list_implies(hyps, Utils.tp concl)
    val thm = prove_sats g_iff thm_refs thm_auto ctxt2
    val name = Binding.name (def_name ^ _iff_sats)
    val thm = Utils.fix_vars thm (map (#1 o dest_Free) vars') lthy
 in
   Local_Theory.note ((name, []), [thm]) lthy |> Utils.display theorem pos
 end

fun synth_thm_tc def_name term hyps vars pos lthy =
let val (_,tm,ctxt1) = Utils.thm_concl_tm lthy term
     val (thm_refs,ctxt2) = Variable.import true [Proof_Context.get_thm lthy term]
ctxt1
                   |>> #2
    val vars' = map (Thm.term_of o #2) vars
```

```
    val tc_attrib = @{attributes [TC]}
    val r_tm = tm |> Utils.dest_lhs_def |> fold (op $`) vars'
    val concl = @{const mem} $ r_tm $ @{const formula}
    val g = Logic.list_implies(hyps, Utils.tp concl)
    val thm = prove_tc_form g thm_refs ctxt2
    val name = Binding.name (def_name ^ _type)
    val thm = Utils.fix_vars thm (map (#1 o dest_Free) vars') ctxt2
 in
   Local_Theory.note ((name, tc_attrib), [thm]) lthy |> Utils.display theorem pos
 end


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

end
›
ML‹

local
  val synth_constdecl =
       Parse.position (Parse.string -- ((Parse.$$$ from_schematic |-- Parse.thm)));

  val _ =
     Outer_Syntax.local_theory command_keyword‹synthesize› ML setup for syn-
thetic definitions
        (synth_constdecl >> (fn ((bndg,thm),p) => synthetic_def bndg thm p true
```

true))

  val _ =
    Outer_Syntax.local_theory ***command_keyword*** ‹*synthesize_notc*› *ML setup for synthetic definitions*
      (*synth_constdecl* >> (*fn* ((*bndg*,*thm*),*p*) => *synthetic_def bndg thm p false false*))

in

end
›

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

end

7 Aids to internalize formulas

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

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

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

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* ∈ *nat*⟧ ⟹ *nth*(*succ*(*n*), *Cons*(*a*,*l*)) = *x*
by *simp*

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

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

lemmas *fm_defs* =
  omega_fm_def limit_ordinal_fm_def empty_fm_def typed_function_fm_def
  pair_fm_def upair_fm_def domain_fm_def function_fm_def succ_fm_def

cons_fm_def fun_apply_fm_def image_fm_def big_union_fm_def union_fm_def relation_fm_def composition_fm_def field_fm_def ordinal_fm_def range_fm_def transset_fm_def subset_fm_def Replace_fm_def

lemmas *formulas_def* = *fm_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

setup‹
fold (*Context.theory_map o Formulas.add_thm*) (*rev* @{*thms formulas_def*})›

end

8 Some enhanced theorems on recursion

theory *Recursion_Thms* **imports** *ZF.Epsilon* **begin**

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

```
lemma fld_restrict_eq : a ∈ A ⟹ (r ∩ A×A)-``{a} = (r-``{a} ∩ A)
  by(force)

lemma fld_restrict_mono : relation(r) ⟹ A ⊆ B ⟹ r ∩ A×A ⊆ r ∩ B×B
  by(auto)

lemma fld_restrict_dom :
  assumes relation(r) domain(r) ⊆ A range(r)⊆ A
  shows r∩ A×A = r
proof (rule equalityI,blast,rule subsetI)
  { fix x
    assume xr: x ∈ r
    from xr assms have ∃ a b . x = ⟨a,b⟩ by (simp add: relation_def)
    then obtain a b where ⟨a,b⟩ ∈ r ⟨a,b⟩ ∈ r∩A×A x ∈ r∩A×A
      using assms xr
      by force
    then have x∈ r ∩ A×A by simp
  }
  then show x ∈ r ⟹ x∈ r∩A×A for x .
qed

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

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

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

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

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

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

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

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

have *tr_down*$(r,x) \subseteq$ *tr_down*(r,a)
using *tr_down_mono x_a rr* **by** *simp*
with *1*
have *tr_down*$(r,x) \subseteq A$ **using** *subset_trans* **by** *force*
have $\langle x,a\rangle \in r$ **using** *x_a underD* **by** *simp*
with *1* ‹*tr_down*$(r,x) \subseteq A$› ‹$x \in A$›
have *wfrec*$(r,x,H) =$ *wfrec*$[A](r,x,H)$ **by** *simp*
}
then
have $x\in r$-‘‘$\{a\} \Longrightarrow$ *wfrec*$(r,x,H) =$ *wfrec*$[A](r,x,H)$ **for** x **.**
then
have *Eq1* :$(\lambda\ x \in r$-‘‘$\{a\}$. *wfrec*$(r,x,H)) = (\lambda\ x \in r$-‘‘$\{a\}$. *wfrec*$[A](r,x,H))$
using *lam_cong* **by** *simp*

from *assms*
have *wfrec*$(r,a,H) = H(a,\lambda\ x \in r$-‘‘$\{a\}$. *wfrec*$(r,x,H))$ **by** (*simp add*:*wfrec*)
also
have ... $= H(a,\lambda\ x \in r$-‘‘$\{a\}$. *wfrec*$[A](r,x,H))$
using *assms Eq1* **by** *simp*
also from *1* ‹r-‘‘$\{a\} \subseteq A$›
have ... $= H(a,\lambda\ x \in (r\cap A\times A)$-‘‘$\{a\}$. *wfrec*$[A](r,x,H))$
using *assms rest_eq* **by** *simp*
also from ‹$a\in A$›
have ... $= H(a,\lambda\ x \in (r$-‘‘$\{a\})\cap A$. *wfrec*$[A](r,x,H))$
using *fld_restrict_eq* **by** *simp*
also from ‹$a\in A$› ‹*wf*$[A](r)$›
have ... $=$ *wfrec*$[A](r,a,H)$ **using** *wfrec_on* **by** *simp*
finally show *?case* **.**
qed

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

lemma *wfrec_trans_restr* : *relation*$(r) \Longrightarrow$ *wf*$(r) \Longrightarrow$ *trans*$(r) \Longrightarrow r$-‘‘$\{a\}\subseteq A \Longrightarrow$ $a \in A \Longrightarrow$
wfrec$(r, a, H) =$ *wfrec*$[A](r, a, H)$
by(*subgoal_tac tr_down*$(r,a) \subseteq A$,*auto simp add* : *wfrec_restr tr_down_def trancl_eq_r*)

lemma *field_trancl* : *field*$(r\hat{}+) =$ *field*(r)
by (*blast intro*: *r_into_trancl dest*!: *trancl_type* [*THEN subsetD*])

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)
unfolding *Rrel_def* **by** *simp*

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

```
  unfolding Rrel_def Memrel_def ..

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

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

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

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


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

  using Rrel_mem field_Rrel by blast

lemma restrict_trancl_Rrel:
  assumes R(w,y)
  shows restrict(f,Rrel(R,d)-``{y})`w
       = restrict(f,(Rrel(R,d)^+)-``{y})`w
proof (cases y∈d)
  let ?r=Rrel(R,d) and ?s=(Rrel(R,d))^+
  case True
  show ?thesis
  proof (cases w∈d)
    case True
    with ‹y∈d› assms
    have ⟨w,y⟩∈?r
      unfolding Rrel_def by blast
    then
    have ⟨w,y⟩∈?s
      using r_subset_trancl[of ?r] relation_Rrel[of R d] by blast
    with ‹⟨w,y⟩∈?r›
    have w∈?r-``{y} w∈?s-``{y}
      using vimage_singleton_iff by simp_all
    then
    show ?thesis by simp
  next
    case False
    then
    have w∉domain(restrict(f,?r-``{y}))
      using subsetD[OF field_Rrel[of R d]] by auto
    moreover from ‹w∉d›
    have w∉domain(restrict(f,?s-``{y}))
      using subsetD[OF field_Rrel[of R d], of w] field_trancl[of ?r]
        fieldI1[of w y ?s] by auto
```

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

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

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

end
```

9 Relativization of the cumulative hierarchy

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

begin

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

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

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

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

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

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

definition
  $is\_HVfrom :: [i{\Rightarrow}o,i,i,i,i] \Rightarrow o$ **where**
  $is\_HVfrom(M,A,x,f,h) \equiv \exists U[M].\ \exists R[M].\ \ union(M,A,U,h)$
    $\wedge\ big\_union(M,R,U) \wedge is\_Replace(M,x,is\_powapply(M,f),R)$

definition
  $is\_Vfrom :: [i{\Rightarrow}o,i,i,i] \Rightarrow o$ **where**

is_Vfrom$(M,A,i,V) \equiv$ *is_transrec*$(M,$*is_HVfrom*$(M,A),i,V)$

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

9.1 Formula synthesis

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

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

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

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

definition
is_Hrank :: $[i \Rightarrow o,i,i,i] \Rightarrow o$ **where**
is_Hrank$(M,x,f,hc) \equiv (\exists R[M].$ *big_union*$(M,R,hc) \wedge$*is_Replace*$(M,x,$*PHrank*$(M,f),R))$

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

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

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

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

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

9.2 Absoluteness results

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

begin

lemma *is_powapply_abs*: ⟦$M(f)$; $M(y)$⟧ $\Longrightarrow$ *is_powapply*(M,f,y,z) $\longleftrightarrow M(z) \wedge z$ = {x∈*Pow*(f‘y). $M(x)$}
unfolding *is_powapply_def* **by** *simp*

lemma ⟦$M(A)$; $M(x)$; $M(f)$; $M(h)$ ⟧ $\Longrightarrow$
is_HVfrom(M,A,x,f,h) $\longleftrightarrow$
(∃ R[M]. $h = A \cup \bigcup R \wedge$ *is_Replace*(M, x,λx y. y = {x ∈ *Pow*(f ‘ x) . $M(x)$}, R))
using *is_powapply_abs* **unfolding** *is_HVfrom_def* **by** *auto*

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

lemma *powapply_closed*:
⟦ $M(y)$; $M(f)$ ⟧ $\Longrightarrow$ M({x ∈ *Pow*(f ‘ y) . $M(x)$})

using *apply_closed power_ax* **unfolding** *power_ax_def* **by** *simp*

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

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

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

lemma *relation2_HVfrom*: $M(A) \Longrightarrow relation2(M,is\_HVfrom(M,A),HVfrom(M,A))$
unfolding *is_HVfrom_def HVfrom_def relation2_def*

```
    using Replace_is_powapply RepFun_is_powapply
          Union_powapply_closed RepFun_powapply_closed by auto

lemma HVfrom_closed :
  M(A) ⟹ ∀x[M]. ∀g[M]. function(g) ⟶ M(HVfrom(M,A,x,g))
  unfolding HVfrom_def using Union_powapply_closed by simp

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

    unfolding HVfrom_def by simp
  finally
  show ?case .
qed

lemma Vfrom_abs: ⟦ M(A); M(i); M(V); Ord(i) ⟧ ⟹ is_Vfrom(M,A,i,V) ⟷
V = {x∈Vfrom(A,i). M(x)}
  unfolding is_Vfrom_def
  using relation2_HVfrom HVfrom_closed HVfrom_replacement
    transrec_abs[of is_HVfrom(M,A) i HVfrom(M,A)] transrec_HVfrom by simp
```

```
lemma Vfrom_closed: ⟦ M(A); M(i); Ord(i) ⟧ ⟹ M({x∈Vfrom(A,i). M(x)})
  unfolding is_Vfrom_def
  using relation2_HVfrom HVfrom_closed HVfrom_replacement
    transrec_closed[of is_HVfrom(M,A) i HVfrom(M,A)] transrec_HVfrom by simp

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

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

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

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

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


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

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

lemma Replace_PHrank_abs:
  assumes
```

M(z) M(f) M(hr)
shows
is_Replace(M,z,PHrank(M,f),hr) ⟷ *hr = Replace(z,PHrank(M,f))*
proof -
have $\bigwedge x\ y.$ ⟦*x*∈*z*; *PHrank(M,f,x,y)* ⟧ ⟹ *M(y)*
using ‹*M(z)*› ‹*M(f)*› **unfolding** *PHrank_def* **by** *simp*
then
show *?thesis* **using** ‹*M(z)*› ‹*M(hr)*› ‹*M(f)*› *univ_PHrank Replace_abs* **by** *simp*
qed

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

lemma *RepFun_PHrank_closed* :
assumes
M(f) M(A)
shows
M(Replace(A,PHrank(M,f)))
proof -
have ⟦ *x*∈*A* ; *PHrank(M,f,x,y)* ⟧ ⟹ *M(y)* **for** *x y*
using *assms* **unfolding** *PHrank_def* **by** *simp*
with *univ_PHrank*
show *?thesis* **using** *assms PHrank_replacement* **by** *simp*
qed

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

lemma *Union_PHrank_closed*:
assumes
M(x) M(f)
shows
M($\bigcup$ *y*∈*x. succ(f‘y)*)
proof -
have *M(succ(f‘y))* **if** *y*∈*x* **for** *y*

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

lemma *is_Hrank_closed* :
M(*A*) ⟹ ∀ *x*[*M*]. ∀ *g*[*M*]. *function*(*g*) ⟶ *M*(*Hrank*(*x*,*g*))
unfolding *Hrank_def* **using** *RepFun_PHrank_closed Union_PHrank_closed* **by** *simp*

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

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

end
end

10 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
begin

syntax
  sats :: [*i*, *i*, *i*] ⇒ *o* ((, _ ⊨ _) [*36*,*36*,*36*] *60*)
translations
  (*M*,*env* ⊨ *φ*) ⇌ *CONST sats*(*M*,*φ*,*env*)

abbreviation
  dec10 :: *i* (*10*) **where** *10* ≡ *succ*(*9*)

abbreviation
  dec11 :: *i* (*11*) **where** *11* ≡ *succ*(*10*)

abbreviation
  dec12 :: *i* (*12*) **where** *12* ≡ *succ*(*11*)

abbreviation
  dec13 :: *i* (*13*) **where** *13* ≡ *succ*(*12*)

abbreviation
  dec14 :: *i* (*14*) **where** *14* ≡ *succ*(*13*)

definition
  infinity_ax :: (*i* ⇒ *o*) ⇒ *o* **where**
  infinity_ax(*M*) ≡
    (∃ *I*[*M*]. (∃ *z*[*M*]. *empty*(*M*,*z*) ∧ *z*∈*I*) ∧ (∀ *y*[*M*]. *y*∈*I* ⟶ (∃ *sy*[*M*]. *successor*(*M*,*y*,*sy*) ∧ *sy*∈*I*)))

definition
  choice_ax :: (*i*⇒*o*) ⇒ *o* **where**
  choice_ax(*M*) ≡ ∀ *x*[*M*]. ∃ *a*[*M*]. ∃ *f*[*M*]. *ordinal*(*M*,*a*) ∧ *surjection*(*M*,*a*,*x*,*f*)

context *M_basic* **begin**

lemma *choice_ax_abs* :
  choice_ax(*M*) ⟷ (∀ *x*[*M*]. ∃ *a*[*M*]. ∃ *f*[*M*]. *Ord*(*a*) ∧ *f* ∈ *surj*(*a*,*x*))

unfolding *choice_ax_def*
by (*simp*)

end

definition
wellfounded_trancl :: [*i*=>*o*,*i*,*i*,*i*] => *o* **where**
wellfounded_trancl(*M*,*Z*,*r*,*p*) ≡
∃ *w*[*M*]. ∃ *wx*[*M*]. ∃ *rp*[*M*].
w ∈ *Z* & *pair*(*M*,*w*,*p*,*wx*) & *tran_closure*(*M*,*r*,*rp*) & *wx* ∈ *rp*

lemma *empty_intf* :
infinity_ax(*M*) ⟹
(∃ *z*[*M*]. *empty*(*M*,*z*))
by (*auto simp add*: *empty_def infinity_ax_def*)

lemma *Transset_intf* :
Transset(*M*) ⟹ *y*∈*x* ⟹ *x* ∈ *M* ⟹ *y* ∈ *M*
by (*simp add*: *Transset_def*,*auto*)

locale *M_ZF* =
fixes *M*
assumes
upair_ax: *upair_ax*(##*M*) **and**
Union_ax: *Union_ax*(##*M*) **and**
power_ax: *power_ax*(##*M*) **and**
extensionality:*extensionality*(##*M*) **and**
foundation_ax: *foundation_ax*(##*M*) **and**
infinity_ax: *infinity_ax*(##*M*) **and**
separation_ax: *φ* ∈ *formula* ⟹ *env* ∈ *list*(*M*) ⟹
arity(*φ*) ≤ *1* #+ *length*(*env*) ⟹
separation(##*M*,*λx*. *sats*(*M*,*φ*,[*x*] @ *env*)) **and**
replacement_ax:*φ* ∈ *formula* ⟹ *env* ∈ *list*(*M*) ⟹
arity(*φ*) ≤ *2* #+ *length*(*env*) ⟹
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*]

10.1 Interface with *M_trivial*

lemma *zero_in_M*: *0* ∈ *M*
proof -
from *infinity_ax*
have (∃ *z*[##*M*]. *empty*(##*M*,*z*))

```
    by (rule empty_intf)
  then obtain z where
    zm: empty(##M,z)  z∈M
    by auto
  then
  have z=0
    using transitivity empty_def by auto
  with zm show ?thesis
    by simp
qed

end

sublocale M_ZF_trans ⊆ M_trans ##M
  using transitivity zero_in_M exI[of λx. x∈M]
  by unfold_locales simp_all

sublocale M_ZF_trans ⊆ M_trivial ##M
  using trans_M M_trivial.intro M_trivial_axioms.intro upair_ax
    Union_ax by unfold_locales

context M_ZF_trans
begin
```

10.2 Interface with *M_basic*

```
schematic_goal inter_fm_auto:
  assumes
    nth(i,env) = x nth(j,env) = B
    i ∈ nat j ∈ nat env ∈ list(A)
  shows
    (∀y∈A . y∈B ⟶ x∈y) ⟷ sats(A,?ifm(i,j),env)
  by (insert assms ; (rule sep_rules | simp)+)

lemma inter_sep_intf :
  assumes
    A∈M
  shows
    separation(##M,λx . ∀y∈M . y∈A ⟶ x∈y)
proof -
  obtain ifm where
    fmsats:⋀env. env∈list(M) ⟹ (∀ y∈M. y∈(nth(1,env)) ⟶ nth(0,env)∈y)
    ⟷ sats(M,ifm(0,1),env)
    and
    ifm(0,1) ∈ formula
    and
    arity(ifm(0,1)) = 2
    using ‹A∈M› inter_fm_auto
    by (simp del:FOL_sats_iff add: nat_simp_union)
```

```
  then
  have ∀ a∈M. separation(##M, λx. sats(M,ifm(0,1) , [x, a]))
    using separation_ax by simp
  moreover
  have (∀ y∈M . y∈a ⟶ x∈y) ⟷ sats(M,ifm(0,1),[x,a])
    if a∈M x∈M for a x
    using that fmsats[of [x,a]] by simp
  ultimately
  have ∀ a∈M. separation(##M, λx . ∀ y∈M . y∈a ⟶ x∈y)
    unfolding separation_def by simp
  with ‹A∈M› show ?thesis by simp
qed


schematic_goal diff_fm_auto:
  assumes
    nth(i,env) = x nth(j,env) = B
    i ∈ nat j ∈ nat env ∈ list(A)
  shows
    x∉B ⟷ sats(A,?dfm(i,j),env)
  by (insert assms ; (rule sep_rules | simp)+)

lemma diff_sep_intf :
  assumes
    B∈M
  shows
    separation(##M,λx . x∉B)
proof -
  obtain dfm where
    fmsats:⋀env. env∈list(M) ⟹ nth(0,env)∉nth(1,env)
    ⟷ sats(M,dfm(0,1),env)
    and
    dfm(0,1) ∈ formula
    and
    arity(dfm(0,1)) = 2
    using ‹B∈M› diff_fm_auto
    by (simp del:FOL_sats_iff add: nat_simp_union)
  then
  have ∀ b∈M. separation(##M, λx. sats(M,dfm(0,1) , [x, b]))
    using separation_ax by simp
  moreover
  have x∉b ⟷ sats(M,dfm(0,1),[x,b])
    if b∈M x∈M for b x
    using that fmsats[of [x,b]] by simp
  ultimately
  have ∀ b∈M. separation(##M, λx . x∉b)
    unfolding separation_def by simp
  with ‹B∈M› show ?thesis by simp
```

```
qed

schematic_goal cprod_fm_auto:
  assumes
    nth(i,env) = z nth(j,env) = B nth(h,env) = C
    i ∈ nat j ∈ nat h ∈ nat env ∈ list(A)
  shows
   (∃x∈A. x∈B ∧ (∃y∈A. y∈C ∧ pair(##A,x,y,z))) ⟷ sats(A,?cpfm(i,j,h),env)
  by (insert assms ; (rule sep_rules | simp)+)


lemma cartprod_sep_intf :
  assumes
    A∈M
    and
    B∈M
  shows
    separation(##M,λz. ∃x∈M. x∈A ∧ (∃y∈M. y∈B ∧ pair(##M,x,y,z)))
proof -
  obtain cpfm where
    fmsats:⋀env. env∈list(M) ⟹
   (∃x∈M. x∈nth(1,env) ∧ (∃y∈M. y∈nth(2,env) ∧ pair(##M,x,y,nth(0,env))))
    ⟷ sats(M,cpfm(0,1,2),env)
    and
    cpfm(0,1,2) ∈ formula
    and
    arity(cpfm(0,1,2)) = 3
   using cprod_fm_auto by (simp del:FOL_sats_iff add: fm_definitions nat_simp_union)
  then
  have ∀a∈M. ∀b∈M. separation(##M, λz. sats(M,cpfm(0,1,2) , [z, a, b]))
    using separation_ax by simp
  moreover
 have (∃x∈M. x∈a ∧ (∃y∈M. y∈b ∧ pair(##M,x,y,z))) ⟷ sats(M,cpfm(0,1,2),[z,a,b])
    if a∈M b∈M z∈M for a b z
    using that fmsats[of [z,a,b]] by simp
  ultimately
   have ∀a∈M. ∀b∈M. separation(##M, λz . (∃x∈M. x∈a ∧ (∃y∈M. y∈b ∧
pair(##M,x,y,z))))
    unfolding separation_def by simp
  with ‹A∈M› ‹B∈M› show ?thesis by simp
qed

schematic_goal im_fm_auto:
  assumes
    nth(i,env) = y nth(j,env) = r nth(h,env) = B
    i ∈ nat j ∈ nat h ∈ nat env ∈ list(A)
  shows
   (∃p∈A. p∈r & (∃x∈A. x∈B & pair(##A,x,y,p))) ⟷ sats(A,?imfm(i,j,h),env)
  by (insert assms ; (rule sep_rules | simp)+)
```

lemma *image_sep_intf* :
assumes
A∈*M*
and
r∈*M*
shows
separation(##*M*, λ*y*. ∃ *p*∈*M*. *p*∈*r* & (∃ *x*∈*M*. *x*∈*A* & *pair*(##*M*,*x*,*y*,*p*)))
proof -
obtain *imfm* **where**
fmsats:⋀*env*. *env*∈*list*(*M*) ⟹
(∃ *p*∈*M*. *p*∈*nth*(*1*,*env*) & (∃ *x*∈*M*. *x*∈*nth*(*2*,*env*) & *pair*(##*M*,*x*,*nth*(*0*,*env*),*p*)))
⟷ *sats*(*M*,*imfm*(*0*,*1*,*2*),*env*)
and
imfm(*0*,*1*,*2*) ∈ *formula*
and
arity(*imfm*(*0*,*1*,*2*)) = *3*
using *im_fm_auto* **by** (*simp del*:*FOL_sats_iff pair_abs add*: *fm_definitions nat_simp_union*)
then
have ∀ *r*∈*M*. ∀ *a*∈*M*. *separation*(##*M*, λ*y*. *sats*(*M*,*imfm*(*0*,*1*,*2*) , [*y*,*r*,*a*]))
using *separation_ax* **by** *simp*
moreover
have (∃ *p*∈*M*. *p*∈*k* & (∃ *x*∈*M*. *x*∈*a* & *pair*(##*M*,*x*,*y*,*p*))) ⟷ *sats*(*M*,*imfm*(*0*,*1*,*2*),[*y*,*k*,*a*])
if *k*∈*M* *a*∈*M* *y*∈*M* **for** *k a y*
using *that fmsats*[*of* [*y*,*k*,*a*]] **by** *simp*
ultimately
have ∀ *k*∈*M*. ∀ *a*∈*M*. *separation*(##*M*, λ*y* . ∃ *p*∈*M*. *p*∈*k* & (∃ *x*∈*M*. *x*∈*a* & *pair*(##*M*,*x*,*y*,*p*)))
unfolding *separation_def* **by** *simp*
with ‹*r*∈*M*› ‹*A*∈*M*› **show** *?thesis* **by** *simp*
qed

schematic_goal *con_fm_auto*:
assumes
nth(*i*,*env*) = *z* *nth*(*j*,*env*) = *R*
i ∈ *nat* *j* ∈ *nat* *env* ∈ *list*(*A*)
shows
(∃ *p*∈*A*. *p*∈*R* & (∃ *x*∈*A*.∃ *y*∈*A*. *pair*(##*A*,*x*,*y*,*p*) & *pair*(##*A*,*y*,*x*,*z*)))
⟷ *sats*(*A*,*?cfm*(*i*,*j*),*env*)
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *converse_sep_intf* :
assumes
R∈*M*
shows
separation(##*M*,λ*z*. ∃ *p*∈*M*. *p*∈*R* & (∃ *x*∈*M*.∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*p*) & *pair*(##*M*,*y*,*x*,*z*)))
proof -

```
  obtain cfm where
    fmsats:⋀env. env∈list(M) ⟹
   (∃p∈M. p∈nth(1,env) & (∃x∈M.∃y∈M. pair(##M,x,y,p) & pair(##M,y,x,nth(0,env))))
    ⟷ sats(M,cfm(0,1),env)
    and
    cfm(0,1) ∈ formula
    and
    arity(cfm(0,1)) = 2
   using con_fm_auto by (simp del:FOL_sats_iff pair_abs add: fm_definitions nat_simp_union)
  then
  have ∀r∈M. separation(##M, λz. sats(M,cfm(0,1) , [z,r]))
    using separation_ax by simp
  moreover
  have (∃p∈M. p∈r & (∃x∈M.∃y∈M. pair(##M,x,y,p) & pair(##M,y,x,z)))
⟷
          sats(M,cfm(0,1),[z,r])
    if z∈M r∈M for z r
    using that fmsats[of [z,r]] by simp
  ultimately
 have ∀r∈M. separation(##M, λz . ∃p∈M. p∈r & (∃x∈M.∃y∈M. pair(##M,x,y,p)
& pair(##M,y,x,z)))
    unfolding separation_def by simp
  with ‹R∈M› show ?thesis by simp
qed


schematic_goal rest_fm_auto:
  assumes
    nth(i,env) = z nth(j,env) = C
    i ∈ nat j ∈ nat env ∈ list(A)
  shows
    (∃x∈A. x∈C & (∃y∈A. pair(##A,x,y,z)))
  ⟷ sats(A,?rfm(i,j),env)
  by (insert assms ; (rule sep_rules | simp)+)


lemma restrict_sep_intf :
  assumes
    A∈M
  shows
    separation(##M,λz. ∃x∈M. x∈A & (∃y∈M. pair(##M,x,y,z)))
proof -
  obtain rfm where
    fmsats:⋀env. env∈list(M) ⟹
    (∃x∈M. x∈nth(1,env) & (∃y∈M. pair(##M,x,y,nth(0,env))))
    ⟷ sats(M,rfm(0,1),env)
    and
    rfm(0,1) ∈ formula
    and
```

arity(*rfm*(*0*,*1*)) = *2*
using *rest_fm_auto* **by** (*simp del*:*FOL_sats_iff pair_abs add*: *fm_definitions nat_simp_union*)
then
have ∀ *a*∈*M*. *separation*(##*M*, λ*z*. *sats*(*M*,*rfm*(*0*,*1*) , [*z*,*a*]))
using *separation_ax* **by** *simp*
moreover
have (∃ *x*∈*M*. *x*∈*a* & (∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*z*))) ⟷
sats(*M*,*rfm*(*0*,*1*),[*z*,*a*])
if *z*∈*M a*∈*M* **for** *z a*
using *that fmsats*[*of* [*z*,*a*]] **by** *simp*
ultimately
have ∀ *a*∈*M*. *separation*(##*M*, λ*z* . ∃ *x*∈*M*. *x*∈*a* & (∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*z*)))
unfolding *separation_def* **by** *simp*
with ⟨*A*∈*M*⟩ **show** *?thesis* **by** *simp*
qed

schematic_goal *comp_fm_auto*:
assumes
nth(*i*,*env*) = *xz nth*(*j*,*env*) = *S nth*(*h*,*env*) = *R*
i ∈ *nat j* ∈ *nat h* ∈ *nat env* ∈ *list*(*A*)
shows
(∃ *x*∈*A*. ∃ *y*∈*A*. ∃ *z*∈*A*. ∃ *xy*∈*A*. ∃ *yz*∈*A*.
pair(##*A*,*x*,*z*,*xz*) & *pair*(##*A*,*x*,*y*,*xy*) & *pair*(##*A*,*y*,*z*,*yz*) & *xy*∈*S* & *yz*∈*R*)
⟷ *sats*(*A*,*?cfm*(*i*,*j*,*h*),*env*)
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *comp_sep_intf* :
assumes
R∈*M*
and
S∈*M*
shows
separation(##*M*,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 -
obtain *cfm* **where**
fmsats:⋀*env*. *env*∈*list*(*M*) ⟹
(∃ *x*∈*M*. ∃ *y*∈*M*. ∃ *z*∈*M*. ∃ *xy*∈*M*. ∃ *yz*∈*M*. *pair*(##*M*,*x*,*z*,*nth*(*0*,*env*)) &
pair(##*M*,*x*,*y*,*xy*) & *pair*(##*M*,*y*,*z*,*yz*) & *xy*∈*nth*(*1*,*env*) & *yz*∈*nth*(*2*,*env*))
⟷ *sats*(*M*,*cfm*(*0*,*1*,*2*),*env*)
and
cfm(*0*,*1*,*2*) ∈ *formula*
and
arity(*cfm*(*0*,*1*,*2*)) = *3*
using *comp_fm_auto* **by** (*simp del*:*FOL_sats_iff pair_abs add*: *fm_definitions*

nat_simp_union)
then
have ∀ *r*∈*M*. ∀ *s*∈*M*. *separation*(##*M*, λ*y*. *sats*(*M*,*cfm*(*0*,*1*,*2*) , [*y*,*s*,*r*]))
using *separation_ax* **by** *simp*
moreover
have (∃ *x*∈*M*. ∃ *y*∈*M*. ∃ *z*∈*M*. ∃ *xy*∈*M*. ∃ *yz*∈*M*.
pair(##*M*,*x*,*z*,*xz*) & *pair*(##*M*,*x*,*y*,*xy*) & *pair*(##*M*,*y*,*z*,*yz*) & *xy*∈*s*
& *yz*∈*r*)
⟷ *sats*(*M*,*cfm*(*0*,*1*,*2*) , [*xz*,*s*,*r*])
if *xz*∈*M* *s*∈*M* *r*∈*M* **for** *xz s r*
using *that fmsats*[*of* [*xz*,*s*,*r*]] **by** *simp*
ultimately
have ∀ *s*∈*M*. ∀ *r*∈*M*. *separation*(##*M*, 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*)
unfolding *separation_def* **by** *simp*
with ‹*S*∈*M*› ‹*R*∈*M*› **show** *?thesis* **by** *simp*
qed

schematic_goal *pred_fm_auto*:
assumes
nth(*i*,*env*) = *y* *nth*(*j*,*env*) = *R* *nth*(*h*,*env*) = *X*
i ∈ *nat* *j* ∈ *nat* *h* ∈ *nat* *env* ∈ *list*(*A*)
shows
(∃ *p*∈*A*. *p*∈*R* & *pair*(##*A*,*y*,*X*,*p*)) ⟷ *sats*(*A*,*?pfm*(*i*,*j*,*h*),*env*)
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *pred_sep_intf*:
assumes
R∈*M*
and
X∈*M*
shows
separation(##*M*, λ*y*. ∃ *p*∈*M*. *p*∈*R* & *pair*(##*M*,*y*,*X*,*p*))
proof -
obtain *pfm* **where**
fmsats:⋀*env*. *env*∈*list*(*M*) ⟹
(∃ *p*∈*M*. *p*∈*nth*(*1*,*env*) & *pair*(##*M*,*nth*(*0*,*env*),*nth*(*2*,*env*),*p*)) ⟷ *sats*(*M*,*pfm*(*0*,*1*,*2*),*env*)
and
pfm(*0*,*1*,*2*) ∈ *formula*
and
arity(*pfm*(*0*,*1*,*2*)) = *3*
using *pred_fm_auto* **by** (*simp del*:*FOL_sats_iff pair_abs add*: *fm_definitions*
nat_simp_union)
then
have ∀ *x*∈*M*. ∀ *r*∈*M*. *separation*(##*M*, λ*y*. *sats*(*M*,*pfm*(*0*,*1*,*2*) , [*y*,*r*,*x*]))

```
    using separation_ax by simp
  moreover
  have (∃ p∈M. p∈r & pair(##M,y,x,p))
        ⟷ sats(M,pfm(0,1,2) , [y,r,x])
    if y∈M r∈M x∈M for y x r
    using that fmsats[of [y,r,x]] by simp
  ultimately
  have ∀ x∈M. ∀ r∈M. separation(##M, λ y . ∃ p∈M. p∈r & pair(##M,y,x,p))
    unfolding separation_def by simp
  with ‹X∈M› ‹R∈M› show ?thesis by simp
qed


schematic_goal mem_fm_auto:
  assumes
    nth(i,env) = z i ∈ nat env ∈ list(A)
  shows
    (∃ x∈A. ∃ y∈A. pair(##A,x,y,z) & x ∈ y) ⟷ sats(A,?mfm(i),env)
  by (insert assms ; (rule sep_rules | simp)+)

lemma memrel_sep_intf:
  separation(##M, λz. ∃ x∈M. ∃ y∈M. pair(##M,x,y,z) & x ∈ y)
proof -
  obtain mfm where
    fmsats:⋀env. env∈list(M) ⟹
   (∃ x∈M. ∃ y∈M. pair(##M,x,y,nth(0,env)) & x ∈ y) ⟷ sats(M,mfm(0),env)
    and
    mfm(0) ∈ formula
    and
    arity(mfm(0)) = 1
     using mem_fm_auto by (simp del:FOL_sats_iff pair_abs add: fm_definitions
nat_simp_union)
  then
  have separation(##M, λz. sats(M,mfm(0) , [z]))
    using separation_ax by simp
  moreover
  have (∃ x∈M. ∃ y∈M. pair(##M,x,y,z) & x ∈ y) ⟷ sats(M,mfm(0),[z])
    if z∈M for z
    using that fmsats[of [z]] by simp
  ultimately
  have separation(##M, λz . ∃ x∈M. ∃ y∈M. pair(##M,x,y,z) & x ∈ y)
    unfolding separation_def by simp
  then show ?thesis by simp
qed

schematic_goal recfun_fm_auto:
  assumes
    nth(i1,env) = x nth(i2,env) = r nth(i3,env) = f nth(i4,env) = g nth(i5,env)
= a
```

```
    nth(i6,env) = b i1∈nat i2∈nat i3∈nat i4∈nat i5∈nat i6∈nat env ∈ list(A)
  shows
    (∃xa∈A. ∃xb∈A. pair(##A,x,a,xa) & xa ∈ r & pair(##A,x,b,xb) & xb ∈ r
&
                (∃fx∈A. ∃gx∈A. fun_apply(##A,f,x,fx) & fun_apply(##A,g,x,gx)
& fx ≠ gx))
    ⟷ sats(A,?rffm(i1,i2,i3,i4,i5,i6),env)
  by (insert assms ; (rule sep_rules | simp)+)


lemma is_recfun_sep_intf :
  assumes
    r∈M f∈M g∈M a∈M b∈M
  shows
    separation(##M,λx. ∃xa∈M. ∃xb∈M.
                pair(##M,x,a,xa) & xa ∈ r & pair(##M,x,b,xb) & xb ∈ r &
              (∃fx∈M. ∃gx∈M. fun_apply(##M,f,x,fx) & fun_apply(##M,g,x,gx)
&
                                   fx ≠ gx))
proof -
  obtain rffm where
    fmsats:⋀env. env∈list(M) ⟹
    (∃xa∈M. ∃xb∈M. pair(##M,nth(0,env),nth(4,env),xa) & xa ∈ nth(1,env)
&
    pair(##M,nth(0,env),nth(5,env),xb) & xb ∈ nth(1,env) & (∃fx∈M. ∃gx∈M.
   fun_apply(##M,nth(2,env),nth(0,env),fx) & fun_apply(##M,nth(3,env),nth(0,env),gx)
& fx ≠ gx))
    ⟷ sats(M,rffm(0,1,2,3,4,5),env)
    and
    rffm(0,1,2,3,4,5) ∈ formula
    and
    arity(rffm(0,1,2,3,4,5)) = 6
     using recfun_fm_auto by (simp del:FOL_sats_iff pair_abs add: fm_definitions
nat_simp_union)
  then
  have ∀a1∈M. ∀a2∈M. ∀a3∈M. ∀a4∈M. ∀a5∈M.
        separation(##M, λx. sats(M,rffm(0,1,2,3,4,5) , [x,a1,a2,a3,a4,a5]))
    using separation_ax by simp
  moreover
  have (∃xa∈M. ∃xb∈M. pair(##M,x,a4,xa) & xa ∈ a1 & pair(##M,x,a5,xb)
& xb ∈ a1 &
          (∃fx∈M. ∃gx∈M. fun_apply(##M,a2,x,fx) & fun_apply(##M,a3,x,gx)
& fx ≠ gx))
          ⟷ sats(M,rffm(0,1,2,3,4,5) , [x,a1,a2,a3,a4,a5])
    if x∈M a1∈M a2∈M a3∈M a4∈M a5∈M  for x a1 a2 a3 a4 a5
    using that fmsats[of [x,a1,a2,a3,a4,a5]] by simp
  ultimately
  have ∀a1∈M. ∀a2∈M. ∀a3∈M. ∀a4∈M. ∀a5∈M. separation(##M, λ x .
         ∃xa∈M. ∃xb∈M. pair(##M,x,a4,xa) & xa ∈ a1 & pair(##M,x,a5,xb)
```

& *xb* ∈ *a1* &
(∃ *fx*∈*M*. ∃ *gx*∈*M*. *fun_apply*(##*M*,*a2*,*x*,*fx*) & *fun_apply*(##*M*,*a3*,*x*,*gx*)
& *fx* ≠ *gx*))
unfolding *separation_def* **by** *simp*
with ‹*r*∈*M*› ‹*f*∈*M*› ‹*g*∈*M*› ‹*a*∈*M*› ‹*b*∈*M*› **show** *?thesis* **by** *simp*
qed

schematic_goal *funsp_fm_auto*:
assumes
nth(*i*,*env*) = *p* *nth*(*j*,*env*) = *z* *nth*(*h*,*env*) = *n*
i ∈ *nat* *j* ∈ *nat* *h* ∈ *nat* *env* ∈ *list*(*A*)
shows
(∃ *f*∈*A*. ∃ *b*∈*A*. ∃ *nb*∈*A*. ∃ *cnbf*∈*A*. *pair*(##*A*,*f*,*b*,*p*) & *pair*(##*A*,*n*,*b*,*nb*) &
is_cons(##*A*,*nb*,*f*,*cnbf*) &
upair(##*A*,*cnbf*,*cnbf*,*z*)) ⟷ *sats*(*A*,*?fsfm*(*i*,*j*,*h*),*env*)
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *funspace_succ_rep_intf* :
assumes
n∈*M*
shows
strong_replacement(##*M*,
λ*p z*. ∃ *f*∈*M*. ∃ *b*∈*M*. ∃ *nb*∈*M*. ∃ *cnbf*∈*M*.
pair(##*M*,*f*,*b*,*p*) & *pair*(##*M*,*n*,*b*,*nb*) & *is_cons*(##*M*,*nb*,*f*,*cnbf*)
&
upair(##*M*,*cnbf*,*cnbf*,*z*))
proof -
obtain *fsfm* **where**
fmsats:*env*∈*list*(*M*) ⟹
(∃ *f*∈*M*. ∃ *b*∈*M*. ∃ *nb*∈*M*. ∃ *cnbf*∈*M*. *pair*(##*M*,*f*,*b*,*nth*(*0*,*env*)) & *pair*(##*M*,*nth*(*2*,*env*),*b*,*nb*)
& *is_cons*(##*M*,*nb*,*f*,*cnbf*) & *upair*(##*M*,*cnbf*,*cnbf*,*nth*(*1*,*env*)))
⟷ *sats*(*M*,*fsfm*(*0*,*1*,*2*),*env*)
and *fsfm*(*0*,*1*,*2*) ∈ *formula* **and** *arity*(*fsfm*(*0*,*1*,*2*)) = *3* **for** *env*
using *funsp_fm_auto*[*of concl*:*M*] **by** (*simp del*:*FOL_sats_iff pair_abs add*: *fm_definitions*
nat_simp_union)
then
have ∀ *n0*∈*M*. *strong_replacement*(##*M*, λ*p z*. *sats*(*M*,*fsfm*(*0*,*1*,*2*) , [*p*,*z*,*n0*]))
using *replacement_ax* **by** *simp*
moreover
have (∃ *f*∈*M*. ∃ *b*∈*M*. ∃ *nb*∈*M*. ∃ *cnbf*∈*M*. *pair*(##*M*,*f*,*b*,*p*) & *pair*(##*M*,*n0*,*b*,*nb*)
&
is_cons(##*M*,*nb*,*f*,*cnbf*) & *upair*(##*M*,*cnbf*,*cnbf*,*z*))
⟷ *sats*(*M*,*fsfm*(*0*,*1*,*2*) , [*p*,*z*,*n0*])
if *p*∈*M* *z*∈*M* *n0*∈*M* **for** *p z n0*
using *that fmsats*[*of* [*p*,*z*,*n0*]] **by** *simp*

ultimately
have ∀ *n0*∈*M*. *strong_replacement*(##*M*, λ *p z*.
∃ *f*∈*M*. ∃ *b*∈*M*. ∃ *nb*∈*M*. ∃ *cnbf*∈*M*. *pair*(##*M*,*f*,*b*,*p*) & *pair*(##*M*,*n0*,*b*,*nb*)
&
is_cons(##*M*,*nb*,*f*,*cnbf*) & *upair*(##*M*,*cnbf*,*cnbf*,*z*))
unfolding *strong_replacement_def univalent_def* **by** *simp*
with ‹*n*∈*M*› **show** *?thesis* **by** *simp*
qed

lemmas *M_basic_sep_instances* =
inter_sep_intf diff_sep_intf cartprod_sep_intf
image_sep_intf converse_sep_intf restrict_sep_intf
pred_sep_intf memrel_sep_intf comp_sep_intf is_recfun_sep_intf

end

sublocale *M_ZF_trans* ⊆ *M_basic* ##*M*
using *trans_M zero_in_M power_ax M_basic_sep_instances funspace_succ_rep_intf*
by *unfold_locales auto*

10.3 Interface with *M_trancl*

schematic_goal *rtran_closure_mem_auto*:
assumes
nth(*i*,*env*) = *p nth*(*j*,*env*) = *r* *nth*(*k*,*env*) = *B*
i ∈ *nat j* ∈ *nat k* ∈ *nat env* ∈ *list*(*A*)
shows
rtran_closure_mem(##*A*,*B*,*r*,*p*) ⟷ *sats*(*A*,*?rcfm*(*i*,*j*,*k*),*env*)
unfolding *rtran_closure_mem_def*
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma (**in** *M_ZF_trans*) *rtrancl_separation_intf*:
assumes
r∈*M*
and
A∈*M*
shows
separation (##*M*, *rtran_closure_mem*(##*M*,*A*,*r*))
proof -
obtain *rcfm* **where**
fmsats:⋀*env*. *env*∈*list*(*M*) ⟹
(*rtran_closure_mem*(##*M*,*nth*(*2*,*env*),*nth*(*1*,*env*),*nth*(*0*,*env*))) ⟷ *sats*(*M*,*rcfm*(*0*,*1*,*2*),*env*)
and
rcfm(*0*,*1*,*2*) ∈ *formula*
and

```
    arity(rcfm(0,1,2)) = 3
   using rtran_closure_mem_auto by (simp del:FOL_sats_iff pair_abs add: fm_definitions
nat_simp_union)
  then
  have ∀ x∈M. ∀ a∈M. separation(##M, λy. sats(M,rcfm(0,1,2) , [y,x,a]))
    using separation_ax by simp
  moreover
  have (rtran_closure_mem(##M,a,x,y))
          ⟷ sats(M,rcfm(0,1,2) , [y,x,a])
    if y∈M x∈M a∈M for y x a
    using that fmsats[of [y,x,a]] by simp
  ultimately
  have ∀ x∈M. ∀ a∈M. separation(##M, rtran_closure_mem(##M,a,x))
    unfolding separation_def by simp
  with ‹r∈M› ‹A∈M› show ?thesis by simp
qed

schematic_goal rtran_closure_fm_auto:
  assumes
    nth(i,env) = r nth(j,env) = rp
    i ∈ nat j ∈ nat env ∈ list(A)
  shows
    rtran_closure(##A,r,rp) ⟷ sats(A,?rtc(i,j),env)
  unfolding rtran_closure_def
  by (insert assms ; (rule sep_rules rtran_closure_mem_auto| simp)+)

schematic_goal trans_closure_fm_auto:
  assumes
    nth(i,env) = r nth(j,env) = rp
    i ∈ nat j ∈ nat env ∈ list(A)
  shows
    tran_closure(##A,r,rp) ⟷ sats(A,?tc(i,j),env)
  unfolding tran_closure_def
  by (insert assms ; (rule sep_rules rtran_closure_fm_auto | simp))+

synthesize trans_closure_fm from_schematic trans_closure_fm_auto

schematic_goal wellfounded_trancl_fm_auto:
  assumes
    nth(i,env) = p nth(j,env) = r  nth(k,env) = B
    i ∈ nat j ∈ nat k ∈ nat env ∈ list(A)
  shows
    wellfounded_trancl(##A,B,r,p) ⟷ sats(A,?wtf(i,j,k),env)
  unfolding  wellfounded_trancl_def
  by (insert assms ; (rule sep_rules trans_closure_fm_iff_sats | simp)+)

context M_ZF_trans
begin
```

lemma *wftrancl_separation_intf*:
  assumes
    r∈*M* **and** *Z*∈*M*
  shows
    separation (##*M*, *wellfounded_trancl*(##*M*,*Z*,*r*))
proof -
  obtain *rcfm* **where**
    fmsats:⋀*env*. *env*∈*list*(*M*) ⟹
   (*wellfounded_trancl*(##*M*,*nth*(*2*,*env*),*nth*(*1*,*env*),*nth*(*0*,*env*))) ⟷ *sats*(*M*,*rcfm*(*0*,*1*,*2*),*env*)
    and
    rcfm(*0*,*1*,*2*) ∈ *formula*
    and
    arity(*rcfm*(*0*,*1*,*2*)) = *3*
   using *wellfounded_trancl_fm_auto*[*of concl*:*M nth*(*2*,_)] **unfolding** *fm_definitions*
    by (*simp del*:*FOL_sats_iff pair_abs add*: *nat_simp_union*)
  then
  have ∀ *x*∈*M*. ∀ *z*∈*M*. *separation*(##*M*, λ*y*. *sats*(*M*,*rcfm*(*0*,*1*,*2*) , [*y*,*x*,*z*]))
    using *separation_ax* **by** *simp*
  moreover
  have (*wellfounded_trancl*(##*M*,*z*,*x*,*y*))
          ⟷ *sats*(*M*,*rcfm*(*0*,*1*,*2*) , [*y*,*x*,*z*])
    if *y*∈*M* *x*∈*M* *z*∈*M* **for** *y x z*
    using *that fmsats*[*of* [*y*,*x*,*z*]] **by** *simp*
  ultimately
  have ∀ *x*∈*M*. ∀ *z*∈*M*. *separation*(##*M*, *wellfounded_trancl*(##*M*,*z*,*x*))
    unfolding *separation_def* **by** *simp*
  with ‹*r*∈*M*› ‹*Z*∈*M*› **show** *?thesis* **by** *simp*
qed

Proof that *nat* ∈ *M*

lemma *finite_sep_intf*: *separation*(##*M*, λ*x*. *x*∈*nat*)
proof -
  have *arity*(*finite_ordinal_fm*(*0*)) = *1*
     unfolding *finite_ordinal_fm_def limit_ordinal_fm_def empty_fm_def succ_fm_def*
cons_fm_def
      union_fm_def upair_fm_def
    by (*simp add*: *nat_union_abs1 Un_commute*)
  with *separation_ax*
  have (∀ *v*∈*M*. *separation*(##*M*,λ*x*. *sats*(*M*,*finite_ordinal_fm*(*0*),[*x*,*v*])))
    by *simp*
  then have (∀ *v*∈*M*. *separation*(##*M*,*finite_ordinal*(##*M*)))
    unfolding *separation_def* **by** *simp*
  then have *separation*(##*M*,*finite_ordinal*(##*M*))
    using *zero_in_M* **by** *auto*
  then show *?thesis* **unfolding** *separation_def* **by** *simp*
qed

lemma *nat_subset_I'*:
  ⟦ *I*∈*M* ; *0*∈*I* ; ⋀*x*. *x*∈*I* ⟹ *succ*(*x*)∈*I* ⟧ ⟹ *nat* ⊆ *I*

by (*rule subsetI*,*induct_tac x*,*simp*+)

lemma *nat_subset_I*: ∃ *I*∈*M*. *nat* ⊆ *I*
proof -
have ∃ *I*∈*M*. *0*∈*I* ∧ (∀ *x*∈*M*. *x*∈*I* ⟶ *succ*(*x*)∈*I*)
using *infinity_ax* **unfolding** *infinity_ax_def* **by** *auto*
then obtain *I* **where**
I∈*M* *0*∈*I* (∀ *x*∈*M*. *x*∈*I* ⟶ *succ*(*x*)∈*I*)
by *auto*
then have ⋀*x*. *x*∈*I* ⟹ *succ*(*x*)∈*I*
using *transitivity* **by** *simp*
then have *nat*⊆*I*
using ‹*I*∈*M*› ‹*0*∈*I*› *nat_subset_I'* **by** *simp*
then show *?thesis* **using** ‹*I*∈*M*› **by** *auto*
qed

lemma *nat_in_M*: *nat* ∈ *M*
proof -
have *1*:{*x*∈*B* . *x*∈*A*}=*A* **if** *A*⊆*B* **for** *A B*
using *that* **by** *auto*
obtain *I* **where**
I∈*M* *nat*⊆*I*
using *nat_subset_I* **by** *auto*
then have {*x*∈*I* . *x*∈*nat*} ∈ *M*
using *finite_sep_intf separation_closed*[*of* λ*x* . *x*∈*nat*] **by** *simp*
then show *?thesis*
using ‹*nat*⊆*I*› *1* **by** *simp*
qed

end

sublocale *M_ZF_trans* ⊆ *M_trancl* *##M*
using *rtrancl_separation_intf wftrancl_separation_intf nat_in_M*
wellfounded_trancl_def **by** *unfold_locales auto*

10.4 Interface with *M_eclose*

lemma *repl_sats*:
assumes
sat:⋀*x z*. *x*∈*M* ⟹ *z*∈*M* ⟹ *sats*(*M*,φ,*Cons*(*x*,*Cons*(*z*,*env*))) ⟷ *P*(*x*,*z*)
shows
strong_replacement(*##M*,λ*x z*. *sats*(*M*,φ,*Cons*(*x*,*Cons*(*z*,*env*)))) ⟷
strong_replacement(*##M*,*P*)
by (*rule strong_replacement_cong*,*simp add*:*sat*)

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

iterates_replacement(##*M*, *is_list_functor*(##*M*,*A*), *0*)
proof -
{
fix *n*
assume *n*∈*nat*
have *succ*(*n*)∈*M*
using ‹*n*∈*nat*› *nat_into_M* **by** *simp*
then have *1*:*Memrel*(*succ*(*n*))∈*M*
using ‹*n*∈*nat*› *Memrel_closed* **by** *simp*
have *0*∈*M*
using *nat_0I nat_into_M* **by** *simp*
then have *is_list_functor*(##*M*, *A*, *a*, *b*)
⟷ *sats*(*M*, *list_functor_fm*(*13*,*1*,*0*), [*b*,*a*,*c*,*d*,*a0*,*a1*,*a2*,*a3*,*a4*,*y*,*x*,*z*,*Memrel*(*succ*(*n*)),*A*,*0*])
if *a*∈*M b*∈*M c*∈*M d*∈*M a0*∈*M a1*∈*M a2*∈*M a3*∈*M a4*∈*M y*∈*M x*∈*M z*∈*M*
for *a b c d a0 a1 a2 a3 a4 y x z*
using *that 1* ‹*A*∈*M*› *list_functor_iff_sats* **by** *simp*
then have *sats*(*M*, *iterates_MH_fm*(*list_functor_fm*(*13*,*1*,*0*),*10*,*2*,*1*,*0*), [*a0*,*a1*,*a2*,*a3*,*a4*,*y*,*x*,*z*,*Memrel*(*succ*
⟷ *iterates_MH*(##*M*,*is_list_functor*(##*M*,*A*),*0*,*a2*, *a1*, *a0*)
if *a0*∈*M a1*∈*M a2*∈*M a3*∈*M a4*∈*M y*∈*M x*∈*M z*∈*M*
for *a0 a1 a2 a3 a4 y x z*
using *that sats_iterates_MH_fm*[*of M is_list_functor*(##*M*,*A*) _] *1* ‹*0*∈*M*›
‹*A*∈*M*› **by** *simp*
then have *2*:*sats*(*M*, *is_wfrec_fm*(*iterates_MH_fm*(*list_functor_fm*(*13*,*1*,*0*),*10*,*2*,*1*,*0*),*3*,*1*,*0*),
[*y*,*x*,*z*,*Memrel*(*succ*(*n*)),*A*,*0*])
⟷
is_wfrec(##*M*, *iterates_MH*(##*M*,*is_list_functor*(##*M*,*A*),*0*) , *Mem-
rel*(*succ*(*n*)), *x*, *y*)
if *y*∈*M x*∈*M z*∈*M* **for** *y x z*
using *that sats_is_wfrec_fm 1* ‹*0*∈*M*› ‹*A*∈*M*› **by** *simp*
let
?f=*Exists*(*And*(*pair_fm*(*1*,*0*,*2*),
is_wfrec_fm(*iterates_MH_fm*(*list_functor_fm*(*13*,*1*,*0*),*10*,*2*,*1*,*0*),*3*,*1*,*0*)))
have *satsf*:*sats*(*M*, *?f*, [*x*,*z*,*Memrel*(*succ*(*n*)),*A*,*0*])
⟷
(∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*z*) &
is_wfrec(##*M*, *iterates_MH*(##*M*,*is_list_functor*(##*M*,*A*),*0*) , *Mem-
rel*(*succ*(*n*)), *x*, *y*))
if *x*∈*M z*∈*M* **for** *x z*
using *that 2 1* ‹*0*∈*M*› ‹*A*∈*M*› **by** (*simp del*:*pair_abs*)
have *arity*(*?f*) = *5*
unfolding *fm_definitions*
by (*simp add*:*nat_simp_union*)
then
have *strong_replacement*(##*M*,λ*x z*. *sats*(*M*,*?f*,[*x*,*z*,*Memrel*(*succ*(*n*)),*A*,*0*]))
using *replacement_ax 1* ‹*A*∈*M*› ‹*0*∈*M*› **by** *simp*
then
have *strong_replacement*(##*M*,λ*x z*.
∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*z*) & *is_wfrec*(##*M*, *iterates_MH*(##*M*,*is_list_functor*(##*M*,*A*),*0*)
,

```
            Memrel(succ(n)), x, y))
    using repl_sats[of M ?f [Memrel(succ(n)),A,0]]  satsf by (simp del:pair_abs)
  }
  then
  show ?thesis unfolding iterates_replacement_def wfrec_replacement_def by simp
qed
```

```
lemma (in M_ZF_trans) iterates_repl_intf :
  assumes
    v∈M and
    isfm:is_F_fm ∈ formula and
    arty:arity(is_F_fm)=2 and
    satsf: ⋀a b env'. ⟦ a∈M ; b∈M ; env'∈list(M) ⟧
             ⟹ is_F(a,b) ⟷ sats(M, is_F_fm, [b,a]@env')
  shows
    iterates_replacement(##M,is_F,v)
proof -
  {
    fix n
    assume n∈nat
    have succ(n)∈M
      using ‹n∈nat› nat_into_M by simp
    then have 1:Memrel(succ(n))∈M
      using ‹n∈nat› Memrel_closed by simp
    {
      fix a0 a1 a2 a3 a4 y x z
      assume as:a0∈M a1∈M a2∈M a3∈M a4∈M y∈M x∈M z∈M
     have sats(M, is_F_fm, Cons(b,Cons(a,Cons(c,Cons(d,[a0,a1,a2,a3,a4,y,x,z,Memrel(succ(n)),v])))))
          ⟷ is_F(a,b)
        if a∈M b∈M c∈M d∈M for a b c d
        using as that 1 satsf[of a b [c,d,a0,a1,a2,a3,a4,y,x,z,Memrel(succ(n)),v]]
‹v∈M› by simp
      then
     have sats(M, iterates_MH_fm(is_F_fm,9,2,1,0), [a0,a1,a2,a3,a4,y,x,z,Memrel(succ(n)),v])
          ⟷ iterates_MH(##M,is_F,v,a2, a1, a0)
        using as
          sats_iterates_MH_fm[of M is_F is_F_fm] 1 ‹v∈M› by simp
    }
    then have 2:sats(M, is_wfrec_fm(iterates_MH_fm(is_F_fm,9,2,1,0),3,1,0),
                        [y,x,z,Memrel(succ(n)),v])
        ⟷
        is_wfrec(##M, iterates_MH(##M,is_F,v),Memrel(succ(n)), x, y)
      if y∈M x∈M z∈M for y x z
      using  that sats_is_wfrec_fm 1 ‹v∈M› by simp
    let
      ?f=Exists(And(pair_fm(1,0,2),
```

```
        is_wfrec_fm(iterates_MH_fm(is_F_fm,9,2,1,0),3,1,0)))
    have satsf:sats(M, ?f, [x,z,Memrel(succ(n)),v])
        ⟷
        (∃ y∈M. pair(##M,x,y,z) &
        is_wfrec(##M, iterates_MH(##M,is_F,v) , Memrel(succ(n)), x, y))
      if x∈M z∈M for x z
      using that 2 1 ‹v∈M› by (simp del:pair_abs)
    have arity(?f) = 4
      unfolding fm_definitions
      using arty by (simp add:nat_simp_union)
    then
    have strong_replacement(##M,λx z. sats(M,?f,[x,z,Memrel(succ(n)),v]))
      using replacement_ax 1 ‹v∈M› ‹is_F_fm∈formula› by simp
    then
    have strong_replacement(##M,λx z.
         ∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, iterates_MH(##M,is_F,v) ,
              Memrel(succ(n)), x, y))
      using repl_sats[of M ?f [Memrel(succ(n)),v]]  satsf by (simp del:pair_abs)
  }
  then
  show ?thesis unfolding iterates_replacement_def wfrec_replacement_def by simp
qed

lemma (in M_ZF_trans) formula_repl1_intf :
  iterates_replacement(##M, is_formula_functor(##M), 0)
proof -
  have 0∈M
    using  nat_0I nat_into_M by simp
  have 1:arity(formula_functor_fm(1,0)) = 2
    unfolding fm_definitions
    by (simp add:nat_simp_union)
  have 2:formula_functor_fm(1,0)∈formula by simp
  have is_formula_functor(##M,a,b) ⟷
      sats(M, formula_functor_fm(1,0), [b,a])
    if a∈M b∈M  for a b
    using that by simp
  then show ?thesis using ‹0∈M› 1 2 iterates_repl_intf by simp
qed

lemma (in M_ZF_trans) nth_repl_intf:
  assumes
    l ∈ M
  shows
    iterates_replacement(##M,λl' t. is_tl(##M,l',t),l)
proof -
  have 1:arity(tl_fm(1,0)) = 2
    unfolding fm_definitions by (simp add:nat_simp_union)
  have 2:tl_fm(1,0)∈formula by simp
  have is_tl(##M,a,b) ⟷ sats(M, tl_fm(1,0), [b,a])
```

if *a∈M b∈M* **for** *a b*
using *that* **by** *simp*
then show *?thesis* **using** ‹*l∈M*› *1 2 iterates_repl_intf* **by** *simp*
qed

lemma (**in** *M_ZF_trans*) *eclose_repl1_intf*:
assumes
A∈M
shows
iterates_replacement(##M, big_union(##M), A)
proof -
have *1*:*arity(big_union_fm(1,0)) = 2*
unfolding *fm_definitions* **by** (*simp add*:*nat_simp_union*)
have *2*:*big_union_fm(1,0)∈formula* **by** *simp*
have *big_union(##M,a,b) ⟷ sats(M, big_union_fm(1,0), [b,a])*
if *a∈M b∈M* **for** *a b*
using *that* **by** *simp*
then show *?thesis* **using** ‹*A∈M*› *1 2 iterates_repl_intf* **by** *simp*
qed

lemma (**in** *M_ZF_trans*) *list_repl2_intf*:
assumes
A∈M
shows
strong_replacement(##M,λn y. n∈nat & is_iterates(##M, is_list_functor(##M,A), 0, n, y))
proof -
have *0∈M*
using *nat_0I nat_into_M* **by** *simp*
have *is_list_functor(##M,A,a,b) ⟷*
sats(M,list_functor_fm(13,1,0),[b,a,c,d,e,f,g,h,i,j,k,n,y,A,0,nat])
if *a∈M b∈M c∈M d∈M e∈M f∈Mg∈Mh∈Mi∈Mj∈M k∈M n∈M y∈M*
for *a b c d e f g h i j k n y*
using *that* ‹*0∈M*› *nat_in_M* ‹*A∈M*› **by** *simp*
then
have *1*:*sats(M, is_iterates_fm(list_functor_fm(13,1,0),3,0,1),[n,y,A,0,nat]) ⟷*
is_iterates(##M, is_list_functor(##M,A), 0, n , y)
if *n∈M y∈M* **for** *n y*
using *that* ‹*0∈M*› ‹*A∈M*› *nat_in_M*
sats_is_iterates_fm[of M is_list_functor(##M,A)] **by** *simp*
let *?f = And(Member(0,4),is_iterates_fm(list_functor_fm(13,1,0),3,0,1))*
have *satsf*:*sats(M, ?f,[n,y,A,0,nat]) ⟷*
n∈nat & is_iterates(##M, is_list_functor(##M,A), 0, n, y)
if *n∈M y∈M* **for** *n y*
using *that* ‹*0∈M*› ‹*A∈M*› *nat_in_M 1* **by** *simp*
have *arity(?f) = 5*
unfolding *fm_definitions*

by (*simp add*:*nat_simp_union*)
then
have *strong_replacement*(##*M*,*λn y*. *sats*(*M*,*?f*,[*n*,*y*,*A*,*0*,*nat*]))
using *replacement_ax 1 nat_in_M* ‹*A*∈*M*› ‹*0*∈*M*› by *simp*
then
show *?thesis* using *repl_sats*[*of M ?f* [*A*,*0*,*nat*]] *satsf* by *simp*
qed

lemma (**in** *M_ZF_trans*) *formula_repl2_intf*:
strong_replacement(##*M*,*λn y*. *n*∈*nat* & *is_iterates*(##*M*, *is_formula_functor*(##*M*), *0*, *n*, *y*))
proof -
have *0*∈*M*
using *nat_0I nat_into_M* **by** *simp*
have *is_formula_functor*(##*M*,*a*,*b*) ⟷
sats(*M*,*formula_functor_fm*(*1*,*0*),[*b*,*a*,*c*,*d*,*e*,*f*,*g*,*h*,*i*,*j*,*k*,*n*,*y*,*0*,*nat*])
if *a*∈*M b*∈*M c*∈*M d*∈*M e*∈*M f*∈*Mg*∈*Mh*∈*Mi*∈*Mj*∈*M k*∈*M n*∈*M y*∈*M*
for *a b c d e f g h i j k n y*
using *that* ‹*0*∈*M*› *nat_in_M* **by** *simp*
then
have *1*:*sats*(*M*, *is_iterates_fm*(*formula_functor_fm*(*1*,*0*),*2*,*0*,*1*),[*n*,*y*,*0*,*nat*]) ⟷
is_iterates(##*M*, *is_formula_functor*(##*M*), *0*, *n* , *y*)
if *n*∈*M y*∈*M* **for** *n y*
using *that* ‹*0*∈*M*› *nat_in_M*
sats_is_iterates_fm[*of M is_formula_functor*(##*M*)] **by** *simp*
let *?f* = *And*(*Member*(*0*,*3*),*is_iterates_fm*(*formula_functor_fm*(*1*,*0*),*2*,*0*,*1*))
have *satsf*:*sats*(*M*, *?f*,[*n*,*y*,*0*,*nat*]) ⟷
n∈*nat* & *is_iterates*(##*M*, *is_formula_functor*(##*M*), *0*, *n*, *y*)
if *n*∈*M y*∈*M* **for** *n y*
using *that* ‹*0*∈*M*› *nat_in_M 1* **by** *simp*
have *artyf*:*arity*(*?f*) = *4*
unfolding *fm_definitions*
by (*simp add*:*nat_simp_union*)
then
have *strong_replacement*(##*M*,*λn y*. *sats*(*M*,*?f*,[*n*,*y*,*0*,*nat*]))
using *replacement_ax 1 artyf* ‹*0*∈*M*› *nat_in_M* **by** *simp*
then
show *?thesis* **using** *repl_sats*[*of M ?f* [*0*,*nat*]] *satsf* **by** *simp*
qed

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

proof -
  have *big_union*(##*M*,*a*,*b*) ⟷
      sats(*M*,*big_union_fm*(*1*,*0*),[*b*,*a*,*c*,*d*,*e*,*f*,*g*,*h*,*i*,*j*,*k*,*n*,*y*,*A*,*nat*])
    if *a*∈*M* *b*∈*M* *c*∈*M* *d*∈*M* *e*∈*M* *f*∈*Mg*∈*Mh*∈*Mi*∈*Mj*∈*M* *k*∈*M* *n*∈*M* *y*∈*M*
    for *a b c d e f g h i j k n y*
    using *that* ‹*A*∈*M*› *nat_in_M* **by** *simp*
  then
  have *1*:*sats*(*M*, *is_iterates_fm*(*big_union_fm*(*1*,*0*),*2*,*0*,*1*),[*n*,*y*,*A*,*nat*]) ⟷
      is_iterates(##*M*, *big_union*(##*M*), *A*, *n* , *y*)
    if *n*∈*M* *y*∈*M* **for** *n y*
    using *that* ‹*A*∈*M*› *nat_in_M*
      sats_is_iterates_fm[*of M big_union*(##*M*)] **by** *simp*
  let *?f* = *And*(*Member*(*0*,*3*),*is_iterates_fm*(*big_union_fm*(*1*,*0*),*2*,*0*,*1*))
  have *satsf*:*sats*(*M*, *?f*,[*n*,*y*,*A*,*nat*]) ⟷
      n∈*nat* & *is_iterates*(##*M*, *big_union*(##*M*), *A*, *n*, *y*)
    if *n*∈*M* *y*∈*M* **for** *n y*
    using *that* ‹*A*∈*M*› *nat_in_M 1* **by** *simp*
  have *artyf*:*arity*(*?f*) = *4*
    unfolding *fm_definitions*
    by (*simp add*:*nat_simp_union*)
  then
  have *strong_replacement*(##*M*,λ*n y*. *sats*(*M*,*?f*,[*n*,*y*,*A*,*nat*]))
    using *replacement_ax 1 artyf* ‹*A*∈*M*› *nat_in_M* **by** *simp*
  then
  show *?thesis* **using** *repl_sats*[*of M ?f* [*A*,*nat*]] *satsf* **by** *simp*
qed

sublocale *M_ZF_trans* ⊆ *M_datatypes* ##*M*
  using *list_repl1_intf list_repl2_intf formula_repl1_intf*
    formula_repl2_intf nth_repl_intf
  by *unfold_locales auto*

sublocale *M_ZF_trans* ⊆ *M_eclose* ##*M*
  using *eclose_repl1_intf eclose_repl2_intf*
  by *unfold_locales auto*

definition
  powerset_fm :: [*i*,*i*] ⇒ *i* **where**
  powerset_fm(*A*,*z*) ≡ *Forall*(*Iff*(*Member*(*0*,*succ*(*z*)),*subset_fm*(*0*,*succ*(*A*))))

lemma *powerset_type* [*TC*]:
  ⟦ *x* ∈ *nat*; *y* ∈ *nat* ⟧ ⟹ *powerset_fm*(*x*,*y*) ∈ *formula*
  by (*simp add*:*powerset_fm_def*)

definition
  is_powapply_fm :: [*i*,*i*,*i*] ⇒ *i* **where**

```
  is_powapply_fm(f,y,z) ≡
     Exists(And(fun_apply_fm(succ(f), succ(y), 0),
          Forall(Iff(Member(0, succ(succ(z))),
          Forall(Implies(Member(0, 1), Member(0, 2)))))))

lemma is_powapply_type [TC] :
  ⟦f∈nat ; y∈nat; z∈nat⟧ ⟹ is_powapply_fm(f,y,z)∈formula
  unfolding is_powapply_fm_def by simp

declare is_powapply_fm_def[fm_definitions add]

lemma sats_is_powapply_fm :
  assumes
    f∈nat y∈nat z∈nat env∈list(A) 0∈A
  shows
    is_powapply(##A,nth(f, env),nth(y, env),nth(z, env))
    ⟷ sats(A,is_powapply_fm(f,y,z),env)
  unfolding is_powapply_def is_powapply_fm_def powerset_def subset_def
  using nth_closed assms by simp


lemma (in M_ZF_trans) powapply_repl :
  assumes
    f∈M
  shows
    strong_replacement(##M,is_powapply(##M,f))
proof -
  have arity(is_powapply_fm(2,0,1)) = 3
    unfolding is_powapply_fm_def
    by (simp add: fm_definitions nat_simp_union)
  then
  have ∀f0∈M. strong_replacement(##M, λp z. sats(M,is_powapply_fm(2,0,1) ,
[p,z,f0]))
    using replacement_ax by simp
  moreover
  have is_powapply(##M,f0,p,z) ⟷ sats(M,is_powapply_fm(2,0,1) , [p,z,f0])
    if p∈M z∈M f0∈M for p z f0
    using that zero_in_M sats_is_powapply_fm[of 2 0 1 [p,z,f0] M] by simp
  ultimately
  have ∀f0∈M. strong_replacement(##M, is_powapply(##M,f0))
    unfolding strong_replacement_def univalent_def by simp
  with ‹f∈M› show ?thesis by simp
qed



definition
  PHrank_fm :: [i,i,i] ⇒ i where
  PHrank_fm(f,y,z) ≡ Exists(And(fun_apply_fm(succ(f),succ(y),0)
```

```
                              ,succ_fm(0,succ(z))))

lemma PHrank_type [TC]:
  ⟦ x ∈ nat; y ∈ nat; z ∈ nat ⟧ ⟹ PHrank_fm(x,y,z) ∈ formula
  by (simp add:PHrank_fm_def)


lemma (in M_ZF_trans) sats_PHrank_fm:
  ⟦ x ∈ nat; y ∈ nat; z ∈ nat;  env ∈ list(M) ⟧
    ⟹ sats(M,PHrank_fm(x,y,z),env) ⟷
        PHrank(##M,nth(x,env),nth(y,env),nth(z,env))
 using zero_in_M Internalizations.nth_closed by (simp add: PHrank_def PHrank_fm_def)


lemma (in M_ZF_trans) phrank_repl :
  assumes
    f∈M
  shows
    strong_replacement(##M,PHrank(##M,f))
proof -
  have arity(PHrank_fm(2,0,1)) = 3
    unfolding PHrank_fm_def
    by (simp add: fm_definitions nat_simp_union)
  then
 have ∀ f0∈M. strong_replacement(##M, λp z. sats(M,PHrank_fm(2,0,1) , [p,z,f0]))
    using replacement_ax by simp
  then
  have ∀ f0∈M. strong_replacement(##M, PHrank(##M,f0))
   unfolding strong_replacement_def univalent_def by (simp add:sats_PHrank_fm)
  with ‹f∈M› show ?thesis by simp
qed



definition
  is_Hrank_fm :: [i,i,i] ⇒ i where
  is_Hrank_fm(x,f,hc) ≡ Exists(And(big_union_fm(0,succ(hc)),
                        Replace_fm(succ(x),PHrank_fm(succ(succ(succ(f))),0,1),0)))

lemma is_Hrank_type [TC]:
  ⟦ x ∈ nat; y ∈ nat; z ∈ nat ⟧ ⟹ is_Hrank_fm(x,y,z) ∈ formula
  by (simp add:is_Hrank_fm_def)

lemma (in M_ZF_trans) sats_is_Hrank_fm:
  ⟦ x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(M)⟧
    ⟹ sats(M,is_Hrank_fm(x,y,z),env) ⟷
        is_Hrank(##M,nth(x,env),nth(y,env),nth(z,env))
  using zero_in_M is_Hrank_def is_Hrank_fm_def sats_Replace_fm
  by (simp add:sats_PHrank_fm)
```

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

```
lemma (in M_ZF_trans) wfrec_rank :
  assumes
    X∈M
  shows
    wfrec_replacement(##M,is_Hrank(##M),rrank(X))
proof -
  have
    is_Hrank(##M,a2, a1, a0) ⟷
          sats(M, is_Hrank_fm(2,1,0), [a0,a1,a2,a3,a4,y,x,z,rrank(X)])
    if a4∈M a3∈M a2∈M a1∈M a0∈M y∈M x∈M z∈M for a4 a3 a2 a1 a0 y x z
    using that rrank_in_M ‹X∈M› by (simp add:sats_is_Hrank_fm)
  then
  have
    1:sats(M, is_wfrec_fm(is_Hrank_fm(2,1,0),3,1,0),[y,x,z,rrank(X)])
  ⟷ is_wfrec(##M, is_Hrank(##M) ,rrank(X), x, y)
    if y∈M x∈M z∈M for y x z
   using that ‹X∈M› rrank_in_M sats_is_wfrec_fm by (simp add:sats_is_Hrank_fm)
  let
    ?f=Exists(And(pair_fm(1,0,2),is_wfrec_fm(is_Hrank_fm(2,1,0),3,1,0)))
  have satsf:sats(M, ?f, [x,z,rrank(X)])
             ⟷ (∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, is_Hrank(##M) ,
rrank(X), x, y))
    if x∈M z∈M for x z
    using that 1 ‹X∈M› rrank_in_M by (simp del:pair_abs)
  have arity(?f) = 3
    unfolding fm_definitions
    by (simp add:nat_simp_union)
  then
  have strong_replacement(##M,λx z. sats(M,?f,[x,z,rrank(X)]))
    using replacement_ax 1 ‹X∈M› rrank_in_M by simp
  then
  have strong_replacement(##M,λx z.
        ∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, is_Hrank(##M) , rrank(X),
x, y))
    using repl_sats[of M ?f [rrank(X)]]  satsf by (simp del:pair_abs)
  then
  show ?thesis unfolding wfrec_replacement_def  by simp
qed
```

```
definition
  is_HVfrom_fm :: [i,i,i,i] ⇒ i where
  is_HVfrom_fm(A,x,f,h) ≡ Exists(Exists(And(union_fm(A #+ 2,1,h #+ 2),
                          And(big_union_fm(0,1),
                          Replace_fm(x #+ 2,is_powapply_fm(f #+ 4,0,1),0)))))
```

declare *is_HVfrom_fm_def*[*fm_definitions add*]

lemma *is_HVfrom_type* [*TC*]:
  ⟦ *A*∈*nat*; *x* ∈ *nat*; *f* ∈ *nat*; *h* ∈ *nat* ⟧ ⟹ *is_HVfrom_fm*(*A*,*x*,*f*,*h*) ∈ *formula*
  by (*simp add*:*is_HVfrom_fm_def*)

lemma *sats_is_HVfrom_fm* :
  ⟦ *a*∈*nat*; *x* ∈ *nat*; *f* ∈ *nat*; *h* ∈ *nat*; *env* ∈ *list*(*A*); *0*∈*A*⟧
    ⟹ *sats*(*A*,*is_HVfrom_fm*(*a*,*x*,*f*,*h*),*env*) ⟷
        is_HVfrom(##*A*,*nth*(*a*,*env*),*nth*(*x*,*env*),*nth*(*f*,*env*),*nth*(*h*,*env*))
  using *is_HVfrom_def is_HVfrom_fm_def sats_Replace_fm*[*OF sats_is_powapply_fm*]
  by *simp*

lemma *is_HVfrom_iff_sats*:
  assumes
    nth(*a*,*env*) = *aa nth*(*x*,*env*) = *xx nth*(*f*,*env*) = *ff nth*(*h*,*env*) = *hh*
    a∈*nat x*∈*nat f*∈*nat h*∈*nat env*∈*list*(*A*) *0*∈*A*
  shows
    is_HVfrom(##*A*,*aa*,*xx*,*ff*,*hh*) ⟷ *sats*(*A*, *is_HVfrom_fm*(*a*,*x*,*f*,*h*), *env*)
  using *assms sats_is_HVfrom_fm* **by** *simp*

schematic_goal *sats_is_Vset_fm_auto*:
  assumes
    i∈*nat v*∈*nat env*∈*list*(*A*) *0*∈*A*
    i < *length*(*env*) *v* < *length*(*env*)
  shows
    is_Vset(##*A*,*nth*(*i*, *env*),*nth*(*v*, *env*))
    ⟷ *sats*(*A*,*?ivs_fm*(*i*,*v*),*env*)
  unfolding *is_Vset_def is_Vfrom_def*
  by (*insert assms*; (*rule sep_rules is_HVfrom_iff_sats is_transrec_iff_sats* | *simp*)+)

schematic_goal *is_Vset_iff_sats*:
  assumes
    nth(*i*,*env*) = *ii nth*(*v*,*env*) = *vv*
    i∈*nat v*∈*nat env*∈*list*(*A*) *0*∈*A*
    i < *length*(*env*) *v* < *length*(*env*)
  shows
    is_Vset(##*A*,*ii*,*vv*) ⟷ *sats*(*A*, *?ivs_fm*(*i*,*v*), *env*)
  unfolding ‹*nth*(*i*,*env*) = *ii*›[*symmetric*] ‹*nth*(*v*,*env*) = *vv*›[*symmetric*]
  by (*rule sats_is_Vset_fm_auto*(*1*); *simp add*:*assms*)

lemma (**in** *M_ZF_trans*) *memrel_eclose_sing* :
  a∈*M* ⟹ ∃ *sa*∈*M*. ∃ *esa*∈*M*. ∃ *mesa*∈*M*.
     upair(##*M*,*a*,*a*,*sa*) & *is_eclose*(##*M*,*sa*,*esa*) & *membership*(##*M*,*esa*,*mesa*)
  using *upair_ax eclose_closed Memrel_closed* **unfolding** *upair_ax_def*
  by (*simp del*:*upair_abs*)

```
lemma (in M_ZF_trans) trans_repl_HVFrom :
  assumes
    A∈M i∈M
  shows
    transrec_replacement(##M,is_HVfrom(##M,A),i)
proof -
  { fix mesa
    assume mesa∈M
    have
      0:is_HVfrom(##M,A,a2, a1, a0) ⟷
      sats(M, is_HVfrom_fm(8,2,1,0), [a0,a1,a2,a3,a4,y,x,z,A,mesa])
      if a4∈M a3∈M a2∈M a1∈M a0∈M y∈M x∈M z∈M for a4 a3 a2 a1 a0 y x
z
      using that zero_in_M sats_is_HVfrom_fm ‹mesa∈M› ‹A∈M› by simp
    have
      1:sats(M, is_wfrec_fm(is_HVfrom_fm(8,2,1,0),4,1,0),[y,x,z,A,mesa])
        ⟷ is_wfrec(##M, is_HVfrom(##M,A),mesa, x, y)
      if y∈M x∈M z∈M for y x z
      using that ‹A∈M› ‹mesa∈M› sats_is_wfrec_fm[OF 0] by simp
    let
      ?f=Exists(And(pair_fm(1,0,2),is_wfrec_fm(is_HVfrom_fm(8,2,1,0),4,1,0)))
    have satsf:sats(M, ?f, [x,z,A,mesa])
          ⟷ (∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, is_HVfrom(##M,A)
, mesa, x, y))
      if x∈M z∈M for x z
      using that 1 ‹A∈M› ‹mesa∈M› by (simp del:pair_abs)
    have arity(?f) = 4
      unfolding fm_definitions
      by (simp add:nat_simp_union)
    then
    have strong_replacement(##M,λx z. sats(M,?f,[x,z,A,mesa]))
      using replacement_ax 1 ‹A∈M› ‹mesa∈M› by simp
    then
    have strong_replacement(##M,λx z.
          ∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, is_HVfrom(##M,A) , mesa,
x, y))
      using repl_sats[of M ?f [A,mesa]]  satsf by (simp del:pair_abs)
    then
    have wfrec_replacement(##M,is_HVfrom(##M,A),mesa)
      unfolding wfrec_replacement_def  by simp
  }
  then show ?thesis unfolding transrec_replacement_def
    using ‹i∈M› memrel_eclose_sing by simp
qed

sublocale M_ZF_trans ⊆ M_eclose_pow ##M
  using power_ax powapply_repl phrank_repl trans_repl_HVFrom
    wfrec_rank by unfold_locales auto
```

```
lemma (in M_ZF_trans) repl_gen :
  assumes
    f_abs: ⋀x y. ⟦ x∈M; y∈M ⟧ ⟹ is_F(##M,x,y) ⟷ y = f(x)
    and
    f_sats: ⋀x y. ⟦x∈M ; y∈M ⟧ ⟹
          sats(M,f_fm,Cons(x,Cons(y,env))) ⟷ is_F(##M,x,y)
    and
    f_form: f_fm ∈ formula
    and
    f_arty: arity(f_fm) = 2
    and
    env∈list(M)
  shows
    strong_replacement(##M, λx y. y = f(x))
proof -
  have sats(M,f_fm,[x,y]@env) ⟷ is_F(##M,x,y) if x∈M y∈M for x y
    using that f_sats[of x y] by simp
  moreover
  from f_form f_arty
  have strong_replacement(##M, λx y. sats(M,f_fm,[x,y]@env))
    using ‹env∈list(M)› replacement_ax by simp
  ultimately
  have strong_replacement(##M, is_F(##M))
   using strong_replacement_cong[of ##M λx y. sats(M,f_fm,[x,y]@env) is_F(##M)]
by simp
  with f_abs show ?thesis
    using strong_replacement_cong[of ##M is_F(##M) λx y. y = f(x)] by simp
qed


lemma (in M_ZF_trans) sep_in_M :
  assumes
    φ ∈ formula env∈list(M)
    arity(φ) ≤ 1 #+ length(env) A∈M and
    satsQ: ⋀x. x∈M ⟹ sats(M,φ,[x]@env) ⟷ Q(x)
  shows
    {y∈A . Q(y)}∈M
proof -
  have separation(##M,λx. sats(M,φ,[x] @ env))
    using assms separation_ax by simp
  then show ?thesis using
      ‹A∈M› satsQ trans_M
      separation_cong[of ##M λy. sats(M,φ,[y]@env) Q]
      separation_closed by simp
qed

end
```

11 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

lemma Transset_M :
  Transset(M) ⟹  y∈x ⟹ x ∈ M ⟹ y ∈ M
  by (simp add: Transset_def,auto)



locale M_ctm = M_ZF_trans +
  fixes enum
  assumes M_countable:      enum∈bij(nat,M)
begin

lemma tuples_in_M: A∈M ⟹ B∈M ⟹ ⟨A,B⟩∈M
  by (simp flip:setclass_iff)
```

11.1 *Collects* in *M*

```
lemma Collect_in_M_0p :
  assumes
    Qfm : Q_fm ∈ formula and
    Qarty : arity(Q_fm) = 1 and
    Qsats : ⋀x. x∈M ⟹ sats(M,Q_fm,[x]) ⟷ is_Q(##M,x) and
    Qabs  : ⋀x. x∈M ⟹ is_Q(##M,x) ⟷ Q(x) and
    A∈M
  shows
    Collect(A,Q)∈M
proof -
  have z∈A ⟹ z∈M for z
    using ‹A∈M› transitivity by simp
  then
  have 1:Collect(A,is_Q(##M)) = Collect(A,Q)
    using Qabs Collect_cong[of A A is_Q(##M) Q] by simp
  have separation(##M,is_Q(##M))
    using separation_ax Qsats Qarty Qfm
      separation_cong[of ##M λy. sats(M,Q_fm,[y]) is_Q(##M)]
    by simp
  then
  have Collect(A,is_Q(##M))∈M
    using separation_closed ‹A∈M› by simp
```

then
show *?thesis* **using** *1* **by** *simp*
qed

lemma *Collect_in_M_2p* :
assumes
Qfm : *Q_fm* ∈ *formula* **and**
Qarty : *arity*(*Q_fm*) = *3* **and**
params_M : *y*∈*M* *z*∈*M* **and**
Qsats : ⋀*x*. *x*∈*M* ⟹ *sats*(*M*,*Q_fm*,[*x*,*y*,*z*]) ⟷ *is_Q*(##*M*,*x*,*y*,*z*) **and**
Qabs : ⋀*x*. *x*∈*M* ⟹ *is_Q*(##*M*,*x*,*y*,*z*) ⟷ *Q*(*x*,*y*,*z*) **and**
A∈*M*
shows
Collect(*A*,λ*x*. *Q*(*x*,*y*,*z*))∈*M*
proof -
have *z*∈*A* ⟹ *z*∈*M* **for** *z*
using ‹*A*∈*M*› *transitivity* **by** *simp*
then
have *1*:*Collect*(*A*,λ*x*. *is_Q*(##*M*,*x*,*y*,*z*)) = *Collect*(*A*,λ*x*. *Q*(*x*,*y*,*z*))
using *Qabs* *Collect_cong*[*of* *A* *A* λ*x*. *is_Q*(##*M*,*x*,*y*,*z*) λ*x*. *Q*(*x*,*y*,*z*)] **by** *simp*
have *separation*(##*M*,λ*x*. *is_Q*(##*M*,*x*,*y*,*z*))
using *separation_ax* *Qsats* *Qarty* *Qfm* *params_M*
separation_cong[*of* ##*M* λ*x*. *sats*(*M*,*Q_fm*,[*x*,*y*,*z*]) λ*x*. *is_Q*(##*M*,*x*,*y*,*z*)]
by *simp*
then
have *Collect*(*A*,λ*x*. *is_Q*(##*M*,*x*,*y*,*z*))∈*M*
using *separation_closed* ‹*A*∈*M*› **by** *simp*
then
show *?thesis* **using** *1* **by** *simp*
qed

lemma *Collect_in_M_4p* :
assumes
Qfm : *Q_fm* ∈ *formula* **and**
Qarty : *arity*(*Q_fm*) = *5* **and**
params_M : *a1*∈*M* *a2*∈*M* *a3*∈*M* *a4*∈*M* **and**
Qsats : ⋀*x*. *x*∈*M* ⟹ *sats*(*M*,*Q_fm*,[*x*,*a1*,*a2*,*a3*,*a4*]) ⟷ *is_Q*(##*M*,*x*,*a1*,*a2*,*a3*,*a4*)
and
Qabs : ⋀*x*. *x*∈*M* ⟹ *is_Q*(##*M*,*x*,*a1*,*a2*,*a3*,*a4*) ⟷ *Q*(*x*,*a1*,*a2*,*a3*,*a4*) **and**
A∈*M*
shows
Collect(*A*,λ*x*. *Q*(*x*,*a1*,*a2*,*a3*,*a4*))∈*M*
proof -
have *z*∈*A* ⟹ *z*∈*M* **for** *z*
using ‹*A*∈*M*› *transitivity* **by** *simp*
then
have *1*:*Collect*(*A*,λ*x*. *is_Q*(##*M*,*x*,*a1*,*a2*,*a3*,*a4*)) = *Collect*(*A*,λ*x*. *Q*(*x*,*a1*,*a2*,*a3*,*a4*))

using *Qabs* *Collect_cong*[*of* *A* *A* λ*x*. *is_Q*(##*M*,*x*,*a1*,*a2*,*a3*,*a4*) λ*x*. *Q*(*x*,*a1*,*a2*,*a3*,*a4*)]

```
    by simp
  have separation(##M,λx. is_Q(##M,x,a1,a2,a3,a4))
    using separation_ax Qsats Qarty Qfm params_M
      separation_cong[of ##M λx. sats(M,Q_fm,[x,a1,a2,a3,a4])
        λx. is_Q(##M,x,a1,a2,a3,a4)]
    by simp
  then
  have Collect(A,λx. is_Q(##M,x,a1,a2,a3,a4))∈M
    using separation_closed ‹A∈M› by simp
  then
  show ?thesis using 1 by simp
qed

lemma Repl_in_M :
  assumes
    f_fm: f_fm ∈ formula and
    f_ar: arity(f_fm)≤ 2 #+ length(env) and
    fsats: ⋀x y. x∈M ⟹ y∈M ⟹ sats(M,f_fm,[x,y]@env) ⟷ is_f(x,y) and
    fabs: ⋀x y. x∈M ⟹ y∈M ⟹ is_f(x,y) ⟷ y = f(x) and
    fclosed: ⋀x. x∈A ⟹ f(x) ∈ M and
    A∈M env∈list(M)
  shows {f(x). x∈A}∈M
proof -
  have strong_replacement(##M, λx y. sats(M,f_fm,[x,y]@env))
    using replacement_ax f_fm f_ar ‹env∈list(M)› by simp
  then
  have strong_replacement(##M, λx y. y = f(x))
    using fsats fabs
      strong_replacement_cong[of ##M λx y. sats(M,f_fm,[x,y]@env) λx y. y =
f(x)]
    by simp
  then
  have { y . x∈A , y = f(x) } ∈ M
    using ‹A∈M› fclosed strong_replacement_closed by simp
  moreover
  have {f(x). x∈A} = { y . x∈A , y = f(x) }
    by auto
  ultimately show ?thesis by simp
qed

end
```

11.2 A forcing locale and generic filters

```
locale forcing_data = forcing_notion + M_ctm +
  assumes P_in_M:          P ∈ M
    and leq_in_M:        leq ∈ M
```

begin

lemma *transD* : *Transset*(*M*) $\Longrightarrow$ *y* ∈ *M* $\Longrightarrow$ *y* ⊆ *M*
by (*unfold Transset_def*, *blast*)

lemmas *P_sub_M* = *transD*[*OF trans_M P_in_M*]

definition
M_generic :: *i*⇒*o* **where**
M_generic(*G*) ≡ *filter*(*G*) ∧ (∀ *D*∈*M*. *D*⊆*P* ∧ *dense*(*D*)⟶*D*∩*G*≠*0*)

lemma *M_genericD* [*dest*]: *M_generic*(*G*) $\Longrightarrow$ *x*∈*G* $\Longrightarrow$ *x*∈*P*
unfolding *M_generic_def* **by** (*blast dest*:*filterD*)

lemma *M_generic_leqD* [*dest*]: *M_generic*(*G*) $\Longrightarrow$ *p*∈*G* $\Longrightarrow$ *q*∈*P* $\Longrightarrow$ *p*≼*q* $\Longrightarrow$ *q*∈*G*
unfolding *M_generic_def* **by** (*blast dest*:*filter_leqD*)

lemma *M_generic_compatD* [*dest*]: *M_generic*(*G*) $\Longrightarrow$ *p*∈*G* $\Longrightarrow$ *r*∈*G* $\Longrightarrow$ ∃ *q*∈*G*. *q*≼*p* ∧ *q*≼*r*
unfolding *M_generic_def* **by** (*blast dest*:*low_bound_filter*)

lemma *M_generic_denseD* [*dest*]: *M_generic*(*G*) $\Longrightarrow$ *dense*(*D*) $\Longrightarrow$ *D*⊆*P* $\Longrightarrow$ *D*∈*M* $\Longrightarrow$ ∃ *q*∈*G*. *q*∈*D*
unfolding *M_generic_def* **by** *blast*

lemma *G_nonempty*: *M_generic*(*G*) $\Longrightarrow$ *G*≠*0*
proof -
have *P*⊆*P* **..**
assume
M_generic(*G*)
with *P_in_M P_dense* ‹*P*⊆*P*› **show**
G ≠ *0*
unfolding *M_generic_def* **by** *auto*
qed

lemma *one_in_G* :
assumes *M_generic*(*G*)
shows *one* ∈ *G*
proof -
from *assms* **have** *G*⊆*P*
unfolding *M_generic_def* **and** *filter_def* **by** *simp*
from ‹*M_generic*(*G*)› **have** *increasing*(*G*)
unfolding *M_generic_def* **and** *filter_def* **by** *simp*
with ‹*G*⊆*P*› **and** ‹*M_generic*(*G*)›
show *?thesis*
using *G_nonempty* **and** *one_in_P* **and** *one_max*
unfolding *increasing_def* **by** *blast*

```
qed

lemma G_subset_M: M_generic(G) ⟹ G ⊆ M
  using transitivity[OF _ P_in_M] by auto

declare iff_trans [trans]

lemma generic_filter_existence:
  p∈P ⟹ ∃ G. p∈G ∧ M_generic(G)
proof -
  assume p∈P
  let ?D=λn∈nat. (if (enum`n⊆P ∧ dense(enum`n))  then enum`n else P)
  have ∀ n∈nat. ?D`n ∈ Pow(P)
    by auto
  then
  have ?D:nat→Pow(P)
    using lam_type by auto
  have Eq4: ∀ n∈nat. dense(?D`n)
  proof(intro ballI)
    fix n
    assume n∈nat
    then
     have dense(?D`n) ⟷ dense(if enum`n ⊆ P ∧ dense(enum`n) then enum`n
else P)
      by simp
    also
    have ... ⟷  (¬(enum`n ⊆ P ∧ dense(enum`n)) ⟶ dense(P))
      using split_if by simp
    finally
    show dense(?D`n)
      using P_dense ‹n∈nat› by auto
  qed
  from ‹?D∈_› and Eq4
  interpret cg: countable_generic P leq one ?D
    by (unfold_locales, auto)
  from ‹p∈P›
  obtain G where Eq6: p∈G ∧ filter(G) ∧ (∀ n∈nat.(?D`n)∩G≠0)
    using cg.countable_rasiowa_sikorski[where M=λ_. M]  P_sub_M
     M_countable[THEN bij_is_fun] M_countable[THEN bij_is_surj, THEN surj_range]

    unfolding cg.D_generic_def by blast
  then
  have Eq7: (∀ D∈M. D⊆P ∧ dense(D)⟶D∩G≠0)
  proof (intro ballI impI)
    fix D
    assume D∈M and Eq9: D ⊆ P ∧ dense(D)
    have ∀ y∈M. ∃ x∈nat. enum`x= y
      using M_countable and  bij_is_surj unfolding surj_def by (simp)
    with ‹D∈M› obtain n where Eq10: n∈nat ∧ enum`n = D
```

```
      by auto
    with Eq9 and if_P
    have ?D`n = D by (simp)
    with Eq6 and Eq10
    show D∩G≠0 by auto
  qed
  with Eq6
  show ?thesis unfolding M_generic_def by auto
qed

end


lemma (in M_trivial) compat_in_abs :
  assumes
    M(A) M(r) M(p) M(q)
  shows
    is_compat_in(M,A,r,p,q) ⟷ compat_in(A,r,p,q)
  using assms unfolding is_compat_in_def compat_in_def by simp

context forcing_data begin

definition
  compat_in_fm :: [i,i,i,i] ⇒ i where
  compat_in_fm(A,r,p,q) ≡
    Exists(And(Member(0,succ(A)),Exists(And(pair_fm(1,p#+2,0),
                                    And(Member(0,r#+2),
                          Exists(And(pair_fm(2,q#+3,0),Member(0,r#+3))))))))

lemma compat_in_fm_type[TC] :
  ⟦ A∈nat;r∈nat;p∈nat;q∈nat⟧ ⟹ compat_in_fm(A,r,p,q)∈formula
  unfolding compat_in_fm_def by simp

lemma sats_compat_in_fm:
  assumes
    A∈nat r∈nat  p∈nat q∈nat env∈list(M)
  shows
    sats(M,compat_in_fm(A,r,p,q),env) ⟷
           is_compat_in(##M,nth(A, env),nth(r, env),nth(p, env),nth(q, env))
  unfolding compat_in_fm_def is_compat_in_def using assms by simp

end

end
```

12 The ZFC axioms, internalized

```
theory Internal_ZFC_Axioms
  imports
```

Forcing_Data

begin

schematic_goal *ZF_union_auto*:
Union_ax(##*A*) ⟷ (*A*, [] ⊨ *?zfunion*)
unfolding *Union_ax_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_union_fm* **from_schematic** *ZF_union_auto*

schematic_goal *ZF_power_auto*:
power_ax(##*A*) ⟷ (*A*, [] ⊨ *?zfpow*)
unfolding *power_ax_def powerset_def subset_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_power_fm* **from_schematic** *ZF_power_auto*

schematic_goal *ZF_pairing_auto*:
upair_ax(##*A*) ⟷ (*A*, [] ⊨ *?zfpair*)
unfolding *upair_ax_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_pairing_fm* **from_schematic** *ZF_pairing_auto*

schematic_goal *ZF_foundation_auto*:
foundation_ax(##*A*) ⟷ (*A*, [] ⊨ *?zfpow*)
unfolding *foundation_ax_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_foundation_fm* **from_schematic** *ZF_foundation_auto*

schematic_goal *ZF_extensionality_auto*:
extensionality(##*A*) ⟷ (*A*, [] ⊨ *?zfpow*)
unfolding *extensionality_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_extensionality_fm* **from_schematic** *ZF_extensionality_auto*

schematic_goal *ZF_infinity_auto*:
infinity_ax(##*A*) ⟷ (*A*, [] ⊨ (*?*φ(*i*,*j*,*h*)))
unfolding *infinity_ax_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_infinity_fm* **from_schematic** *ZF_infinity_auto*

schematic_goal *ZF_choice_auto*:
choice_ax(##*A*) ⟷ (*A*, [] ⊨ (*?*φ(*i*,*j*,*h*)))
unfolding *choice_ax_def*

by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_choice_fm* **from_schematic** *ZF_choice_auto*

syntax
_choice :: *i* (*AC*)
translations
AC ⇀ *CONST ZF_choice_fm*

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 ≡ { *ZF_extensionality_fm*, *ZF_foundation_fm*, *ZF_pairing_fm*, *ZF_union_fm*, *ZF_infinity_fm*, *ZF_power_fm* }

definition
ZFC_fin :: *i* **where**
ZFC_fin ≡ *ZF_fin* ∪ {*ZF_choice_fm*}

lemma *ZFC_fin_type* : *ZFC_fin* ⊆ *formula*
unfolding *ZFC_fin_def ZF_fin_def ZFC_fm_defs* **by** (*auto*)

12.1 The Axiom of Separation, internalized

lemma *iterates_Forall_type* [*TC*]:
⟦ *n* ∈ *nat*; *p* ∈ *formula* ⟧ ⟹ *Forall^n*(*p*) ∈ *formula*
by (*induct set*:*nat*, *auto*)

lemma *last_init_eq* :
assumes *l* ∈ *list*(*A*) *length*(*l*) = *succ*(*n*)
shows ∃ *a*∈*A*. ∃ *l'*∈*list*(*A*). *l* = *l'*@[*a*]
proof-
from ‹*l*∈_› ‹*length*(_) = _›
have *rev*(*l*) ∈ *list*(*A*) *length*(*rev*(*l*)) = *succ*(*n*)
by *simp_all*
then
obtain *a l'* **where** *a*∈*A* *l'*∈*list*(*A*) *rev*(*l*) = *Cons*(*a*,*l'*)
by (*cases*;*simp*)
then
have *l* = *rev*(*l'*) @ [*a*] *rev*(*l'*) ∈ *list*(*A*)
using *rev_rev_ident*[*OF* ‹*l*∈_›] **by** *auto*
with ‹*a*∈_›
show *?thesis* **by** *blast*
qed

```
lemma take_drop_eq :
  assumes l∈list(M)
  shows ⋀ n . n < succ(length(l)) ⟹ l = take(n,l) @ drop(n,l)
  using ‹l∈list(M)›
proof induct
  case Nil
  then show ?case by auto
next
  case (Cons a l)
  then show ?case
  proof -
    {
      fix i
      assume i<succ(succ(length(l)))
      with ‹l∈list(M)›
      consider (lt) i = 0 | (eq) ∃k∈nat. i = succ(k) ∧ k < succ(length(l))
        using ‹l∈list(M)›  le_natI nat_imp_quasinat
        by (cases rule:nat_cases[of i];auto)
      then
      have take(i,Cons(a,l)) @ drop(i,Cons(a,l)) = Cons(a,l)
        using Cons
        by (cases;auto)
    }
    then show ?thesis using Cons by auto
  qed
qed

lemma list_split :
assumes n ≤ succ(length(rest)) rest ∈ list(M)
shows  ∃re∈list(M). ∃st∈list(M). rest = re @ st ∧ length(re) = pred(n)
proof -
  from assms
  have pred(n) ≤ length(rest)
    using pred_mono[OF _ ‹n≤_›] pred_succ_eq by auto
  with ‹rest∈_›
  have pred(n)∈nat rest = take(pred(n),rest) @ drop(pred(n),rest) (is _ = ?re @
?st)
    using take_drop_eq[OF ‹rest∈_›] le_natI by auto
  then
  have length(?re) = pred(n) ?re∈list(M) ?st∈list(M)
    using length_take[rule_format,OF _ ‹pred(n)∈_›] ‹pred(n) ≤ _› ‹rest∈_›
    unfolding min_def
    by auto
  then
  show ?thesis
   using rev_bexI[of _ _ λ re. ∃st∈list(M). rest = re @ st ∧ length(re) = pred(n)]
     ‹length(?re) = _› ‹rest = _›
    by auto
```

qed

lemma *sats_nForall*:
  assumes
    $\varphi \in formula$
  shows
    $n{\in}nat \Longrightarrow ms \in list(M) \Longrightarrow$
      $M,\ ms \models (Forall\hat{\ }n(\varphi)) \longleftrightarrow$
      $(\forall\, rest \in list(M).\ length(rest) = n \longrightarrow M,\ rest\ @\ ms \models \varphi)$
proof (*induct n arbitrary*:*ms set*:*nat*)
  case *0*
  with *assms*
  show *?case* **by** *simp*
next
  case (*succ n*)
  have $(\forall\, rest{\in}list(M).\ length(rest) = succ(n) \longrightarrow P(rest,n)) \longleftrightarrow$
       $(\forall\, t{\in}M.\ \forall\, res{\in}list(M).\ length(res) = n \longrightarrow P(res\ @\ [t],n))$
    if $n{\in}nat$ **for** *n P*
    using *that last_init_eq* **by** *force*
  from *this*[*of* _ λ*rest* _. (*M*, *rest* @ *ms* ⊨ φ)] ‹*n*∈*nat*›
  have $(\forall\, rest{\in}list(M).\ length(rest) = succ(n) \longrightarrow M,\ rest\ @\ ms \models \varphi) \longleftrightarrow$
       $(\forall\, t{\in}M.\ \forall\, res{\in}list(M).\ length(res) = n \longrightarrow\ M,\ (res\ @\ [t])\ @\ ms \models \varphi)$
    by *simp*
    with *assms succ*(*1*,*3*) *succ*(*2*)[*of Cons*(_,*ms*)]
  show *?case*
    using *arity_sats_iff*[*of* φ _ *M Cons*(_, *ms* @ _)] *app_assoc*
    by (*simp*)
qed

definition
  $sep\_body\_fm :: i \Rightarrow i$ **where**
  $sep\_body\_fm(p) \equiv Forall(Exists(Forall($
              $Iff(Member(0,1),And(Member(0,2),$
                   $incr\_bv1\hat{\ }2(p))))))$

lemma *sep_body_fm_type* [*TC*]: $p \in formula \Longrightarrow sep\_body\_fm(p) \in formula$
  by (*simp add*: *sep_body_fm_def*)

lemma *sats_sep_body_fm*:
  assumes
    $\varphi \in formula\ ms{\in}list(M)\ rest{\in}list(M)$
  shows
    $M,\ rest\ @\ ms \models sep\_body\_fm(\varphi) \longleftrightarrow$
     $separation(\#\#M,\lambda x.\ M,\ [x]\ @\ rest\ @\ ms \models \varphi)$
  using *assms formula_add_params1*[*of* _ *2* _ _ [_,_]]
  unfolding *sep_body_fm_def separation_def* **by** *simp*

definition
  $ZF\_separation\_fm :: i \Rightarrow i$ **where**

```
  ZF_separation_fm(p) ≡ Forall^(pred(arity(p)))(sep_body_fm(p))

lemma ZF_separation_fm_type [TC]: p ∈ formula ⟹ ZF_separation_fm(p) ∈ formula
  by (simp add: ZF_separation_fm_def)

lemma sats_ZF_separation_fm_iff:
  assumes
    φ∈formula
  shows
  (M, [] ⊨ (ZF_separation_fm(φ)))
  ⟷
   (∀ env∈list(M). arity(φ) ≤ 1 #+ length(env) ⟶
      separation(##M,λx. M, [x] @ env ⊨ φ))
proof (intro iffI ballI impI)
  let ?n=Arith.pred(arity(φ))
  fix env
  assume M, [] ⊨ ZF_separation_fm(φ)
  assume arity(φ) ≤ 1 #+ length(env) env∈list(M)
  moreover from this
  have arity(φ) ≤ succ(length(env)) by simp
  then
  obtain some rest where some∈list(M) rest∈list(M)
    env = some @ rest length(some) = Arith.pred(arity(φ))
    using list_split[OF ‹arity(φ) ≤ succ(_)› ‹env∈_›] by force
  moreover from ‹φ∈_›
  have arity(φ) ≤ succ(Arith.pred(arity(φ)))
   using succpred_leI by simp
  moreover
  note assms
  moreover
  assume M, [] ⊨ ZF_separation_fm(φ)
  moreover from calculation
  have M, some ⊨ sep_body_fm(φ)
    using sats_nForall[of sep_body_fm(φ) ?n]
    unfolding ZF_separation_fm_def by simp
  ultimately
  show separation(##M, λx. M, [x] @ env ⊨ φ)
    unfolding ZF_separation_fm_def
    using sats_sep_body_fm[of φ [] M some]
      arity_sats_iff[of φ rest M [_] @ some]
      separation_cong[of ##M λx. M, Cons(x, some @ rest) ⊨ φ _ ]
    by simp
next — almost equal to the previous implication
  let ?n=Arith.pred(arity(φ))
  assume asm:∀ env∈list(M). arity(φ) ≤ 1 #+ length(env) ⟶
    separation(##M, λx. M, [x] @ env ⊨ φ)
  {
    fix some
    assume some∈list(M) length(some) = Arith.pred(arity(φ))
```

moreover
note ‹φ∈_›
moreover from *calculation*
have *arity*(φ) ≤ *1* #+ *length*(*some*)
using *le_trans*[*OF succpred_leI*] *succpred_leI* **by** *simp*
moreover from *calculation* **and** *asm*
have *separation*(##*M*, λ*x*. *M*, [*x*] @ *some* ⊨ φ) **by** *blast*
ultimately
have *M*, *some* ⊨ *sep_body_fm*(φ)
using *sats_sep_body_fm*[*of* φ [] *M some*]
arity_sats_iff [*of* φ _ *M* [_,_] @ *some*]
strong_replacement_cong[*of* ##*M* λ*x y*. *M*, *Cons*(*x*, *Cons*(*y*, *some* @ _)) ⊨ φ _]
by *simp*
}
with ‹φ∈_›
show *M*, [] ⊨ *ZF_separation_fm*(φ)
using *sats_nForall*[*of sep_body_fm*(φ) *?n*]
unfolding *ZF_separation_fm_def*
by *simp*
qed

12.2 The Axiom of Replacement, internalized

schematic_goal *sats_univalent_fm_auto*:
assumes

Q_iff_sats:⋀*x y z*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *z*∈*A* ⟹
Q(*x*,*z*) ⟷ (*A*,*Cons*(*z*,*Cons*(*y*,*Cons*(*x*,*env*))) ⊨ *Q1_fm*)
⋀*x y z*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *z*∈*A* ⟹
Q(*x*,*y*) ⟷ (*A*,*Cons*(*z*,*Cons*(*y*,*Cons*(*x*,*env*))) ⊨ *Q2_fm*)
and
asms: *nth*(*i*,*env*) = *B i* ∈ *nat env* ∈ *list*(*A*)
shows
univalent(##*A*,*B*,*Q*) ⟷ *A*,*env* ⊨ *?ufm*(*i*)
unfolding *univalent_def*
by (*insert asms*; (*rule sep_rules Q_iff_sats* | *simp*)+)

synthesize_notc *univalent_fm* **from_schematic** *sats_univalent_fm_auto*

lemma *univalent_fm_type* [*TC*]: *q1*∈ *formula* ⟹ *q2*∈*formula* ⟹ *i*∈*nat* ⟹
univalent_fm(*q2*,*q1*,*i*) ∈*formula*
by (*simp add*:*univalent_fm_def*)

lemma *sats_univalent_fm* :
assumes
Q_iff_sats:⋀*x y z*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *z*∈*A* ⟹
Q(*x*,*z*) ⟷ (*A*,*Cons*(*z*,*Cons*(*y*,*Cons*(*x*,*env*))) ⊨ *Q1_fm*)
⋀*x y z*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *z*∈*A* ⟹

$Q(x,y) \longleftrightarrow (A,Cons(z,Cons(y,Cons(x,env))) \models Q2\_fm)$
and
asms: $nth(i,env) = B$ $i \in nat$ $env \in list(A)$
shows
$A,env \models univalent\_fm(Q1\_fm,Q2\_fm,i) \longleftrightarrow univalent(\#\#A,B,Q)$
unfolding *univalent_fm_def* **using** *asms sats_univalent_fm_auto[OF Q_iff_sats]*
by *simp*

definition
swap_vars :: $i \Rightarrow i$ **where**
$swap\_vars(\varphi) \equiv$
$Exists(Exists(And(Equal(0,3),And(Equal(1,2),iterates(\lambda p.\ incr\_bv(p)`2\ ,\ 2,\ \varphi)))))$

lemma *swap_vars_type*[*TC*] :
$\varphi \in formula \Longrightarrow swap\_vars(\varphi) \in formula$
unfolding *swap_vars_def* **by** *simp*

lemma *sats_swap_vars* :
$[x,y]$ @ $env \in list(M) \Longrightarrow \varphi \in formula \Longrightarrow$
$M,\ [x,y]$ @ $env \models swap\_vars(\varphi) \longleftrightarrow M,[y,x]$ @ $env \models \varphi$
unfolding *swap_vars_def*
using *sats_incr_bv_iff* [*of* _ _ *M* _ [*y*,*x*]] **by** *simp*

definition
univalent_Q1 :: $i \Rightarrow i$ **where**
$univalent\_Q1(\varphi) \equiv incr\_bv1(swap\_vars(\varphi))$

definition
univalent_Q2 :: $i \Rightarrow i$ **where**
$univalent\_Q2(\varphi) \equiv incr\_bv(swap\_vars(\varphi))`0$

lemma *univalent_Qs_type* [*TC*]:
assumes $\varphi \in formula$
shows $univalent\_Q1(\varphi) \in formula$ $univalent\_Q2(\varphi) \in formula$
unfolding *univalent_Q1_def univalent_Q2_def* **using** *assms* **by** *simp_all*

lemma *sats_univalent_fm_assm*:
assumes
$x \in A$ $y \in A$ $z \in A$ $env \in list(A)$ $\varphi \in formula$
shows
$(A,\ ([x,z]$ @ $env) \models \varphi) \longleftrightarrow (A,\ Cons(z,Cons(y,Cons(x,env))) \models (univalent\_Q1(\varphi)))$
$(A,\ ([x,y]$ @ $env) \models \varphi) \longleftrightarrow (A,\ Cons(z,Cons(y,Cons(x,env))) \models (univalent\_Q2(\varphi)))$
unfolding *univalent_Q1_def univalent_Q2_def*
using
sats_incr_bv_iff[*of* _ _ *A* _ []] — simplifies iterates of $\lambda x.\ incr\_bv(x)\ `\ 0$
sats_incr_bv1_iff[*of* _ $Cons(x,env)$ *A z y*]
sats_swap_vars assms
by *simp_all*

```
definition
  rep_body_fm :: i ⇒ 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
  by (simp add: rep_body_fm_def)

lemmas ZF_replacement_simps = formula_add_params1[of φ 2 _ M [_,_] ]
  sats_incr_bv_iff[of _ _ M _ []] — simplifies iterates of λx. incr_bv(x) ` 0
  sats_incr_bv_iff[of _ _ M _ [_,_]]— simplifies λx. 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,λx y. M, [x,y] @ rest @ ms ⊨ φ)
  using assms ZF_replacement_simps
  unfolding rep_body_fm_def strong_replacement_def univalent_def
  unfolding univalent_fm_def univalent_Q1_def univalent_Q2_def
  by simp

definition
  ZF_replacement_fm :: i ⇒ i 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
  by (simp add: ZF_replacement_fm_def)

lemma sats_ZF_replacement_fm_iff:
  assumes
    φ∈formula
  shows
  (M, [] ⊨ (ZF_replacement_fm(φ)))
   ⟷
   (∀ env∈list(M). arity(φ) ≤ 2 #+ length(env) ⟶
      strong_replacement(##M,λx y. M,[x,y] @ env ⊨ φ))
proof (intro iffI ballI impI)
  let ?n=Arith.pred(Arith.pred(arity(φ)))
  fix env
 assume M, [] ⊨ ZF_replacement_fm(φ) arity(φ) ≤ 2 #+ length(env) env∈list(M)
  moreover from this
  have arity(φ) ≤ succ(succ(length(env))) by (simp)
```

```
  moreover from calculation
  have pred(arity(φ)) ≤ succ(length(env))
    using pred_mono[OF _ ‹arity(φ)≤succ(_)›] pred_succ_eq by simp
  moreover from calculation
  obtain some rest where some∈list(M) rest∈list(M)
    env = some @ rest length(some) = Arith.pred(Arith.pred(arity(φ)))
    using list_split[OF ‹pred(_) ≤ _› ‹env∈_›] by auto
  moreover
  note ‹φ∈_›
  moreover from this
  have arity(φ) ≤ succ(succ(Arith.pred(Arith.pred(arity(φ)))))
    using le_trans[OF succpred_leI] succpred_leI by simp
  moreover from calculation
  have M, some ⊨ rep_body_fm(φ)
    using sats_nForall[of rep_body_fm(φ) ?n]
    unfolding ZF_replacement_fm_def
    by simp
  ultimately
  show strong_replacement(##M, λx y. M, [x, y] @ env ⊨ φ)
    using sats_rep_body_fm[of φ [] M some]
      arity_sats_iff[of φ rest M [_,_] @ some]
      strong_replacement_cong[of ##M λx y. M, Cons(x, Cons(y, some @ rest))
⊨ φ _ ]
    by simp
next — almost equal to the previous implication
  let ?n=Arith.pred(Arith.pred(arity(φ)))
  assume asm:∀ env∈list(M). arity(φ) ≤ 2 #+ length(env) ⟶
    strong_replacement(##M, λx y. M, [x, y] @ env ⊨ φ)
  {
    fix some
    assume some∈list(M) length(some) = Arith.pred(Arith.pred(arity(φ)))
    moreover
    note ‹φ∈_›
    moreover from calculation
    have arity(φ) ≤ 2 #+ length(some)
      using le_trans[OF succpred_leI] succpred_leI by simp
    moreover from calculation and asm
    have strong_replacement(##M, λx y. M, [x, y] @ some ⊨ φ) by blast
    ultimately
    have M, some ⊨ rep_body_fm(φ)
    using sats_rep_body_fm[of φ [] M some]
      arity_sats_iff[of φ _ M [_,_] @ some]
      strong_replacement_cong[of ##M λx y. M, Cons(x, Cons(y, some @ _)) ⊨
φ _ ]
    by simp
  }
  with ‹φ∈_›
  show M, [] ⊨ ZF_replacement_fm(φ)
    using sats_nForall[of rep_body_fm(φ) ?n]
```

unfolding *ZF_replacement_fm_def*
by *simp*
qed

definition
ZF_inf :: *i* **where**
ZF_inf ≡ {*ZF_separation_fm*(*p*) . *p* ∈ *formula* } ∪ {*ZF_replacement_fm*(*p*) . *p* ∈ *formula* }

lemma *Un_subset_formula*: *A*⊆*formula* ∧ *B*⊆*formula* ⟹ *A*∪*B* ⊆ *formula*
by *auto*

lemma *ZF_inf_subset_formula* : *ZF_inf* ⊆ *formula*
unfolding *ZF_inf_def* **by** *auto*

definition
ZFC :: *i* **where**
ZFC ≡ *ZF_inf* ∪ *ZFC_fin*

definition
ZF :: *i* **where**
ZF ≡ *ZF_inf* ∪ *ZF_fin*

definition
ZF_minus_P :: *i* **where**
ZF_minus_P ≡ *ZF* - { *ZF_power_fm* }

lemma *ZFC_subset_formula*: *ZFC* ⊆ *formula*
by (*simp add*:*ZFC_def Un_subset_formula ZF_inf_subset_formula ZFC_fin_type*)

Satisfaction of a set of sentences

definition
satT :: [*i*,*i*] ⇒ *o* (_ ⊨ _ [*36*,*36*] *60*) **where**
A ⊨ Φ ≡ ∀φ∈Φ. (*A*,[] ⊨ φ)

lemma *satTI* [*intro*!]:
assumes ⋀φ. φ∈Φ ⟹ *A*,[] ⊨ φ
shows *A* ⊨ Φ
using *assms* **unfolding** *satT_def* **by** *simp*

lemma *satTD* [*dest*] :*A* ⊨ Φ ⟹ φ∈Φ ⟹ *A*,[] ⊨ φ
unfolding *satT_def* **by** *simp*

lemma *sats_ZFC_iff_sats_ZF_AC*:
(*N* ⊨ *ZFC*) ⟷ (*N* ⊨ *ZF*) ∧ (*N*, [] ⊨ *AC*)
unfolding *ZFC_def ZFC_fin_def ZF_def* **by** *auto*

lemma *M_ZF_iff_M_satT*: *M_ZF*(*M*) ⟷ (*M* ⊨ *ZF*)
proof

assume *M* ⊨ *ZF*
then
have *fin*: *upair_ax*(##*M*) *Union_ax*(##*M*) *power_ax*(##*M*)
extensionality(##*M*) *foundation_ax*(##*M*) *infinity_ax*(##*M*)
unfolding *ZF_def ZF_fin_def ZFC_fm_defs satT_def*
using *ZFC_fm_sats*[*of M*] **by** *simp_all*
{
fix *φ env*
assume *φ* ∈ *formula env*∈*list*(*M*)
moreover from ‹*M* ⊨ *ZF*›
have ∀ *p*∈*formula*. (*M*, [] ⊨ (*ZF_separation_fm*(*p*)))
∀ *p*∈*formula*. (*M*, [] ⊨ (*ZF_replacement_fm*(*p*)))
unfolding *ZF_def ZF_inf_def* **by** *auto*
moreover from *calculation*
have *arity*(*φ*) ≤ *succ*(*length*(*env*)) ⟹ *separation*(##*M*, *λx*. (*M*, *Cons*(*x*, *env*) ⊨ *φ*))
arity(*φ*) ≤ *succ*(*succ*(*length*(*env*))) ⟹ *strong_replacement*(##*M*,*λx y*. *sats*(*M*,*φ*,*Cons*(*x*,*Cons*(*y*, *env*))))
using *sats_ZF_separation_fm_iff sats_ZF_replacement_fm_iff* **by** *simp_all*
}
with *fin*
show *M_ZF*(*M*)
unfolding *M_ZF_def* **by** *simp*
next
assume ‹*M_ZF*(*M*)›
then
have *M* ⊨ *ZF_fin*
unfolding *M_ZF_def ZF_fin_def ZFC_fm_defs satT_def*
using *ZFC_fm_sats*[*of M*] **by** *blast*
moreover from ‹*M_ZF*(*M*)›
have ∀ *p*∈*formula*. (*M*, [] ⊨ (*ZF_separation_fm*(*p*)))
∀ *p*∈*formula*. (*M*, [] ⊨ (*ZF_replacement_fm*(*p*)))
unfolding *M_ZF_def* **using** *sats_ZF_separation_fm_iff*
sats_ZF_replacement_fm_iff **by** *simp_all*
ultimately
show *M* ⊨ *ZF*
unfolding *ZF_def ZF_inf_def* **by** *blast*
qed

end

13 Renaming of variables in internalized formulas

theory *Renaming*
imports
Nat_Miscellanea
ZF-Constructible.Formula
begin

```
lemma app_nm :
  assumes n∈nat m∈nat f∈n→m x ∈ nat
  shows f`x ∈ nat
proof(cases x∈n)
  case True
  then show ?thesis using assms in_n_in_nat apply_type by simp
next
  case False
  then show ?thesis using assms apply_0 domain_of_fun by simp
qed
```

13.1 Renaming of free variables

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

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

using *that lamE*[*of x m* ∪ *p* _ *x* ∈ (*m* ∪ *p*) × (*n* ∪ *q*)] *D* **unfolding** *union_fun_def*
by *auto*
then have *B*: *?h* ⊆ (*m* ∪ *p*) × (*n* ∪ *q*) **..**
have *m* ∪ *p* ⊆ *domain*(*?h*)
unfolding *union_fun_def* **using** *domain_lam* **by** *simp*
with *A B*
show *?thesis* **using** *Pi_iff* [*THEN iffD2*] **by** *simp*
qed

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

lemma *id_fn_type* :
assumes *n* ∈ *nat*
shows *id*(*n*) ∈ *n* → *n*

unfolding *id_def* **using** ‹$n \in nat$› **by** *simp*

lemma *id_fn_action*:
assumes $n \in nat$ $env \in list(M)$
shows $\bigwedge j$. $j < n \Longrightarrow nth(j,env) = nth(id(n)`j,env)$
proof -
show $nth(j,env) = nth(id(n)`j,env)$ **if** $j < n$ **for** j **using** *that* ‹$n \in nat$› *ltD* **by** *simp*
qed

definition
sum :: [*i*,*i*,*i*,*i*,*i*] ⇒ *i* **where**
$sum(f,g,m,n,p) \equiv \lambda j \in m\#+p$. *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 $sum(f,g,m,n,p)`x = f`x$
proof -
from ‹$m \in nat$›
have $m \le m\#+p$
using *add_le_self*[*of m*] **by** *simp*
with *assms*
have $x \in m\#+p$
using *ltI*[*of x m*] *lt_trans2*[*of x m m#+p*] *ltD* **by** *simp*
from *assms*
have $x<m$
using *ltI* **by** *simp*
with ‹$x \in m\#+p$›
show *?thesis* **unfolding** *sum_def* **by** *simp*
qed

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 $sum(f,g,m,n,p)`x = g`(x\#-m)\#+n$
proof -
from *assms*
have $x \in nat$
using *in_n_in_nat*[*of m#+p*] *ltD*
by *simp*
with *assms*
have $\neg\ x<m$
using *not_lt_iff_le*[*THEN iffD2*] **by** *simp*
from *assms*
have $x \in m\#+p$
using *ltD* **by** *simp*
with ‹$\neg\ x<m$›

```
  show ?thesis unfolding sum_def by simp
qed


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

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

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

then show *?thesis* **using** *1 that* **by** *simp*
next
case *False*
with ‹*m*∈*nat*› **have** *m*≤*x* **using** *not_lt_iff_le that in_n_in_nat*[*of m*#+*p*] **by** *simp*
then show *?thesis* **using** *2 that* **by** *simp*
qed
have *A*:*function*(*?h*) **unfolding** *sum_def* **using** *function_lam* **by** *simp*
have *x*∈ (*m* #+ *p*) × (*n* #+ *q*) **if** *x*∈ *?h* **for** *x*
using *that lamE*[*of x m*#+*p* _ *x* ∈ (*m* #+ *p*) × (*n* #+ *q*)] *D* **unfolding** *sum_def*
by *auto*
then have *B*:*?h* ⊆ (*m* #+ *p*) × (*n* #+ *q*) **..**
have *m* #+ *p* ⊆ *domain*(*?h*)
unfolding *sum_def* **using** *domain_lam* **by** *simp*
with *A B*
show *?thesis* **using** *Pi_iff* [*THEN iffD2*] **by** *simp*
qed

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

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

lemma *sum_action_id* :
assumes
env ∈ *list*(*M*)
env' ∈ *list*(*M*)
f ∈ *length*(*env*)→*length*(*env'*)
env1 ∈ *list*(*M*)
⋀ *i* . *i* < *length*(*env*) ⟹ *nth*(*i*,*env*) = *nth*(*f*‘*i*,*env'*)

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

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

definition
sum_id :: [*i*,*i*] ⇒ *i* **where**
sum_id(*m*,*f*) ≡ *sum*(λ*x*∈*1*.*x*,*f*,*1*,*1*,*m*)

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

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

simp_all add:ltI,simp_all add: app_nm in_n_in_nat)

lemma *sum_id_tc_aux* :
$p \in nat \Longrightarrow q \in nat \Longrightarrow f \in p \to q \Longrightarrow sum\_id(p,f) \in 1\#{+}p \to 1\#{+}q$
by (*unfold sum_id_def,rule sum_type,simp_all*)

lemma *sum_id_tc* :
$n \in nat \Longrightarrow m \in nat \Longrightarrow f \in n \to m \Longrightarrow sum\_id(n,f) \in succ(n) \to succ(m)$
by(*rule ssubst*[*of succ(n) → succ(m) 1#+n → 1#+m*],
simp,rule sum_id_tc_aux,simp_all)

13.2 Renaming of formulas

consts *ren* :: *i⇒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* : $l \in nat \Longrightarrow Member(x,y) \in formula \Longrightarrow arity(Member(x,y)) \le l \Longrightarrow x \in l$
by(*simp,rule subsetD,rule le_imp_subset,assumption,simp*)
lemma *arity_memr* : $l \in nat \Longrightarrow Member(x,y) \in formula \Longrightarrow arity(Member(x,y)) \le l \Longrightarrow y \in l$
by(*simp,rule subsetD,rule le_imp_subset,assumption,simp*)
lemma *arity_eql* : $l \in nat \Longrightarrow Equal(x,y) \in formula \Longrightarrow arity(Equal(x,y)) \le l \Longrightarrow x \in l$
by(*simp,rule subsetD,rule le_imp_subset,assumption,simp*)
lemma *arity_eqr* : $l \in nat \Longrightarrow Equal(x,y) \in formula \Longrightarrow arity(Equal(x,y)) \le l \Longrightarrow y \in l$
by(*simp,rule subsetD,rule le_imp_subset,assumption,simp*)
lemma *nand_ar1* : $p \in formula \Longrightarrow q{\in}formula \Longrightarrow arity(p) \le arity(Nand(p,q))$
by (*simp,rule Un_upper1_le,simp+*)
lemma *nand_ar2* : $p \in formula \Longrightarrow q{\in}formula \Longrightarrow arity(q) \le arity(Nand(p,q))$
by (*simp,rule Un_upper2_le,simp+*)

lemma *nand_ar1D* : $p \in formula \Longrightarrow q{\in}formula \Longrightarrow arity(Nand(p,q)) \le n \Longrightarrow arity(p) \le n$
by(*auto simp add: le_trans*[*OF Un_upper1_le*[*of arity(p) arity(q)*]])
lemma *nand_ar2D* : $p \in formula \Longrightarrow q{\in}formula \Longrightarrow arity(Nand(p,q)) \le n \Longrightarrow arity(q) \le n$

by(*auto simp add*: *le_trans*[*OF Un_upper2_le*[*of arity*(*p*) *arity*(*q*)]])

lemma *ren_tc* : $p \in formula \Longrightarrow$
($\bigwedge$ $n\ m\ f\ .\ n \in nat \Longrightarrow m \in nat \Longrightarrow f \in n \to m \Longrightarrow ren(p)`n`m`f \in formula$)
by (*induct set*:*formula*,*auto simp add*: *app_nm sum_id_tc*)

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

lemma *arity_forallE* : $p \in formula \Longrightarrow m \in nat \Longrightarrow arity(Forall(p)) \leq m \Longrightarrow arity(p) \leq succ(m)$

by(*rule_tac n*=*arity*(*p*) **in** *natE*,*erule arity_type*,*simp*+)

lemma *env_coincidence_sum_id* :
  assumes $m \in nat$ $n \in nat$
    $\varrho \in list(A)$ $\varrho' \in list(A)$
    $f \in n \to m$
    $\bigwedge i . i < n \Longrightarrow nth(i,\varrho) = nth(f`i,\varrho')$
    $a \in A$ $j \in succ(n)$
  shows $nth(j,Cons(a,\varrho)) = nth(sum\_id(n,f)`j,Cons(a,\varrho'))$
proof -
  let *?g*=*sum_id*(*n*,*f*)
  have $succ(n) \in nat$ **using** ‹$n\in nat$› **by** *simp*
  then have $j \in nat$ **using** ‹$j\in succ(n)$› *in_n_in_nat* **by** *blast*
  then have $nth(j,Cons(a,\varrho)) = nth(?g`j,Cons(a,\varrho'))$
  proof (*cases rule*:*natE*[*OF* ‹$j\in nat$›])
    case *1*
    then show *?thesis* **using** *assms sum_id0* **by** *simp*
  next
    case (*2 i*)
    with ‹$j\in succ(n)$› **have** $succ(i)\in succ(n)$ **by** *simp*
    with ‹$n\in nat$› **have** $i \in n$ **using** *nat_succD assms* **by** *simp*
    have $f`i\in m$ **using** ‹$f\in n\to m$› *apply_type* ‹$i\in n$› **by** *simp*
    then have $f`i \in nat$ **using** *in_n_in_nat* ‹$m\in nat$› **by** *simp*
    have $nth(succ(i),Cons(a,\varrho)) = nth(i,\varrho)$ **using** ‹$i\in nat$› **by** *simp*
    also have ... $= nth(f`i,\varrho')$ **using** *assms* ‹$i\in n$› *ltI* **by** *simp*
    also have ... $= nth(succ(f`i),Cons(a,\varrho'))$ **using** ‹$f`i\in nat$› **by** *simp*
    also have ... $= nth(?g`succ(i),Cons(a,\varrho'))$
     using *assms sum_idS*[*OF* ‹$n\in nat$› ‹$m\in nat$› ‹$f\in n\to m$› ‹$i \in n$›] *cases* **by** *simp*
    finally have $nth(succ(i),Cons(a,\varrho)) = nth(?g`succ(i),Cons(a,\varrho'))$ **.**
    then show *?thesis* **using** ‹$j=succ(i)$› **by** *simp*
  qed
  then show *?thesis* **.**
qed

lemma *sats_iff_sats_ren* :
  fixes φ
  assumes $\varphi \in formula$
  shows $[\![n \in nat ; m \in nat ; \varrho \in list(M) ; \varrho' \in list(M) ; f \in n \to m ;$
      $arity(\varphi) \le n ;$
      $\bigwedge i . i < n \Longrightarrow nth(i,\varrho) = nth(f`i,\varrho')]\!] \Longrightarrow$
    $sats(M,\varphi,\varrho) \longleftrightarrow sats(M,ren(\varphi)`n`m`f,\varrho')$
  using ‹$\varphi \in formula$›
proof(*induct* φ *arbitrary*:n m ϱ ϱ' f)
  case (*Member x y*)
  have $ren(Member(x,y))`n`m`f = Member(f`x,f`y)$ **using** *Member assms arity_type*
by *force*
  moreover
  have $x \in n$ **using** *Member arity_memI* **by** *simp*
  moreover

have $y \in n$ **using** *Member arity_memr* **by** *simp*
ultimately
show *?case* **using** *Member ltI* **by** *simp*
next
case (*Equal x y*)
have $ren(Equal(x,y))`n`m`f = Equal(f`x,f`y)$ **using** *Equal assms arity_type* **by** *force*
moreover
have $x \in n$ **using** *Equal arity_eql* **by** *simp*
moreover
have $y \in n$ **using** *Equal arity_eqr* **by** *simp*
ultimately show *?case* **using** *Equal ltI* **by** *simp*
next
case (*Nand p q*)
have $ren(Nand(p,q))`n`m`f = Nand(ren(p)`n`m`f,ren(q)`n`m`f)$ **using** *Nand* **by** *simp*
moreover
have $arity(p) \le n$ **using** *Nand nand_ar1D* **by** *simp*
moreover from *this*
have $i \in arity(p) \Longrightarrow i \in n$ **for** i **using** *subsetD[OF le_imp_subset[OF* ‹$arity(p) \le n$›*]]* **by** *simp*
moreover from *this*
have $i \in arity(p) \Longrightarrow nth(i,\varrho) = nth(f`i,\varrho')$ **for** i **using** *Nand ltI* **by** *simp*
moreover from *this*
have $sats(M,p,\varrho) \longleftrightarrow sats(M,ren(p)`n`m`f,\varrho')$ **using** ‹$arity(p)\le n$› *Nand* **by** *simp*
have $arity(q) \le n$ **using** *Nand nand_ar2D* **by** *simp*
moreover from *this*
have $i \in arity(q) \Longrightarrow i \in n$ **for** i **using** *subsetD[OF le_imp_subset[OF* ‹$arity(q) \le n$›*]]* **by** *simp*
moreover from *this*
have $i \in arity(q) \Longrightarrow nth(i,\varrho) = nth(f`i,\varrho')$ **for** i **using** *Nand ltI* **by** *simp*
moreover from *this*
have $sats(M,q,\varrho) \longleftrightarrow sats(M,ren(q)`n`m`f,\varrho')$ **using** *assms* ‹$arity(q)\le n$› *Nand* **by** *simp*
ultimately
show *?case* **using** *Nand* **by** *simp*
next
case (*Forall p*)
have $0{:}ren(Forall(p))`n`m`f = Forall(ren(p)`succ(n)`succ(m)`sum\_id(n,f))$
using *Forall* **by** *simp*
have $1{:}sum\_id(n,f) \in succ(n) \to succ(m)$ (**is** $?g \in \_$) **using** *sum_id_tc Forall* **by** *simp*
then have 2: $arity(p) \le succ(n)$
using *Forall le_trans[of _ succ(pred(arity(p)))] succpred_leI* **by** *simp*
have $succ(n)\in nat$ $succ(m)\in nat$ **using** *Forall* **by** *auto*
then have $A{:}\bigwedge j .j < succ(n) \Longrightarrow nth(j,\ Cons(a,\ \varrho)) = nth(?g`j,\ Cons(a,\ \varrho'))$ **if** $a\in M$ **for** a
using *that env_coincidence_sum_id Forall ltD* **by** *force*

```
  have
      sats(M,p,Cons(a,ϱ)) ⟷ sats(M,ren(p)`succ(n)`succ(m)`?g,Cons(a,ϱ')) if
a∈M for a
  proof -
    have C:Cons(a,ϱ) ∈ list(M) Cons(a,ϱ')∈list(M) using Forall that by auto
    have sats(M,p,Cons(a,ϱ)) ⟷ sats(M,ren(p)`succ(n)`succ(m)`?g,Cons(a,ϱ'))
        using Forall(2)[OF ‹succ(n)∈nat› ‹succ(m)∈nat› C(1) C(2) 1 2 A[OF
‹a∈M›]] by simp
    then show ?thesis .
  qed
  then show ?case using Forall 0 1 2 by simp
qed

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

begin

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

  fun renaming_def mk_ren name from to ctxt =
    let val to = to |> Syntax.read_term ctxt
        val from = from |> Syntax.read_term ctxt
        val (tc_lemma,action_lemma,fvs,r) = mk_ren from to ctxt
            val (tc_lemma,action_lemma) = (fix_vars tc_lemma fvs ctxt , fix_vars
action_lemma fvs ctxt)
        val ren_fun_name = Binding.name (name ^ _fn)
        val ren_fun_def =  Binding.name (name ^ _fn_def)
        val ren_thm = Binding.name (name ^ _thm)
    in
```

```
    Local_Theory.note  ((ren_thm, []), [tc_lemma,action_lemma]) ctxt |> snd |>
     Local_Theory.define ((ren_fun_name, NoSyn), ((ren_fun_def, []), r)) |> snd

  end;
›
```

ML‹
```
local

  val ren_parser = Parse.position (Parse.string --
     (Parse.$$$ src |-- Parse.string --| Parse.$$$ tgt -- Parse.string));

  val _ =
   Outer_Syntax.local_theory command_keyword‹rename› ML setup for synthetic
definitions
     (ren_parser >> (fn ((name,(from,to)),_) => renaming_def sum_rename name
from to ))

  val _ =
    Outer_Syntax.local_theory command_keyword‹simple_rename› ML setup for
synthetic definitions
     (ren_parser >> (fn ((name,(from,to)),_) => renaming_def ren_thm name from
to ))

in
end
›
```
end

14 Automatic relativization of terms.

theory *Relativization*
  imports *ZF-Constructible.Formula*
    ZF-Constructible.Relative
    ZF-Constructible.Datatype_absolute
  keywords
    relativize :: *thy_decl* % *ML*
    and
    relativize_tm :: *thy_decl* % *ML*
    and
    reldb_add :: *thy_decl* % *ML*

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

setup⟨*Absoluteness.setup*⟩
lemmas *relative_abs* =
M_trans.empty_abs
M_trans.pair_abs
M_trivial.cartprod_abs
M_trans.union_abs
M_trans.inter_abs
M_trans.setdiff_abs
M_trans.Union_abs
M_trivial.cons_abs

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

M_trivial.nat_case_abs

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

M_basic.composition_abs
M_trans.restriction_abs
M_trans.Inter_abs
M_trivial.is_funspace_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

lemmas *datatype_abs* =
M_datatypes.list_N_abs
M_datatypes.list_abs
M_datatypes.formula_N_abs

```
  M_datatypes.formula_abs
  M_eclose.is_eclose_n_abs
  M_eclose.eclose_abs
  M_datatypes.length_abs
  M_datatypes.nth_abs
  M_trivial.Member_abs
  M_trivial.Equal_abs
  M_trivial.Nand_abs
  M_trivial.Forall_abs
  M_datatypes.depth_abs
  M_datatypes.formula_case_abs
```

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

ML‹

```
signature Relativization =
  sig
    structure Data: GENERIC_DATA
    val Rel_add: attribute
    val Rel_del: attribute
     val add_rel_const : string -> term -> term -> Proof.context -> Data.T ->
Data.T
    val add_constant : string -> string -> Proof.context -> Proof.context
    val db: (term * term) list
    val init_db : (term * term) list -> theory -> theory
    val get_db : Proof.context -> (term * term) list
     val relativ_fm: term -> (term * term) list -> (term * (term * term)) list *
Proof.context -> term -> term
     val relativ_tm: term -> (term * term) list -> (term * (term * term)) list *
Proof.context -> term -> term * (term * (term * term)) list * Proof.context
    val read_new_const : Proof.context -> string -> term
     val relativ_tm_frm': term -> (term * term) list -> Proof.context -> term ->
term option * term
   val relativize_def: bstring -> string -> Position.T -> Proof.context -> Proof.context
   val relativize_tm: bstring -> string -> Position.T -> Proof.context -> Proof.context
  end

structure Relativization : Relativization = struct
type relset = { db_rels: (term * term) list};

  (* relativization db of relation constructors *)
  val db =
          [ (@{const relation}, @{const Relative.is_relation})
          , (@{const function}, @{const Relative.is_function})
          , (@{const mem}, @{const mem})
          , (@{const True}, @{const True})
          , (@{const False}, @{const False})
          , (@{const Memrel}, @{const membership})
```

```
          , (@{const trancl}, @{const tran_closure})
          , (@{const IFOL.eq(i)}, @{const IFOL.eq(i)})
          , (@{const Subset}, @{const Relative.subset})
          , (@{const quasinat}, @{const Relative.is_quasinat})
          , (@{const apply}, @{const Relative.fun_apply})
          , (@{const Upair}, @{const Relative.upair})
          ]

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

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

(∗ instantiated with diferent types than lookup_tm ∗)
val lookup_rel=  AList.lookup (op aconv)

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

(∗ generic data ∗)
structure Data = Generic_Data
(
  type T = relset;
  val empty = {db_rels = []}; (∗ Should we initialize this outside this file? ∗)
  val extend = I;
  fun merge ({db_rels = db},  {db_rels = db'}) =
     {db_rels = AList.join (op aconv) (K #1) (db', db)};
);

fun init_db db = Context.theory_map (Data.put {db_rels = db })
fun get_db thy = let val db = Data.get (Context.Proof thy)
                 in #db_rels db
                 end

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

fun add_rel_const thm_name c t ctxt (rs as {db_rels = db}) =
  case lookup_rel db c of
    SOME t' =>
    (warning (Ignoring duplicate relativization rule ^
             const_name c ^   ^ Syntax.string_of_term ctxt t ^
             ( ^  Syntax.string_of_term ctxt t' ^  in   ^ thm_name ^ )); rs)
  | NONE => {db_rels = (c, t) :: db};

fun get_consts thm =
  let val (c_rel, rhs) = Thm.concl_of thm |> Utils.dest_trueprop |>
```

```
                    Utils.dest_iff_tms |>> head_of
  in case try Utils.dest_eq_tms rhs of
      SOME tm => (c_rel, tm |> #2 |> head_of)
    | NONE => (c_rel, rhs |> Utils.dest_mem_tms |> #2 |> head_of)
  end

fun add_rule ctxt thm rs =
  let val (c_rel,c_abs) = get_consts thm
      val thm_name = Proof_Context.pretty_fact ctxt ( , [thm]) |> Pretty.string_of
  in add_rel_const thm_name c_abs c_rel ctxt rs
end


fun add_constant rel abs thy =
  let val c_abs = read_new_const thy abs
      val c_rel = read_new_const thy rel
  in Local_Theory.target (Context.proof_map
    (Data.map (fn db => {db_rels = (c_rel,c_abs) :: #db_rels db}))) thy
 end

fun del_rel_const c (rs as {db_rels = db}) =
  case lookup_rel db c of
    SOME c' =>
    { db_rels = AList.delete (fn (_,b) => b = c) c' db}
  | NONE => (warning (The constant  ^
           const_name c ^  doesn't have a relativization rule associated); rs) ;

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


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

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

(* *)


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

(* Produces a relativized existential quantification of the term t *)
fun rex p t (Free v) = @{const rex} $ p $ lambda (Free v) t
  | rex _ t (Bound _) = t
```

```
  | rex _ t tm = raise TERM (rex shouldn't handle this.,[tm,t])

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

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

fun relativ_tms _ _ _ ctxt' [] = ([], [], ctxt')
  | relativ_tms pred rel_db rs' ctxt' (u :: us) =
      let val (w_u, rs_u, ctxt_u) = relativ_tm pred rel_db (rs', ctxt') u
          val (w_us, rs_us, ctxt_us) = relativ_tms pred rel_db rs_u ctxt_u us
      in (w_u :: w_us, join_tm (rs_u , rs_us), ctxt_us)
      end
and
    (* The result of the relativization of a term is a triple consisting of
      a. the relativized term (it can be a free or a bound variable but also a Collect)
      b. a list of (term * (term, term)), taken as a map, which is used
         to reuse relativization of different occurrences of the same term. The
         first element is the original term, the second its relativized version,
         and the last one is the predicate corresponding to it.
      c. the resulting context of created variables.
    *)
    relativ_tm pred rel_db (rs,ctxt) tm =
      let
      (* relativization of a fully applied constant *)
      fun mk_rel_const c args abs_args ctxt =
        case lookup_rel rel_db c of
          SOME p =>
            let val frees = fold_aterms (fn t => if is_Free t then cons t else I) p []
                val args' = List.filter (not o Utils.inList frees) args
                val (v, ctxt1) = Variable.variant_fixes [] ctxt |>> var_i o hd
                val r_tm = list_comb (p, pred :: args' @ abs_args @ [v])
            in (v, r_tm, ctxt1)
```

```
        end
      | NONE => raise TERM (Constant ^ const_name c ^ is not present in the db. , nil)

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

    (* relativization of non dependent product and sum *)
    fun relativ_app_no_dep tm c t t' =
       if loose_bvar1 (t', 0)
       then raise TERM(A dependency was found when trying to relativize, [tm])
       else relativ_app tm [] c [t, t']

    fun go (Var _) = raise TERM (Var: Is this possible?,[])
      | go (@{const Replace} $ t $ pc) =
        let val pc' = relativ_fm pred rel_db (rs,ctxt) pc
        in relativ_app tm [pc'] @{const Replace} [t]
        end
      | go (@{const Collect} $ t $ pc) =
        let val pc' = relativ_fm pred rel_db (rs,ctxt) pc
        in relativ_app tm [pc'] @{const Collect} [t]
        end
      | go (tm as @{const Sigma} $ t $ Abs (_,_,t')) =
        relativ_app_no_dep tm @{const Sigma} t t'
      | go (tm as @{const Pi} $ t $ Abs (_,_,t')) =
        relativ_app_no_dep tm @{const Pi} t t'
      | go (tm as @{const bool_of_o} $ t) =
        let val t' = relativ_fm pred rel_db (rs,ctxt) t
        in relativ_app tm [t'] @{const bool_of_o} []
        end
      | go (tm as Const _) = relativ_app tm [] tm []
      | go (tm as _ $ _) = strip_comb tm |> uncurry (relativ_app tm [])
      | go tm = (tm, update_tm (tm,(tm,tm)) rs, ctxt)

    (* we first check if the term has been already relativized as a variable *)
    in case lookup_tm rs tm of
        NONE => go tm
      | SOME (w, _) => (w, rs, ctxt)
    end
and
```

```
relativ_fm pred rel_db (rs,ctxt) fm =
let

(* relativization of a fully applied constant *)
fun relativ_app ctxt c args = case lookup_rel rel_db c of
  SOME p =>
    let (* flag indicates whether the relativized constant is absolute or not. *)
      val flag = not (exists (curry op aconv c) absolute_rels)
      val frees = fold_aterms (fn t => if is_Free t then cons t else I) p []
      val (args, rs_ts, _) = relativ_tms pred rel_db rs ctxt args
      val args' = List.filter (not o Utils.inList frees) args
      val tm = list_comb (p, if flag then pred :: args' else args')
     in close_rel_tm pred (SOME tm) NONE rs_ts
     end
 | NONE   => raise TERM (Constant  ^ const_name c ^  is not present in the db. , nil)

(* Handling of bounded quantifiers. *)
fun bquant ctxt quant conn dom pred =
  let val (v,pred') = Term.dest_abs pred |>> var_i
  in
    go ctxt (quant $ lambda v (conn $ (@{const mem} $ v $ dom) $ pred'))
  end
and
(* We could share relativizations of terms occuring inside propositional connectives. *)
     go ctxt (@{const IFOL.conj} $ f $ f') = @{const IFOL.conj} $ go ctxt f $ go ctxt f'
  | go ctxt (@{const IFOL.disj} $ f $ f') = @{const IFOL.disj} $ go ctxt f $ go ctxt f'
  | go ctxt (@{const IFOL.Not} $ f) = @{const IFOL.Not} $ go ctxt f
  | go ctxt (@{const IFOL.iff} $ f $ f') = @{const IFOL.iff} $ go ctxt f $ go ctxt f'
  | go ctxt (@{const IFOL.imp} $ f $ f') = @{const IFOL.imp} $ go ctxt f $ go ctxt f'
  | go ctxt (@{const IFOL.All(i)} $ f) = @{const OrdQuant.rall} $ pred $ go ctxt f
  | go ctxt (@{const IFOL.Ex(i)} $ f) = @{const OrdQuant.rex} $ pred $ go ctxt f
  | go ctxt (@{const Bex} $ f $ Abs p) = bquant ctxt @{const Ex(i)} @{const IFOL.conj} f p
  | go ctxt (@{const Ball} $ f $ Abs p) = bquant ctxt @{const All(i)} @{const IFOL.imp} f p
  | go ctxt (Const c) = relativ_app ctxt (Const c) []
  | go ctxt (tm as _ $ _) = strip_comb tm |> uncurry (relativ_app ctxt)
  | go ctxt (Abs body) =
    let
      val (v, t) = Term.dest_abs body
     val new_ctxt = if Variable.is_fixed ctxt v then ctxt else #2 (Variable.add_fixes
```

```
[v] ctxt)
      in
        lambda (var_i v) (go new_ctxt t)
      end
    | go _ t = raise TERM (Relativization of formulas cannot handle this case.,[t])
  in go ctxt fm
  end


fun relativ_tm_frm' cls_pred db ctxt tm =
  let val ty = fastype_of tm
  in case ty of
       @{typ i} =>
         let val (w, rs, _) =  relativ_tm cls_pred db ([],ctxt) tm
         in (SOME w, close_rel_tm cls_pred NONE (SOME w) rs)
         end
     | @{typ o} => (NONE, relativ_fm cls_pred db ([],ctxt) tm)
     | ty' => raise TYPE (We can relativize only terms of types i and o,[ty'],[tm])
  end

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

fun relativize_def def_name thm_ref pos lthy =
  let
    val ctxt = lthy
    val (vars,tm,ctxt1) = Utils.thm_concl_tm ctxt (thm_ref ^ _def)
    val ({db_rels = db'}) = Data.get (Context.Proof lthy)
    val tm = tm |> #2 o Utils.dest_eq_tms' o Utils.dest_trueprop
    val (cls_pred, ctxt1) = Variable.variant_fixes [N] ctxt1 |>> var_io o hd
    val (v,t) = relativ_tm_frm' cls_pred db' ctxt1 tm
    val t_vars = Term.add_free_names tm []
    val vs' = List.filter (#1 #> #1 #> #1 #> Utils.inList t_vars) vars
    val vs = cls_pred :: map (Thm.term_of o #2) vs' @ the_list v
    val at = List.foldr (uncurry lambda) t vs
    val abs_const = read_const lthy (lname lthy thm_ref)
in
   lthy |>
   Local_Theory.define ((Binding.name def_name, NoSyn),
                        ((Binding.name (def_name ^ _def), []), at)) |>>
   (#2 #> (fn (s,t) => (s,[t]))) |> Utils.display theorem pos |>
   Local_Theory.target (
       fn ctxt' => Context.proof_map
        (Data.map (add_rel_const  abs_const (read_new_const ctxt' def_name) ctxt'))
ctxt')
  end

fun relativize_tm def_name term pos lthy =
  let
    val ctxt = lthy
```

```
    val (cls_pred, ctxt1) = Variable.variant_fixes [N] ctxt |>> var_io o hd
    val tm = Syntax.read_term ctxt1 term
    val ({db_rels = db'}) = Data.get (Context.Proof lthy)
    val vs' = Variable.add_frees ctxt1 tm []
   fun update_ctxt (v,_) c = if Variable.is_fixed c v then c else #2 (Variable.add_fixes
[v] c)
    val ctxt2 = fold update_ctxt vs' ctxt1
    val (v,t) = relativ_tm_frm' cls_pred db' ctxt2 tm
    val vs = cls_pred :: map Free vs' @ the_list v
    val at = List.foldr (uncurry lambda) t vs
in
   lthy |>
    Local_Theory.define ((Binding.name def_name, NoSyn),
                        ((Binding.name (def_name ^ _def), []), at)) |>>
  (#2 #> (fn (s,t) => (s,[t]))) |> Utils.display theorem pos
  end


end
›
```

ML‹

```
local
  val relativize_parser =
       Parse.position (Parse.string -- Parse.string);

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



  val _ =
     Outer_Syntax.local_theory command_keyword‹relativize› ML setup for rela-
tivizing definitions
       (relativize_parser >> (fn ((bndg,thm),pos) =>
          Relativization.relativize_def thm bndg pos))

  val _ =
      Outer_Syntax.local_theory command_keyword‹relativize_tm› ML setup for
relativizing definitions
       (relativize_parser >> (fn ((bndg,term),pos) =>
          Relativization.relativize_tm term bndg pos))
val _ =
  Theory.setup
    (Attrib.setup binding‹Rel› (Attrib.add_del Relativization.Rel_add Relativiza-
tion.Rel_del)
      declaration of relativization rule) ;
```

in
end
⟩
setup⟨*Relativization.init_db Relativization.db* ⟩

declare *relative_abs*[*Rel*]

declare *datatype_abs*[*Rel*]

end

15 Names and generic extensions

theory *Names*
  imports
    Forcing_Data
    Interface
    Recursion_Thms
    Relativization
    Synthetic_Definition
begin

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

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

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

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

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

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

lemma *SepReplace_pred_implies* :
  ∀*x*. *Q*(*x*)⟶ *b*(*x*) = *b*′(*x*)⟹ {*b*(*x*) .. *x*∈*A*, *Q*(*x*)}={*b*′(*x*) .. *x*∈*A*, *Q*(*x*)}
  by (*force simp add*:*SepReplace_def*)

15.1 The well-founded relation *ed*

```
lemma eclose_sing : x ∈ eclose(a) ⟹ x ∈ eclose({a})
  by(rule subsetD[OF mem_eclose_subset],simp+)

lemma ecloseE :
  assumes  x ∈ eclose(A)
  shows  x ∈ A ∨ (∃ B ∈ A . x ∈ eclose(B))
  using assms
proof (induct rule:eclose_induct_down)
  case (1 y)
  then
  show ?case
    using arg_into_eclose by auto
next
  case (2 y z)
  from ‹y ∈ A ∨ (∃ B∈A. y ∈ eclose(B))›
  consider (inA) y ∈ A | (exB) (∃ B∈A. y ∈ eclose(B))
    by auto
  then show ?case
  proof (cases)
    case inA
    then
    show ?thesis using 2 arg_into_eclose by auto
  next
    case exB
    then obtain B where y ∈ eclose(B) B∈A
      by auto
    then
    show ?thesis using 2 ecloseD[of y B z] by auto
  qed
qed

lemma eclose_singE : x ∈ eclose({a}) ⟹ x = a ∨ x ∈ eclose(a)
  by(blast dest: ecloseE)

lemma in_eclose_sing :
  assumes x ∈ eclose({a}) a ∈ eclose(z)
  shows x ∈ eclose({z})
proof -
  from ‹x∈eclose({a})›
  consider (eq) x=a | (lt) x∈eclose(a)
    using eclose_singE by auto
  then
  show ?thesis
    using eclose_sing mem_eclose_trans assms
    by (cases, auto)
qed

lemma in_dom_in_eclose :
```

assumes $x \in domain(z)$
shows $x \in eclose(z)$
proof -
from *assms*
obtain y **where** $\langle x,y\rangle \in z$
unfolding *domain_def* **by** *auto*
then
show *?thesis*
unfolding *Pair_def*
using *ecloseD*[*of* {*x*,*x*}] *ecloseD*[*of* {{*x*,*x*},{*x*,*y*}}] *arg_into_eclose*
by *auto*
qed

term*ed* is the well-founded relation on which *val* is defined.

definition
$ed :: [i,i] \Rightarrow o$ **where**
$ed(x,y) \equiv x \in domain(y)$

definition
$edrel :: i \Rightarrow i$ **where**
$edrel(A) \equiv Rrel(ed,A)$

lemma *edI*[*intro*!]: $t \in domain(x) \Longrightarrow ed(t,x)$
unfolding *ed_def* **.**

lemma *edD*[*dest*!]: $ed(t,x) \Longrightarrow t \in domain(x)$
unfolding *ed_def* **.**

lemma *rank_ed*:
assumes $ed(y,x)$
shows $succ(rank(y)) \leq rank(x)$
proof
from *assms*
obtain p **where** $\langle y,p\rangle \in x$ **by** *auto*
moreover
obtain s **where** $y \in s$ $s \in \langle y,p\rangle$ **unfolding** *Pair_def* **by** *auto*
ultimately
have $rank(y) < rank(s)$ $rank(s) < rank(\langle y,p\rangle)$ $rank(\langle y,p\rangle) < rank(x)$
using *rank_lt* **by** *blast+*
then
show $rank(y) < rank(x)$
using *lt_trans* **by** *blast*
qed

lemma *edrel_dest* [*dest*]: $x \in edrel(A) \Longrightarrow \exists\ a \in A.\ \exists\ b \in A.\ x = \langle a,b\rangle$
by(*auto simp add*:*ed_def edrel_def Rrel_def*)

lemma *edrelD* : $x \in edrel(A) \Longrightarrow \exists\ a\in A.\ \exists\ b \in A.\ x = \langle a,b\rangle \wedge a \in domain(b)$
  by(*auto simp add*:*ed_def edrel_def Rrel_def*)

lemma *edrelI* [*intro*!]: $x\in A \Longrightarrow y\in A \Longrightarrow x \in domain(y) \Longrightarrow \langle x,y\rangle\in edrel(A)$
  by (*simp add*:*ed_def edrel_def Rrel_def*)

lemma *edrel_trans*: $Transset(A) \Longrightarrow y\in A \Longrightarrow x \in domain(y) \Longrightarrow \langle x,y\rangle\in edrel(A)$
  by (*rule edrelI*, *auto simp add*:*Transset_def domain_def Pair_def*)

lemma *domain_trans*: $Transset(A) \Longrightarrow y\in A \Longrightarrow x \in domain(y) \Longrightarrow x\in A$
  by (*auto simp add*: *Transset_def domain_def Pair_def*)

lemma *relation_edrel* : *relation*(*edrel*(*A*))
  by(*auto simp add*: *relation_def*)

lemma *field_edrel* : $field(edrel(A))\subseteq A$
  by *blast*

lemma *edrel_sub_memrel*: $edrel(A) \subseteq trancl(Memrel(eclose(A)))$
proof
  fix *z*
  assume
    $z\in edrel(A)$
  then obtain *x y* **where**
    Eq1: $x\in A$ $y\in A$ $z=\langle x,y\rangle$ $x\in domain(y)$
    using *edrelD*
    by *blast*
  then obtain *u v* **where**
    Eq2: $x\in u$ $u\in v$ $v\in y$
    unfolding *domain_def Pair_def* **by** *auto*
  with *Eq1* **have**
    Eq3: $x\in eclose(A)$ $y\in eclose(A)$ $u\in eclose(A)$ $v\in eclose(A)$
   by (*auto*, *rule_tac* [*3-4*] *ecloseD*, *rule_tac* [*3*] *ecloseD*, *simp_all add*:*arg_into_eclose*)
  let
    ?r=trancl(*Memrel*(*eclose*(*A*)))
  from *Eq2* **and** *Eq3* **have**
    $\langle x,u\rangle\in ?r$ $\langle u,v\rangle\in ?r$ $\langle v,y\rangle\in ?r$
    by (*auto simp add*: *r_into_trancl*)
  then **have**
    $\langle x,y\rangle\in ?r$
    by (*rule_tac trancl_trans*, *rule_tac* [*2*] *trancl_trans*, *simp*)
  with *Eq1* **show** $z\in ?r$ **by** *simp*
qed

lemma *wf_edrel* : *wf*(*edrel*(*A*))
  using *wf_subset* [*of trancl*(*Memrel*(*eclose*(*A*)))] *edrel_sub_memrel*
    wf_trancl wf_Memrel
  by *auto*

```
lemma ed_induction:
  assumes ⋀x. ⟦⋀y.  ed(y,x) ⟹ Q(y) ⟧ ⟹ Q(x)
  shows Q(a)
proof(induct rule: wf_induct2[OF wf_edrel[of eclose({a})] ,of a eclose({a})])
  case 1
  then show ?case using arg_into_eclose by simp
next
  case 2
  then show ?case using field_edrel .
next
  case (3 x)
  then
  show ?case
    using assms[of x] edrelI domain_trans[OF Transset_eclose 3(1)] by blast
qed

lemma dom_under_edrel_eclose: edrel(eclose({x})) -`` {x} = domain(x)
proof
  show edrel(eclose({x})) -`` {x} ⊆ domain(x)
    unfolding edrel_def Rrel_def ed_def
    by auto
next
  show domain(x) ⊆ edrel(eclose({x})) -`` {x}
    unfolding edrel_def Rrel_def
    using in_dom_in_eclose eclose_sing arg_into_eclose
    by blast
qed

lemma ed_eclose : ⟨y,z⟩ ∈ edrel(A) ⟹ y ∈ eclose(z)
  by(drule edrelD,auto simp add:domain_def in_dom_in_eclose)

lemma tr_edrel_eclose : ⟨y,z⟩ ∈ edrel(eclose({x}))^+ ⟹ y ∈ eclose(z)
  by(rule trancl_induct,(simp add: ed_eclose mem_eclose_trans)+)


lemma restrict_edrel_eq :
  assumes z ∈ domain(x)
  shows edrel(eclose({x})) ∩ eclose({z})×eclose({z}) = edrel(eclose({z}))
proof(intro equalityI subsetI)
  let ?ec=λ y . edrel(eclose({y}))
  let ?ez=eclose({z})
  let ?rr=?ec(x) ∩ ?ez × ?ez
  fix y
  assume yr:y ∈ ?rr
  with yr obtain a b where 1:⟨a,b⟩ ∈ ?rr a ∈ ?ez b ∈ ?ez ⟨a,b⟩ ∈ ?ec(x) y=⟨a,b⟩
    by blast
  moreover
  from this
  have a ∈ domain(b) using edrelD by blast
```

```
  ultimately
  show y ∈ edrel(eclose({z})) by blast
next
  let ?ec=λ y . edrel(eclose({y}))
  let ?ez=eclose({z})
  let ?rr=?ec(x) ∩ ?ez × ?ez
  fix y
  assume yr:y ∈ edrel(?ez)
  then obtain a b where a ∈ ?ez b ∈ ?ez y=⟨a,b⟩ a ∈ domain(b)
    using edrelD by blast
  moreover
  from this assms
  have z ∈ eclose(x) using in_dom_in_eclose by simp
  moreover
  from assms calculation
  have a ∈ eclose({x}) b ∈ eclose({x}) using in_eclose_sing by simp_all
  moreover
  from this ‹a∈domain(b)›
  have ⟨a,b⟩ ∈ edrel(eclose({x})) by blast
  ultimately
  show y ∈ ?rr by simp
qed

lemma tr_edrel_subset :
  assumes z ∈ domain(x)
  shows   tr_down(edrel(eclose({x})),z) ⊆ eclose({z})
proof(intro subsetI)
  let ?r=λ x . edrel(eclose({x}))
  fix y
  assume  y ∈ tr_down(?r(x),z)
  then
  have ⟨y,z⟩ ∈ ?r(x) ^+ using tr_downD by simp
  with assms
  show y ∈ eclose({z}) using tr_edrel_eclose eclose_sing by simp
qed

definition
  Hv :: [i,i,i,i]⇒i where
  Hv(P,G,x,f) ≡ { f`y .. y∈ domain(x), ∃ p∈P. ⟨y,p⟩ ∈ x ∧ p ∈ 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]⇒i where
  val(P,G,τ) ≡ wfrec(edrel(eclose({τ})), τ ,Hv(P,G))

definition
  GenExt :: [i,i,i]⇒i     (_^_[_] [71,1])
  where M^P[G] ≡ {val(P,G,τ). τ ∈ M}
```

abbreviation (**in** *forcing_notion*)
  GenExt_at_P :: *i*⇒*i*⇒*i*   (_[_] [*71*,*1*])
  where $M[G] \equiv M^P[G]$

context *M_ctm*
begin

lemma *upairM* : $x \in M \Longrightarrow y \in M \Longrightarrow \{x,y\} \in M$
  by (*simp flip*: *setclass_iff*)

lemma *singletonM* : $a \in M \Longrightarrow \{a\} \in M$
  by (*simp flip*: *setclass_iff*)

end

15.2 Values and check-names

context *forcing_data*
begin

definition
  Hcheck :: [*i*,*i*] ⇒ *i* **where**
  $Hcheck(z,f) \equiv \{ \langle f`y,one\rangle . y \in z\}$

definition
  check :: *i* ⇒ *i* **where**
  $check(x) \equiv transrec(x , Hcheck)$

lemma *checkD*:
  $check(x) = wfrec(Memrel(eclose(\{x\})), x, Hcheck)$
  unfolding *check_def transrec_def* **..**

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

lemma *Hcheck_trancl*:$Hcheck(y, restrict(f,Memrel(eclose(\{x\}))\text{-``}\{y\}))$
      $= Hcheck(y, restrict(f,(Memrel(eclose(\{x\}))\hat{}+)\text{-``}\{y\}))$
  unfolding *Hcheck_def*
  using *restrict_trans_eq* **by** *simp*

lemma *check_trancl*: $check(x) = wfrec(rcheck(x), x, Hcheck)$
  using *checkD wf_eq_trancl Hcheck_trancl* **unfolding** *rcheck_def* **by** *simp*

lemma *rcheck_in_M* :
  $x \in M \Longrightarrow rcheck(x) \in M$
  unfolding *rcheck_def* **by** (*simp flip*: *setclass_iff*)

lemma *aux_def_check*: $x \in y \Longrightarrow$
wfrec(*Memrel*(*eclose*({*y*})), *x*, *Hcheck*) =
wfrec(*Memrel*(*eclose*({*x*})), *x*, *Hcheck*)
by (*rule wfrec_eclose_eq*,*auto simp add*: *arg_into_eclose eclose_sing*)

lemma *def_check* : *check*(*y*) = { ⟨*check*(*w*),*one*⟩ . *w* ∈ *y*}
proof -
let
?*r*=λ*y*. *Memrel*(*eclose*({*y*}))
have *wfr*: ∀ *w* . *wf*(?*r*(*w*))
using *wf_Memrel* **..**
then
have *check*(*y*)= *Hcheck*(*y*, λ*x*∈?*r*(*y*) -‘‘ {*y*}. *wfrec*(?*r*(*y*), *x*, *Hcheck*))
using *wfrec*[*of* ?*r*(*y*) *y Hcheck*] *checkD* **by** *simp*
also
have ... = *Hcheck*(*y*, λ*x*∈*y*. *wfrec*(?*r*(*y*), *x*, *Hcheck*))
using *under_Memrel_eclose arg_into_eclose* **by** *simp*
also
have ... = *Hcheck*(*y*, λ*x*∈*y*. *check*(*x*))
using *aux_def_check checkD* **by** *simp*
finally show ?*thesis* **using** *Hcheck_def* **by** *simp*
qed

lemma *def_checkS* :
fixes *n*
assumes *n* ∈ *nat*
shows *check*(*succ*(*n*)) = *check*(*n*) ∪ {⟨*check*(*n*),*one*⟩}
proof -
have *check*(*succ*(*n*)) = {⟨*check*(*i*),*one*⟩ . *i* ∈ *succ*(*n*)}
using *def_check* **by** *blast*
also have ... = {⟨*check*(*i*),*one*⟩ . *i* ∈ *n*} ∪ {⟨*check*(*n*),*one*⟩}
by *blast*
also have ... = *check*(*n*) ∪ {⟨*check*(*n*),*one*⟩}
using *def_check*[*of n*,*symmetric*] **by** *simp*
finally show ?*thesis* **.**
qed

lemma *field_Memrel2* :
assumes *x* ∈ *M*
shows *field*(*Memrel*(*eclose*({*x*}))) ⊆ *M*
proof -
have *field*(*Memrel*(*eclose*({*x*}))) ⊆ *eclose*({*x*}) *eclose*({*x*}) ⊆ *M*
using *Ordinal.Memrel_type field_rel_subset assms eclose_least*[*OF trans_M*] **by**
auto
then
show ?*thesis* **using** *subset_trans* **by** *simp*
qed

lemma *aux_def_val*:
  assumes $z \in domain(x)$
  shows $wfrec(edrel(eclose(\{x\})),z,Hv(P,G)) = wfrec(edrel(eclose(\{z\})),z,Hv(P,G))$
proof -
  let $?r=\lambda x . edrel(eclose(\{x\}))$
  have $z\in eclose(\{z\})$ **using** *arg_in_eclose_sing* **.**
  moreover
  have $relation(?r(x))$ **using** *relation_edrel* **.**
  moreover
  have $wf(?r(x))$ **using** *wf_edrel* **.**
  moreover from *assms*
  have $tr\_down(?r(x),z) \subseteq eclose(\{z\})$ **using** *tr_edrel_subset* **by** *simp*
  ultimately
  have $wfrec(?r(x),z,Hv(P,G)) = wfrec[eclose(\{z\})](?r(x),z,Hv(P,G))$
    using *wfrec_restr* **by** *simp*
  also from ‹$z\in domain(x)$›
  have ... $= wfrec(?r(z),z,Hv(P,G))$
    using *restrict_edrel_eq wfrec_restr_eq* **by** *simp*
  finally show *?thesis* **.**
qed

The next lemma provides the usual recursive expresion for the definition of term*val*.

lemma *def_val*: $val(P,G,x) = \{val(P,G,t) .. t\in domain(x) , \exists p\in P . \langle t,p\rangle\in x \wedge p \in G \}$
proof -
  let
    $?r=\lambda\tau . edrel(eclose(\{\tau\}))$
  let
    $?f=\lambda z\in ?r(x)-``\{x\}. wfrec(?r(x),z,Hv(P,G))$
  have $\forall\tau. wf(?r(\tau))$ **using** *wf_edrel* **by** *simp*
  with *wfrec [of _ x]*
  have $val(P,G,x) = Hv(P,G,x,?f)$ **using** *val_def* **by** *simp*
  also
  have ... $= Hv(P,G,x,\lambda z\in domain(x). wfrec(?r(x),z,Hv(P,G)))$
    using *dom_under_edrel_eclose* **by** *simp*
  also
  have ... $= Hv(P,G,x,\lambda z\in domain(x). val(P,G,z))$
    using *aux_def_val val_def* **by** *simp*
  finally
  show *?thesis* **using** *Hv_def SepReplace_def* **by** *simp*
qed

lemma *val_mono* : $x\subseteq y \Longrightarrow val(P,G,x) \subseteq val(P,G,y)$
  by (*subst (1 2) def_val, force*)

Check-names are the canonical names for elements of the ground model. Here we show that this is the case.

lemma *valcheck* : $one \in G \Longrightarrow\ one \in P \Longrightarrow val(P,G,check(y))\ = y$
proof (*induct rule*:*eps_induct*)
  case (*1 y*)
  then show *?case*
  proof -
    have $check(y) = \{\ \langle check(w),\ one\rangle\ .\ w \in y\}$ (**is** _ = *?C*)
      using *def_check* **.**
    then
    have $val(P,G,check(y)) = val(P,G,\ \{\langle check(w),\ one\rangle\ .\ w \in y\})$
      by *simp*
    also
    have ... $= \{val(P,G,t)\ ..\ t{\in}domain(?C)\ ,\ \exists\, p{\in}P\ .\ \ \langle t,\ p\rangle{\in}?C \wedge p \in G\ \}$
      using *def_val* **by** *blast*
    also
    have ... $= \ \{val(P,G,t)\ ..\ t{\in}domain(?C)\ ,\ \exists\, w{\in}y.\ t{=}check(w)\ \}$
      using *1* **by** *simp*
    also
    have ... $= \{val(P,G,check(w))\ .\ w{\in}y\ \}$
      by *force*
    finally
    show $val(P,G,check(y)) = y$
      using *1* **by** *simp*
  qed
qed

lemma *val_of_name* :
  $val(P,G,\{x{\in}A{\times}P.\ Q(x)\}) = \{val(P,G,t)\ ..\ t{\in}A\ ,\ \exists\, p{\in}P\ .\ \ Q(\langle t,p\rangle) \wedge p \in G\ \}$
proof -
  let
    $?n{=}\{x{\in}A{\times}P.\ Q(x)\}$ **and**
    $?r{=}\lambda\tau\ .\ edrel(eclose(\{\tau\}))$
  let
    $?f{=}\lambda z{\in}?r(?n)\text{-``}\{?n\}.\ val(P,G,z)$
  have
    wfR : $wf(?r(\tau))$ **for** τ
    by (*simp add*: *wf_edrel*)
  have $domain(?n) \subseteq A$ **by** *auto*
  { fix t
    assume $H{:}t \in domain(\{x \in A \times P\ .\ Q(x)\})$
    then have $?f\ `\ t = (if\ t \in\ ?r(?n)\text{-``}\{?n\}\ then\ val(P,G,t)\ else\ 0)$
      by *simp*
    moreover have ... $= val(P,G,t)$
      using *dom_under_edrel_eclose H if_P* **by** *auto*
  }
  then
  have $Eq1{:}\ t \in domain(\{x \in A \times P\ .\ Q(x)\}) \Longrightarrow val(P,G,t) =\ ?f`\ t$ **for** t
    by *simp*
  have $val(P,G,?n) = \{val(P,G,t)\ ..\ t{\in}domain(?n),\ \exists\, p \in P\ .\ \langle t,p\rangle \in\ ?n \wedge p \in G\}$

by (*subst def_val*,*simp*)
also
have ... = {*?f'‘t* .. *t*∈*domain*(*?n*), ∃ *p*∈*P* . ⟨*t*,*p*⟩∈*?n* ∧ *p*∈*G*}
unfolding *Hv_def*
by (*subst SepReplace_dom_implies*,*auto simp add*:*Eq1*)
also
have ... = { (*if t*∈*?r*(*?n*)-‘‘{*?n*} *then val*(*P*,*G*,*t*) *else 0*) .. *t*∈*domain*(*?n*), ∃ *p*∈*P* . ⟨*t*,*p*⟩∈*?n* ∧ *p*∈*G*}
by (*simp*)
also
have *Eq2*: ... = { *val*(*P*,*G*,*t*) .. *t*∈*domain*(*?n*), ∃ *p*∈*P* . ⟨*t*,*p*⟩∈*?n* ∧ *p*∈*G*}
proof -
have *domain*(*?n*) ⊆ *?r*(*?n*)-‘‘{*?n*}
using *dom_under_edrel_eclose* **by** *simp*
then
have ∀ *t*∈*domain*(*?n*). (*if t*∈*?r*(*?n*)-‘‘{*?n*} *then val*(*P*,*G*,*t*) *else 0*) = *val*(*P*,*G*,*t*)
by *auto*
then
show { (*if t*∈*?r*(*?n*)-‘‘{*?n*} *then val*(*P*,*G*,*t*) *else 0*) .. *t*∈*domain*(*?n*), ∃ *p*∈*P* . ⟨*t*,*p*⟩∈*?n* ∧ *p*∈*G*} =
{ *val*(*P*,*G*,*t*) .. *t*∈*domain*(*?n*), ∃ *p*∈*P* . ⟨*t*,*p*⟩∈*?n* ∧ *p*∈*G*}
by *auto*
qed
also
have ... = { *val*(*P*,*G*,*t*) .. *t*∈*A*, ∃ *p*∈*P* . ⟨*t*,*p*⟩∈*?n* ∧ *p*∈*G*}
by *force*
finally
show *val*(*P*,*G*,*?n*) = { *val*(*P*,*G*,*t*) .. *t*∈*A*, ∃ *p*∈*P* . *Q*(⟨*t*,*p*⟩) ∧ *p*∈*G*}
by *auto*
qed

lemma *val_of_name_alt* :
val(*P*,*G*,{*x*∈*A*×*P*. *Q*(*x*)}) = {*val*(*P*,*G*,*t*) .. *t*∈*A* , ∃ *p*∈*P*∩*G* . *Q*(⟨*t*,*p*⟩) }
using *val_of_name* **by** *force*

lemma *val_only_names*: *val*(*P*,*F*,τ) = *val*(*P*,*F*,{*x*∈τ. ∃ *t*∈*domain*(τ). ∃ *p*∈*P*. *x*=⟨*t*,*p*⟩})
(**is** _ = *val*(*P*,*F*,*?name*))
proof -
have *val*(*P*,*F*,*?name*) = {*val*(*P*,*F*, *t*).. *t*∈*domain*(*?name*), ∃ *p*∈*P*. ⟨*t*, *p*⟩ ∈ *?name* ∧ *p* ∈ *F*}
using *def_val* **by** *blast*
also
have ... = {*val*(*P*,*F*, *t*). *t*∈{*y*∈*domain*(*?name*). ∃ *p*∈*P*. ⟨*y*, *p*⟩ ∈ *?name* ∧ *p* ∈ *F*}}
using *Sep_and_Replace* **by** *simp*
also
have ... = {*val*(*P*,*F*, *t*). *t*∈{*y*∈*domain*(τ). ∃ *p*∈*P*. ⟨*y*, *p*⟩ ∈ τ ∧ *p* ∈ *F*}}
by *blast*
also

```
  have ... = {val(P,F, t).. t∈domain(τ), ∃p∈P. ⟨t, p⟩ ∈ τ ∧ p ∈ F}
    using Sep_and_Replace by simp
  also
  have ... = val(P,F, τ)
    using def_val[symmetric] by blast
  finally
  show ?thesis ..
qed

lemma val_only_pairs: val(P,F,τ) = val(P,F,{x∈τ. ∃t p. x=⟨t,p⟩})
proof
  have val(P,F,τ) = val(P,F,{x∈τ. ∃t∈domain(τ). ∃p∈P. x=⟨t,p⟩})
    (is _ = val(P,F,?name))
    using val_only_names .
  also
  have ... ⊆ val(P,F,{x∈τ. ∃t p. x=⟨t,p⟩})
    using val_mono[of ?name {x∈τ. ∃t p. x=⟨t,p⟩}] by auto
  finally
  show val(P,F,τ) ⊆ val(P,F,{x∈τ. ∃t p. x=⟨t,p⟩}) by simp
next
  show val(P,F,{x∈τ. ∃t p. x=⟨t,p⟩}) ⊆ val(P,F,τ)
    using val_mono[of {x∈τ. ∃t p. x=⟨t,p⟩}] by auto
qed

lemma val_subset_domain_times_range: val(P,F,τ) ⊆ val(P,F,domain(τ)×range(τ))
  using val_only_pairs[THEN equalityD1]
    val_mono[of {x ∈ τ . ∃t p. x = ⟨t, p⟩} domain(τ)×range(τ)] by blast

lemma val_subset_domain_times_P: val(P,F,τ) ⊆ val(P,F,domain(τ)×P)
 using val_only_names[of F τ] val_mono[of {x∈τ. ∃t∈domain(τ). ∃p∈P. x=⟨t,p⟩}
domain(τ)×P F]
  by auto

lemma val_of_elem: ⟨ϑ,p⟩ ∈ π ⟹ p∈G ⟹ p∈P ⟹ val(P,G,ϑ) ∈ val(P,G,π)
proof -
  assume
    ⟨ϑ,p⟩ ∈ π
  then
  have ϑ∈domain(π) by auto
  assume p∈G p∈P
  with ‹ϑ∈domain(π)› ‹⟨ϑ,p⟩ ∈ π›
  have val(P,G,ϑ) ∈ {val(P,G,t) .. t∈domain(π) , ∃p∈P . ⟨t, p⟩∈π ∧ p ∈ G }
    by auto
  then
  show ?thesis by (subst def_val)
qed

lemma elem_of_val: x∈val(P,G,π) ⟹ ∃ϑ∈domain(π). val(P,G,ϑ) = x
  by (subst (asm) def_val,auto)
```

lemma *elem_of_val_pair*: $x{\in}val(P,G,\pi) \Longrightarrow \exists\vartheta.\ \exists p{\in}G.\ \langle\vartheta,p\rangle{\in}\pi \wedge val(P,G,\vartheta) = x$
by (*subst* (*asm*) *def_val*,*auto*)

lemma *elem_of_val_pair'*:
assumes $\pi{\in}M$ $x{\in}val(P,G,\pi)$
shows $\exists\vartheta{\in}M.\ \exists p{\in}G.\ \langle\vartheta,p\rangle{\in}\pi \wedge val(P,G,\vartheta) = x$
proof -
from *assms*
obtain ϑ p **where** $p{\in}G$ $\langle\vartheta,p\rangle{\in}\pi$ $val(P,G,\vartheta) = x$
using *elem_of_val_pair* **by** *blast*
moreover from *this* ‹$\pi{\in}M$›
have $\vartheta{\in}M$
using *pair_in_M_iff*[*THEN iffD1*, *THEN conjunct1*, *simplified*]
transitivity **by** *blast*
ultimately
show *?thesis* **by** *blast*
qed

lemma *GenExtD*:
$x \in M[G] \Longrightarrow \exists\tau{\in}M.\ x = val(P,G,\tau)$
by (*simp add*:*GenExt_def*)

lemma *GenExtI*:
$x \in M \Longrightarrow val(P,G,x) \in M[G]$
by (*auto simp add*: *GenExt_def*)

lemma *Transset_MG* : *Transset*($M[G]$)
proof -
{ fix *vc y*
assume $vc \in M[G]$ **and** $y \in vc$
then obtain c **where** $c{\in}M$ $val(P,G,c){\in}M[G]$ $y \in val(P,G,c)$
using *GenExtD* **by** *auto*
from ‹$y \in val(P,G,c)$›
obtain ϑ **where** $\vartheta{\in}domain(c)$ $val(P,G,\vartheta) = y$
using *elem_of_val* **by** *blast*
with *trans_M* ‹$c{\in}M$›
have $y \in M[G]$
using *domain_trans GenExtI* **by** *blast*
}
then
show *?thesis* **using** *Transset_def* **by** *auto*
qed

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

lemma *check_n_M* :

fixes *n*
assumes $n \in nat$
shows $check(n) \in M$
using ‹$n{\in}nat$›
proof (*induct n*)
case *0*
then show *?case* **using** *zero_in_M* **by** (*subst def_check*,*simp*)
next
case (*succ x*)
have $one \in M$ **using** *one_in_P P_sub_M subsetD* **by** *simp*
with ‹$check(x){\in}M$›
have $\langle check(x),one\rangle \in M$
using *tuples_in_M* **by** *simp*
then
have $\{\langle check(x),one\rangle\} \in M$
using *singletonM* **by** *simp*
with ‹$check(x){\in}M$›
have $check(x) \cup \{\langle check(x),one\rangle\} \in M$
using *Un_closed* **by** *simp*
then show *?case* **using** ‹$x{\in}nat$› *def_checkS* **by** *simp*
qed

definition
PHcheck :: $[i,i,i,i] \Rightarrow o$ **where**
$PHcheck(o,f,y,p) \equiv p{\in}M \wedge (\exists fy[\#\#M].\ fun\_apply(\#\#M,f,y,fy) \wedge pair(\#\#M,fy,o,p))$

definition
is_Hcheck :: $[i,i,i,i] \Rightarrow o$ **where**
$is\_Hcheck(o,z,f,hc) \equiv is\_Replace(\#\#M,z,PHcheck(o,f),hc)$

lemma *one_in_M*: $one \in M$
by (*insert one_in_P P_in_M*, *simp add*: *transitivity*)

lemma *def_PHcheck*:
assumes
$z{\in}M$ $f{\in}M$
shows
$Hcheck(z,f) = Replace(z,PHcheck(one,f))$
proof -
from *assms*
have $\langle f`x,one\rangle \in M$ $f`x{\in}M$ **if** $x{\in}z$ **for** *x*
using *tuples_in_M one_in_M transitivity that apply_closed* **by** *simp_all*
then
have $\{y\ .\ x \in z,\ y = \langle f\ `\ x,\ one\rangle\} = \{y\ .\ x \in z,\ y = \langle f\ `\ x,\ one\rangle \wedge y{\in}M \wedge f`x{\in}M\}$
by *simp*
then
show *?thesis*

using ‹*z*∈*M*› ‹*f*∈*M*› *transitivity*
unfolding *Hcheck_def PHcheck_def RepFun_def*
by *auto*
qed

definition
PHcheck_fm :: [*i*,*i*,*i*,*i*] ⇒ *i* **where**
PHcheck_fm(*o*,*f*,*y*,*p*) ≡ *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* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *u* ∈ *nat* ⟧ ⟹ *PHcheck_fm*(*x*,*y*,*z*,*u*) ∈ *formula*
by (*simp add*:*PHcheck_fm_def*)

lemma *sats_PHcheck_fm* [*simp*]:
⟦ *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *u* ∈ *nat* ; *env* ∈ *list*(*M*)⟧
⟹ *sats*(*M*,*PHcheck_fm*(*x*,*y*,*z*,*u*),*env*) ⟷
PHcheck(*nth*(*x*,*env*),*nth*(*y*,*env*),*nth*(*z*,*env*),*nth*(*u*,*env*))
using *zero_in_M Internalizations.nth_closed* **by** (*simp add*: *PHcheck_def PHcheck_fm_def*)

definition
is_Hcheck_fm :: [*i*,*i*,*i*,*i*] ⇒ *i* **where**
is_Hcheck_fm(*o*,*z*,*f*,*hc*) ≡ *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* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *u* ∈ *nat* ⟧ ⟹ *is_Hcheck_fm*(*x*,*y*,*z*,*u*) ∈ *formula*
by (*simp add*:*is_Hcheck_fm_def*)

lemma *sats_is_Hcheck_fm* [*simp*]:
⟦ *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *u* ∈ *nat* ; *env* ∈ *list*(*M*)⟧
⟹ *sats*(*M*,*is_Hcheck_fm*(*x*,*y*,*z*,*u*),*env*) ⟷
is_Hcheck(*nth*(*x*,*env*),*nth*(*y*,*env*),*nth*(*z*,*env*),*nth*(*u*,*env*))
using *sats_Replace_fm* **unfolding** *is_Hcheck_def is_Hcheck_fm_def*
by *simp*

lemma *wfrec_Hcheck* :
assumes
X∈*M*
shows
wfrec_replacement(##*M*,*is_Hcheck*(*one*),*rcheck*(*X*))
proof -

```
  have is_Hcheck(one,a,b,c) ⟷
      sats(M,is_Hcheck_fm(8,2,1,0),[c,b,a,d,e,y,x,z,one,rcheck(x)])
    if a∈M b∈M c∈M d∈M e∈M y∈M x∈M z∈M
    for a b c d e y x z
    using that one_in_M ‹X∈M› rcheck_in_M by simp
  then have 1:sats(M,is_wfrec_fm(is_Hcheck_fm(8,2,1,0),4,1,0),
                  [y,x,z,one,rcheck(X)]) ⟷
          is_wfrec(##M, is_Hcheck(one),rcheck(X), x, y)
    if x∈M y∈M z∈M for x y z
    using  that sats_is_wfrec_fm ‹X∈M› rcheck_in_M one_in_M by simp
  let
    ?f=Exists(And(pair_fm(1,0,2),
           is_wfrec_fm(is_Hcheck_fm(8,2,1,0),4,1,0)))
  have satsf:sats(M, ?f, [x,z,one,rcheck(X)]) ⟷
          (∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, is_Hcheck(one),rcheck(X),
x, y))
    if x∈M z∈M for x z
    using that 1 ‹X∈M› rcheck_in_M one_in_M by (simp del:pair_abs)
  have artyf:arity(?f) = 4
    unfolding fm_definitions
    by (simp add:nat_simp_union)
  then
  have strong_replacement(##M,λx z. sats(M,?f,[x,z,one,rcheck(X)]))
    using replacement_ax 1 artyf ‹X∈M› rcheck_in_M one_in_M by simp
  then
  have strong_replacement(##M,λx z.
          ∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, is_Hcheck(one),rcheck(X),
x, y))
    using repl_sats[of M ?f [one,rcheck(X)]] satsf by (simp del:pair_abs)
  then
  show ?thesis unfolding wfrec_replacement_def by simp
qed

lemma repl_PHcheck :
  assumes
    f∈M
  shows
    strong_replacement(##M,PHcheck(one,f))
proof -
  have arity(PHcheck_fm(2,3,0,1)) = 4
   unfolding PHcheck_fm_def fun_apply_fm_def big_union_fm_def pair_fm_def image_fm_def
      upair_fm_def
    by (simp add:nat_simp_union)
  with ‹f∈M›
  have strong_replacement(##M,λx y. sats(M,PHcheck_fm(2,3,0,1),[x,y,one,f]))
    using replacement_ax one_in_M by simp
  with ‹f∈M›
   show ?thesis using one_in_M unfolding strong_replacement_def univalent_def
by simp
```

qed

lemma *univ_PHcheck* : ⟦ $z \in M$; $f \in M$ ⟧ ⟹ *univalent*(##*M*,*z*,*PHcheck*(*one*,*f*))
unfolding *univalent_def PHcheck_def* **by** *simp*

lemma *relation2_Hcheck* :
relation2(##*M*,*is_Hcheck*(*one*),*Hcheck*)
proof -
have *1*:⟦$x \in z$; *PHcheck*(*one*,*f*,*x*,*y*) ⟧ ⟹ (##*M*)(*y*)
if $z \in M$ $f \in M$ **for** *z f x y*
using *that* **unfolding** *PHcheck_def* **by** *simp*
have *is_Replace*(##*M*,*z*,*PHcheck*(*one*,*f*),*hc*) ⟷ *hc* = *Replace*(*z*,*PHcheck*(*one*,*f*))
if $z \in M$ $f \in M$ $hc \in M$ **for** *z f hc*
using *that Replace_abs*[*OF* _ _ *univ_PHcheck 1*] **by** *simp*
with *def_PHcheck*
show *?thesis*
unfolding *relation2_def is_Hcheck_def Hcheck_def* **by** *simp*
qed

lemma *PHcheck_closed* :
⟦$z \in M$; $f \in M$; $x \in z$; *PHcheck*(*one*,*f*,*x*,*y*) ⟧ ⟹ (##*M*)(*y*)
unfolding *PHcheck_def* **by** *simp*

lemma *Hcheck_closed* :
∀ $y \in M$. ∀ $g \in M$. *function*(*g*) ⟶ *Hcheck*(*y*,*g*)∈*M*
proof -
have *Replace*(*y*,*PHcheck*(*one*,*f*))∈*M* **if** $f \in M$ $y \in M$ **for** *f y*
using *that repl_PHcheck PHcheck_closed*[*of y f*] *univ_PHcheck*
strong_replacement_closed
by (*simp flip*: *setclass_iff*)
then show *?thesis* **using** *def_PHcheck* **by** *auto*
qed

lemma *wf_rcheck* : $x \in M$ ⟹ *wf*(*rcheck*(*x*))
unfolding *rcheck_def* **using** *wf_trancl*[*OF wf_Memrel*] **.**

lemma *trans_rcheck* : $x \in M$ ⟹ *trans*(*rcheck*(*x*))
unfolding *rcheck_def* **using** *trans_trancl* **.**

lemma *relation_rcheck* : $x \in M$ ⟹ *relation*(*rcheck*(*x*))
unfolding *rcheck_def* **using** *relation_trancl* **.**

lemma *check_in_M* : $x \in M$ ⟹ *check*(*x*) ∈ *M*
unfolding *transrec_def*
using *wfrec_Hcheck*[*of x*] *check_trancl wf_rcheck trans_rcheck relation_rcheck rcheck_in_M*
Hcheck_closed relation2_Hcheck trans_wfrec_closed[*of rcheck*(*x*) *x is_Hcheck*(*one*)
Hcheck]
by (*simp flip*: *setclass_iff*)

end

definition
is_singleton :: $[i\Rightarrow o,i,i] \Rightarrow o$ **where**
$is\_singleton(A,x,z) \equiv \exists\, c[A].\ empty(A,c) \wedge is\_cons(A,x,c,z)$

lemma (**in** *M_trivial*) *singleton_abs*[*simp*] : $[\![M(x)\ ;\ M(s)\ ]\!] \Longrightarrow is\_singleton(M,x,s) \longleftrightarrow s = \{x\}$
unfolding *is_singleton_def* **using** *nonempty* **by** *simp*

definition
singleton_fm :: $[i,i] \Rightarrow i$ **where**
$singleton\_fm(i,j) \equiv Exists(And(empty\_fm(0),\ cons\_fm(succ(i),0,succ(j))))$

declare *singleton_fm_def*[*fm_definitions*]

lemma *singleton_type*[*TC*] : $[\![x \in nat;\ y \in nat\ ]\!] \Longrightarrow singleton\_fm(x,y) \in formula$
unfolding *singleton_fm_def* **by** *simp*

lemma *is_singleton_iff_sats*:
$[\![nth(i,env) = x;\ nth(j,env) = y;$
$i \in nat;\ j\in nat\ ;\ env \in list(A)]\!]$
$\Longrightarrow is\_singleton(\#\#A,x,y) \longleftrightarrow sats(A,\ singleton\_fm(i,j),\ env)$
unfolding *is_singleton_def singleton_fm_def* **by** *simp*

context *forcing_data* **begin**

definition
is_rcheck :: $[i,i] \Rightarrow o$ **where**
$is\_rcheck(x,z) \equiv \exists\, r\in M.\ tran\_closure(\#\#M,r,z) \wedge (\exists\, ec\in M.\ membership(\#\#M,ec,r)$
$\wedge$
$(\exists\, s\in M.\ is\_singleton(\#\#M,x,s) \wedge\ \ is\_eclose(\#\#M,s,ec)))$

lemma *rcheck_abs*[*Rel*] :
$[\![x\in M\ ;\ r\in M\ ]\!] \Longrightarrow is\_rcheck(x,r) \longleftrightarrow r = rcheck(x)$
unfolding *rcheck_def is_rcheck_def*
using *singletonM trancl_closed Memrel_closed eclose_closed* **by** *simp*

schematic_goal *rcheck_fm_auto*:
assumes
$i \in nat\ j \in nat\ env \in list(M)$
shows
$is\_rcheck(nth(i,env),nth(j,env)) \longleftrightarrow sats(M,?rch(i,j),env)$
unfolding *is_rcheck_def*
by (*insert assms* ; (*rule sep_rules is_singleton_iff_sats is_eclose_iff_sats*
trans_closure_fm_iff_sats | *simp*)+)

synthesize *rcheck_fm* **from_schematic** *rcheck_fm_auto*

definition
  is_check :: [*i*,*i*] ⇒ *o* **where**
 is_check(*x*,*z*) ≡ ∃ *rch*∈*M*. *is_rcheck*(*x*,*rch*) ∧ *is_wfrec*(##*M*,*is_Hcheck*(*one*),*rch*,*x*,*z*)

lemma *check_abs*[*Rel*] :
  assumes
    x∈*M* *z*∈*M*
  shows
    is_check(*x*,*z*) ⟷ *z* = *check*(*x*)
proof -
  have
    is_check(*x*,*z*) ⟷ *is_wfrec*(##*M*,*is_Hcheck*(*one*),*rcheck*(*x*),*x*,*z*)
    unfolding *is_check_def* **using** *assms rcheck_abs rcheck_in_M*
    unfolding *check_trancl is_check_def* **by** *simp*
  then show *?thesis*
    unfolding *check_trancl*
   using *assms wfrec_Hcheck*[*of x*] *wf_rcheck trans_rcheck relation_rcheck rcheck_in_M*
     Hcheck_closed relation2_Hcheck trans_wfrec_abs[*of rcheck*(*x*) *x z is_Hcheck*(*one*)
Hcheck]
    by (*simp flip*: *setclass_iff*)
qed

definition
  check_fm :: [*i*,*i*,*i*] ⇒ *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*)))

lemma *check_fm_type*[*TC*] :
  ⟦*x*∈*nat*;*o*∈*nat*;*z*∈*nat*⟧ ⟹ *check_fm*(*x*,*o*,*z*)∈*formula*
  unfolding *check_fm_def* **by** *simp*

lemma *sats_check_fm* :
  assumes
    nth(*o*,*env*) = *one* *x*∈*nat* *z*∈*nat* *o*∈*nat* *env*∈*list*(*M*) *x* < *length*(*env*) *z* <
length(*env*)
  shows
    sats(*M*, *check_fm*(*x*,*o*,*z*), *env*) ⟷ *is_check*(*nth*(*x*,*env*),*nth*(*z*,*env*))
proof -
  have *sats_is_Hcheck_fm*:
    ⋀*a0 a1 a2 a3 a4*. ⟦ *a0*∈*M*; *a1*∈*M*; *a2*∈*M*; *a3*∈*M*; *a4*∈*M* ⟧ ⟹
       is_Hcheck(*one*,*a2*, *a1*, *a0*) ⟷
       sats(*M*, *is_Hcheck_fm*(*6*#+*o*,*2*,*1*,*0*), [*a0*,*a1*,*a2*,*a3*,*a4*,*r*]@*env*) **if** *r*∈*M* **for**
r
    using *that one_in_M assms* **by** *simp*

then
have *sats*(*M*, *is_wfrec_fm*(*is_Hcheck_fm*(*6*#+*o*,*2*,*1*,*0*),*0*,*1*#+*x*,*1*#+*z*),*Cons*(*r*,*env*))
⟷ *is_wfrec*(##*M*,*is_Hcheck*(*one*),*r*,*nth*(*x*,*env*),*nth*(*z*,*env*)) **if** *r*∈*M* **for** *r*
using *that assms one_in_M sats_is_wfrec_fm* **by** *simp*
then
show *?thesis* **unfolding** *is_check_def check_fm_def*
using *assms rcheck_in_M one_in_M rcheck_fm_iff_sats*[*symmetric*] **by** *simp*
qed

lemma *check_replacement*:
{*check*(*x*). *x*∈*P*} ∈ *M*
proof -
have *arity*(*check_fm*(*0*,*2*,*1*)) = *3*
unfolding *eclose_n_fm_def is_eclose_fm_def mem_eclose_fm_def fm_definitions*
by (*simp add*:*nat_simp_union*)
moreover
have *check*(*x*)∈*M* **if** *x*∈*P* **for** *x*
using *that transitivity check_in_M P_in_M* **by** *simp*
ultimately
show *?thesis* **using** *sats_check_fm check_abs P_in_M check_in_M one_in_M*
Repl_in_M[*of check_fm*(*0*,*2*,*1*) [*one*] *is_check check*] **by** *simp*
qed

lemma *pair_check* : ⟦ *p*∈*M* ; *y*∈*M* ⟧ ⟹ (∃ *c*∈*M*. *is_check*(*p*,*c*) ∧ *pair*(##*M*,*c*,*p*,*y*))
⟷ *y* = ⟨*check*(*p*),*p*⟩
using *check_abs check_in_M tuples_in_M* **by** *simp*

lemma *M_subset_MG* : *one* ∈ *G* ⟹ *M* ⊆ *M*[*G*]
using *check_in_M one_in_P GenExtI*
by (*intro subsetI*, *subst valcheck* [*of G*,*symmetric*], *auto*)

The name for the generic filter

definition
G_dot :: *i* **where**
G_dot ≡ {⟨*check*(*p*),*p*⟩ . *p*∈*P*}

lemma *G_dot_in_M* :
G_dot ∈ *M*
proof -
let *?is_pcheck* = λ*x y*. ∃ *ch*∈*M*. *is_check*(*x*,*ch*) ∧ *pair*(##*M*,*ch*,*x*,*y*)
let *?pcheck_fm* = *Exists*(*And*(*check_fm*(*1*,*3*,*0*),*pair_fm*(*0*,*1*,*2*)))
have *sats*(*M*,*?pcheck_fm*,[*x*,*y*,*one*]) ⟷ *?is_pcheck*(*x*,*y*) **if** *x*∈*M* *y*∈*M* **for** *x y*
using *sats_check_fm that one_in_M* **by** *simp*
moreover
have *?is_pcheck*(*x*,*y*) ⟷ *y* = ⟨*check*(*x*),*x*⟩ **if** *x*∈*M* *y*∈*M* **for** *x y*
using *that check_abs check_in_M* **by** *simp*
moreover

have *?pcheck_fm∈formula* **by** *simp*
moreover
have *arity(?pcheck_fm)=3*
unfolding *is_eclose_fm_def mem_eclose_fm_def eclose_n_fm_def fm_definitions*
by (*simp add:nat_simp_union*)
moreover
from *P_in_M check_in_M tuples_in_M P_sub_M*
have ⟨*check(p),p*⟩ ∈ *M* **if** *p∈P* **for** *p*
using *that* **by** *auto*
ultimately
show *?thesis*
unfolding *G_dot_def*
using *one_in_M P_in_M Repl_in_M[of ?pcheck_fm [one]]*
by *simp*
qed

lemma *val_G_dot* :
assumes $G \subseteq P$
one ∈ *G*
shows *val(P,G,G_dot)* = *G*
proof (*intro equalityI subsetI*)
fix *x*
assume *x∈val(P,G,G_dot)*
then obtain *ϑ p* **where** *p∈G* ⟨*ϑ,p*⟩ ∈ *G_dot val(P,G,ϑ)* = *x ϑ* = *check(p)*
unfolding *G_dot_def* **using** *elem_of_val_pair G_dot_in_M*
by *force*
with ‹*one∈G*› ‹*G⊆P*› **show**
x ∈ *G*
using *valcheck P_sub_M* **by** *auto*
next
fix *p*
assume *p∈G*
have ⟨*check(q),q*⟩ ∈ *G_dot* **if** *q∈P* **for** *q*
unfolding *G_dot_def* **using** *that* **by** *simp*
with ‹*p∈G*› ‹*G⊆P*›
have *val(P,G,check(p))* ∈ *val(P,G,G_dot)*
using *val_of_elem G_dot_in_M* **by** *blast*
with ‹*p∈G*› ‹*G⊆P*› ‹*one∈G*›
show *p* ∈ *val(P,G,G_dot)*
using *P_sub_M valcheck* **by** *auto*
qed

lemma *G_in_Gen_Ext* :
assumes $G \subseteq P$ **and** *one* ∈ *G*
shows *G* ∈ *M[G]*
using *assms val_G_dot GenExtI[of _ G] G_dot_in_M*
by *force*

lemma *fst_snd_closed*: *p*∈*M* ⟹ *fst*(*p*) ∈ *M* ∧ *snd*(*p*)∈ *M*
proof (*cases* ∃ *a*. ∃ *b*. *p* = ⟨*a*, *b*⟩)
  case *False*
  then
  show *fst*(*p*) ∈ *M* ∧ *snd*(*p*) ∈ *M* **unfolding** *fst_def snd_def* **using** *zero_in_M* **by** *auto*
next
  case *True*
  then
  obtain *a b* **where** *p* = ⟨*a*, *b*⟩ **by** *blast*
  with *True*
  have *fst*(*p*) = *a snd*(*p*) = *b* **unfolding** *fst_def snd_def* **by** *simp_all*
  moreover
  assume *p*∈*M*
  moreover from *this*
  have *a*∈*M*
    unfolding ‹*p* = _› *Pair_def* **by** (*force intro*:*Transset_M*[*OF trans_M*])
  moreover from ‹*p*∈*M*›
  have *b*∈*M*
     using *Transset_M*[*OF trans_M*, *of* {*a*,*b*} *p*] *Transset_M*[*OF trans_M*, *of b* {*a*,*b*}]
    unfolding ‹*p* = _› *Pair_def* **by** (*simp*)
  ultimately
  show *?thesis* **by** *simp*
qed

end

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

lemma *zero_in_MG* :
  0 ∈ *M*[*G*]
proof -
  have *0* = *val*(*P*,*G*,*0*)
    using *zero_in_M elem_of_val* **by** *auto*
  also
  have ... ∈ *M*[*G*]
    using *GenExtI zero_in_M* **by** *simp*
  finally show *?thesis* **.**
qed

lemma *G_nonempty*: *G*≠*0*
proof -
  have *P*⊆*P* **..**
  with *P_in_M P_dense* ‹*P*⊆*P*›

show $G \neq 0$
using *generic* **unfolding** *M_generic_def* **by** *auto*
qed

end
end

16 Well-founded relation on names

theory *FrecR* **imports** *Names Synthetic_Definition* **begin**

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

frecR is the well-founded relation on names that allows us to define forcing for atomic formulas.

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

lemma (**in** *M_trivial*) *hcomp_abs*:
assumes
is_f_abs: $\bigwedge a\ z.\ M(a) \Longrightarrow M(z) \Longrightarrow is\_f(a,z) \longleftrightarrow z = f(a)$ **and**
is_g_abs: $\bigwedge a\ z.\ M(a) \Longrightarrow M(z) \Longrightarrow is\_g(a,z) \longleftrightarrow z = g(a)$ **and**
g_closed: $\bigwedge a.\ M(a) \Longrightarrow M(g(a))$
$M(a)\ M(w)$
shows
$is\_hcomp(M,is\_f,is\_g,a,w) \longleftrightarrow w = f(g(a))$
unfolding *is_hcomp_def* **using** *assms* **by** *simp*

definition
hcomp_fm :: $[i \Rightarrow i \Rightarrow i, i \Rightarrow i \Rightarrow i, i, i] \Rightarrow i$ **where**
$hcomp\_fm(pf,pg,a,w) \equiv Exists(And(pg(succ(a),0),pf(0,succ(w))))$

lemma *sats_hcomp_fm*:
assumes
f_iff_sats: $\bigwedge a\ b\ z.\ a \in nat \Longrightarrow b \in nat \Longrightarrow z \in M \Longrightarrow$
$is\_f(nth(a,Cons(z,env)),nth(b,Cons(z,env))) \longleftrightarrow sats(M,pf(a,b),Cons(z,env))$
and
g_iff_sats: $\bigwedge a\ b\ z.\ a \in nat \Longrightarrow b \in nat \Longrightarrow z \in M \Longrightarrow$
$is\_g(nth(a,Cons(z,env)),nth(b,Cons(z,env))) \longleftrightarrow sats(M,pg(a,b),Cons(z,env))$
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 -
have $sats(M,\ pf(0,\ succ(w)),\ Cons(x,\ env)) \longleftrightarrow is\_f(x,nth(w,env))$ **if** $x \in M$ $w \in nat$ **for** $x\ w$
using *f_iff_sats*[*of 0 succ(w) x*] *that* **by** *simp*

moreover
have *sats*(*M*, *pg*(*succ*(*a*), *0*), *Cons*(*x*, *env*)) ⟷ *is_g*(*nth*(*a*,*env*),*x*) **if** *x*∈*M* *a*∈*nat* **for** *x a*
using *g_iff_sats*[*of succ*(*a*) *0 x*] *that* **by** *simp*
ultimately
show *?thesis* **unfolding** *hcomp_fm_def is_hcomp_def* **using** *assms* **by** *simp*
qed

definition
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*⇒*i* **where**
cond_of(*x*) ≡ *snd*(*snd*(*snd*((*x*))))

lemma *components_simp*:
ftype(⟨*f*,*n1*,*n2*,*c*⟩) = *f*
name1(⟨*f*,*n1*,*n2*,*c*⟩) = *n1*
name2(⟨*f*,*n1*,*n2*,*c*⟩) = *n2*
cond_of(⟨*f*,*n1*,*n2*,*c*⟩) = *c*
unfolding *ftype_def name1_def name2_def cond_of_def*
by *simp_all*

definition *eclose_n* :: [*i*⇒*i*,*i*] ⇒ *i* **where**
eclose_n(*name*,*x*) = *eclose*({*name*(*x*)})

definition
ecloseN :: *i* ⇒ *i* **where**
ecloseN(*x*) = *eclose_n*(*name1*,*x*) ∪ *eclose_n*(*name2*,*x*)

lemma *components_in_eclose* :
n1 ∈ *ecloseN*(⟨*f*,*n1*,*n2*,*c*⟩)
n2 ∈ *ecloseN*(⟨*f*,*n1*,*n2*,*c*⟩)
unfolding *ecloseN_def eclose_n_def*
using *components_simp arg_into_eclose* **by** *auto*

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

```
lemma ecloseNI1 :
  assumes x ∈ eclose(n1) ∨ x∈eclose(n2)
  shows x ∈ ecloseN(⟨f,n1,n2,c⟩)
  unfolding ecloseN_def eclose_n_def
  using assms eclose_sing names_simp
  by auto

lemmas ecloseNI = ecloseNI1

lemma ecloseN_mono :
  assumes u ∈ ecloseN(x) name1(x) ∈ ecloseN(y) name2(x) ∈ ecloseN(y)
  shows u ∈ ecloseN(y)
proof -
  from ‹u∈_›
  consider (a) u∈eclose({name1(x)}) | (b) u ∈ eclose({name2(x)})
    unfolding ecloseN_def  eclose_n_def by auto
  then
  show ?thesis
  proof cases
    case a
    with ‹name1(x) ∈ _›
    show ?thesis
      unfolding ecloseN_def  eclose_n_def
      using eclose_singE[OF a] mem_eclose_trans[of u name1(x) ] by auto
  next
    case b
    with ‹name2(x) ∈ _›
    show ?thesis
      unfolding ecloseN_def eclose_n_def
      using eclose_singE[OF b] mem_eclose_trans[of u name2(x)] by auto
  qed
qed

definition
  is_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))

definition
  fst_fm :: [i,i] ⇒ i where
  fst_fm(x,t) ≡ Or(Exists(pair_fm(succ(t),0,succ(x))),
                 And(Neg(Exists(Exists(pair_fm(0,1,2 #+ x)))),empty_fm(t)))

lemma sats_fst_fm :
  ⟦ x ∈ nat; y ∈ nat;env ∈ list(A) ⟧
    ⟹ sats(A, fst_fm(x,y), env) ⟷
```

```
      is_fst(##A, nth(x,env), nth(y,env))
  by (simp add: fst_fm_def is_fst_def)

definition
  is_ftype :: (i⇒o)⇒i⇒i⇒o where
  is_ftype ≡ is_fst

definition
  ftype_fm :: [i,i] ⇒ i where
  ftype_fm ≡ fst_fm

lemma is_ftype_iff_sats:
  assumes
    nth(a,env) = aa nth(b,env) = bb a∈nat b∈nat env ∈ list(A)
  shows
    is_ftype(##A,aa,bb)  ⟷ sats(A,ftype_fm(a,b), env)
  unfolding ftype_fm_def is_ftype_def
  using assms sats_fst_fm
  by simp

definition
  is_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))

definition
  snd_fm :: [i,i] ⇒ i where
  snd_fm(x,t) ≡ Or(Exists(pair_fm(0,succ(t),succ(x))),
                And(Neg(Exists(Exists(pair_fm(1,0,2 #+ x)))),empty_fm(t)))

lemma sats_snd_fm :
  ⟦ x ∈ nat; y ∈ nat;env ∈ list(A) ⟧
    ⟹ sats(A, snd_fm(x,y), env) ⟷
      is_snd(##A, nth(x,env), nth(y,env))
  by (simp add: snd_fm_def is_snd_def)

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 :
  ⟦ x ∈ nat; y ∈ nat;env ∈ list(A) ⟧
    ⟹ sats(A, name1_fm(x,y), env) ⟷
      is_name1(##A, nth(x,env), nth(y,env))
  unfolding name1_fm_def is_name1_def using sats_fst_fm sats_snd_fm
```

sats_hcomp_fm[of A is_fst(##A) _ fst_fm is_snd(##A)] **by** *simp*

lemma *is_name1_iff_sats*:
assumes
nth(a,env) = aa nth(b,env) = bb a∈nat b∈nat env ∈ list(A)
shows
is_name1(##A,aa,bb) ⟷ sats(A,name1_fm(a,b), env)
using *assms sats_name1_fm*
by *simp*

definition
is_snd_snd :: *(i⇒o)⇒i⇒i⇒o* **where**
is_snd_snd(M,x,t) ≡ is_hcomp(M,is_snd(M),is_snd(M),x,t)

definition
snd_snd_fm :: *[i,i]⇒i* **where**
snd_snd_fm(x,t) ≡ hcomp_fm(snd_fm,snd_fm,x,t)

lemma *sats_snd2_fm* :
⟦ *x ∈ nat; y ∈ nat;env ∈ list(A)* ⟧
⟹ *sats(A,snd_snd_fm(x,y), env)* ⟷
is_snd_snd(##A, nth(x,env), nth(y,env))
unfolding *snd_snd_fm_def is_snd_snd_def* **using** *sats_snd_fm*
sats_hcomp_fm[of A is_snd(##A) _ snd_fm is_snd(##A)] **by** *simp*

definition
is_name2 :: *(i⇒o)⇒i⇒i⇒o* **where**
is_name2(M,x,t3) ≡ is_hcomp(M,is_fst(M),is_snd_snd(M),x,t3)

definition
name2_fm :: *[i,i] ⇒ i* **where**
name2_fm(x,t3) ≡ hcomp_fm(fst_fm,snd_snd_fm,x,t3)

lemma *sats_name2_fm* :
⟦ *x ∈ nat; y ∈ nat;env ∈ list(A)* ⟧
⟹ *sats(A,name2_fm(x,y), env)* ⟷
is_name2(##A, nth(x,env), nth(y,env))
unfolding *name2_fm_def is_name2_def* **using** *sats_fst_fm sats_snd2_fm*
sats_hcomp_fm[of A is_fst(##A) _ fst_fm is_snd_snd(##A)] **by** *simp*

lemma *is_name2_iff_sats*:
assumes
nth(a,env) = aa nth(b,env) = bb a∈nat b∈nat env ∈ list(A)
shows
is_name2(##A,aa,bb) ⟷ sats(A,name2_fm(a,b), env)
using *assms*
by *(simp add:sats_name2_fm)*

definition

is_cond_of :: (*i*⇒*o*)⇒*i*⇒*i*⇒*o* **where**
is_cond_of(*M*,*x*,*t4*) ≡ *is_hcomp*(*M*,*is_snd*(*M*),*is_snd_snd*(*M*),*x*,*t4*)

definition
cond_of_fm :: [*i*,*i*] ⇒ *i* **where**
cond_of_fm(*x*,*t4*) ≡ *hcomp_fm*(*snd_fm*,*snd_snd_fm*,*x*,*t4*)

lemma *sats_cond_of_fm* :
⟦ *x* ∈ *nat*; *y* ∈ *nat*;*env* ∈ *list*(*A*) ⟧
⟹ *sats*(*A*,*cond_of_fm*(*x*,*y*), *env*) ⟷
is_cond_of(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
unfolding *cond_of_fm_def is_cond_of_def* **using** *sats_snd_fm sats_snd2_fm*
sats_hcomp_fm[*of A is_snd*(##*A*) _ *snd_fm is_snd_snd*(##*A*)] **by** *simp*

lemma *is_cond_of_iff_sats*:
assumes
nth(*a*,*env*) = *aa nth*(*b*,*env*) = *bb a*∈*nat b*∈*nat env* ∈ *list*(*A*)
shows
is_cond_of(##*A*,*aa*,*bb*) ⟷ *sats*(*A*,*cond_of_fm*(*a*,*b*), *env*)
using *assms*
by (*simp add*:*sats_cond_of_fm*)

lemma *components_type*[*TC*] :
assumes *a*∈*nat b*∈*nat*
shows
ftype_fm(*a*,*b*)∈*formula*
name1_fm(*a*,*b*)∈*formula*
name2_fm(*a*,*b*)∈*formula*
cond_of_fm(*a*,*b*)∈*formula*
using *assms*
unfolding *ftype_fm_def fst_fm_def snd_fm_def snd_snd_fm_def name1_fm_def name2_fm_def*
cond_of_fm_def hcomp_fm_def
by *simp_all*

lemmas *components_iff_sats* = *is_ftype_iff_sats is_name1_iff_sats is_name2_iff_sats*
is_cond_of_iff_sats

lemmas *components_defs* = *fst_fm_def ftype_fm_def snd_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*,*en*,*t*) ≡ ∃ *en1*[*N*].∃ *en2*[*N*].
  is_eclose_n(*N*,*is_name1*,*en1*,*t*) ∧ *is_eclose_n*(*N*,*is_name2*,*en2*,*t*)∧
  union(*N*,*en1*,*en2*,*en*)

definition
  ecloseN_fm :: [*i*,*i*] ⇒ *i* **where**
  ecloseN_fm(*en*,*t*) ≡ *Exists*(*Exists*(*And*(*eclose_n1_fm*(*1*,*t*#+*2*),
  And(*eclose_n2_fm*(*0*,*t*#+*2*),*union_fm*(*1*,*0*,*en*#+*2*)))))
lemma *ecloseN_fm_type* [*TC*] :
  ⟦ *en* ∈ *nat* ; *t* ∈ *nat* ⟧ ⟹ *ecloseN_fm*(*en*,*t*) ∈ *formula*
  unfolding *ecloseN_fm_def eclose_n1_fm_def eclose_n2_fm_def* **by** *simp*

lemma *sats_ecloseN_fm* [*simp*]:
  ⟦ *en* ∈ *nat*; *t* ∈ *nat* ; *env* ∈ *list*(*A*) ⟧
  ⟹ *sats*(*A*, *ecloseN_fm*(*en*,*t*), *env*) ⟷ *is_ecloseN*(##*A*,*nth*(*en*,*env*),*nth*(*t*,*env*))
  unfolding *ecloseN_fm_def is_ecloseN_def eclose_n1_fm_def eclose_n2_fm_def is_eclose_n_def*
  using *nth_0 nth_ConsI sats_name1_fm sats_name2_fm*
  is_singleton_iff_sats[*symmetric*]
  by *auto*

definition
  frecR :: *i* ⇒ *i* ⇒ *o* **where**
  frecR(*x*,*y*) ≡
  (*ftype*(*x*) = *1* ∧ *ftype*(*y*) = *0*
  ∧ (*name1*(*x*) ∈ *domain*(*name1*(*y*)) ∪ *domain*(*name2*(*y*)) ∧ (*name2*(*x*) = *name1*(*y*) ∨ *name2*(*x*) = *name2*(*y*))))
  ∨ (*ftype*(*x*) = *0* ∧ *ftype*(*y*) = *1* ∧ *name1*(*x*) = *name1*(*y*) ∧ *name2*(*x*) ∈ *domain*(*name2*(*y*)))

lemma *frecR_ftypeD* :
  assumes *frecR*(*x*,*y*)
  shows (*ftype*(*x*) = *0* ∧ *ftype*(*y*) = *1*) ∨ (*ftype*(*x*) = *1* ∧ *ftype*(*y*) = *0*)
  using *assms* **unfolding** *frecR_def* **by** *auto*

lemma *frecRI1*: *s* ∈ *domain*(*n1*) ∨ *s* ∈ *domain*(*n2*) ⟹ *frecR*(⟨*1*, *s*, *n1*, *q*⟩, ⟨*0*, *n1*, *n2*, *q'*⟩)
  unfolding *frecR_def* **by** (*simp add*:*components_simp*)

lemma *frecRI1'*: *s* ∈ *domain*(*n1*) ∪ *domain*(*n2*) ⟹ *frecR*(⟨*1*, *s*, *n1*, *q*⟩, ⟨*0*, *n1*,

n2, *q'*⟩)
unfolding *frecR_def* **by** (*simp add*:*components_simp*)

lemma *frecRI2*: *s* ∈ *domain*(*n1*) ∨ *s* ∈ *domain*(*n2*) ⟹ *frecR*(⟨*1*, *s*, *n2*, *q*⟩, ⟨*0*, *n1*, *n2*, *q'*⟩)
unfolding *frecR_def* **by** (*simp add*:*components_simp*)

lemma *frecRI2'*: *s* ∈ *domain*(*n1*) ∪ *domain*(*n2*) ⟹ *frecR*(⟨*1*, *s*, *n2*, *q*⟩, ⟨*0*, *n1*, *n2*, *q'*⟩)
unfolding *frecR_def* **by** (*simp add*:*components_simp*)

lemma *frecRI3*: ⟨*s*, *r*⟩ ∈ *n2* ⟹ *frecR*(⟨*0*, *n1*, *s*, *q*⟩, ⟨*1*, *n1*, *n2*, *q'*⟩)
unfolding *frecR_def* **by** (*auto simp add*:*components_simp*)

lemma *frecRI3'*: *s* ∈ *domain*(*n2*) ⟹ *frecR*(⟨*0*, *n1*, *s*, *q*⟩, ⟨*1*, *n1*, *n2*, *q'*⟩)
unfolding *frecR_def* **by** (*auto simp add*:*components_simp*)

lemma *frecR_iff* :
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*)))
unfolding *frecR_def* **..**

lemma *frecR_D1* :
frecR(*x*,*y*) ⟹ *ftype*(*y*) = *0* ⟹ *ftype*(*x*) = *1* ∧
(*name1*(*x*) ∈ *domain*(*name1*(*y*)) ∪ *domain*(*name2*(*y*)) ∧ (*name2*(*x*) = *name1*(*y*) ∨ *name2*(*x*) = *name2*(*y*)))
using *frecR_iff*
by *auto*

lemma *frecR_D2* :
frecR(*x*,*y*) ⟹ *ftype*(*y*) = *1* ⟹ *ftype*(*x*) = *0* ∧
ftype(*x*) = *0* ∧ *ftype*(*y*) = *1* ∧ *name1*(*x*) = *name1*(*y*) ∧ *name2*(*x*) ∈ *domain*(*name2*(*y*))
using *frecR_iff*
by *auto*

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*)
using *assms* **unfolding** *frecR_def* **by** (*force simp add*:*components_simp*)

definition
is_frecR :: [*i*⇒*o*,*i*,*i*] ⇒ *o* **where**

is_frecR(*M*,*x*,*y*) ≡ ∃ *ftx*[*M*]. ∃ *n1x*[*M*]. ∃ *n2x*[*M*]. ∃ *fty*[*M*]. ∃ *n1y*[*M*]. ∃ *n2y*[*M*]. ∃ *dn1*[*M*]. ∃ *dn2*[*M*].
is_ftype(*M*,*x*,*ftx*) ∧ *is_name1*(*M*,*x*,*n1x*)∧ *is_name2*(*M*,*x*,*n2x*) ∧
is_ftype(*M*,*y*,*fty*) ∧ *is_name1*(*M*,*y*,*n1y*) ∧ *is_name2*(*M*,*y*,*n2y*)
∧ *is_domain*(*M*,*n1y*,*dn1*) ∧ *is_domain*(*M*,*n2y*,*dn2*) ∧
((*number1*(*M*,*ftx*) ∧ *empty*(*M*,*fty*) ∧ (*n1x* ∈ *dn1* ∨ *n1x* ∈ *dn2*) ∧ (*n2x* = *n1y* ∨ *n2x* = *n2y*))
∨ (*empty*(*M*,*ftx*) ∧ *number1*(*M*,*fty*) ∧ *n1x* = *n1y* ∧ *n2x* ∈ *dn2*))

schematic_goal *sats_frecR_fm_auto*:
assumes
i∈*nat* *j*∈*nat* *env*∈*list*(*A*) *nth*(*i*,*env*) = *a* *nth*(*j*,*env*) = *b*
shows
is_frecR(##*A*,*a*,*b*) ⟷ *sats*(*A*,*?fr_fm*(*i*,*j*),*env*)
unfolding *is_frecR_def*
by (*insert assms* ; (*rule sep_rules′ cartprod_iff_sats components_iff_sats*
| *simp del*:*sats_cartprod_fm*)+)

synthesize *frecR_fm* **from_schematic** *sats_frecR_fm_auto*

lemma *eq_ftypep_not_frecrR*:
assumes *ftype*(*x*) = *ftype*(*y*)
shows ¬ *frecR*(*x*,*y*)
using *assms frecR_ftypeD* **by** *force*

definition
rank_names :: *i* ⇒ *i* **where**
rank_names(*x*) ≡ *max*(*rank*(*name1*(*x*)),*rank*(*name2*(*x*)))

lemma *rank_names_types* [*TC*]:
shows *Ord*(*rank_names*(*x*))
unfolding *rank_names_def max_def* **using** *Ord_rank Ord_Un* **by** *auto*

definition
mtype_form :: *i* ⇒ *i* **where**
mtype_form(*x*) ≡ *if rank*(*name1*(*x*)) < *rank*(*name2*(*x*)) *then 0 else 2*

definition
type_form :: *i* ⇒ *i* **where**
type_form(*x*) ≡ *if ftype*(*x*) = *0 then 1 else mtype_form*(*x*)

lemma *type_form_tc* [*TC*]:
shows *type_form*(*x*) ∈ *3*
unfolding *type_form_def mtype_form_def* **by** *auto*

lemma *frecR_le_rnk_names* :
assumes *frecR*(*x*,*y*)

shows *rank_names*(x)≤*rank_names*(y)
proof -
obtain *a b c d* **where**
H: *a* = *name1*(x) *b* = *name2*(x)
c = *name1*(y) *d* = *name2*(y)
(a ∈ *domain*(c)∪*domain*(d) ∧ (b=c ∨ b = d)) ∨ (a = c ∧ b ∈ *domain*(d))
using *assms* **unfolding** *frecR_def* **by** *force*
then
consider
(m) a ∈ *domain*(c) ∧ (b = c ∨ b = d)
| (n) a ∈ *domain*(d) ∧ (b = c ∨ b = d)
| (o) b ∈ *domain*(d) ∧ a = c
by *auto*
then show *?thesis* **proof**(*cases*)
case *m*
then
have *rank*(a) < *rank*(c)
using *eclose_rank_lt* *in_dom_in_eclose* **by** *simp*
with ‹*rank*(a) < *rank*(c)› *H m*
show *?thesis* **unfolding** *rank_names_def* **using** *Ord_rank max_cong max_cong2* *leI* **by** *auto*
next
case *n*
then
have *rank*(a) < *rank*(d)
using *eclose_rank_lt in_dom_in_eclose* **by** *simp*
with ‹*rank*(a) < *rank*(d)› *H n*
show *?thesis* **unfolding** *rank_names_def*
using *Ord_rank max_cong2 max_cong max_commutes*[*of rank*(c) *rank*(d)] *leI* **by** *auto*
next
case *o*
then
have *rank*(b) < *rank*(d) (**is** *?b* < *?d*) *rank*(a) = *rank*(c) (**is** *?a* = _)
using *eclose_rank_lt in_dom_in_eclose* **by** *simp_all*
with *H*
show *?thesis* **unfolding** *rank_names_def*
using *Ord_rank max_commutes max_cong2*[*OF leI*[*OF* ‹*?b* < *?d*›], *of ?a*] **by** *simp*
qed
qed

definition
Γ :: *i* ⇒ *i* **where**
Γ(x) = *3* ∗∗ *rank_names*(x) ++ *type_form*(x)

lemma Γ*_type* [*TC*]:
shows *Ord*(Γ(x))

unfolding Γ_*def* **by** *simp*

lemma Γ_*mono* :
assumes *frecR*(x,y)
shows Γ(x) < Γ(y)
proof -
have *F*: *type_form*(x) < *3* *type_form*(y) < *3*
using *ltI* **by** *simp_all*
from *assms*
have *A*: *rank_names*(x) ≤ *rank_names*(y) (**is** *?x* ≤ *?y*)
using *frecR_le_rnk_names* **by** *simp*
then
have *Ord*(*?y*) **unfolding** *rank_names_def* **using** *Ord_rank max_def* **by** *simp*
note *leE*[*OF* ‹*?x*≤*?y*›]
then
show *?thesis*
proof(*cases*)
case *1*
then
show *?thesis* **unfolding** Γ_*def* **using** *oadd_lt_mono2* ‹*?x* < *?y*› *F* **by** *auto*
next
case *2*
consider (a) *ftype*(x) = *0* ∧ *ftype*(y) = *1* | (b) *ftype*(x) = *1* ∧ *ftype*(y) = *0*
using *frecR_ftypeD*[*OF* ‹*frecR*(x,y)›] **by** *auto*
then show *?thesis* **proof**(*cases*)
case b
then
have *type_form*(y) = *1*
using *type_form_def* **by** *simp*
from b
have *H*: *name2*(x) = *name1*(y) ∨ *name2*(x) = *name2*(y) (**is** *?τ* = *?σ′* ∨ *?τ* = *?τ′*)
name1(x) ∈ *domain*(*name1*(y)) ∪ *domain*(*name2*(y))
(**is** *?σ* ∈ *domain*(*?σ′*) ∪ *domain*(*?τ′*))
using *assms* **unfolding** *type_form_def frecR_def* **by** *auto*
then
have *E*: *rank*(*?τ*) = *rank*(*?σ′*) ∨ *rank*(*?τ*) = *rank*(*?τ′*) **by** *auto*
from *H*
consider (a) *rank*(*?σ*) < *rank*(*?σ′*) | (b) *rank*(*?σ*) < *rank*(*?τ′*)
using *eclose_rank_lt in_dom_in_eclose* **by** *force*
then
have *rank*(*?σ*) < *rank*(*?τ*) **proof** (*cases*)
case a
with ‹*rank_names*(x) = *rank_names*(y) ›
show *?thesis* **unfolding** *rank_names_def mtype_form_def type_form_def* **using** *max_D2*[*OF E a*]
E assms Ord_rank **by** *simp*
next

case *b*
with ⟨*rank_names*(x) = *rank_names*(y)⟩
show *?thesis* **unfolding** *rank_names_def mtype_form_def type_form_def*
using *max_D2*[*OF* _ *b*] *max_commutes E assms Ord_rank disj_commute* **by** *auto*
qed
with *b*
have *type_form*(x) = *0* **unfolding** *type_form_def mtype_form_def* **by** *simp*
with ⟨*rank_names*(x) = *rank_names*(y)⟩ ⟨*type_form*(y) = *1*⟩ ⟨*type_form*(x) = *0*⟩
show *?thesis*
unfolding Γ*_def* **by** *auto*
next
case *a*
then
have *name1*(x) = *name1*(y) (**is** *?σ* = *?σ'*)
name2(x) ∈ *domain*(*name2*(y)) (**is** *?τ* ∈ *domain*(*?τ'*))
type_form(x) = *1*
using *assms* **unfolding** *type_form_def frecR_def* **by** *auto*
then
have *rank*(*?σ*) = *rank*(*?σ'*) *rank*(*?τ*) < *rank*(*?τ'*)
using *eclose_rank_lt in_dom_in_eclose* **by** *simp_all*
with ⟨*rank_names*(x) = *rank_names*(y)⟩
have *rank*(*?τ'*) ≤ *rank*(*?σ'*)
unfolding *rank_names_def* **using** *Ord_rank max_D1* **by** *simp*
with *a*
have *type_form*(y) = *2*
unfolding *type_form_def mtype_form_def* **using** *not_lt_iff_le assms* **by** *simp*
with ⟨*rank_names*(x) = *rank_names*(y)⟩ ⟨*type_form*(y) = *2*⟩ ⟨*type_form*(x) = *1*⟩
show *?thesis*
unfolding Γ*_def* **by** *auto*
qed
qed
qed

definition
frecrel :: *i* ⇒ *i* **where**
frecrel(*A*) ≡ *Rrel*(*frecR*,*A*)

lemma *frecrelI* :
assumes $x \in A$ $y{\in}A$ *frecR*(x,y)
shows ⟨x,y⟩∈*frecrel*(*A*)
using *assms* **unfolding** *frecrel_def Rrel_def* **by** *auto*

lemma *frecrelD* :
assumes ⟨x,y⟩ ∈ *frecrel*(*A1*×*A2*×*A3*×*A4*)
shows *ftype*(x) ∈ *A1* *ftype*(x) ∈ *A1*
name1(x) ∈ *A2* *name1*(y) ∈ *A2* *name2*(x) ∈ *A3* *name2*(x) ∈ *A3*

cond_of(*x*) ∈ *A4* *cond_of*(*y*) ∈ *A4*
frecR(*x*,*y*)
using *assms* **unfolding** *frecrel_def Rrel_def ftype_def* **by** (*auto simp add*:*components_simp*)

lemma *wf_frecrel* :
shows *wf*(*frecrel*(*A*))
proof -
have *frecrel*(*A*) ⊆ *measure*(*A*,Γ)
unfolding *frecrel_def Rrel_def measure_def*
using Γ*_mono* **by** *force*
then show *?thesis* **using** *wf_subset wf_measure* **by** *auto*
qed

lemma *core_induction_aux*:
fixes *A1 A2* :: *i*
assumes
Transset(*A1*)
⋀τ ϑ *p*. *p* ∈ *A2* ⟹ ⟦⋀*q* σ. ⟦ *q*∈*A2* ; σ∈*domain*(ϑ)⟧ ⟹ *Q*(*0*,τ,σ,*q*)⟧ ⟹ *Q*(*1*,τ,ϑ,*p*)
⋀τ ϑ *p*. *p* ∈ *A2* ⟹ ⟦⋀*q* σ. ⟦ *q*∈*A2* ; σ∈*domain*(τ) ∪ *domain*(ϑ)⟧ ⟹ *Q*(*1*,σ,τ,*q*) ∧ *Q*(*1*,σ,ϑ,*q*)⟧ ⟹ *Q*(*0*,τ,ϑ,*p*)
shows *a*∈*2*×*A1*×*A1*×*A2* ⟹ *Q*(*ftype*(*a*),*name1*(*a*),*name2*(*a*),*cond_of*(*a*))
proof (*induct a rule*:*wf_induct*[*OF wf_frecrel*[*of 2*×*A1*×*A1*×*A2*]])
case (*1 x*)
let *?*τ = *name1*(*x*)
let *?*ϑ = *name2*(*x*)
let *?D* = *2*×*A1*×*A1*×*A2*
assume *x* ∈ *?D*
then
have *cond_of*(*x*)∈*A2*
by (*auto simp add*:*components_simp*)
from ‹*x*∈*?D*›
consider (*eq*) *ftype*(*x*)=*0* | (*mem*) *ftype*(*x*)=*1*
by (*auto simp add*:*components_simp*)
then
show *?case*
proof *cases*
case *eq*
then
have *Q*(*1*, σ, *?*τ, *q*) ∧ *Q*(*1*, σ, *?*ϑ, *q*) **if** σ ∈ *domain*(*?*τ) ∪ *domain*(*?*ϑ) **and** *q*∈*A2* **for** *q* σ
proof -
from *1*
have *A*: *?*τ∈*A1* *?*ϑ∈*A1* *?*τ∈*eclose*(*A1*) *?*ϑ∈*eclose*(*A1*)
using *arg_into_eclose* **by** (*auto simp add*:*components_simp*)
with ‹*Transset*(*A1*)› *that*(*1*)
have σ∈*eclose*(*?*τ) ∪ *eclose*(*?*ϑ)
using *in_dom_in_eclose* **by** *auto*
then

```
      have σ∈A1
        using mem_eclose_subset[OF ‹?τ∈A1›] mem_eclose_subset[OF ‹?ϑ∈A1›]
          Transset_eclose_eq_arg[OF ‹Transset(A1)›]
        by auto
      with ‹q∈A2› ‹?ϑ ∈ A1› ‹cond_of(x)∈A2› ‹?τ∈A1›
      have frecR(⟨1, σ, ?τ, q⟩, x) (is frecR(?T,_))
        frecR(⟨1, σ, ?ϑ, q⟩, x) (is frecR(?U,_))
        using frecRI1'[OF that(1)] frecR_DI ‹ftype(x) = 0›
          frecRI2'[OF that(1)]
        by (auto simp add:components_simp)
      with ‹x∈?D› ‹σ∈A1› ‹q∈A2›
      have ⟨?T,x⟩∈ frecrel(?D) ⟨?U,x⟩∈ frecrel(?D)
      using frecrelI[of ?T ?D x] frecrelI[of ?U ?D x] by (auto simp add:components_simp)
      with ‹q∈A2› ‹σ∈A1› ‹?τ∈A1› ‹?ϑ∈A1›
      have Q(1, σ, ?τ, q) using 1 by (force simp add:components_simp)
      moreover from ‹q∈A2› ‹σ∈A1› ‹?τ∈A1› ‹?ϑ∈A1› ‹⟨?U,x⟩∈ frecrel(?D)›
      have Q(1, σ, ?ϑ, q) using 1 by (force simp add:components_simp)
      ultimately
      show ?thesis using A by simp
    qed
    then show ?thesis using assms(3) ‹ftype(x) = 0› ‹cond_of(x)∈A2› by auto
  next
    case mem
    have Q(0, ?τ, σ, q) if σ ∈ domain(?ϑ) and q∈A2 for q σ
    proof -
      from 1 assms
      have ?τ∈A1 ?ϑ∈A1 cond_of(x)∈A2 ?τ∈eclose(A1) ?ϑ∈eclose(A1)
        using arg_into_eclose by (auto simp add:components_simp)
      with ‹Transset(A1)› that(1)
      have σ∈ eclose(?ϑ)
        using in_dom_in_eclose by auto
      then
      have σ∈A1
      using mem_eclose_subset[OF ‹?ϑ∈A1›] Transset_eclose_eq_arg[OF ‹Transset(A1)›]
        by auto
      with ‹q∈A2› ‹?ϑ ∈ A1› ‹cond_of(x)∈A2› ‹?τ∈A1›
      have frecR(⟨0, ?τ, σ, q⟩, x) (is frecR(?T,_))
        using frecRI3'[OF that(1)] frecR_DI ‹ftype(x) = 1›
        by (auto simp add:components_simp)
      with ‹x∈?D› ‹σ∈A1› ‹q∈A2› ‹?τ∈A1›
      have ⟨?T,x⟩∈ frecrel(?D) ?T∈?D
        using frecrelI[of ?T ?D x] by (auto simp add:components_simp)
      with ‹q∈A2› ‹σ∈A1› ‹?τ∈A1› ‹?ϑ∈A1› 1
      show ?thesis by (force simp add:components_simp)
    qed
    then show ?thesis using assms(2) ‹ftype(x) = 1› ‹cond_of(x)∈A2› by auto
  qed
qed
```

lemma *def_frecrel* : *frecrel*(*A*) = {*z*∈*A*×*A*. ∃ *x y*. *z* = ⟨*x*, *y*⟩ ∧ *frecR*(*x*,*y*)}
unfolding *frecrel_def Rrel_def* **..**

lemma *frecrel_fst_snd*:
frecrel(*A*) = {*z* ∈ *A*×*A* .
ftype(*fst*(*z*)) = *1* ∧
ftype(*snd*(*z*)) = *0* ∧ *name1*(*fst*(*z*)) ∈ *domain*(*name1*(*snd*(*z*))) ∪ *domain*(*name2*(*snd*(*z*))) ∧
(*name2*(*fst*(*z*)) = *name1*(*snd*(*z*)) ∨ *name2*(*fst*(*z*)) = *name2*(*snd*(*z*)))
∨ (*ftype*(*fst*(*z*)) = *0* ∧
ftype(*snd*(*z*)) = *1* ∧ *name1*(*fst*(*z*)) = *name1*(*snd*(*z*)) ∧ *name2*(*fst*(*z*)) ∈ *domain*(*name2*(*snd*(*z*))))}
unfolding *def_frecrel frecR_def*
by (*intro equalityI subsetI CollectI*; *elim CollectE*; *auto*)

end

17 Arities of internalized formulas

theory *Arities*
imports *FrecR*
begin

lemma *arity_upair_fm* : ⟦ *t1*∈*nat* ; *t2*∈*nat* ; *up*∈*nat* ⟧ ⟹
arity(*upair_fm*(*t1*,*t2*,*up*)) = ⋃ {*succ*(*t1*),*succ*(*t2*),*succ*(*up*)}
unfolding *upair_fm_def*
using *nat_union_abs1 nat_union_abs2 pred_Un*
by *auto*

lemma *arity_pair_fm* : ⟦ *t1*∈*nat* ; *t2*∈*nat* ; *p*∈*nat* ⟧ ⟹
arity(*pair_fm*(*t1*,*t2*,*p*)) = ⋃ {*succ*(*t1*),*succ*(*t2*),*succ*(*p*)}
unfolding *pair_fm_def*
using *arity_upair_fm nat_union_abs1 nat_union_abs2 pred_Un*
by *auto*

lemma *arity_composition_fm* :
⟦ *r*∈*nat* ; *s*∈*nat* ; *t*∈*nat* ⟧ ⟹ *arity*(*composition_fm*(*r*,*s*,*t*)) = ⋃ {*succ*(*r*), *succ*(*s*), *succ*(*t*)}
unfolding *composition_fm_def*
using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_domain_fm* :
⟦ *r*∈*nat* ; *z*∈*nat* ⟧ ⟹ *arity*(*domain_fm*(*r*,*z*)) = *succ*(*r*) ∪ *succ*(*z*)
unfolding *domain_fm_def*
using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_range_fm* :
⟦ $r \in nat$; $z \in nat$ ⟧ ⟹ $arity(range\_fm(r,z)) = succ(r) \cup succ(z)$
unfolding *range_fm_def*
using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_union_fm* :
⟦ $x \in nat$; $y \in nat$; $z \in nat$ ⟧ ⟹ $arity(union\_fm(x,y,z)) = \bigcup \{succ(x), succ(y), succ(z)\}$
unfolding *union_fm_def*
using *nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_image_fm* :
⟦ $x \in nat$; $y \in nat$; $z \in nat$ ⟧ ⟹ $arity(image\_fm(x,y,z)) = \bigcup \{succ(x), succ(y), succ(z)\}$
unfolding *image_fm_def*
using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_pre_image_fm* :
⟦ $x \in nat$; $y \in nat$; $z \in nat$ ⟧ ⟹ $arity(pre\_image\_fm(x,y,z)) = \bigcup \{succ(x), succ(y), succ(z)\}$
unfolding *pre_image_fm_def*
using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_big_union_fm* :
⟦ $x \in nat$; $y \in nat$ ⟧ ⟹ $arity(big\_union\_fm(x,y)) = succ(x) \cup succ(y)$
unfolding *big_union_fm_def*
using *nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_fun_apply_fm* :
⟦ $x \in nat$; $y \in nat$; $f \in nat$ ⟧ ⟹
$arity(fun\_apply\_fm(f,x,y)) = succ(f) \cup succ(x) \cup succ(y)$
unfolding *fun_apply_fm_def*
using *arity_upair_fm arity_image_fm arity_big_union_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_field_fm* :
⟦ $r \in nat$; $z \in nat$ ⟧ ⟹ $arity(field\_fm(r,z)) = succ(r) \cup succ(z)$
unfolding *field_fm_def*
using *arity_pair_fm arity_domain_fm arity_range_fm arity_union_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_empty_fm* :

⟦ r∈nat ⟧ ⟹ *arity*(*empty_fm*(*r*)) = *succ*(*r*)
unfolding *empty_fm_def*
using *nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *simp*

lemma *arity_succ_fm* :
⟦x∈nat;y∈nat⟧ ⟹ *arity*(*succ_fm*(*x*,*y*)) = *succ*(*x*) ∪ *succ*(*y*)
unfolding *succ_fm_def cons_fm_def*
using *arity_upair_fm arity_union_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *number1arity__fm* :
⟦ r∈nat ⟧ ⟹ *arity*(*number1_fm*(*r*)) = *succ*(*r*)
unfolding *number1_fm_def*
using *arity_empty_fm arity_succ_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *simp*

lemma *arity_function_fm* :
⟦ r∈nat ⟧ ⟹ *arity*(*function_fm*(*r*)) = *succ*(*r*)
unfolding *function_fm_def*
using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *simp*

lemma *arity_relation_fm* :
⟦ r∈nat ⟧ ⟹ *arity*(*relation_fm*(*r*)) = *succ*(*r*)
unfolding *relation_fm_def*
using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *simp*

lemma *arity_restriction_fm* :
⟦ r∈nat ; z∈nat ; A∈nat ⟧ ⟹ *arity*(*restriction_fm*(*A*,*z*,*r*)) = *succ*(*A*) ∪ *succ*(*r*) ∪ *succ*(*z*)
unfolding *restriction_fm_def*
using *arity_pair_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_typed_function_fm* :
⟦ x∈nat ; y∈nat ; f∈nat ⟧ ⟹
arity(*typed_function_fm*(*f*,*x*,*y*)) = ⋃ {*succ*(*f*), *succ*(*x*), *succ*(*y*)}
unfolding *typed_function_fm_def*
using *arity_pair_fm arity_relation_fm arity_function_fm arity_domain_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_subset_fm* :
⟦x∈nat ; y∈nat⟧ ⟹ *arity*(*subset_fm*(*x*,*y*)) = *succ*(*x*) ∪ *succ*(*y*)

unfolding *subset_fm_def*
using *nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_transset_fm* :
⟦*x*∈*nat*⟧ ⟹ *arity*(*transset_fm*(*x*)) = *succ*(*x*)
unfolding *transset_fm_def*
using *arity_subset_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_ordinal_fm* :
⟦*x*∈*nat*⟧ ⟹ *arity*(*ordinal_fm*(*x*)) = *succ*(*x*)
unfolding *ordinal_fm_def*
using *arity_transset_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_limit_ordinal_fm* :
⟦*x*∈*nat*⟧ ⟹ *arity*(*limit_ordinal_fm*(*x*)) = *succ*(*x*)
unfolding *limit_ordinal_fm_def*
using *arity_ordinal_fm arity_succ_fm arity_empty_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_finite_ordinal_fm* :
⟦*x*∈*nat*⟧ ⟹ *arity*(*finite_ordinal_fm*(*x*)) = *succ*(*x*)
unfolding *finite_ordinal_fm_def*
using *arity_ordinal_fm arity_limit_ordinal_fm arity_succ_fm arity_empty_fm*
nat_union_abs2 pred_Un_distrib
by *auto*

lemma *arity_omega_fm* :
⟦*x*∈*nat*⟧ ⟹ *arity*(*omega_fm*(*x*)) = *succ*(*x*)
unfolding *omega_fm_def*
using *arity_limit_ordinal_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_cartprod_fm* :
⟦ *A*∈*nat* ; *B*∈*nat* ; *z*∈*nat* ⟧ ⟹ *arity*(*cartprod_fm*(*A*,*B*,*z*)) = *succ*(*A*) ∪ *succ*(*B*)
∪ *succ*(*z*)
unfolding *cartprod_fm_def*
using *arity_pair_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_fst_fm* :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*fst_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
unfolding *fst_fm_def*
using *arity_pair_fm arity_empty_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_snd_fm* :

⟦x∈nat ; t∈nat⟧ ⟹ *arity*(*snd_fm*(x,t)) = *succ*(x) ∪ *succ*(t)
unfolding *snd_fm_def*
using *arity_pair_fm arity_empty_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_snd_snd_fm* :
⟦x∈nat ; t∈nat⟧ ⟹ *arity*(*snd_snd_fm*(x,t)) = *succ*(x) ∪ *succ*(t)
unfolding *snd_snd_fm_def hcomp_fm_def*
using *arity_snd_fm arity_empty_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_ftype_fm* :
⟦x∈nat ; t∈nat⟧ ⟹ *arity*(*ftype_fm*(x,t)) = *succ*(x) ∪ *succ*(t)
unfolding *ftype_fm_def*
using *arity_fst_fm*
by *auto*

lemma *name1arity__fm* :
⟦x∈nat ; t∈nat⟧ ⟹ *arity*(*name1_fm*(x,t)) = *succ*(x) ∪ *succ*(t)
unfolding *name1_fm_def hcomp_fm_def*
using *arity_fst_fm arity_snd_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *name2arity__fm* :
⟦x∈nat ; t∈nat⟧ ⟹ *arity*(*name2_fm*(x,t)) = *succ*(x) ∪ *succ*(t)
unfolding *name2_fm_def hcomp_fm_def*
using *arity_fst_fm arity_snd_snd_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_cond_of_fm* :
⟦x∈nat ; t∈nat⟧ ⟹ *arity*(*cond_of_fm*(x,t)) = *succ*(x) ∪ *succ*(t)
unfolding *cond_of_fm_def hcomp_fm_def*
using *arity_snd_fm arity_snd_snd_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_singleton_fm* :
⟦x∈nat ; t∈nat⟧ ⟹ *arity*(*singleton_fm*(x,t)) = *succ*(x) ∪ *succ*(t)
unfolding *singleton_fm_def cons_fm_def*
using *arity_union_fm arity_upair_fm arity_empty_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_Memrel_fm* :
⟦x∈nat ; t∈nat⟧ ⟹ *arity*(*Memrel_fm*(x,t)) = *succ*(x) ∪ *succ*(t)
unfolding *Memrel_fm_def*
using *arity_pair_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_quasinat_fm* :
⟦x∈nat⟧ ⟹ *arity*(*quasinat_fm*(x)) = *succ*(x)

unfolding *quasinat_fm_def cons_fm_def*
using *arity_succ_fm arity_empty_fm*
nat_union_abs2 pred_Un_distrib
by *auto*

lemma *arity_is_recfun_fm* :
$[\![p \in formula ; v \in nat ; n \in nat; Z \in nat; i \in nat]\!] \Longrightarrow arity(p) = i \Longrightarrow$
$arity(is\_recfun\_fm(p,v,n,Z)) = succ(v) \cup succ(n) \cup succ(Z) \cup pred(pred(pred(pred(i))))$
unfolding *is_recfun_fm_def*
using *arity_upair_fm arity_pair_fm arity_pre_image_fm arity_restriction_fm*
nat_union_abs2 pred_Un_distrib
by *auto*

lemma *arity_is_wfrec_fm* :
$[\![p \in formula ; v \in nat ; n \in nat; Z \in nat ; i \in nat]\!] \Longrightarrow arity(p) = i \Longrightarrow$
$arity(is\_wfrec\_fm(p,v,n,Z)) = succ(v) \cup succ(n) \cup succ(Z) \cup pred(pred(pred(pred(pred(i)))))$
unfolding *is_wfrec_fm_def*
using *arity_succ_fm arity_is_recfun_fm*
nat_union_abs2 pred_Un_distrib
by *auto*

lemma *arity_is_nat_case_fm* :
$[\![p \in formula ; v \in nat ; n \in nat; Z \in nat; i \in nat]\!] \Longrightarrow arity(p) = i \Longrightarrow$
$arity(is\_nat\_case\_fm(v,p,n,Z)) = succ(v) \cup succ(n) \cup succ(Z) \cup pred(pred(i))$
unfolding *is_nat_case_fm_def*
using *arity_succ_fm arity_empty_fm arity_quasinat_fm*
nat_union_abs2 pred_Un_distrib
by *auto*

lemma *arity_iterates_MH_fm* :
assumes $isF \in formula$ $v \in nat$ $n \in nat$ $g \in nat$ $z \in nat$ $i \in nat$
$arity(isF) = i$
shows $arity(iterates\_MH\_fm(isF,v,n,g,z)) =$
$succ(v) \cup succ(n) \cup succ(g) \cup succ(z) \cup pred(pred(pred(pred(i))))$
proof -
let $?\varphi = Exists(And(fun\_apply\_fm(succ(succ(succ(g))), 2, 0), Forall(Implies(Equal(0,$
$2), isF))))$
let $?ar = succ(succ(succ(g))) \cup pred(pred(i))$
from *assms*
have $arity(?\varphi) = ?ar$ $?\varphi \in formula$
using *arity_fun_apply_fm*
nat_union_abs1 nat_union_abs2 pred_Un_distrib succ_Un_distrib Un_assoc[symmetric]
by *simp_all*
then
show *?thesis*
unfolding *iterates_MH_fm_def*
using *arity_is_nat_case_fm[OF* $\langle ?\varphi \in \_ \rangle$ _ _ _ _ $\langle arity(?\varphi) = \_ \rangle$*] assms pred_succ_eq*
pred_Un_distrib
by *auto*

qed

lemma *arity_is_iterates_fm* :
  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 -
  let *?φ = iterates_MH_fm(p, 7#+v, 2, 1, 0)*
  let *?ψ = is_wfrec_fm(?φ, 0, succ(succ(n)),succ(succ(Z)))*
  from ‹*v∈_*›
  have *arity(?φ) = (8#+v) ∪ pred(pred(pred(pred(i)))) ?φ∈formula*
    using *assms arity_iterates_MH_fm nat_union_abs2*
    by *simp_all*
  then
  have *arity(?ψ) = succ(succ(succ(n))) ∪ succ(succ(succ(Z))) ∪ (3#+v) ∪*
      pred(pred(pred(pred(pred(pred(pred(pred(pred(i)))))))))
    using *assms arity_is_wfrec_fm[OF ‹?φ∈_› _ _ _ _ ‹arity(?φ) = _›] nat_union_abs1*
pred_Un_distrib
    by *auto*
  then
  show *?thesis*
    unfolding *is_iterates_fm_def*
    using *arity_Memrel_fm arity_succ_fm assms nat_union_abs1 pred_Un_distrib*
    by *auto*
qed

lemma *arity_eclose_n_fm* :
  assumes *A∈nat x∈nat t∈nat*
  shows *arity(eclose_n_fm(A,x,t)) = succ(A) ∪ succ(x) ∪ succ(t)*
proof -
  let *?φ = big_union_fm(1,0)*
  have *arity(?φ) = 2 ?φ∈formula*
    using *arity_big_union_fm nat_union_abs2*
    by *simp_all*
  with *assms*
  show *?thesis*
    unfolding *eclose_n_fm_def*
    using *arity_is_iterates_fm[OF ‹?φ∈_› _ _ _,of _ _ _ 2]*
    by *auto*
qed

lemma *arity_mem_eclose_fm* :
  assumes *x∈nat t∈nat*
  shows *arity(mem_eclose_fm(x,t)) = succ(x) ∪ succ(t)*
proof -
  let *?φ=eclose_n_fm(x #+ 2, 1, 0)*
  from ‹*x∈nat*›
  have *arity(?φ) = x#+3*

using *arity_eclose_n_fm nat_union_abs2*
by *simp*
with *assms*
show *?thesis*
unfolding *mem_eclose_fm_def*
using *arity_finite_ordinal_fm nat_union_abs2 pred_Un_distrib*
by *simp*
qed

lemma *arity_is_eclose_fm* :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*is_eclose_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
unfolding *is_eclose_fm_def*
using *arity_mem_eclose_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *eclose_n1arity__fm* :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*eclose_n1_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
unfolding *eclose_n1_fm_def*
using *arity_is_eclose_fm arity_singleton_fm name1arity__fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *eclose_n2arity__fm* :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*eclose_n2_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
unfolding *eclose_n2_fm_def*
using *arity_is_eclose_fm arity_singleton_fm name2arity__fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_ecloseN_fm* :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*ecloseN_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
unfolding *ecloseN_fm_def*
using *eclose_n1arity__fm eclose_n2arity__fm arity_union_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_frecR_fm* :
⟦*a*∈*nat*;*b*∈*nat*⟧ ⟹ *arity*(*frecR_fm*(*a*,*b*)) = *succ*(*a*) ∪ *succ*(*b*)
unfolding *frecR_fm_def*
using *arity_ftype_fm name1arity__fm name2arity__fm arity_domain_fm number1arity__fm arity_empty_fm nat_union_abs2 pred_Un_distrib*
by *auto*

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

end

18 The definition of *forces*

theory *Forces_Definition* **imports** *Arities FrecR Synthetic_Definition* **begin**

This is the core of our development.

18.1 The relation *frecrel*

definition
frecrelP :: [*i*⇒*o*,*i*] ⇒ *o* **where**
frecrelP(*M*,*xy*) ≡ (∃ *x*[*M*]. ∃ *y*[*M*]. *pair*(*M*,*x*,*y*,*xy*) ∧ *is_frecR*(*M*,*x*,*y*))

definition
frecrelP_fm :: *i* ⇒ *i* **where**
frecrelP_fm(*a*) ≡ *Exists*(*Exists*(*And*(*pair_fm*(*1*,*0*,*a*#+*2*),*frecR_fm*(*1*,*0*))))

lemma *arity_frecrelP_fm* :
a∈*nat* ⟹ *arity*(*frecrelP_fm*(*a*)) = *succ*(*a*)
unfolding *frecrelP_fm_def*
using *arity_frecR_fm arity_pair_fm pred_Un_distrib*
by *simp*

lemma *frecrelP_fm_type*[*TC*] :
a∈*nat* ⟹ *frecrelP_fm*(*a*)∈*formula*
unfolding *frecrelP_fm_def* **by** *simp*

lemma *sats_frecrelP_fm* :
assumes *a*∈*nat env*∈*list*(*A*)
shows *sats*(*A*,*frecrelP_fm*(*a*),*env*) ⟷ *frecrelP*(##*A*,*nth*(*a*, *env*))
unfolding *frecrelP_def frecrelP_fm_def*
using *assms* **by** (*auto simp add*:*frecR_fm_iff_sats*[*symmetric*])

lemma *frecrelP_iff_sats*:
assumes
nth(*a*,*env*) = *aa a*∈ *nat* *env* ∈ *list*(*A*)
shows
frecrelP(##*A*,*aa*) ⟷ *sats*(*A*, *frecrelP_fm*(*a*), *env*)
using *assms*
by (*simp add*:*sats_frecrelP_fm*)

definition
is_frecrel :: [*i*⇒*o*,*i*,*i*] ⇒ *o* **where**
is_frecrel(*M*,*A*,*r*) ≡ ∃ *A2*[*M*]. *cartprod*(*M*,*A*,*A*,*A2*) ∧ *is_Collect*(*M*,*A2*, *frecrelP*(*M*)
,*r*)

definition
frecrel_fm :: [*i*,*i*] ⇒ *i* **where**
frecrel_fm(*a*,*r*) ≡ *Exists*(*And*(*cartprod_fm*(*a*#+*1*,*a*#+*1*,*0*),*Collect_fm*(*0*,*frecrelP_fm*(*0*),*r*#+*1*)))

lemma *frecrel_fm_type*[*TC*] :

$[\![a \in nat; b \in nat]\!] \Longrightarrow$ *frecrel_fm(a,b)∈formula*
unfolding *frecrel_fm_def* **by** *simp*

lemma *arity_frecrel_fm* :
assumes *a∈nat b∈nat*
shows *arity(frecrel_fm(a,b)) = succ(a) ∪ succ(b)*
unfolding *frecrel_fm_def*
using *assms arity_Collect_fm arity_cartprod_fm arity_frecrelP_fm pred_Un_distrib*
by *auto*

lemma *sats_frecrel_fm* :
assumes
a∈nat r∈nat env∈list(A)
shows
sats(A,frecrel_fm(a,r),env)
⟷ is_frecrel(##A,nth(a, env),nth(r, env))
unfolding *is_frecrel_def frecrel_fm_def*
using *assms*
by *(simp add:sats_Collect_fm sats_frecrelP_fm)*

lemma *is_frecrel_iff_sats*:
assumes
nth(a,env) = aa nth(r,env) = rr a∈ nat r∈ nat env ∈ list(A)
shows
is_frecrel(##A, aa,rr) ⟷ sats(A, frecrel_fm(a,r), env)
using *assms*
by *(simp add:sats_frecrel_fm)*

definition
names_below :: *i ⇒ i ⇒ i* **where**
names_below(P,x) ≡ 2×ecloseN(x)×ecloseN(x)×P

lemma *names_belowsD*:
assumes *x ∈ names_below(P,z)*
obtains *f n1 n2 p* **where**
x = ⟨f,n1,n2,p⟩ f∈2 n1∈ecloseN(z) n2∈ecloseN(z) p∈P
using *assms* **unfolding** *names_below_def* **by** *auto*

definition
is_names_below :: *[i⇒o,i,i,i] ⇒ o* **where**
is_names_below(M,P,x,nb) ≡ ∃ p1[M]. ∃ p0[M]. ∃ t[M]. ∃ ec[M].
is_ecloseN(M,ec,x) ∧ number2(M,t) ∧ cartprod(M,ec,P,p0) ∧ cartprod(M,ec,p0,p1)
∧ cartprod(M,t,p1,nb)

definition
number2_fm :: *i⇒i* **where**
number2_fm(a) ≡ Exists(And(number1_fm(0), succ_fm(0,succ(a))))

lemma *number2_fm_type*[*TC*] :
  a∈*nat* ⟹ *number2_fm*(*a*) ∈ *formula*
  unfolding *number2_fm_def* **by** *simp*

lemma *number2arity__fm* :
  a∈*nat* ⟹ *arity*(*number2_fm*(*a*)) = *succ*(*a*)
  unfolding *number2_fm_def*
  using *number1arity__fm arity_succ_fm nat_union_abs2 pred_Un_distrib*
  by *simp*

lemma *sats_number2_fm* [*simp*]:
  ⟦ *x* ∈ *nat*; *env* ∈ *list*(*A*) ⟧
    ⟹ *sats*(*A*, *number2_fm*(*x*), *env*) ⟷ *number2*(##*A*, *nth*(*x*,*env*))
  by (*simp add*: *number2_fm_def number2_def*)

definition
  is_names_below_fm :: [*i*,*i*,*i*] ⇒ *i* **where**
  is_names_below_fm(*P*,*x*,*nb*) ≡ *Exists*(*Exists*(*Exists*(*Exists*(
            And(*ecloseN_fm*(*0*,*x* #+ *4*),*And*(*number2_fm*(*1*),
        And(*cartprod_fm*(*0*,*P* #+ *4*,*2*),*And*(*cartprod_fm*(*0*,*2*,*3*),*cartprod_fm*(*1*,*3*,*nb*#+*4*))))))))

lemma *arity_is_names_below_fm* :
  ⟦*P*∈*nat*;*x*∈*nat*;*nb*∈*nat*⟧ ⟹ *arity*(*is_names_below_fm*(*P*,*x*,*nb*)) = *succ*(*P*) ∪ *succ*(*x*)
∪ *succ*(*nb*)
  unfolding *is_names_below_fm_def*
  using *arity_cartprod_fm number2arity__fm arity_ecloseN_fm nat_union_abs2 pred_Un_distrib*
  by *auto*

lemma *is_names_below_fm_type*[*TC*]:
  ⟦*P*∈*nat*;*x*∈*nat*;*nb*∈*nat*⟧ ⟹ *is_names_below_fm*(*P*,*x*,*nb*)∈*formula*
  unfolding *is_names_below_fm_def* **by** *simp*

lemma *sats_is_names_below_fm* :
  assumes
    P∈*nat x*∈*nat nb*∈*nat env*∈*list*(*A*)
  shows
    sats(*A*,*is_names_below_fm*(*P*,*x*,*nb*),*env*)
    ⟷ *is_names_below*(##*A*,*nth*(*P*, *env*),*nth*(*x*, *env*),*nth*(*nb*, *env*))
  unfolding *is_names_below_fm_def is_names_below_def* **using** *assms* **by** *simp*

definition
  is_tuple :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
  is_tuple(*M*,*z*,*t1*,*t2*,*p*,*t*) ≡ ∃ *t1t2p*[*M*]. ∃ *t2p*[*M*]. *pair*(*M*,*t2*,*p*,*t2p*) ∧ *pair*(*M*,*t1*,*t2p*,*t1t2p*)
∧
                                        pair(*M*,*z*,*t1t2p*,*t*)

definition
is_tuple_fm :: [*i*,*i*,*i*,*i*,*i*] ⇒ *i* **where**
is_tuple_fm(*z*,*t1*,*t2*,*p*,*tup*) = *Exists*(*Exists*(*And*(*pair_fm*(*t2* #+ *2*,*p* #+ *2*,*0*),
And(*pair_fm*(*t1* #+ *2*,*0*,*1*),*pair_fm*(*z* #+ *2*,*1*,*tup* #+ *2*)))))

lemma *arity_is_tuple_fm* : ⟦ *z*∈*nat* ; *t1*∈*nat* ; *t2*∈*nat* ; *p*∈*nat* ; *tup*∈*nat* ⟧ ⟹
arity(*is_tuple_fm*(*z*,*t1*,*t2*,*p*,*tup*)) = ⋃ {*succ*(*z*),*succ*(*t1*),*succ*(*t2*),*succ*(*p*),*succ*(*tup*)}
unfolding *is_tuple_fm_def*
using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *is_tuple_fm_type*[*TC*] :
z∈*nat* ⟹ *t1*∈*nat* ⟹ *t2*∈*nat* ⟹ *p*∈*nat* ⟹ *tup*∈*nat* ⟹ *is_tuple_fm*(*z*,*t1*,*t2*,*p*,*tup*)∈*formula*
unfolding *is_tuple_fm_def* **by** *simp*

lemma *sats_is_tuple_fm* :
assumes
z∈*nat* *t1*∈*nat* *t2*∈*nat* *p*∈*nat* *tup*∈*nat* *env*∈*list*(*A*)
shows
sats(*A*,*is_tuple_fm*(*z*,*t1*,*t2*,*p*,*tup*),*env*)
⟷ *is_tuple*(##*A*,*nth*(*z*, *env*),*nth*(*t1*, *env*),*nth*(*t2*, *env*),*nth*(*p*, *env*),*nth*(*tup*, *env*))
unfolding *is_tuple_def is_tuple_fm_def* **using** *assms* **by** *simp*

lemma *is_tuple_iff_sats*:
assumes
nth(*a*,*env*) = *aa* *nth*(*b*,*env*) = *bb* *nth*(*c*,*env*) = *cc* *nth*(*d*,*env*) = *dd* *nth*(*e*,*env*) = *ee*
a∈*nat* *b*∈*nat* *c*∈*nat* *d*∈*nat* *e*∈*nat* *env* ∈ *list*(*A*)
shows
is_tuple(##*A*,*aa*,*bb*,*cc*,*dd*,*ee*) ⟷ *sats*(*A*, *is_tuple_fm*(*a*,*b*,*c*,*d*,*e*), *env*)
using *assms* **by** (*simp add*: *sats_is_tuple_fm*)

18.2 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*))

definition
is_eq_case :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_eq_case(*M*,*t1*,*t2*,*p*,*P*,*leq*,*f*) ≡
∀ *s*[*M*]. (∃ *d*[*M*]. *is_domain*(*M*,*t1*,*d*) ∧ *s*∈*d*) ∨ (∃ *d*[*M*]. *is_domain*(*M*,*t2*,*d*) ∧ *s*∈*d*)
⟶ (∀ *q*[*M*]. *q*∈*P* ∧ (∃ *qp*[*M*]. *pair*(*M*,*q*,*p*,*qp*) ∧ *qp*∈*leq*) ⟶
(∃ *ost1q*[*M*]. ∃ *ost2q*[*M*]. ∃ *o*[*M*]. ∃ *vf1*[*M*]. ∃ *vf2*[*M*].

is_tuple(*M*,*o*,*s*,*t1*,*q*,*ost1q*) ∧
is_tuple(*M*,*o*,*s*,*t2*,*q*,*ost2q*) ∧ *number1*(*M*,*o*) ∧
fun_apply(*M*,*f*,*ost1q*,*vf1*) ∧ *fun_apply*(*M*,*f*,*ost2q*,*vf2*) ∧
(*vf1* = *o* ⟷ *vf2* = *o*)))

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*)

definition
is_mem_case :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_mem_case(*M*,*t1*,*t2*,*p*,*P*,*leq*,*f*) ≡ ∀ *v*[*M*]. ∀ *vp*[*M*]. *v*∈*P* ∧ *pair*(*M*,*v*,*p*,*vp*) ∧ *vp*∈*leq* ⟶
(∃ *q*[*M*]. ∃ *s*[*M*]. ∃ *r*[*M*]. ∃ *qv*[*M*]. ∃ *sr*[*M*]. ∃ *qr*[*M*]. ∃ *z*[*M*]. ∃ *zt1sq*[*M*]. ∃ *o*[*M*].
r∈ *P* ∧ *q*∈*P* ∧ *pair*(*M*,*q*,*v*,*qv*) ∧ *pair*(*M*,*s*,*r*,*sr*) ∧ *pair*(*M*,*q*,*r*,*qr*) ∧
empty(*M*,*z*) ∧ *is_tuple*(*M*,*z*,*t1*,*s*,*q*,*zt1sq*) ∧
number1(*M*,*o*) ∧ *qv*∈*leq* ∧ *sr*∈*t2* ∧ *qr*∈*leq* ∧ *fun_apply*(*M*,*f*,*zt1sq*,*o*))

schematic_goal *sats_is_mem_case_fm_auto*:
assumes
n1∈*nat* *n2*∈*nat* *p*∈*nat* *P*∈*nat* *leq*∈*nat* *f*∈*nat* *env*∈*list*(*A*)
shows
is_mem_case(##*A*, *nth*(*n1*, *env*),*nth*(*n2*, *env*),*nth*(*p*, *env*),*nth*(*P*, *env*), *nth*(*leq*, *env*),*nth*(*f*,*env*))
⟷ *sats*(*A*,*?imc_fm*(*n1*,*n2*,*p*,*P*,*leq*,*f*),*env*)
unfolding *is_mem_case_def*
by (*insert assms* ; (*rule sep_rules′ is_tuple_iff_sats* | *simp*)+)

synthesize *mem_case_fm* **from_schematic** *sats_is_mem_case_fm_auto*

lemma *arity_mem_case_fm* :
assumes
n1∈*nat* *n2*∈*nat* *p*∈*nat* *P*∈*nat* *leq*∈*nat* *f*∈*nat*
shows
arity(*mem_case_fm*(*n1*,*n2*,*p*,*P*,*leq*,*f*)) =
succ(*n1*) ∪ *succ*(*n2*) ∪ *succ*(*p*) ∪ *succ*(*P*) ∪ *succ*(*leq*) ∪ *succ*(*f*)
unfolding *mem_case_fm_def*
using *assms arity_pair_fm arity_is_tuple_fm number1arity__fm arity_fun_apply_fm arity_empty_fm*
pred_Un_distrib
by *auto*

schematic_goal *sats_is_eq_case_fm_auto*:
assumes

n1∈nat n2∈nat p∈nat P∈nat leq∈nat f∈nat env∈list(A)
shows
is_eq_case(##A, nth(n1, env),nth(n2, env),nth(p, env),nth(P, env), nth(leq, env),nth(f,env))
⟷ *sats(A,?iec_fm(n1,n2,p,P,leq,f),env)*
unfolding *is_eq_case_def*
by (*insert assms* ; (*rule sep_rules′ is_tuple_iff_sats* | *simp*)+)

synthesize *eq_case_fm* **from_schematic** *sats_is_eq_case_fm_auto*

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)*
unfolding *eq_case_fm_def*
using *assms arity_pair_fm arity_is_tuple_fm number1arity__fm arity_fun_apply_fm arity_empty_fm*
arity_domain_fm pred_Un_distrib
by *auto*

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)*)

definition
is_Hfrc :: [*i⇒o,i,i,i,i*] ⇒ *o* **where**
is_Hfrc(M,P,leq,fnnc,f) ≡
∃ *ft[M]*. ∃ *n1[M]*. ∃ *n2[M]*. ∃ *co[M]*.
co∈P ∧ *is_tuple(M,ft,n1,n2,co,fnnc)* ∧
((*empty(M,ft)* ∧ *is_eq_case(M,n1,n2,co,P,leq,f)*)
∨ (*number1(M,ft)* ∧ *is_mem_case(M,n1,n2,co,P,leq,f)*))

definition
Hfrc_fm :: [*i,i,i,i*] ⇒ *i* **where**
Hfrc_fm(P,leq,fnnc,f) ≡
Exists(Exists(Exists(Exists(
And(Member(0,P #+ 4),And(is_tuple_fm(3,2,1,0,fnnc #+ 4),
Or(And(empty_fm(3),eq_case_fm(2,1,0,P #+ 4,leq #+ 4,f #+ 4)),
And(number1_fm(3),mem_case_fm(2,1,0,P #+ 4,leq #+ 4,f #+ 4)))))))))

declare *Hfrc_fm_def[fm_definitions]*

lemma *Hfrc_fm_type[TC]* :
⟦*P∈nat;leq∈nat;fnnc∈nat;f∈nat*⟧ ⟹ *Hfrc_fm(P,leq,fnnc,f)∈formula*
unfolding *Hfrc_fm_def* **by** *simp*

lemma *arity_Hfrc_fm* :
  assumes
    $P\in nat$ $leq\in nat$ $fnnc\in nat$ $f\in nat$
  shows
    $arity(Hfrc\_fm(P,leq,fnnc,f)) = succ(P) \cup succ(leq) \cup succ(fnnc) \cup succ(f)$
  unfolding *Hfrc_fm_def*
  using *assms arity_is_tuple_fm arity_mem_case_fm arity_eq_case_fm*
    arity_empty_fm number1arity__fm pred_Un_distrib
  by *auto*

lemma *sats_Hfrc_fm*:
  assumes
    $P\in nat$ $leq\in nat$ $fnnc\in nat$ $f\in nat$ $env\in list(A)$
  shows
    $sats(A,Hfrc\_fm(P,leq,fnnc,f),env)$
    $\longleftrightarrow is\_Hfrc(\#\#A,nth(P, env), nth(leq, env), nth(fnnc, env),nth(f, env))$
  unfolding *is_Hfrc_def Hfrc_fm_def*
  using *assms*
  by (*simp add*: *sats_is_tuple_fm eq_case_fm_iff_sats*[*symmetric*] *mem_case_fm_iff_sats*[*symmetric*])

lemma *Hfrc_iff_sats*:
  assumes
    $P\in nat$ $leq\in nat$ $fnnc\in nat$ $f\in nat$ $env\in list(A)$
    $nth(P,env) = PP$ $nth(leq,env) = lleq$ $nth(fnnc,env) = ffnnc$ $nth(f,env) = ff$
  shows
    $is\_Hfrc(\#\#A, PP, lleq,ffnnc,ff)$
    $\longleftrightarrow sats(A,Hfrc\_fm(P,leq,fnnc,f),env)$
  using *assms*
  by (*simp add*:*sats_Hfrc_fm*)

definition
  $is\_Hfrc\_at :: [i\Rightarrow o,i,i,i,i,i] \Rightarrow o$ **where**
  $is\_Hfrc\_at(M,P,leq,fnnc,f,z) \equiv$
      $(empty(M,z) \wedge \neg\ is\_Hfrc(M,P,leq,fnnc,f))$
    $\vee\ (number1(M,z) \wedge is\_Hfrc(M,P,leq,fnnc,f))$

definition
  $Hfrc\_at\_fm :: [i,i,i,i,i] \Rightarrow i$ **where**
  $Hfrc\_at\_fm(P,leq,fnnc,f,z) \equiv Or(And(empty\_fm(z),Neg(Hfrc\_fm(P,leq,fnnc,f))),$
      $And(number1\_fm(z),Hfrc\_fm(P,leq,fnnc,f)))$
declare *Hfrc_at_fm_def*[*fm_definitions*]

lemma *arity_Hfrc_at_fm* :
  assumes
    $P\in nat$ $leq\in nat$ $fnnc\in nat$ $f\in nat$ $z\in nat$
  shows
    $arity(Hfrc\_at\_fm(P,leq,fnnc,f,z)) = succ(P) \cup succ(leq) \cup succ(fnnc) \cup succ(f)$
$\cup\ succ(z)$

unfolding *Hfrc_at_fm_def*
using *assms arity_Hfrc_fm arity_empty_fm number1arity__fm pred_Un_distrib*
by *auto*

lemma *Hfrc_at_fm_type*[*TC*] :
⟦*P*∈*nat*;*leq*∈*nat*;*fnnc*∈*nat*;*f*∈*nat*;*z*∈*nat*⟧ ⟹ *Hfrc_at_fm*(*P*,*leq*,*fnnc*,*f*,*z*)∈*formula*
unfolding *Hfrc_at_fm_def* **by** *simp*

lemma *sats_Hfrc_at_fm*:
assumes
P∈*nat leq*∈*nat fnnc*∈*nat f*∈*nat z*∈*nat env*∈*list*(*A*)
shows
sats(*A*,*Hfrc_at_fm*(*P*,*leq*,*fnnc*,*f*,*z*),*env*)
⟷ *is_Hfrc_at*(##*A*,*nth*(*P*, *env*), *nth*(*leq*, *env*), *nth*(*fnnc*, *env*),*nth*(*f*, *env*),*nth*(*z*, *env*))
unfolding *is_Hfrc_at_def Hfrc_at_fm_def* **using** *assms sats_Hfrc_fm*
by *simp*

lemma *is_Hfrc_at_iff_sats*:
assumes
P∈*nat leq*∈*nat fnnc*∈*nat f*∈*nat z*∈*nat env*∈*list*(*A*)
nth(*P*,*env*) = *PP* *nth*(*leq*,*env*) = *lleq nth*(*fnnc*,*env*) = *ffnnc*
nth(*f*,*env*) = *ff nth*(*z*,*env*) = *zz*
shows
is_Hfrc_at(##*A*, *PP*, *lleq*,*ffnnc*,*ff*,*zz*)
⟷ *sats*(*A*,*Hfrc_at_fm*(*P*,*leq*,*fnnc*,*f*,*z*),*env*)
using *assms* **by** (*simp add*:*sats_Hfrc_at_fm*)

lemma *arity_tran_closure_fm* :
⟦*x*∈*nat*;*f*∈*nat*⟧ ⟹ *arity*(*trans_closure_fm*(*x*,*f*)) = *succ*(*x*) ∪ *succ*(*f*)
unfolding *trans_closure_fm_def*
using *arity_omega_fm arity_field_fm arity_typed_function_fm arity_pair_fm arity_empty_fm arity_fun_apply_fm*
arity_composition_fm arity_succ_fm nat_union_abs2 pred_Un_distrib
by *auto*

18.3 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*)
unfolding *forcerel_fm_def*
using *arity_frecrel_fm arity_tran_closure_fm arity_is_names_below_fm pred_Un_distrib*
by *auto*

lemma *forcerel_fm_type*[*TC*]:
⟦*p*∈*nat*;*x*∈*nat*;*z*∈*nat*⟧ ⟹ *forcerel_fm*(*p*,*x*,*z*)∈*formula*
unfolding *forcerel_fm_def* **by** *simp*

lemma *sats_forcerel_fm*:
assumes
p∈*nat x*∈*nat z*∈*nat env*∈*list*(*A*)
shows
sats(*A*,*forcerel_fm*(*p*,*x*,*z*),*env*) ⟷ *is_forcerel*(##*A*,*nth*(*p*,*env*),*nth*(*x*, *env*),*nth*(*z*, *env*))
proof -
have *sats*(*A*,*trans_closure_fm*(*1*,*z* #+ *2*),*Cons*(*nb*,*Cons*(*r*,*env*))) ⟷
tran_closure(##*A*, *r*, *nth*(*z*, *env*)) **if** *r*∈*A nb*∈*A* **for** *r nb*
using *that assms trans_closure_fm_iff_sats*[*of 1* [*nb*,*r*]@*env* _ *z*#+*2*,*symmetric*]
by *simp*
moreover
have *sats*(*A*, *is_names_below_fm*(*succ*(*succ*(*p*)), *succ*(*succ*(*x*)), *0*), *Cons*(*nb*, *Cons*(*r*, *env*))) ⟷
is_names_below(##*A*, *nth*(*p*,*env*), *nth*(*x*, *env*), *nb*)
if *r*∈*A nb*∈*A* **for** *nb r*
using *assms that sats_is_names_below_fm*[*of p* #+ *2 x* #+ *2 0* [*nb*,*r*]@*env*] **by** *simp*
moreover
have *sats*(*A*, *frecrel_fm*(*0*, *1*), *Cons*(*nb*, *Cons*(*r*, *env*))) ⟷
is_frecrel(##*A*, *nb*, *r*)
if *r*∈*A nb*∈*A* **for** *r nb*
using *assms that sats_frecrel_fm*[*of 0 1* [*nb*,*r*]@*env*] **by** *simp*
ultimately
show *?thesis* **using** *assms* **unfolding** *is_forcerel_def forcerel_fm_def* **by** *simp*
qed

18.4 *frc_at*, forcing for atomic formulas

definition
frc_at :: [*i*,*i*,*i*] ⇒ *i* **where**
frc_at(*P*,*leq*,*fnnc*) ≡ *wfrec*(*frecrel*(*names_below*(*P*,*fnnc*)),*fnnc*,
λ*x f*. *bool_of_o*(*Hfrc*(*P*,*leq*,*x*,*f*)))

```
definition
  is_frc_at :: [i⇒o,i,i,i,i] ⇒ o where
  is_frc_at(M,P,leq,x,z) ≡ ∃ r[M]. is_forcerel(M,P,x,r) ∧
                                  is_wfrec(M,is_Hfrc_at(M,P,leq),r,x,z)

definition
  frc_at_fm :: [i,i,i,i] ⇒ i where
  frc_at_fm(p,l,x,z) ≡ Exists(And(forcerel_fm(succ(p),succ(x),0),
          is_wfrec_fm(Hfrc_at_fm(6#+p,6#+l,2,1,0),0,succ(x),succ(z))))

lemma frc_at_fm_type [TC] :
  ⟦p∈nat;l∈nat;x∈nat;z∈nat⟧ ⟹ frc_at_fm(p,l,x,z)∈formula
  unfolding frc_at_fm_def by simp

lemma arity_frc_at_fm :
  assumes p∈nat l∈nat x∈nat z∈nat
  shows arity(frc_at_fm(p,l,x,z)) = succ(p) ∪ succ(l) ∪ succ(x) ∪ succ(z)
proof -
  let ?φ = Hfrc_at_fm(6 #+ p, 6 #+ l, 2, 1, 0)
  from assms
  have  arity(?φ) = (7#+p) ∪ (7#+l) ?φ ∈ formula
    using arity_Hfrc_at_fm nat_simp_union
    by auto
  with assms
   have W: arity(is_wfrec_fm(?φ, 0, succ(x), succ(z))) = 2#+p ∪ (2#+l) ∪
(2#+x) ∪ (2#+z)
     using arity_is_wfrec_fm[OF ‹?φ∈_› _ _ _ _ ‹arity(?φ) = _›] pred_Un_distrib
pred_succ_eq
      nat_union_abs1
    by auto
  from assms
  have arity(forcerel_fm(succ(p),succ(x),0)) = succ(succ(p)) ∪ succ(succ(x))
    using arity_forcerel_fm nat_simp_union
    by auto
  with W assms
  show ?thesis
    unfolding frc_at_fm_def
    using arity_forcerel_fm pred_Un_distrib
    by auto
qed

lemma sats_frc_at_fm :
  assumes
    p∈nat l∈nat i∈nat j∈nat env∈list(A) i < length(env) j < length(env)
  shows
    sats(A,frc_at_fm(p,l,i,j),env) ⟷
     is_frc_at(##A,nth(p,env),nth(l,env),nth(i,env),nth(j,env))
proof -
  {
```

fix *r pp ll*
assume *r∈A*
have *0:is_Hfrc_at(##A,nth(p,env),nth(l,env),a2, a1, a0)* ⟷
sats(A, Hfrc_at_fm(6#+p,6#+l,2,1,0), [a0,a1,a2,a3,a4,r]@env)
if *a0∈A a1∈A a2∈A a3∈A a4∈A* **for** *a0 a1 a2 a3 a4*
using *that assms ‹r∈A›*
is_Hfrc_at_iff_sats[of 6#+p 6#+l 2 1 0 [a0,a1,a2,a3,a4,r]@env A] **by** *simp*
have *sats(A,is_wfrec_fm(Hfrc_at_fm(6 #+ p, 6 #+ l, 2, 1, 0), 0, succ(i), succ(j)),[r]@env)* ⟷
is_wfrec(##A, is_Hfrc_at(##A, nth(p,env), nth(l,env)), r,nth(i, env), nth(j, env))
using *assms ‹r∈A›*
sats_is_wfrec_fm[OF 0[simplified]]
by *simp*
}
moreover
have *sats(A, forcerel_fm(succ(p), succ(i), 0), Cons(r, env))* ⟷
is_forcerel(##A,nth(p,env),nth(i,env),r) **if** *r∈A* **for** *r*
using *assms sats_forcerel_fm that* **by** *simp*
ultimately
show *?thesis* **unfolding** *is_frc_at_def frc_at_fm_def*
using *assms* **by** *simp*
qed

definition
forces_eq' :: [*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
forces_eq'(P,l,p,t1,t2) ≡ *frc_at(P,l,⟨0,t1,t2,p⟩) = 1*

definition
forces_mem' :: [*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
forces_mem'(P,l,p,t1,t2) ≡ *frc_at(P,l,⟨1,t1,t2,p⟩) = 1*

definition
forces_neq' :: [*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
forces_neq'(P,l,p,t1,t2) ≡ ¬ (∃ *q∈P*. ⟨*q,p*⟩∈*l* ∧ *forces_eq'(P,l,q,t1,t2)*)

definition
forces_nmem' :: [*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
forces_nmem'(P,l,p,t1,t2) ≡ ¬ (∃ *q∈P*. ⟨*q,p*⟩∈*l* ∧ *forces_mem'(P,l,q,t1,t2)*)

definition
is_forces_eq' :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_forces_eq'(M,P,l,p,t1,t2) ≡ ∃ *o[M]*. ∃ *z[M]*. ∃ *t[M]*. *number1(M,o)* ∧ *empty(M,z)* ∧
is_tuple(M,z,t1,t2,p,t) ∧ *is_frc_at(M,P,l,t,o)*

definition
is_forces_mem' :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_forces_mem'(M,P,l,p,t1,t2) ≡ ∃ *o[M]*. ∃ *t[M]*. *number1(M,o)* ∧

is_tuple(*M*,*o*,*t1*,*t2*,*p*,*t*) ∧ *is_frc_at*(*M*,*P*,*l*,*t*,*o*)

definition
is_forces_neq' :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_forces_neq'(*M*,*P*,*l*,*p*,*t1*,*t2*) ≡
¬ (∃ *q*[*M*]. *q*∈*P* ∧ (∃ *qp*[*M*]. *pair*(*M*,*q*,*p*,*qp*) ∧ *qp*∈*l* ∧ *is_forces_eq'*(*M*,*P*,*l*,*q*,*t1*,*t2*)))

definition
is_forces_nmem' :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_forces_nmem'(*M*,*P*,*l*,*p*,*t1*,*t2*) ≡
¬ (∃ *q*[*M*]. ∃ *qp*[*M*]. *q*∈*P* ∧ *pair*(*M*,*q*,*p*,*qp*) ∧ *qp*∈*l* ∧ *is_forces_mem'*(*M*,*P*,*l*,*q*,*t1*,*t2*))

definition
forces_eq_fm :: [*i*,*i*,*i*,*i*,*i*] ⇒ *i* **where**
forces_eq_fm(*p*,*l*,*q*,*t1*,*t2*) ≡
Exists(*Exists*(*Exists*(*And*(*number1_fm*(*2*),*And*(*empty_fm*(*1*),
And(*is_tuple_fm*(*1*,*t1* #+*3*,*t2* #+*3*,*q* #+*3*,*0*),*frc_at_fm*(*p* #+*3*,*l* #+*3*,*0*,*2*)
))))))

definition
forces_mem_fm :: [*i*,*i*,*i*,*i*,*i*] ⇒ *i* **where**
forces_mem_fm(*p*,*l*,*q*,*t1*,*t2*) ≡ *Exists*(*Exists*(*And*(*number1_fm*(*1*),
And(*is_tuple_fm*(*1*,*t1* #+*2*,*t2* #+*2*,*q* #+*2*,*0*),*frc_at_fm*(*p* #+*2*,*l* #+*2*,*0*,*1*)))))

definition
forces_neq_fm :: [*i*,*i*,*i*,*i*,*i*] ⇒ *i* **where**
forces_neq_fm(*p*,*l*,*q*,*t1*,*t2*) ≡ *Neg*(*Exists*(*Exists*(*And*(*Member*(*1*,*p* #+*2*),
And(*pair_fm*(*1*,*q* #+*2*,*0*),*And*(*Member*(*0*,*l* #+*2*),*forces_eq_fm*(*p* #+*2*,*l* #+*2*,*1*,*t1* #+*2*,*t2* #+*2*)))))))

definition
forces_nmem_fm :: [*i*,*i*,*i*,*i*,*i*] ⇒ *i* **where**
forces_nmem_fm(*p*,*l*,*q*,*t1*,*t2*) ≡ *Neg*(*Exists*(*Exists*(*And*(*Member*(*1*,*p* #+*2*),
And(*pair_fm*(*1*,*q* #+*2*,*0*),*And*(*Member*(*0*,*l* #+*2*),*forces_mem_fm*(*p* #+*2*,*l* #+*2*,*1*,*t1* #+*2*,*t2* #+*2*)))))))

lemma *forces_eq_fm_type* [*TC*]:
⟦ *p*∈*nat*;*l*∈*nat*;*q*∈*nat*;*t1*∈*nat*;*t2*∈*nat*⟧ ⟹ *forces_eq_fm*(*p*,*l*,*q*,*t1*,*t2*) ∈ *formula*
unfolding *forces_eq_fm_def*
by *simp*

lemma *forces_mem_fm_type* [*TC*]:
⟦ *p*∈*nat*;*l*∈*nat*;*q*∈*nat*;*t1*∈*nat*;*t2*∈*nat*⟧ ⟹ *forces_mem_fm*(*p*,*l*,*q*,*t1*,*t2*) ∈ *formula*
unfolding *forces_mem_fm_def*
by *simp*

lemma *forces_neq_fm_type* [*TC*]:
⟦ *p*∈*nat*;*l*∈*nat*;*q*∈*nat*;*t1*∈*nat*;*t2*∈*nat*⟧ ⟹ *forces_neq_fm*(*p*,*l*,*q*,*t1*,*t2*) ∈ *formula*
unfolding *forces_neq_fm_def*
by *simp*

lemma *forces_nmem_fm_type* [*TC*]:
⟦ *p*∈*nat*;*l*∈*nat*;*q*∈*nat*;*t1*∈*nat*;*t2*∈*nat*⟧ ⟹ *forces_nmem_fm*(*p*,*l*,*q*,*t1*,*t2*) ∈ *formula*
unfolding *forces_nmem_fm_def*
by *simp*

lemma *arity_forces_eq_fm* :
p∈*nat* ⟹ *l*∈*nat* ⟹ *q*∈*nat* ⟹ *t1* ∈ *nat* ⟹ *t2* ∈ *nat* ⟹
arity(*forces_eq_fm*(*p*,*l*,*q*,*t1*,*t2*)) = *succ*(*t1*) ∪ *succ*(*t2*) ∪ *succ*(*q*) ∪ *succ*(*p*) ∪ *succ*(*l*)
unfolding *forces_eq_fm_def*
using *number1arity__fm arity_empty_fm arity_is_tuple_fm arity_frc_at_fm pred_Un_distrib*
by *auto*

lemma *arity_forces_mem_fm* :
p∈*nat* ⟹ *l*∈*nat* ⟹ *q*∈*nat* ⟹ *t1* ∈ *nat* ⟹ *t2* ∈ *nat* ⟹
arity(*forces_mem_fm*(*p*,*l*,*q*,*t1*,*t2*)) = *succ*(*t1*) ∪ *succ*(*t2*) ∪ *succ*(*q*) ∪ *succ*(*p*) ∪ *succ*(*l*)
unfolding *forces_mem_fm_def*
using *number1arity__fm arity_empty_fm arity_is_tuple_fm arity_frc_at_fm pred_Un_distrib*
by *auto*

lemma *sats_forces_eq'_fm*:
assumes *p*∈*nat* *l*∈*nat* *q*∈*nat* *t1*∈*nat* *t2*∈*nat* *env*∈*list*(*M*)
shows *sats*(*M*,*forces_eq_fm*(*p*,*l*,*q*,*t1*,*t2*),*env*) ⟷
is_forces_eq'(##*M*,*nth*(*p*,*env*),*nth*(*l*,*env*),*nth*(*q*,*env*),*nth*(*t1*,*env*),*nth*(*t2*,*env*))
unfolding *forces_eq_fm_def is_forces_eq'_def* **using** *assms sats_is_tuple_fm sats_frc_at_fm*
by *simp*

lemma *sats_forces_mem'_fm*:
assumes *p*∈*nat* *l*∈*nat* *q*∈*nat* *t1*∈*nat* *t2*∈*nat* *env*∈*list*(*M*)
shows *sats*(*M*,*forces_mem_fm*(*p*,*l*,*q*,*t1*,*t2*),*env*) ⟷
is_forces_mem'(##*M*,*nth*(*p*,*env*),*nth*(*l*,*env*),*nth*(*q*,*env*),*nth*(*t1*,*env*),*nth*(*t2*,*env*))
unfolding *forces_mem_fm_def is_forces_mem'_def* **using** *assms sats_is_tuple_fm sats_frc_at_fm*
by *simp*

lemma *sats_forces_neq'_fm*:
assumes *p*∈*nat* *l*∈*nat* *q*∈*nat* *t1*∈*nat* *t2*∈*nat* *env*∈*list*(*M*)
shows *sats*(*M*,*forces_neq_fm*(*p*,*l*,*q*,*t1*,*t2*),*env*) ⟷
is_forces_neq'(##*M*,*nth*(*p*,*env*),*nth*(*l*,*env*),*nth*(*q*,*env*),*nth*(*t1*,*env*),*nth*(*t2*,*env*))
unfolding *forces_neq_fm_def is_forces_neq'_def*
using *assms sats_forces_eq'_fm sats_is_tuple_fm sats_frc_at_fm*
by *simp*

lemma *sats_forces_nmem'_fm*:
assumes *p*∈*nat* *l*∈*nat* *q*∈*nat* *t1*∈*nat* *t2*∈*nat* *env*∈*list*(*M*)

shows *sats*(*M*,*forces_nmem_fm*(*p*,*l*,*q*,*t1*,*t2*),*env*) ⟷
is_forces_nmem'(##*M*,*nth*(*p*,*env*),*nth*(*l*,*env*),*nth*(*q*,*env*),*nth*(*t1*,*env*),*nth*(*t2*,*env*))
unfolding *forces_nmem_fm_def is_forces_nmem'_def*
using *assms sats_forces_mem'_fm sats_is_tuple_fm sats_frc_at_fm*
by *simp*

context *forcing_data*
begin

lemma *fst_abs* [*simp*]:
⟦*x*∈*M*; *y*∈*M* ⟧ ⟹ *is_fst*(##*M*,*x*,*y*) ⟷ *y* = *fst*(*x*)
unfolding *fst_def is_fst_def* **using** *pair_in_M_iff zero_in_M*
by (*auto*;*rule_tac the_0 the_0*[*symmetric*],*auto*)

lemma *snd_abs* [*simp*]:
⟦*x*∈*M*; *y*∈*M* ⟧ ⟹ *is_snd*(##*M*,*x*,*y*) ⟷ *y* = *snd*(*x*)
unfolding *snd_def is_snd_def* **using** *pair_in_M_iff zero_in_M*
by (*auto*;*rule_tac the_0 the_0*[*symmetric*],*auto*)

lemma *ftype_abs*:
⟦*x*∈*M*; *y*∈*M* ⟧ ⟹ *is_ftype*(##*M*,*x*,*y*) ⟷ *y* = *ftype*(*x*) **unfolding** *ftype_def*
is_ftype_def **by** *simp*

lemma *name1_abs*:
⟦*x*∈*M*; *y*∈*M* ⟧ ⟹ *is_name1*(##*M*,*x*,*y*) ⟷ *y* = *name1*(*x*)
unfolding *name1_def is_name1_def*
by (*rule hcomp_abs*[*OF fst_abs*];*simp_all add*:*fst_snd_closed*)

lemma *snd_snd_abs*:
⟦*x*∈*M*; *y*∈*M* ⟧ ⟹ *is_snd_snd*(##*M*,*x*,*y*) ⟷ *y* = *snd*(*snd*(*x*))
unfolding *is_snd_snd_def*
by (*rule hcomp_abs*[*OF snd_abs*];*simp_all add*:*fst_snd_closed*)

lemma *name2_abs*:
⟦*x*∈*M*; *y*∈*M* ⟧ ⟹ *is_name2*(##*M*,*x*,*y*) ⟷ *y* = *name2*(*x*)
unfolding *name2_def is_name2_def*
by (*rule hcomp_abs*[*OF fst_abs snd_snd_abs*];*simp_all add*:*fst_snd_closed*)

lemma *cond_of_abs*:
⟦*x*∈*M*; *y*∈*M* ⟧ ⟹ *is_cond_of*(##*M*,*x*,*y*) ⟷ *y* = *cond_of*(*x*)
unfolding *cond_of_def is_cond_of_def*
by (*rule hcomp_abs*[*OF snd_abs snd_snd_abs*];*simp_all add*:*fst_snd_closed*)

lemma *tuple_abs*:
⟦*z*∈*M*;*t1*∈*M*;*t2*∈*M*;*p*∈*M*;*t*∈*M*⟧ ⟹
is_tuple(##*M*,*z*,*t1*,*t2*,*p*,*t*) ⟷ *t* = ⟨*z*,*t1*,*t2*,*p*⟩
unfolding *is_tuple_def* **using** *tuples_in_M* **by** *simp*

```
lemmas components_abs = ftype_abs name1_abs name2_abs cond_of_abs
  tuple_abs

lemma oneN_in_M[simp]: 1∈M
  by (simp flip: setclass_iff)

lemma twoN_in_M : 2∈M
  by (simp flip: setclass_iff)

lemma comp_in_M:
  p ≼ q ⟹ p∈M
  p ≼ q ⟹ q∈M
  using leq_in_M transitivity[of _ leq] pair_in_M_iff by auto



lemma eq_case_abs [simp]:
  assumes
    t1∈M t2∈M p∈M f∈M
  shows
    is_eq_case(##M,t1,t2,p,P,leq,f) ⟷ eq_case(t1,t2,p,P,leq,f)
proof -
  have q ≼ p ⟹ q∈M for q
    using comp_in_M by simp
  moreover
  have ⟨s,y⟩∈t ⟹ s∈domain(t) if t∈M for s y t
    using that unfolding domain_def by auto
  ultimately
  have
    (∀s∈M. s ∈ domain(t1) ∨ s ∈ domain(t2) ⟶ (∀q∈M. q∈P ∧ q ≼ p ⟶
                           (f ` ⟨1, s, t1, q⟩ =1 ⟷ f ` ⟨1, s, t2, q⟩=1))) ⟷
    (∀s. s ∈ domain(t1) ∨ s ∈ domain(t2) ⟶ (∀q. q∈P ∧ q ≼ p ⟶
                              (f ` ⟨1, s, t1, q⟩ =1 ⟷ f ` ⟨1, s, t2, q⟩=1)))
    using assms domain_trans[OF trans_M,of t1]
      domain_trans[OF trans_M,of t2] by auto
  then show ?thesis
    unfolding eq_case_def is_eq_case_def
    using assms pair_in_M_iff nat_into_M[of 1] domain_closed tuples_in_M
      apply_closed leq_in_M
    by (simp add:components_abs)
qed

lemma mem_case_abs [simp]:
  assumes
    t1∈M t2∈M p∈M f∈M
  shows
    is_mem_case(##M,t1,t2,p,P,leq,f) ⟷ mem_case(t1,t2,p,P,leq,f)
proof
  {
```

fix *v*
assume *v*∈*P* *v* ⪯ *p* *is_mem_case*(##*M*,*t1*,*t2*,*p*,*P*,*leq*,*f*)
moreover
from *this*
have *v*∈*M* ⟨*v*,*p*⟩ ∈ *M* (##*M*)(*v*)
using *transitivity*[*OF* _ *P_in_M*,*of v*] *transitivity*[*OF* _ *leq_in_M*]
by *simp_all*
moreover
from *calculation assms*
obtain *q r s* **where**
r ∈ *P* ∧ *q* ∈ *P* ∧ ⟨*q*, *v*⟩ ∈ *M* ∧ ⟨*s*, *r*⟩ ∈ *M* ∧ ⟨*q*, *r*⟩ ∈ *M* ∧ *0* ∈ *M* ∧
⟨*0*, *t1*, *s*, *q*⟩ ∈ *M* ∧ *q* ⪯ *v* ∧ ⟨*s*, *r*⟩ ∈ *t2* ∧ *q* ⪯ *r* ∧ *f* ‘ ⟨*0*, *t1*, *s*, *q*⟩ = *1*
unfolding *is_mem_case_def* **by** (*auto simp add*:*components_abs*)
then
have ∃ *q s r*. *r* ∈ *P* ∧ *q* ∈ *P* ∧ *q* ⪯ *v* ∧ ⟨*s*, *r*⟩ ∈ *t2* ∧ *q* ⪯ *r* ∧ *f* ‘ ⟨*0*, *t1*, *s*, *q*⟩ = *1*
by *auto*
}
then
show *mem_case*(*t1*, *t2*, *p*, *P*, *leq*, *f*) **if** *is_mem_case*(##*M*, *t1*, *t2*, *p*, *P*, *leq*, *f*)
unfolding *mem_case_def* **using** *that assms* **by** *auto*
next
{ fix *v*
assume *v* ∈ *M* *v* ∈ *P* ⟨*v*, *p*⟩ ∈ *M* *v* ⪯ *p* *mem_case*(*t1*, *t2*, *p*, *P*, *leq*, *f*)
moreover
from *this*
obtain *q s r* **where** *r* ∈ *P* ∧ *q* ∈ *P* ∧ *q* ⪯ *v* ∧ ⟨*s*, *r*⟩ ∈ *t2* ∧ *q* ⪯ *r* ∧ *f* ‘ ⟨*0*, *t1*, *s*, *q*⟩ = *1*
unfolding *mem_case_def* **by** *auto*
moreover
from *this* ‹*t2*∈*M*›
have *r*∈*M* *q*∈*M* *s*∈*M* *r* ∈ *P* ∧ *q* ∈ *P* ∧ *q* ⪯ *v* ∧ ⟨*s*, *r*⟩ ∈ *t2* ∧ *q* ⪯ *r* ∧ *f* ‘ ⟨*0*, *t1*, *s*, *q*⟩ = *1*
using *transitivity P_in_M domain_closed*[*of t2*] **by** *auto*
moreover
note ‹*t1*∈*M*›
ultimately
have ∃ *q*∈*M* . ∃ *s*∈*M*. ∃ *r*∈*M*.
r ∈ *P* ∧ *q* ∈ *P* ∧ ⟨*q*, *v*⟩ ∈ *M* ∧ ⟨*s*, *r*⟩ ∈ *M* ∧ ⟨*q*, *r*⟩ ∈ *M* ∧ *0* ∈ *M* ∧
⟨*0*, *t1*, *s*, *q*⟩ ∈ *M* ∧ *q* ⪯ *v* ∧ ⟨*s*, *r*⟩ ∈ *t2* ∧ *q* ⪯ *r* ∧ *f* ‘ ⟨*0*, *t1*, *s*, *q*⟩ = *1*
using *tuples_in_M zero_in_M* **by** *auto*
}
then
show *is_mem_case*(##*M*, *t1*, *t2*, *p*, *P*, *leq*, *f*) **if** *mem_case*(*t1*, *t2*, *p*, *P*, *leq*, *f*)
unfolding *is_mem_case_def* **using** *assms that* **by** (*auto simp add*:*components_abs*)
qed

lemma *Hfrc_abs*:

⟦*fnnc*∈*M*; *f*∈*M*⟧ ⟹
is_Hfrc(##*M*,*P*,*leq*,*fnnc*,*f*) ⟷ *Hfrc*(*P*,*leq*,*fnnc*,*f*)
unfolding *is_Hfrc_def Hfrc_def* **using** *pair_in_M_iff*
by (*auto simp add*:*components_abs*)

lemma *Hfrc_at_abs*:
⟦*fnnc*∈*M*; *f*∈*M* ; *z*∈*M*⟧ ⟹
is_Hfrc_at(##*M*,*P*,*leq*,*fnnc*,*f*,*z*) ⟷ *z* = *bool_of_o*(*Hfrc*(*P*,*leq*,*fnnc*,*f*))
unfolding *is_Hfrc_at_def* **using** *Hfrc_abs*
by *auto*

lemma *components_closed* :
x∈*M* ⟹ *ftype*(*x*)∈*M*
x∈*M* ⟹ *name1*(*x*)∈*M*
x∈*M* ⟹ *name2*(*x*)∈*M*
x∈*M* ⟹ *cond_of*(*x*)∈*M*
unfolding *ftype_def name1_def name2_def cond_of_def* **using** *fst_snd_closed* **by** *simp_all*

lemma *ecloseN_closed*:
(##*M*)(*A*) ⟹ (##*M*)(*ecloseN*(*A*))
(##*M*)(*A*) ⟹ (##*M*)(*eclose_n*(*name1*,*A*))
(##*M*)(*A*) ⟹ (##*M*)(*eclose_n*(*name2*,*A*))
unfolding *ecloseN_def eclose_n_def*
using *components_closed eclose_closed singletonM Un_closed* **by** *auto*

lemma *eclose_n_abs* :
assumes *x*∈*M ec*∈*M*
shows *is_eclose_n*(##*M*,*is_name1*,*ec*,*x*) ⟷ *ec* = *eclose_n*(*name1*,*x*)
is_eclose_n(##*M*,*is_name2*,*ec*,*x*) ⟷ *ec* = *eclose_n*(*name2*,*x*)
unfolding *is_eclose_n_def eclose_n_def*
using *assms name1_abs name2_abs eclose_abs singletonM components_closed*
by *auto*

lemma *is_ecloseN_abs* :
⟦*x*∈*M*;*ec*∈*M*⟧ ⟹ *is_ecloseN*(##*M*,*ec*,*x*) ⟷ *ec* = *ecloseN*(*x*)
unfolding *is_ecloseN_def ecloseN_def*
using *eclose_n_abs Un_closed union_abs ecloseN_closed*
by *auto*

lemma *frecR_abs* :
x∈*M* ⟹ *y*∈*M* ⟹ *frecR*(*x*,*y*) ⟷ *is_frecR*(##*M*,*x*,*y*)
unfolding *frecR_def is_frecR_def* **using** *components_closed domain_closed*
by (*simp add*:*components_abs*)

lemma *frecrelP_abs* :
z∈*M* ⟹ *frecrelP*(##*M*,*z*) ⟷ (∃ *x y*. *z* = ⟨*x*,*y*⟩ ∧ *frecR*(*x*,*y*))
using *pair_in_M_iff frecR_abs* **unfolding** *frecrelP_def* **by** *auto*

lemma *frecrel_abs*:
  assumes
    $A \in M$ $r \in M$
  shows
    $is\_frecrel(\#\#M,A,r) \longleftrightarrow r = frecrel(A)$
proof -
  from ‹$A \in M$›
  have $z \in M$ **if** $z \in A \times A$ **for** z
    using *cartprod_closed transitivity that* **by** *simp*
  then
  have $Collect(A \times A, frecrelP(\#\#M)) = Collect(A \times A, \lambda z. (\exists x\ y.\ z = \langle x,y \rangle \wedge frecR(x,y)))$
    using *Collect_cong*[*of* $A \times A$ $A \times A$ *frecrelP*($\#\#M$)] *assms frecrelP_abs* **by** *simp*
  with *assms*
  show *?thesis* **unfolding** *is_frecrel_def def_frecrel* **using** *cartprod_closed*
    by *simp*
qed

lemma *frecrel_closed*:
  assumes
    $x \in M$
  shows
    $frecrel(x) \in M$
proof -
  have $Collect(x \times x, \lambda z. (\exists x\ y.\ z = \langle x,y \rangle \wedge frecR(x,y))) \in M$
    using *Collect_in_M_0p*[*of frecrelP_fm*(*0*)] *arity_frecrelP_fm sats_frecrelP_fm*
      frecrelP_abs ‹$x \in M$› *cartprod_closed* **by** *simp*
  then show *?thesis*
    unfolding *frecrel_def Rrel_def frecrelP_def* **by** *simp*
qed

lemma *field_frecrel* : $field(frecrel(names\_below(P,x))) \subseteq names\_below(P,x)$
  unfolding *frecrel_def*
  using *field_Rrel* **by** *simp*

lemma *forcerelD* : $uv \in forcerel(P,x) \Longrightarrow uv \in names\_below(P,x) \times names\_below(P,x)$
  unfolding *forcerel_def*
  using *trancl_type field_frecrel* **by** *blast*

lemma *wf_forcerel* :
  $wf(forcerel(P,x))$
  unfolding *forcerel_def* **using** *wf_trancl wf_frecrel* **.**

lemma *restrict_trancl_forcerel*:
  assumes $frecR(w,y)$
  shows $restrict(f,frecrel(names\_below(P,x))\text{-}``\{y\})`w$
      $= restrict(f,forcerel(P,x)\text{-}``\{y\})`w$
  unfolding *forcerel_def frecrel_def* **using** *assms restrict_trancl_Rrel*[*of frecR*]

```
  by simp

lemma names_belowI :
  assumes frecR(⟨ft,n1,n2,p⟩,⟨a,b,c,d⟩) p∈P
  shows ⟨ft,n1,n2,p⟩ ∈ names_below(P,⟨a,b,c,d⟩) (is ?x ∈ names_below(_,?y))
proof -
  from assms
  have ft ∈ 2 a ∈ 2
    unfolding frecR_def by (auto simp add:components_simp)
  from assms
  consider (e) n1 ∈ domain(b) ∪ domain(c) ∧ (n2 = b ∨ n2 =c)
    | (m) n1 = b ∧ n2 ∈ domain(c)
    unfolding frecR_def by (auto simp add:components_simp)
  then show ?thesis
  proof cases
    case e
    then
    have n1 ∈ eclose(b) ∨ n1 ∈ eclose(c)
      using Un_iff in_dom_in_eclose by auto
    with e
    have n1 ∈ ecloseN(?y) n2 ∈ ecloseN(?y)
      using ecloseNI components_in_eclose by auto
    with ‹ft∈2› ‹p∈P›
    show ?thesis unfolding names_below_def by  auto
  next
    case m
    then
    have n1 ∈ ecloseN(?y) n2 ∈ ecloseN(?y)
      using mem_eclose_trans  ecloseNI
        in_dom_in_eclose components_in_eclose by auto
    with ‹ft∈2› ‹p∈P›
    show ?thesis unfolding names_below_def
      by auto
  qed
qed

lemma names_below_tr :
  assumes x∈ names_below(P,y)
    y∈ names_below(P,z)
  shows x∈ names_below(P,z)
proof -
  let ?A=λy . names_below(P,y)
  from assms
  obtain fx x1 x2 px where
    x = ⟨fx,x1,x2,px⟩ fx∈2 x1∈ecloseN(y) x2∈ecloseN(y) px∈P
    unfolding names_below_def by auto
  from assms
  obtain fy y1 y2 py where
    y = ⟨fy,y1,y2,py⟩ fy∈2 y1∈ecloseN(z) y2∈ecloseN(z) py∈P
```

```
    unfolding names_below_def by auto
  from ‹x1∈_› ‹x2∈_› ‹y1∈_› ‹y2∈_› ‹x=_› ‹y=_›
  have x1∈ecloseN(z) x2∈ecloseN(z)
    using ecloseN_mono names_simp by auto
  with ‹fx∈2› ‹px∈P› ‹x=_›
  have x∈?A(z)
    unfolding names_below_def by simp
  then show ?thesis using subsetI by simp
qed

lemma arg_into_names_below2 :
  assumes ⟨x,y⟩ ∈ frecrel(names_below(P,z))
  shows  x ∈ names_below(P,y)
proof -
  {
    from assms
    have x∈names_below(P,z) y∈names_below(P,z) frecR(x,y)
      unfolding frecrel_def Rrel_def
      by auto
    obtain f n1 n2 p where
      x = ⟨f,n1,n2,p⟩ f∈2 n1∈ecloseN(z) n2∈ecloseN(z) p∈P
      using ‹x∈names_below(P,z)›
      unfolding names_below_def by auto
    moreover
    obtain fy m1 m2 q where
      q∈P y = ⟨fy,m1,m2,q⟩
      using ‹y∈names_below(P,z)›
      unfolding names_below_def by auto
    moreover
    note ‹frecR(x,y)›
    ultimately
    have x∈names_below(P,y) using names_belowI by simp
  }
  then show ?thesis .
qed

lemma arg_into_names_below :
  assumes ⟨x,y⟩ ∈ frecrel(names_below(P,z))
  shows  x ∈ names_below(P,x)
proof -
  {
    from assms
    have x∈names_below(P,z)
      unfolding frecrel_def Rrel_def
      by auto
    from ‹x∈names_below(P,z)›
    obtain f n1 n2 p where
      x = ⟨f,n1,n2,p⟩ f∈2 n1∈ecloseN(z) n2∈ecloseN(z) p∈P
      unfolding names_below_def by auto
```

```
    then
    have n1∈ecloseN(x) n2∈ecloseN(x)
      using components_in_eclose by simp_all
    with ‹f∈2› ‹p∈P› ‹x = ⟨f,n1,n2,p⟩›
    have x∈names_below(P,x)
      unfolding names_below_def by simp
  }
  then show ?thesis .
qed

lemma forcerel_arg_into_names_below :
  assumes ⟨x,y⟩ ∈ forcerel(P,z)
  shows  x ∈ names_below(P,x)
  using assms
  unfolding forcerel_def
  by(rule trancl_induct;auto simp add: arg_into_names_below)

lemma names_below_mono :
  assumes ⟨x,y⟩ ∈ frecrel(names_below(P,z))
  shows names_below(P,x) ⊆ names_below(P,y)
proof -
  from assms
  have x∈names_below(P,y)
    using arg_into_names_below2 by simp
  then
  show ?thesis
    using names_below_tr subsetI by simp
qed

lemma frecrel_mono :
  assumes ⟨x,y⟩ ∈ frecrel(names_below(P,z))
  shows frecrel(names_below(P,x)) ⊆ frecrel(names_below(P,y))
  unfolding frecrel_def
  using Rrel_mono names_below_mono assms by simp

lemma forcerel_mono2 :
  assumes ⟨x,y⟩ ∈ frecrel(names_below(P,z))
  shows forcerel(P,x) ⊆ forcerel(P,y)
  unfolding forcerel_def
  using trancl_mono frecrel_mono assms by simp

lemma forcerel_mono_aux :
  assumes ⟨x,y⟩ ∈ frecrel(names_below(P, w)) ^+
  shows forcerel(P,x) ⊆ forcerel(P,y)
  using assms
  by (rule trancl_induct,simp_all add: subset_trans forcerel_mono2)

lemma forcerel_mono :
  assumes ⟨x,y⟩ ∈ forcerel(P,z)
```

shows *forcerel*(*P*,*x*) ⊆ *forcerel*(*P*,*y*)
using *forcerel_mono_aux assms* **unfolding** *forcerel_def* **by** *simp*

lemma *aux*: *x* ∈ *names_below*(*P*, *w*) ⟹ ⟨*x*,*y*⟩ ∈ *forcerel*(*P*,*z*) ⟹
(*y* ∈ *names_below*(*P*, *w*) ⟶ ⟨*x*,*y*⟩ ∈ *forcerel*(*P*,*w*))
unfolding *forcerel_def*
proof(*rule_tac a*=*x* **and** *b*=*y* **and** *P*=λ *y* . *y* ∈ *names_below*(*P*, *w*) ⟶ ⟨*x*,*y*⟩ ∈ *frecrel*(*names_below*(*P*,*w*))^+ **in** *trancl_induct*,*simp*)
let *?A*=λ *a* . *names_below*(*P*, *a*)
let *?R*=λ *a* . *frecrel*(*?A*(*a*))
let *?fR*=λ *a* .*forcerel*(*a*)
show *u*∈*?A*(*w*) ⟶ ⟨*x*,*u*⟩∈*?R*(*w*)^+ **if** *x*∈*?A*(*w*) ⟨*x*,*y*⟩∈*?R*(*z*)^+ ⟨*x*,*u*⟩∈*?R*(*z*) **for** *u*
using *that frecrelD frecrelI r_into_trancl* **unfolding** *names_below_def* **by** *simp*
{
fix *u v*
assume *x* ∈ *?A*(*w*)
⟨*x*, *y*⟩ ∈ *?R*(*z*)^+
⟨*x*, *u*⟩ ∈ *?R*(*z*)^+
⟨*u*, *v*⟩ ∈ *?R*(*z*)
u ∈ *?A*(*w*) ⟹ ⟨*x*, *u*⟩ ∈ *?R*(*w*)^+
then
have *v* ∈ *?A*(*w*) ⟹ ⟨*x*, *v*⟩ ∈ *?R*(*w*)^+
proof -
assume *v* ∈*?A*(*w*)
from ‹⟨*u*,*v*⟩∈_›
have *u*∈*?A*(*v*)
using *arg_into_names_below2* **by** *simp*
with ‹*v* ∈*?A*(*w*)›
have *u*∈*?A*(*w*)
using *names_below_tr* **by** *simp*
with ‹*v*∈_› ‹⟨*u*,*v*⟩∈_›
have ⟨*u*,*v*⟩∈ *?R*(*w*)
using *frecrelD frecrelI r_into_trancl* **unfolding** *names_below_def* **by** *simp*
with ‹*u* ∈ *?A*(*w*) ⟹ ⟨*x*, *u*⟩ ∈ *?R*(*w*)^+› ‹*u*∈*?A*(*w*)›
have ⟨*x*, *u*⟩ ∈ *?R*(*w*)^+ **by** *simp*
with ‹⟨*u*,*v*⟩∈ *?R*(*w*)›
show ⟨*x*,*v*⟩∈ *?R*(*w*)^+ **using** *trancl_trans r_into_trancl*
by *simp*
qed
}
then show *v* ∈ *?A*(*w*) ⟶ ⟨*x*, *v*⟩ ∈ *?R*(*w*)^+
if *x* ∈ *?A*(*w*)
⟨*x*, *y*⟩ ∈ *?R*(*z*)^+
⟨*x*, *u*⟩ ∈ *?R*(*z*)^+
⟨*u*, *v*⟩ ∈ *?R*(*z*)
u ∈ *?A*(*w*) ⟶ ⟨*x*, *u*⟩ ∈ *?R*(*w*)^+ **for** *u v*
using *that* **by** *simp*
qed

lemma *forcerel_eq* :
assumes $\langle z,x\rangle \in$ *forcerel*(P,x)
shows *forcerel*(P,z) = *forcerel*$(P,x) \cap$ *names_below*$(P,z)\times$*names_below*(P,z)
using *assms aux forcerelD forcerel_mono*[*of z x x*] *subsetI*
by *auto*

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)
using *assms*(*2*)
unfolding *forcerel_def*
proof(*rule trancl_induct*)
show $u \in$ *names_below*(P,y) **if** $\langle u, y\rangle \in$ *frecrel*(*names_below*(P, x)) **for** y
using *that vimage_singleton_iff arg_into_names_below2* **by** *simp*
next
show $u \in$ *names_below*(P,z)
if $\langle u, y\rangle \in$ *frecrel*(*names_below*(P, x)) ^+
$\langle y, z\rangle \in$ *frecrel*(*names_below*(P, x))
$u \in$ *names_below*(P, y)
for y z
using *that arg_into_names_below2*[*of y z x*] *names_below_tr* **by** *simp*
qed

lemma *forcerel_below* :
assumes $\langle z,x\rangle \in$ *forcerel*(P,x)
shows *forcerel*(P,x) -‘‘ $\{z\} \subseteq$ *names_below*(P,z)
using *vimage_singleton_iff assms forcerel_below_aux* **by** *auto*

lemma *relation_forcerel* :
shows *relation*(*forcerel*(P,z)) *trans*(*forcerel*(P,z))
unfolding *forcerel_def* **using** *relation_trancl trans_trancl* **by** *simp_all*

lemma *Hfrc_restrict_trancl*: *bool_of_o*(*Hfrc*(P, *leq*, y, *restrict*(f,*frecrel*(*names_below*(P,x))-‘‘$\{y\}$)))
= *bool_of_o*(*Hfrc*(P, *leq*, y, *restrict*(f,(*frecrel*(*names_below*(P,x)) ^+)-‘‘$\{y\}$)))
unfolding *Hfrc_def bool_of_o_def eq_case_def mem_case_def*
using *restrict_trancl_forcerel frecRI1 frecRI2 frecRI3*
unfolding *forcerel_def*
by *simp*

lemma *frc_at_trancl*: *frc_at*(P,leq,z) = *wfrec*(*forcerel*(P,z),z,$\lambda x f$. *bool_of_o*(*Hfrc*(P,leq,x,f))))
unfolding *frc_at_def forcerel_def* **using** *wf_eq_trancl Hfrc_restrict_trancl* **by** *simp*

lemma *forcerelI1* :
assumes $n1 \in$ *domain*(b) $\vee$ $n1 \in$ *domain*(c) $p\in P$ $d\in P$
shows $\langle\langle 1, n1, b, p\rangle, \langle 0,b,c,d\rangle\rangle\in$ *forcerel*$(P,\langle 0,b,c,d\rangle)$

proof -
  let *?x*=⟨*1*, *n1*, *b*, *p*⟩
  let *?y*=⟨*0*,*b*,*c*,*d*⟩
  from *assms*
  have *frecR*(*?x*,*?y*)
    using *frecRI1* **by** *simp*
  then
  have *?x*∈*names_below*(*P*,*?y*) *?y* ∈ *names_below*(*P*,*?y*)
    using *names_belowI* *assms components_in_eclose*
    unfolding *names_below_def* **by** *auto*
  with ‹*frecR*(*?x*,*?y*)›
  show *?thesis*
    unfolding *forcerel_def frecrel_def*
    using *subsetD*[*OF r_subset_trancl*[*OF relation_Rrel*]] *RrelI*
    by *auto*
qed

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 -
  let *?x*=⟨*1*, *n1*, *c*, *p*⟩
  let *?y*=⟨*0*,*b*,*c*,*d*⟩
  from *assms*
  have *frecR*(*?x*,*?y*)
    using *frecRI2* **by** *simp*
  then
  have *?x*∈*names_below*(*P*,*?y*) *?y* ∈ *names_below*(*P*,*?y*)
    using *names_belowI* *assms components_in_eclose*
    unfolding *names_below_def* **by** *auto*
  with ‹*frecR*(*?x*,*?y*)›
  show *?thesis*
    unfolding *forcerel_def frecrel_def*
    using *subsetD*[*OF r_subset_trancl*[*OF relation_Rrel*]] *RrelI*
    by *auto*
qed

lemma *forcerelI3* :
  assumes ⟨*n2*, *r*⟩ ∈ *c* *p*∈*P* *d*∈*P* *r* ∈ *P*
  shows ⟨⟨*0*, *b*, *n2*, *p*⟩,⟨*1*, *b*, *c*, *d*⟩⟩ ∈ *forcerel*(*P*,⟨*1*,*b*,*c*,*d*⟩)
proof -
  let *?x*=⟨*0*, *b*, *n2*, *p*⟩
  let *?y*=⟨*1*, *b*, *c*, *d*⟩
  from *assms*
  have *frecR*(*?x*,*?y*)
    using *assms frecRI3* **by** *simp*
  then
  have *?x*∈*names_below*(*P*,*?y*) *?y* ∈ *names_below*(*P*,*?y*)
    using *names_belowI* *assms components_in_eclose*

unfolding *names_below_def* **by** *auto*
with ‹*frecR*(*?x*,*?y*)›
show *?thesis*
unfolding *forcerel_def frecrel_def*
using *subsetD*[*OF r_subset_trancl*[*OF relation_Rrel*]] *RrelI*
by *auto*
qed

lemmas *forcerelI* = *forcerelI1*[*THEN vimage_singleton_iff* [*THEN iffD2*]]
forcerelI2[*THEN vimage_singleton_iff* [*THEN iffD2*]]
forcerelI3[*THEN vimage_singleton_iff* [*THEN iffD2*]]

lemma *aux_def_frc_at*:
assumes $z \in forcerel(P,x)$ -‘‘ $\{x\}$
shows $wfrec(forcerel(P,x),\ z,\ H) = \ wfrec(forcerel(P,z),\ z,\ H)$
proof -
let *?A*=*names_below*(*P*,*z*)
from *assms*
have $\langle z,x\rangle \in forcerel(P,x)$
using *vimage_singleton_iff* **by** *simp*
then
have $z \in$ *?A*
using *forcerel_arg_into_names_below* **by** *simp*
from ‹$\langle z,x\rangle \in forcerel(P,x)$›
have *E*:$forcerel(P,z) = forcerel(P,x) \cap$ (*?A*×*?A*)
$forcerel(P,x)$ -‘‘ $\{z\} \subseteq$ *?A*
using *forcerel_eq forcerel_below*
by *auto*
with ‹$z\in$*?A*›
have $wfrec(forcerel(P,x),\ z,\ H) = wfrec$[*?A*]$(forcerel(P,x),\ z,\ H)$
using *wfrec_trans_restr*[*OF relation_forcerel*(*1*) *wf_forcerel relation_forcerel*(*2*),
of x z ?A]
by *simp*
then show *?thesis*
using *E wfrec_restr_eq* **by** *simp*
qed

18.5 Recursive expression of *frc_at*

lemma *def_frc_at* :
assumes $p\in P$
shows
frc_at(*P*,*leq*,⟨*ft*,*n1*,*n2*,*p*⟩) =
bool_of_o($p \in P$ ∧
(*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* ⟶
(∃ *q*. ∃ *s*. ∃ *r*. *r*∈*P* ∧ *q*∈*P* ∧ *q* ≼ *v* ∧ ⟨*s*,*r*⟩ ∈ *n2* ∧ *q* ≼ *r* ∧ *frc_at*(*P*,*leq*,⟨*0*,*n1*,*s*,*q*⟩)

= *1*))))
proof -
  let *?r=λy. forcerel(P,y)* **and** *?Hf=λx f. bool_of_o(Hfrc(P,leq,x,f))*
  let *?t=λy. ?r(y) -`` {y}*
  let *?arg=⟨ft,n1,n2,p⟩*
  from *wf_forcerel*
  have *wfr*: ∀ *w . wf(?r(w))* **..**
  with *wfrec [of ?r(?arg) ?arg ?Hf]*
  have *frc_at(P,leq,?arg) = ?Hf(?arg, λx∈?r(?arg) -`` {?arg}. wfrec(?r(?arg), x, ?Hf))*
    using *frc_at_trancl* **by** *simp*
  also
  have ... = *?Hf(?arg, λx∈?r(?arg) -`` {?arg}. frc_at(P,leq,x))*
    using *aux_def_frc_at frc_at_trancl* **by** *simp*
  finally
  show *?thesis*
    unfolding *Hfrc_def mem_case_def eq_case_def*
    using *forcerelI assms*
    by *auto*
qed

18.6 Absoluteness of *frc_at*

lemma *trans_forcerel_t* : *trans(forcerel(P,x))*
  unfolding *forcerel_def* **using** *trans_trancl* **.**

lemma *relation_forcerel_t* : *relation(forcerel(P,x))*
  unfolding *forcerel_def* **using** *relation_trancl* **.**

lemma *forcerel_in_M* :
  assumes
    x∈M
  shows
    forcerel(P,x)∈M
  unfolding *forcerel_def def_frecrel names_below_def*
proof -
  let *?Q = 2 × ecloseN(x) × ecloseN(x) × P*
  have *?Q × ?Q ∈ M*
    using ‹*x∈M*› *P_in_M twoN_in_M ecloseN_closed cartprod_closed* **by** *simp*
  moreover
  have *separation(##M,λz. ∃ x y. z = ⟨x, y⟩ ∧ frecR(x, y))*
  proof -
    have *arity(frecrelP_fm(0)) = 1*
      unfolding *number1_fm_def frecrelP_fm_def*
      by (*simp del:FOL_sats_iff pair_abs empty_abs*
          add: fm_definitions components_defs nat_simp_union)
    then
    have *separation(##M, λz. sats(M,frecrelP_fm(0) , [z]))*

using *separation_ax* **by** *simp*
moreover
have *frecrelP(##M,z)* ⟷ *sats(M,frecrelP_fm(0),[z])*
if *z∈M* **for** *z*
using *that sats_frecrelP_fm[of 0 [z]]* **by** *simp*
ultimately
have *separation(##M,frecrelP(##M))*
unfolding *separation_def* **by** *simp*
then
show *?thesis* **using** *frecrelP_abs*
separation_cong[of ##M frecrelP(##M) λz. ∃ x y. z = ⟨x, y⟩ ∧ frecR(x, y)]
by *simp*
qed
ultimately
show *{z ∈ ?Q × ?Q . ∃ x y. z = ⟨x, y⟩ ∧ frecR(x, y)}^+ ∈ M*
using *separation_closed frecrelP_abs trancl_closed* **by** *simp*
qed

lemma *relation2_Hfrc_at_abs*:
relation2(##M,is_Hfrc_at(##M,P,leq),λx f. bool_of_o(Hfrc(P,leq,x,f)))
unfolding *relation2_def* **using** *Hfrc_at_abs*
by *simp*

lemma *Hfrc_at_closed* :
∀ x∈M. ∀ g∈M. function(g) ⟶ bool_of_o(Hfrc(P,leq,x,g))∈M
unfolding *bool_of_o_def* **using** *zero_in_M nat_into_M[of 1]* **by** *simp*

lemma *wfrec_Hfrc_at* :
assumes
X∈M
shows
wfrec_replacement(##M,is_Hfrc_at(##M,P,leq),forcerel(P,X))
proof -
have *0:is_Hfrc_at(##M,P,leq,a,b,c)* ⟷
sats(M,Hfrc_at_fm(8,9,2,1,0),[c,b,a,d,e,y,x,z,P,leq,forcerel(P,X)])
if *a∈M b∈M c∈M d∈M e∈M y∈M x∈M z∈M*
for *a b c d e y x z*
using *that P_in_M leq_in_M ‹X∈M› forcerel_in_M*
is_Hfrc_at_iff_sats[of concl:M P leq a b c 8 9 2 1 0
[c,b,a,d,e,y,x,z,P,leq,forcerel(P,X)]] **by** *simp*
have *1:sats(M,is_wfrec_fm(Hfrc_at_fm(8,9,2,1,0),5,1,0),[y,x,z,P,leq,forcerel(P,X)])*
⟷
is_wfrec(##M, is_Hfrc_at(##M,P,leq),forcerel(P,X), x, y)
if *x∈M y∈M z∈M* **for** *x y z*
using *that ‹X∈M› forcerel_in_M P_in_M leq_in_M*
sats_is_wfrec_fm[OF 0]
by *simp*
let

?f=Exists(And(pair_fm(1,0,2),is_wfrec_fm(Hfrc_at_fm(8,9,2,1,0),5,1,0)))
have *satsf*:*sats*(*M*, ?*f*, [*x*,*z*,*P*,*leq*,*forcerel*(*P*,*X*)]) ⟷
(∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*z*) & *is_wfrec*(##*M*, *is_Hfrc_at*(##*M*,*P*,*leq*),*forcerel*(*P*,*X*), *x*, *y*))
if *x*∈*M* *z*∈*M* **for** *x z*
using *that 1* ⟨*X*∈*M*⟩ *forcerel_in_M P_in_M leq_in_M* **by** (*simp del*:*pair_abs*)
have *artyf*:*arity*(?*f*) = 5
unfolding *fm_definitions PHcheck_fm_def is_tuple_fm_def*
by (*simp add*:*nat_simp_union*)
moreover
have ?*f*∈*formula*
unfolding *fm_definitions* **by** *simp*
ultimately
have *strong_replacement*(##*M*,λ*x z*. *sats*(*M*,?*f*,[*x*,*z*,*P*,*leq*,*forcerel*(*P*,*X*)]))
using *replacement_ax 1 artyf* ⟨*X*∈*M*⟩ *forcerel_in_M P_in_M leq_in_M* **by** *simp*
then
have *strong_replacement*(##*M*,λ*x z*.
∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*z*) & *is_wfrec*(##*M*, *is_Hfrc_at*(##*M*,*P*,*leq*),*forcerel*(*P*,*X*), *x*, *y*))
using *repl_sats*[*of M* ?*f* [*P*,*leq*,*forcerel*(*P*,*X*)]] *satsf* **by** (*simp del*:*pair_abs*)
then
show ?*thesis* **unfolding** *wfrec_replacement_def* **by** *simp*
qed

lemma *names_below_abs* :
⟦*Q*∈*M*;*x*∈*M*;*nb*∈*M*⟧ ⟹ *is_names_below*(##*M*,*Q*,*x*,*nb*) ⟷ *nb* = *names_below*(*Q*,*x*)
unfolding *is_names_below_def names_below_def*
using *succ_in_M_iff zero_in_M cartprod_closed is_ecloseN_abs ecloseN_closed*
by *auto*

lemma *names_below_closed*:
⟦*Q*∈*M*;*x*∈*M*⟧ ⟹ *names_below*(*Q*,*x*) ∈ *M*
unfolding *names_below_def*
using *zero_in_M cartprod_closed ecloseN_closed succ_in_M_iff*
by *simp*

lemma *names_below_productE* :
assumes *Q* ∈ *M x* ∈ *M*
⋀*A1 A2 A3 A4*. *A1* ∈ *M* ⟹ *A2* ∈ *M* ⟹ *A3* ∈ *M* ⟹ *A4* ∈ *M* ⟹ *R*(*A1* × *A2* × *A3* × *A4*)
shows *R*(*names_below*(*Q*,*x*))
unfolding *names_below_def* **using** *assms zero_in_M ecloseN_closed*[*of x*] *twoN_in_M*
by *auto*

lemma *forcerel_abs* :
⟦*x*∈*M*;*z*∈*M*⟧ ⟹ *is_forcerel*(##*M*,*P*,*x*,*z*) ⟷ *z* = *forcerel*(*P*,*x*)
unfolding *is_forcerel_def forcerel_def*
using *frecrel_abs names_below_abs trancl_abs P_in_M twoN_in_M ecloseN_closed names_below_closed*

names_below_productE[of concl:λp. is_frecrel(##M,p,_) ⟷ _ = frecrel(p)]
frecrel_closed
by simp

lemma frc_at_abs:
assumes fnnc∈M z∈M
shows is_frc_at(##M,P,leq,fnnc,z) ⟷ z = frc_at(P,leq,fnnc)
proof -
from assms
have (∃ r∈M. is_forcerel(##M,P,fnnc, r) ∧ is_wfrec(##M, is_Hfrc_at(##M,
P, leq), r, fnnc, z))
⟷ is_wfrec(##M, is_Hfrc_at(##M, P, leq), forcerel(P,fnnc), fnnc, z)
using forcerel_abs forcerel_in_M **by** simp
then
show ?thesis
unfolding frc_at_trancl is_frc_at_def
using assms wfrec_Hfrc_at[of fnnc] wf_forcerel trans_forcerel_t relation_forcerel_t
forcerel_in_M
Hfrc_at_closed relation2_Hfrc_at_abs
trans_wfrec_abs[of forcerel(P,fnnc) fnnc z is_Hfrc_at(##M,P,leq) λx f. bool_of_o(Hfrc(P,leq,x,f))]
by (simp flip:setclass_iff)
qed

lemma forces_eq'_abs :
⟦p∈M ; t1∈M ; t2∈M⟧ ⟹ is_forces_eq'(##M,P,leq,p,t1,t2) ⟷ forces_eq'(P,leq,p,t1,t2)
unfolding is_forces_eq'_def forces_eq'_def
using frc_at_abs zero_in_M tuples_in_M **by** (auto simp add:components_abs)

lemma forces_mem'_abs :
⟦p∈M ; t1∈M ; t2∈M⟧ ⟹ is_forces_mem'(##M,P,leq,p,t1,t2) ⟷ forces_mem'(P,leq,p,t1,t2)
unfolding is_forces_mem'_def forces_mem'_def
using frc_at_abs zero_in_M tuples_in_M **by** (auto simp add:components_abs)

lemma forces_neq'_abs :
assumes
p∈M t1∈M t2∈M
shows
is_forces_neq'(##M,P,leq,p,t1,t2) ⟷ forces_neq'(P,leq,p,t1,t2)
proof -
have q∈M **if** q∈P **for** q
using that transitivity P_in_M **by** simp
then show ?thesis
unfolding is_forces_neq'_def forces_neq'_def
using assms forces_eq'_abs pair_in_M_iff
by (auto simp add:components_abs,blast)
qed

lemma forces_nmem'_abs :

assumes
  $p \in M$ $t1 \in M$ $t2 \in M$
shows
  is_forces_nmem'(##*M*,*P*,*leq*,*p*,*t1*,*t2*) ⟷ *forces_nmem'*(*P*,*leq*,*p*,*t1*,*t2*)
proof -
  have $q \in M$ **if** $q \in P$ **for** *q*
    using *that transitivity P_in_M* **by** *simp*
  then show *?thesis*
    unfolding *is_forces_nmem'_def forces_nmem'_def*
    using *assms forces_mem'_abs pair_in_M_iff*
    by (*auto simp add*:*components_abs*,*blast*)
qed

end

18.7 Forcing for general formulas

definition
  ren_forces_nand :: *i*⇒*i* **where**
  ren_forces_nand(φ) ≡ *Exists*(*And*(*Equal*(*0*,*1*),*iterates*(λ*p*. *incr_bv*(*p*)‘*1* , *2*, φ)))

lemma *ren_forces_nand_type*[*TC*] :
  φ∈*formula* ⟹ *ren_forces_nand*(φ) ∈*formula*
  unfolding *ren_forces_nand_def*
  by *simp*

lemma *arity_ren_forces_nand* :
  assumes φ∈*formula*
  shows *arity*(*ren_forces_nand*(φ)) ≤ *succ*(*arity*(φ))
proof -
  consider (*lt*) *1*<*arity*(φ) | (*ge*) ¬ *1* < *arity*(φ)
    by *auto*
  then
  show *?thesis*
  proof *cases*
    case *lt*
    with ‹φ∈_›
    have *2* < *succ*(*arity*(φ)) *2*<*arity*(φ)#+*2*
      using *succ_ltI* **by** *auto*
    with ‹φ∈_›
    have *arity*(*iterates*(λ*p*. *incr_bv*(*p*)‘*1*,*2*,φ)) = *2*#+*arity*(φ)
      using *arity_incr_bv_lemma lt*
      by *auto*
    with ‹φ∈_›
    show *?thesis*
      unfolding *ren_forces_nand_def*
      using *lt pred_Un_distrib nat_union_abs1 Un_assoc*[*symmetric*] *Un_le_compat*
      by *simp*
  next

case *ge*
with ‹φ∈_›
have *arity*(φ) ≤ *1* *pred*(*arity*(φ)) ≤ *1*
using *not_lt_iff_le le_trans*[*OF le_pred*]
by *simp_all*
with ‹φ∈_›
have *arity*(*iterates*(λp. *incr_bv*(*p*)‘*1*,*2*,φ)) = (*arity*(φ))
using *arity_incr_bv_lemma ge*
by *simp*
with ‹*arity*(φ) ≤ *1*› ‹φ∈_› ‹*pred*(_) ≤ *1*›
show *?thesis*
unfolding *ren_forces_nand_def*
using *pred_Un_distrib nat_union_abs1 Un_assoc*[*symmetric*] *nat_union_abs2*
by *simp*
qed
qed

lemma *sats_ren_forces_nand*:
[*q*,*P*,*leq*,*o*,*p*] @ *env* ∈ *list*(*M*) ⟹ φ∈*formula* ⟹
sats(*M*, *ren_forces_nand*(φ),[*q*,*p*,*P*,*leq*,*o*] @ *env*) ⟷ *sats*(*M*, φ,[*q*,*P*,*leq*,*o*] @ *env*)
unfolding *ren_forces_nand_def*
using *sats_incr_bv_iff* [*of* _ _ *M* _ [*q*]]
by *simp*

definition
ren_forces_forall :: *i*⇒*i* **where**
ren_forces_forall(φ) ≡
Exists(*Exists*(*Exists*(*Exists*(*Exists*(
And(*Equal*(*0*,*6*),*And*(*Equal*(*1*,*7*),*And*(*Equal*(*2*,*8*),*And*(*Equal*(*3*,*9*),
And(*Equal*(*4*,*5*),*iterates*(λp. *incr_bv*(*p*)‘*5* , *5*, φ)))))))))))

lemma *arity_ren_forces_all* :
assumes φ∈*formula*
shows *arity*(*ren_forces_forall*(φ)) = *5* ∪ *arity*(φ)
proof -
consider (*lt*) *5*<*arity*(φ) | (*ge*) ¬ *5* < *arity*(φ)
by *auto*
then
show *?thesis*
proof *cases*
case *lt*
with ‹φ∈_›
have *5* < *succ*(*arity*(φ)) *5*<*arity*(φ)#+*2* *5*<*arity*(φ)#+*3* *5*<*arity*(φ)#+*4*
using *succ_ltI* **by** *auto*
with ‹φ∈_›
have *arity*(*iterates*(λp. *incr_bv*(*p*)‘*5*,*5*,φ)) = *5*#+*arity*(φ)
using *arity_incr_bv_lemma lt*

by *simp*
with ‹φ∈_›
show *?thesis*
unfolding *ren_forces_forall_def*
using *pred_Un_distrib nat_union_abs1 Un_assoc*[*symmetric*] *nat_union_abs2*
by *simp*
next
case *ge*
with ‹φ∈_›
have *arity*(φ) ≤ *5 pred^5*(*arity*(φ)) ≤ *5*
using *not_lt_iff_le le_trans*[*OF le_pred*]
by *simp_all*
with ‹φ∈_›
have *arity*(*iterates*(λ*p. incr_bv*(*p*)‘*5*,*5*,φ)) = *arity*(φ)
using *arity_incr_bv_lemma ge*
by *simp*
with ‹*arity*(φ) ≤ *5*› ‹φ∈_› ‹*pred^5*(_) ≤ *5*›
show *?thesis*
unfolding *ren_forces_forall_def*
using *pred_Un_distrib nat_union_abs1 Un_assoc*[*symmetric*] *nat_union_abs2*
by *simp*
qed
qed

lemma *ren_forces_forall_type*[*TC*] :
φ∈*formula* ⟹ *ren_forces_forall*(φ) ∈*formula*
unfolding *ren_forces_forall_def* **by** *simp*

lemma *sats_ren_forces_forall* :
[*x*,*P*,*leq*,*o*,*p*] @ *env* ∈ *list*(*M*) ⟹ φ∈*formula* ⟹
sats(*M*, *ren_forces_forall*(φ),[*x*,*p*,*P*,*leq*,*o*] @ *env*) ⟷ *sats*(*M*, φ,[*p*,*P*,*leq*,*o*,*x*] @ *env*)
unfolding *ren_forces_forall_def*
using *sats_incr_bv_iff* [*of* _ _ *M* _ [*p*,*P*,*leq*,*o*,*x*]]
by *simp*

definition
is_leq :: [*i*⇒*o*,*i*,*i*,*i*] ⇒ *o* **where**
is_leq(*A*,*l*,*q*,*p*) ≡ ∃ *qp*[*A*]. (*pair*(*A*,*q*,*p*,*qp*) ∧ *qp*∈*l*)

lemma (**in** *forcing_data*) *leq_abs*:
⟦ *l*∈*M* ; *q*∈*M* ; *p*∈*M* ⟧ ⟹ *is_leq*(##*M*,*l*,*q*,*p*) ⟷ ⟨*q*,*p*⟩∈*l*
unfolding *is_leq_def* **using** *pair_in_M_iff* **by** *simp*

definition
leq_fm :: [*i*,*i*,*i*] ⇒ *i* **where**
leq_fm(*leq*,*q*,*p*) ≡ *Exists*(*And*(*pair_fm*(*q*#+*1*,*p*#+*1*,*0*),*Member*(*0*,*leq*#+*1*)))

lemma *arity_leq_fm* :
⟦*leq*∈*nat*;*q*∈*nat*;*p*∈*nat*⟧ ⟹ *arity*(*leq_fm*(*leq*,*q*,*p*)) = *succ*(*q*) ∪ *succ*(*p*) ∪ *succ*(*leq*)
unfolding *leq_fm_def*
using *arity_pair_fm pred_Un_distrib nat_simp_union*
by *auto*

lemma *leq_fm_type*[*TC*] :
⟦*leq*∈*nat*;*q*∈*nat*;*p*∈*nat*⟧ ⟹ *leq_fm*(*leq*,*q*,*p*)∈*formula*
unfolding *leq_fm_def* **by** *simp*

lemma *sats_leq_fm* :
⟦ *leq*∈*nat*;*q*∈*nat*;*p*∈*nat*;*env*∈*list*(*A*) ⟧ ⟹
sats(*A*,*leq_fm*(*leq*,*q*,*p*),*env*) ⟷ *is_leq*(##*A*,*nth*(*leq*,*env*),*nth*(*q*,*env*),*nth*(*p*,*env*))
unfolding *leq_fm_def is_leq_def* **by** *simp*

18.7.1 The primitive recursion

consts *forces'* :: *i*⇒*i*
primrec
forces'(*Member*(*x*,*y*)) = *forces_mem_fm*(*1*,*2*,*0*,*x*#+*4*,*y*#+*4*)
forces'(*Equal*(*x*,*y*)) = *forces_eq_fm*(*1*,*2*,*0*,*x*#+*4*,*y*#+*4*)
forces'(*Nand*(*p*,*q*)) =
Neg(*Exists*(*And*(*Member*(*0*,*2*),*And*(*leq_fm*(*3*,*0*,*1*),*And*(*ren_forces_nand*(*forces'*(*p*)),
ren_forces_nand(*forces'*(*q*)))))))
forces'(*Forall*(*p*)) = *Forall*(*ren_forces_forall*(*forces'*(*p*)))

definition
forces :: *i*⇒*i* **where**
forces(*φ*) ≡ *And*(*Member*(*0*,*1*),*forces'*(*φ*))

lemma *forces'_type* [*TC*]: *φ*∈*formula* ⟹ *forces'*(*φ*) ∈ *formula*
by (*induct φ set*:*formula*; *simp*)

lemma *forces_type*[*TC*] : *φ*∈*formula* ⟹ *forces*(*φ*) ∈ *formula*
unfolding *forces_def* **by** *simp*

context *forcing_data*
begin

18.8 Forcing for atomic formulas in context

definition
forces_eq :: [*i*,*i*,*i*] ⇒ *o* **where**
forces_eq ≡ *forces_eq'*(*P*,*leq*)

definition
forces_mem :: [*i*,*i*,*i*] ⇒ *o* **where**
forces_mem ≡ *forces_mem'*(*P*,*leq*)

definition
is_forces_eq :: $[i,i,i] \Rightarrow o$ **where**
is_forces_eq $\equiv$ *is_forces_eq'*(##*M*,*P*,*leq*)

definition
is_forces_mem :: $[i,i,i] \Rightarrow o$ **where**
is_forces_mem $\equiv$ *is_forces_mem'*(##*M*,*P*,*leq*)

lemma *def_forces_eq*: $p \in P \Longrightarrow$ *forces_eq*$(p,t1,t2) \longleftrightarrow$
$(\forall s \in domain(t1) \cup domain(t2).\ \forall q.\ q \in P \wedge q \preceq p \longrightarrow$
$(forces\_mem(q,s,t1) \longleftrightarrow forces\_mem(q,s,t2)))$
unfolding *forces_eq_def forces_mem_def forces_eq'_def forces_mem'_def*
using *def_frc_at*[*of p 0 t1 t2*] **unfolding** *bool_of_o_def*
by *auto*

lemma *def_forces_mem*: $p \in P \Longrightarrow$ *forces_mem*$(p,t1,t2) \longleftrightarrow$
$(\forall v \in P.\ v \preceq p \longrightarrow$
$(\exists q.\ \exists s.\ \exists r.\ r \in P \wedge q \in P \wedge q \preceq v \wedge \langle s,r \rangle \in t2 \wedge q \preceq r \wedge forces\_eq(q,t1,s)))$
unfolding *forces_eq'_def forces_mem'_def forces_eq_def forces_mem_def*
using *def_frc_at*[*of p 1 t1 t2*] **unfolding** *bool_of_o_def*
by *auto*

lemma *forces_eq_abs* :
$[\![p \in M\ ;\ t1 \in M\ ;\ t2 \in M]\!] \Longrightarrow is\_forces\_eq(p,t1,t2) \longleftrightarrow forces\_eq(p,t1,t2)$
unfolding *is_forces_eq_def forces_eq_def*
using *forces_eq'_abs* **by** *simp*

lemma *forces_mem_abs* :
$[\![p \in M\ ;\ t1 \in M\ ;\ t2 \in M]\!] \Longrightarrow is\_forces\_mem(p,t1,t2) \longleftrightarrow forces\_mem(p,t1,t2)$
unfolding *is_forces_mem_def forces_mem_def*
using *forces_mem'_abs* **by** *simp*

lemma *sats_forces_eq_fm*:
assumes $p \in nat\ l \in nat\ q \in nat\ t1 \in nat\ t2 \in nat\ \ env \in list(M)$
$nth(p,env)=P\ nth(l,env)=leq$
shows $sats(M,forces\_eq\_fm(p,l,q,t1,t2),env) \longleftrightarrow$
$is\_forces\_eq(nth(q,env),nth(t1,env),nth(t2,env))$
unfolding *forces_eq_fm_def is_forces_eq_def is_forces_eq'_def*
using *assms sats_is_tuple_fm sats_frc_at_fm*
by *simp*

lemma *sats_forces_mem_fm*:
assumes $p \in nat\ l \in nat\ q \in nat\ t1 \in nat\ t2 \in nat\ \ env \in list(M)$
$nth(p,env)=P\ nth(l,env)=leq$
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*))
unfolding *forces_mem_fm_def is_forces_mem_def is_forces_mem′_def*
using *assms sats_is_tuple_fm sats_frc_at_fm*
by *simp*

definition
forces_neq :: [*i*,*i*,*i*] ⇒ *o* **where**
forces_neq(*p*,*t1*,*t2*) ≡ ¬ (∃ *q*∈*P*. *q*≼*p* ∧ *forces_eq*(*q*,*t1*,*t2*))

definition
forces_nmem :: [*i*,*i*,*i*] ⇒ *o* **where**
forces_nmem(*p*,*t1*,*t2*) ≡ ¬ (∃ *q*∈*P*. *q*≼*p* ∧ *forces_mem*(*q*,*t1*,*t2*))

lemma *forces_neq* :
forces_neq(*p*,*t1*,*t2*) ⟷ *forces_neq′*(*P*,*leq*,*p*,*t1*,*t2*)
unfolding *forces_neq_def forces_neq′_def forces_eq_def* **by** *simp*

lemma *forces_nmem* :
forces_nmem(*p*,*t1*,*t2*) ⟷ *forces_nmem′*(*P*,*leq*,*p*,*t1*,*t2*)
unfolding *forces_nmem_def forces_nmem′_def forces_mem_def* **by** *simp*

lemma *sats_forces_Member* :
assumes *x*∈*nat* *y*∈*nat* *env*∈*list*(*M*)
nth(*x*,*env*)=*xx* *nth*(*y*,*env*)=*yy* *q*∈*M*
shows *sats*(*M*,*forces*(*Member*(*x*,*y*)),[*q*,*P*,*leq*,*one*]@*env*) ⟷
(*q*∈*P* ∧ *is_forces_mem*(*q*,*xx*,*yy*))
unfolding *forces_def*
using *assms sats_forces_mem_fm P_in_M leq_in_M one_in_M*
by *simp*

lemma *sats_forces_Equal* :
assumes *x*∈*nat* *y*∈*nat* *env*∈*list*(*M*)
nth(*x*,*env*)=*xx* *nth*(*y*,*env*)=*yy* *q*∈*M*
shows *sats*(*M*,*forces*(*Equal*(*x*,*y*)),[*q*,*P*,*leq*,*one*]@*env*) ⟷
(*q*∈*P* ∧ *is_forces_eq*(*q*,*xx*,*yy*))
unfolding *forces_def*
using *assms sats_forces_eq_fm P_in_M leq_in_M one_in_M*
by *simp*

lemma *sats_forces_Nand* :
assumes *φ*∈*formula* *ψ*∈*formula* *env*∈*list*(*M*) *p*∈*M*
shows *sats*(*M*,*forces*(*Nand*(*φ*,*ψ*)),[*p*,*P*,*leq*,*one*]@*env*) ⟷
(*p*∈*P* ∧ ¬(∃ *q*∈*M*. *q*∈*P* ∧ *is_leq*(##*M*,*leq*,*q*,*p*) ∧
(*sats*(*M*,*forces′*(*φ*),[*q*,*P*,*leq*,*one*]@*env*) ∧ *sats*(*M*,*forces′*(*ψ*),[*q*,*P*,*leq*,*one*]@*env*))))
unfolding *forces_def* **using** *sats_leq_fm assms sats_ren_forces_nand P_in_M leq_in_M*
one_in_M

by *simp*

lemma *sats_forces_Neg* :
assumes $\varphi \in formula$ $env \in list(M)$ $p \in M$
shows $sats(M,forces(Neg(\varphi)),[p,P,leq,one]@env) \longleftrightarrow$
$(p \in P \wedge \neg(\exists q \in M.\ q \in P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
$(sats(M,forces'(\varphi),[q,P,leq,one]@env))))$
unfolding *Neg_def* **using** *assms sats_forces_Nand*
by *simp*

lemma *sats_forces_Forall* :
assumes $\varphi \in formula$ $env \in list(M)$ $p \in M$
shows $sats(M,forces(Forall(\varphi)),[p,P,leq,one]@env) \longleftrightarrow$
$p \in P \wedge (\forall x \in M.\ sats(M,forces'(\varphi),[p,P,leq,one,x]@env))$
unfolding *forces_def* **using** *assms sats_ren_forces_forall P_in_M leq_in_M one_in_M*
by *simp*

end

18.9 The arity of *forces*

lemma *arity_forces_at*:
assumes $x \in nat$ $y \in nat$
shows $arity(forces(Member(x,\ y))) = (succ(x) \cup succ(y))\ \#+\ 4$
$arity(forces(Equal(x,\ y))) = (succ(x) \cup succ(y))\ \#+\ 4$
unfolding *forces_def*
using *assms arity_forces_mem_fm arity_forces_eq_fm succ_Un_distrib nat_simp_union*
by *auto*

lemma *arity_forces'*:
assumes $\varphi \in formula$
shows $arity(forces'(\varphi)) \leq arity(\varphi)\ \#+\ 4$
using *assms*
proof (*induct set:formula*)
case (*Member x y*)
then
show *?case*
using *arity_forces_mem_fm succ_Un_distrib nat_simp_union*
by *simp*
next
case (*Equal x y*)
then
show *?case*
using *arity_forces_eq_fm succ_Un_distrib nat_simp_union*
by *simp*
next
case (*Nand* φ ψ)
let $?\varphi' = ren\_forces\_nand(forces'(\varphi))$
let $?\psi' = ren\_forces\_nand(forces'(\psi))$

```
  have arity(leq_fm(3, 0, 1)) = 4
    using arity_leq_fm succ_Un_distrib nat_simp_union
    by simp
  have 3 ≤ (4#+arity(φ)) ∪ (4#+arity(ψ)) (is _ ≤ ?rhs)
    using nat_simp_union by simp
  from ‹φ∈_› Nand
  have pred(arity(?φ')) ≤ ?rhs  pred(arity(?ψ')) ≤ ?rhs
  proof -
    from ‹φ∈_› ‹ψ∈_›
    have A:pred(arity(?φ')) ≤ arity(forces'(φ))
      pred(arity(?ψ')) ≤ arity(forces'(ψ))
      using pred_mono[OF _  arity_ren_forces_nand] pred_succ_eq
      by simp_all
    from Nand
    have 3 ∪ arity(forces'(φ)) ≤ arity(φ) #+ 4
      3 ∪ arity(forces'(ψ)) ≤ arity(ψ) #+ 4
      using Un_le by simp_all
    with Nand
    show pred(arity(?φ')) ≤ ?rhs
      pred(arity(?ψ')) ≤ ?rhs
      using le_trans[OF A(1)] le_trans[OF A(2)] le_Un_iff
      by simp_all
  qed
  with Nand ‹_=4›
  show ?case
   using pred_Un_distrib Un_assoc[symmetric] succ_Un_distrib nat_union_abs1 Un_leI3[OF
‹3 ≤ ?rhs›]
    by simp
next
  case (Forall φ)
  let ?φ' = ren_forces_forall(forces'(φ))
  show ?case
  proof (cases arity(φ) = 0)
    case True
    with Forall
    show ?thesis
    proof -
      from Forall True
      have arity(forces'(φ)) ≤ 5
        using le_trans[of _ 4 5] by auto
      with ‹φ∈_›
      have arity(?φ') ≤ 5
        using arity_ren_forces_all[OF forces'_type[OF ‹φ∈_›]] nat_union_abs2
        by auto
      with Forall True
      show ?thesis
        using pred_mono[OF _ ‹arity(?φ') ≤ 5›]
        by simp
    qed
```

```
  next
    case False
    with Forall
    show ?thesis
    proof -
      from Forall False
      have arity(?φ') = 5 ∪ arity(forces'(φ))
        arity(forces'(φ)) ≤ 5 #+ arity(φ)
        4 ≤ succ(succ(succ(arity(φ))))
        using Ord_0_lt arity_ren_forces_all
          le_trans[OF _ add_le_mono[of 4 5, OF _ le_refl]]
        by auto
      with ‹φ∈_›
      have 5 ∪ arity(forces'(φ)) ≤ 5#+arity(φ)
        using nat_simp_union by auto
      with ‹φ∈_› ‹arity(?φ') = 5 ∪ _›
      show ?thesis
        using pred_Un_distrib succ_pred_eq[OF _ ‹arity(φ)≠0›]
          pred_mono[OF _ Forall(2)] Un_le[OF ‹4≤succ(_)›]
        by simp
    qed
  qed
qed

lemma arity_forces :
  assumes φ∈formula
  shows arity(forces(φ)) ≤ 4#+arity(φ)
  unfolding forces_def
  using assms arity_forces' le_trans nat_simp_union by auto

lemma arity_forces_le :
  assumes φ∈formula n∈nat arity(φ) ≤ n
  shows arity(forces(φ)) ≤ 4#+n
  using assms le_trans[OF _ add_le_mono[OF le_refl[of 5] ‹arity(φ)≤_›]] arity_forces
  by auto

end
```

19 The Forcing Theorems

```
theory Forcing_Theorems
  imports
    Forces_Definition

begin

context forcing_data
begin
```

19.1 The forcing relation in context

abbreviation *Forces* :: $[i, i, i] \Rightarrow o$ (_ ⊩ _ _ [*36*,*36*,*36*] *60*) **where**
p ⊩ φ *env* ≡ *M*, ([*p*,*P*,*leq*,*one*] @ *env*) ⊨ *forces*(φ)

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 -
have *z*∈*P* ⟹ *z*∈*M* **for** *z*
using *P_in_M transitivity*[*of z P*] **by** *simp*
moreover
have *separation*(##*M*,λ*p*. (*p* ⊩ φ *env*))
using *separation_ax arity_forces far fty P_in_M leq_in_M one_in_M envty arity_forces_le*
by *simp*
then
have *Collect*(*P*,λ*p*. (*p* ⊩ φ *env*))∈*M*
using *separation_closed P_in_M* **by** *simp*
then show *?thesis* **by** *simp*
qed

lemma *forces_mem_iff_dense_below*: *p*∈*P* ⟹ *forces_mem*(*p*,*t1*,*t2*) ⟷ *dense_below*(
{*q*∈*P*. ∃ *s*. ∃ *r*. *r*∈*P* ∧ ⟨*s*,*r*⟩ ∈ *t2* ∧ *q*≼*r* ∧ *forces_eq*(*q*,*t1*,*s*)}
,*p*)
using *def_forces_mem*[*of p t1 t2*] **by** *blast*

19.2 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_eq*:
assumes *p*∈*P* *r*∈*P* *r*≼*p* *forces_eq*(*p*,*t1*,*t2*)
shows *forces_eq*(*r*,*t1*,*t2*)
using *assms def_forces_eq*[*of _ t1 t2*] *leq_transD* **by** *blast*

19.3 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_mem*:
assumes *p*∈*P* *r*∈*P* *r*≼*p* *forces_mem*(*p*,*t1*,*t2*)
shows *forces_mem*(*r*,*t1*,*t2*)
using *assms forces_mem_iff_dense_below dense_below_under* **by** *auto*

19.4 Kunen 2013, Lemma IV.2.37(b)

lemma *density_mem*:
assumes *p*∈*P*
shows *forces_mem*(*p*,*t1*,*t2*) ⟷ *dense_below*({*q*∈*P*. *forces_mem*(*q*,*t1*,*t2*)},*p*)

proof
  assume *forces_mem*$(p,t1,t2)$
  with *assms*
  show *dense_below*$(\{q\in P.$ *forces_mem*$(q,t1,t2)\},p)$
    using *forces_mem_iff_dense_below strengthening_mem*[*of p*] *ideal_dense_below* **by** *auto*
next
  assume *dense_below*$(\{q \in P$. *forces_mem*$(q, t1, t2)\}, p)$
  with *assms*
  have *dense_below*$(\{q\in P.$
    dense_below$(\{q'\in P. \exists s\ r.\ r \in P \wedge \langle s,r\rangle\in t2 \wedge q'\preceq r \wedge$ *forces_eq*$(q',t1,s)\},q)$
    $\},p)$
    using *forces_mem_iff_dense_below* **by** *simp*
  with *assms*
  show *forces_mem*$(p,t1,t2)$
    using *dense_below_dense_below forces_mem_iff_dense_below*[*of p t1 t2*] **by** *blast*
qed

lemma *aux_density_eq*:
  assumes
    dense_below(
    $\{q'\in P. \forall q.\ q\in P \wedge q\preceq q' \longrightarrow$ *forces_mem*$(q,s,t1) \longleftrightarrow$ *forces_mem*$(q,s,t2)\}$
    $,p)$
    forces_mem$(q,s,t1)$ $q\in P$ $p\in P$ $q\preceq p$
  shows
    dense_below$(\{r\in P.$ *forces_mem*$(r,s,t2)\},q)$
proof
  fix r
  assume $r\in P$ $r\preceq q$
  moreover from *this* **and** ‹$p\in P$› ‹$q\preceq p$› ‹$q\in P$›
  have $r\preceq p$
    using *leq_transD* **by** *simp*
  moreover
  note ‹*forces_mem*$(q,s,t1)$› ‹*dense_below*$(\_,p)$› ‹$q\in P$›
  ultimately
  obtain $q1$ **where** $q1\preceq r$ $q1\in P$ *forces_mem*$(q1,s,t2)$
    using *strengthening_mem*[*of q _ s t1*] *refl_leq leq_transD*[*of _ r q*] **by** *blast*
  then
  show $\exists d\in\{r \in P$. *forces_mem*$(r, s, t2)\}.\ d \in P \wedge d\preceq r$
    by *blast*
qed

lemma *density_eq*:
  assumes $p\in P$
  shows *forces_eq*$(p,t1,t2) \longleftrightarrow$ *dense_below*$(\{q\in P.$ *forces_eq*$(q,t1,t2)\},p)$
proof
  assume *forces_eq*$(p,t1,t2)$
  with ‹$p\in P$›

```
  show dense_below({q∈P. forces_eq(q,t1,t2)},p)
    using strengthening_eq ideal_dense_below by auto
next
  assume dense_below({q∈P. forces_eq(q,t1,t2)},p)
  {
    fix s q
    let ?D1={q'∈P. ∀s∈domain(t1) ∪ domain(t2). ∀q. q ∈ P ∧ q≼q' ⟶
          forces_mem(q,s,t1)⟷forces_mem(q,s,t2)}
    let ?D2={q'∈P. ∀q. q∈P ∧ q≼q' ⟶ forces_mem(q,s,t1) ⟷ forces_mem(q,s,t2)}
    assume s∈domain(t1) ∪ domain(t2)
    then
    have ?D1⊆?D2 by blast
    with ‹dense_below(_,p)›
     have dense_below({q'∈P. ∀s∈domain(t1) ∪ domain(t2). ∀q. q ∈ P ∧ q≼q'
⟶
          forces_mem(q,s,t1)⟷forces_mem(q,s,t2)},p)
      using dense_below_cong'[OF ‹p∈P› def_forces_eq[of _ t1 t2]] by simp
    with ‹p∈P› ‹?D1⊆?D2›
    have dense_below({q'∈P. ∀q. q∈P ∧ q≼q' ⟶
          forces_mem(q,s,t1) ⟷ forces_mem(q,s,t2)},p)
      using dense_below_mono by simp
    moreover from this
    have  dense_below({q'∈P. ∀q. q∈P ∧ q≼q' ⟶
          forces_mem(q,s,t2) ⟷ forces_mem(q,s,t1)},p)
      by blast
    moreover
    assume q ∈ P q≼p
    moreover
    note ‹p∈P›
    ultimately
    have forces_mem(q,s,t1) ⟹ dense_below({r∈P. forces_mem(r,s,t2)},q)
        forces_mem(q,s,t2) ⟹ dense_below({r∈P. forces_mem(r,s,t1)},q)
      using aux_density_eq by simp_all
    then
    have forces_mem(q, s, t1) ⟷ forces_mem(q, s, t2)
      using density_mem[OF ‹q∈P›] by blast
  }
  with ‹p∈P›
  show forces_eq(p,t1,t2) using def_forces_eq by blast
qed
```

19.5 Kunen 2013, Lemma IV.2.38

```
lemma not_forces_neq:
  assumes p∈P
  shows forces_eq(p,t1,t2) ⟷ ¬ (∃q∈P. q≼p ∧ forces_neq(q,t1,t2))
  using assms density_eq unfolding forces_neq_def by blast
```

lemma *not_forces_nmem*:
  assumes *p*∈*P*
  shows *forces_mem*(*p*,*t1*,*t2*) ⟷ ¬ (∃ *q*∈*P*. *q*≼*p* ∧ *forces_nmem*(*q*,*t1*,*t2*))
  using *assms density_mem* **unfolding** *forces_nmem_def* **by** *blast*

lemma *sats_forces_Nand′*:
  assumes
    p∈*P* φ∈*formula* ψ∈*formula env* ∈ *list*(*M*)
  shows
    M, [*p*,*P*,*leq*,*one*] @ *env* ⊨ *forces*(*Nand*(φ,ψ)) ⟷
     ¬(∃ *q*∈*M*. *q*∈*P* ∧ *is_leq*(##*M*,*leq*,*q*,*p*) ∧
          M, [*q*,*P*,*leq*,*one*] @ *env* ⊨ *forces*(φ) ∧
          M, [*q*,*P*,*leq*,*one*] @ *env* ⊨ *forces*(ψ))
  using *assms sats_forces_Nand*[*OF assms*(*2-4*) *transitivity*[*OF* ‹*p*∈*P*›]]
  P_in_M leq_in_M one_in_M **unfolding** *forces_def*
  by *simp*

lemma *sats_forces_Neg′*:
  assumes
    p∈*P env* ∈ *list*(*M*) φ∈*formula*
  shows
    M, [*p*,*P*,*leq*,*one*] @ *env* ⊨ *forces*(*Neg*(φ))   ⟷
     ¬(∃ *q*∈*M*. *q*∈*P* ∧ *is_leq*(##*M*,*leq*,*q*,*p*) ∧
         M, [*q*,*P*,*leq*,*one*]@*env* ⊨ *forces*(φ))
  using *assms sats_forces_Neg transitivity*
  P_in_M leq_in_M one_in_M **unfolding** *forces_def*
  by (*simp*, *blast*)

lemma *sats_forces_Forall′*:
  assumes
    p∈*P env* ∈ *list*(*M*) φ∈*formula*
  shows
    M,[*p*,*P*,*leq*,*one*] @ *env* ⊨ *forces*(*Forall*(φ)) ⟷
     (∀ *x*∈*M*.   *M*, [*p*,*P*,*leq*,*one*,*x*] @ *env* ⊨ *forces*(φ))
  using *assms sats_forces_Forall transitivity*
  P_in_M leq_in_M one_in_M sats_ren_forces_forall **unfolding** *forces_def*
  by *simp*

19.6 The relation of forcing and atomic formulas

lemma *Forces_Equal*:
  assumes
   p∈*P t1*∈*M t2*∈*M env*∈*list*(*M*) *nth*(*n*,*env*) = *t1 nth*(*m*,*env*) = *t2 n*∈*nat m*∈*nat*

  shows

(p ⊩ Equal(n,m) env) ⟷ forces_eq(p,t1,t2)
using assms sats_forces_Equal forces_eq_abs transitivity P_in_M
by simp

lemma *Forces_Member*:
assumes
$p{\in}P$ $t1{\in}M$ $t2{\in}M$ $env{\in}list(M)$ $nth(n,env) = t1$ $nth(m,env) = t2$ $n{\in}nat$ $m{\in}nat$

shows
(p ⊩ Member(n,m) env) ⟷ forces_mem(p,t1,t2)
using *assms sats_forces_Member forces_mem_abs transitivity P_in_M*
by *simp*

lemma *Forces_Neg*:
assumes
$p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$
shows
(p ⊩ Neg(φ) env) ⟷ ¬(∃ q∈M. q∈P ∧ q≼p ∧ (q ⊩ φ env))
using *assms sats_forces_Neg' transitivity*
P_in_M pair_in_M_iff leq_in_M leq_abs **by** *simp*

19.7 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 ⊩ Nand(φ,ψ) env) ⟷ ¬(∃ q∈M. q∈P ∧ q≼p ∧ (q ⊩ φ env) ∧ (q ⊩ ψ env))
using *assms sats_forces_Nand' transitivity*
P_in_M pair_in_M_iff leq_in_M leq_abs **by** *simp*

lemma *Forces_And_aux*:
assumes
$p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$ $\psi{\in}formula$
shows
p ⊩ And(φ,ψ) env ⟷
(∀ q∈M. q∈P ∧ q≼p ⟶ (∃ r∈M. r∈P ∧ r≼q ∧ (r ⊩ φ env) ∧ (r ⊩ ψ env)))
unfolding *And_def* **using** *assms Forces_Neg Forces_Nand* **by** (*auto simp only*:)

lemma *Forces_And_iff_dense_below*:
assumes
$p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$ $\psi{\in}formula$
shows
(p ⊩ And(φ,ψ) env) ⟷ dense_below({r∈P. (r ⊩ φ env) ∧ (r ⊩ ψ env) },p)
unfolding *dense_below_def* **using** *Forces_And_aux assms*
by (*auto dest*:*transitivity*[*OF _ P_in_M*]; *rename_tac q*; *drule_tac x=q* **in** *bspec*)+

lemma *Forces_Forall*:

assumes
p∈*P* *env* ∈ *list*(*M*) *φ*∈*formula*
shows
(*p* ⊩ *Forall*(*φ*) *env*) ⟷ (∀ *x*∈*M*. (*p* ⊩ *φ* ([*x*] @ *env*)))
using *sats_forces_Forall′ assms* **by** *simp*

bundle *some_rules* = *elem_of_val_pair* [*dest*] *SepReplace_iff* [*simp del*] *SepReplace_iff* [*iff*]

context
includes *some_rules*
begin

lemma *elem_of_valI*: ∃ *ϑ*. ∃ *p*∈*P*. *p*∈*G* ∧ ⟨*ϑ*,*p*⟩∈*π* ∧ *val*(*P*,*G*,*ϑ*) = *x* ⟹ *x*∈*val*(*P*,*G*,*π*)
by (*subst def_val*, *auto*)

lemma *GenExtD*: *x*∈*M*[*G*] ⟷ (∃ *τ*∈*M*. *x* = *val*(*P*,*G*,*τ*))
unfolding *GenExt_def* **by** *simp*

lemma *left_in_M* : *tau*∈*M* ⟹ ⟨*a*,*b*⟩∈*tau* ⟹ *a*∈*M*
using *fst_snd_closed*[*of* ⟨*a*,*b*⟩] *transitivity* **by** *auto*

19.8 Kunen 2013, Lemma IV.2.29

lemma *generic_inter_dense_below*:
assumes *D*∈*M* *M_generic*(*G*) *dense_below*(*D*,*p*) *p*∈*G*
shows *D* ∩ *G* ≠ *0*
proof -
let *?D*={*q*∈*P*. *p*⊥*q* ∨ *q*∈*D*}
have *dense*(*?D*)
proof
fix *r*
assume *r*∈*P*
show ∃ *d*∈{*q* ∈ *P* . *p* ⊥ *q* ∨ *q* ∈ *D*}. *d* ≼ *r*
proof (*cases p* ⊥ *r*)
case *True*
with ‹*r*∈*P*›

show *?thesis* **using** *refl_leq*[*of r*] **by** (*intro bexI*) (*blast+*)
next
case *False*
then
obtain *s* **where** *s*∈*P* *s*≼*p* *s*≼*r* **by** *blast*
with *assms* ‹*r*∈*P*›
show *?thesis*
using *dense_belowD*[*OF assms*(*3*), *of s*] *leq_transD*[*of _ s r*]
by *blast*
qed
qed

```
  have ?D⊆P by auto

  let ?d_fm=Or(Neg(compat_in_fm(1,2,3,0)),Member(0,4))
  have 1:p∈M
    using ‹M_generic(G)› M_genericD transitivity[OF _ P_in_M]
          ‹p∈G› by simp
  moreover
  have ?d_fm∈formula by simp
  moreover
  have arity(?d_fm) = 5 unfolding compat_in_fm_def pair_fm_def upair_fm_def
    by (simp add: nat_union_abs1 Un_commute)
  moreover
  have (M, [q,P,leq,p,D] ⊨ ?d_fm) ⟷ (¬ is_compat_in(##M,P,leq,p,q) ∨ q∈D)
    if q∈M for q
    using that sats_compat_in_fm P_in_M leq_in_M 1 ‹D∈M› by simp
  moreover
  have (¬ is_compat_in(##M,P,leq,p,q) ∨ q∈D) ⟷ p⊥q ∨ q∈D if q∈M for q
    unfolding compat_def using that compat_in_abs P_in_M leq_in_M 1 by simp
  ultimately
  have ?D∈M using Collect_in_M_4p[of ?d_fm _ _ _ _ λx y z w h. w⊥x ∨ x∈h]
                  P_in_M leq_in_M ‹D∈M› by simp
  note asm = ‹M_generic(G)› ‹dense(?D)› ‹?D⊆P› ‹?D∈M›
  obtain x where x∈G x∈?D using M_generic_denseD[OF asm]
    by force
  moreover from this and ‹M_generic(G)›
  have x∈D
    using M_generic_compatD[OF _ ‹p∈G›, of x]
      refl_leq compatI[of _ p x] by force
  ultimately
  show ?thesis by auto
qed
```

19.9 Auxiliary results for Lemma IV.2.40(a)

```
lemma IV240a_mem_Collect:
  assumes
    π∈M τ∈M
  shows
    {q∈P. ∃σ. ∃r. r∈P ∧ ⟨σ,r⟩ ∈ τ ∧ q≼r ∧ forces_eq(q,π,σ)}∈M
proof -
  let ?rel_pred= λM x a1 a2 a3 a4. ∃σ[M]. ∃r[M]. ∃σr[M].
           r∈a1 ∧ pair(M,σ,r,σr) ∧ σr∈a4 ∧ is_leq(M,a2,x,r) ∧ is_forces_eq'(M,a1,a2,x,a3,σ)
  let ?φ=Exists(Exists(Exists(And(Member(1,4),And(pair_fm(2,1,0),
         And(Member(0,7),And(leq_fm(5,3,1),forces_eq_fm(4,5,3,6,2)))))))
  have σ∈M ∧ r∈M if ⟨σ, r⟩ ∈ τ  for σ r
    using that ‹τ∈M› pair_in_M_iff transitivity[of ⟨σ,r⟩ τ] by simp
  then
   have ?rel_pred(##M,q,P,leq,π,τ) ⟷ (∃σ. ∃r. r∈P ∧ ⟨σ,r⟩ ∈ τ ∧ q≼r ∧
forces_eq(q,π,σ))
```

if $q \in M$ **for** q
unfolding *forces_eq_def* **using** *assms that P_in_M leq_in_M leq_abs forces_eq'_abs pair_in_M_iff*
by *auto*
moreover
have $(M, [q,P,leq,\pi,\tau] \models ?\varphi) \longleftrightarrow ?rel\_pred(\#\#M,q,P,leq,\pi,\tau)$ **if** $q \in M$ **for** q
using *assms that sats_forces_eq'_fm sats_leq_fm P_in_M leq_in_M* **by** *simp*
moreover
have $?\varphi \in formula$ **by** *simp*
moreover
have $arity(?\varphi)=5$
unfolding *leq_fm_def pair_fm_def upair_fm_def*
using *arity_forces_eq_fm* **by** (*simp add:nat_simp_union Un_commute*)
ultimately
show *?thesis*
unfolding *forces_eq_def* **using** *P_in_M leq_in_M assms*
Collect_in_M_4p[*of ?φ _ _ _ _ _*
$\lambda q\ a1\ a2\ a3\ a4.\ \exists \sigma.\ \exists r.\ r \in a1 \wedge \langle \sigma,r \rangle \in \tau \wedge q \preceq r \wedge forces\_eq'(a1,a2,q,a3,\sigma)$]
by *simp*
qed

lemma *IV240a_mem*:
assumes
$M\_generic(G)$ $p \in G$ $\pi \in M$ $\tau \in M$ $forces\_mem(p,\pi,\tau)$
$\bigwedge q\ \sigma.\ q \in P \Longrightarrow q \in G \Longrightarrow \sigma \in domain(\tau) \Longrightarrow forces\_eq(q,\pi,\sigma) \Longrightarrow$
$val(P,G,\pi) = val(P,G,\sigma)$
shows
$val(P,G,\pi) \in val(P,G,\tau)$
proof (*intro elem_of_valI*)
let $?D=\{q \in P.\ \exists \sigma.\ \exists r.\ r \in P \wedge \langle \sigma,r \rangle \in \tau \wedge q \preceq r \wedge forces\_eq(q,\pi,\sigma)\}$
from ‹$M\_generic(G)$› ‹$p \in G$›
have $p \in P$ **by** *blast*
moreover
note ‹$\pi \in M$› ‹$\tau \in M$›
ultimately
have $?D \in M$ **using** *IV240a_mem_Collect* **by** *simp*
moreover from *assms* ‹$p \in P$›
have $dense\_below(?D,p)$
using *forces_mem_iff_dense_below* **by** *simp*
moreover
note ‹$M\_generic(G)$› ‹$p \in G$›
ultimately
obtain q **where** $q \in G$ $q \in ?D$ **using** *generic_inter_dense_below* **by** *blast*
then
obtain σ r **where** $r \in P$ $\langle \sigma,r \rangle \in \tau$ $q \preceq r$ $forces\_eq(q,\pi,\sigma)$ **by** *blast*
moreover from *this* **and** ‹$q \in G$› *assms*
have $r \in G$ $val(P,G,\pi) = val(P,G,\sigma)$ **by** *blast+*
ultimately

show $\exists\ \sigma.\ \exists p{\in}P.\ p \in G \wedge \langle\sigma,\ p\rangle \in \tau \wedge val(P,G,\ \sigma) = val(P,G,\ \pi)$ **by** *auto*
qed

lemma *refl_forces_eq*:$p{\in}P \Longrightarrow forces\_eq(p,x,x)$
using *def_forces_eq* **by** *simp*

lemma *forces_memI*: $\langle\sigma,r\rangle{\in}\tau \Longrightarrow p{\in}P \Longrightarrow r{\in}P \Longrightarrow p{\preceq}r \Longrightarrow forces\_mem(p,\sigma,\tau)$
using *refl_forces_eq*[*of* _ σ] *leq_transD refl_leq*
by (*blast intro*:*forces_mem_iff_dense_below*[*THEN iffD2*])

lemma *IV240a_eq_1st_incl*:
assumes
$M\_generic(G)$ $p{\in}G$ $forces\_eq(p,\tau,\vartheta)$
and
$IH{:}\bigwedge q\ \sigma.\ q{\in}P \Longrightarrow q{\in}G \Longrightarrow \sigma{\in}domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(forces\_mem(q,\sigma,\tau) \longrightarrow val(P,G,\sigma) \in val(P,G,\tau)) \wedge$
$(forces\_mem(q,\sigma,\vartheta) \longrightarrow val(P,G,\sigma) \in val(P,G,\vartheta))$

shows
$val(P,G,\tau) \subseteq val(P,G,\vartheta)$
proof
fix x
assume $x{\in}val(P,G,\tau)$
then
obtain σ r **where** $\langle\sigma,r\rangle{\in}\tau$ $r{\in}G$ $val(P,G,\sigma){=}x$ **by** *blast*
moreover from *this* **and** ‹$p{\in}G$› ‹$M\_generic(G)$›
obtain q **where** $q{\in}G$ $q{\preceq}p$ $q{\preceq}r$ **by** *force*
moreover from *this* **and** ‹$p{\in}G$› ‹$M\_generic(G)$›
have $q{\in}P$ $p{\in}P$ **by** *blast+*
moreover from *calculation* **and** ‹$M\_generic(G)$›
have $forces\_mem(q,\sigma,\tau)$
using *forces_memI* **by** *blast*
moreover
note ‹$forces\_eq(p,\tau,\vartheta)$›
ultimately
have $forces\_mem(q,\sigma,\vartheta)$
using *def_forces_eq* **by** *blast*
with ‹$q{\in}P$› ‹$q{\in}G$› IH[*of* q σ] ‹$\langle\sigma,r\rangle{\in}\tau$› ‹$val(P,G,\sigma) = x$›
show $x{\in}val(P,G,\vartheta)$ **by** (*blast*)
qed

lemma *IV240a_eq_2nd_incl*:
assumes
$M\_generic(G)$ $p{\in}G$ $forces\_eq(p,\tau,\vartheta)$
and

IH:$\bigwedge q\ \sigma.\ q{\in}P \Longrightarrow q{\in}G \Longrightarrow \sigma{\in}domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(forces\_mem(q,\sigma,\tau) \longrightarrow val(P,G,\sigma) \in val(P,G,\tau)) \wedge$
$(forces\_mem(q,\sigma,\vartheta) \longrightarrow val(P,G,\sigma) \in val(P,G,\vartheta))$
shows
$val(P,G,\vartheta) \subseteq val(P,G,\tau)$
proof
fix x
assume $x{\in}val(P,G,\vartheta)$
then
obtain $\sigma\ r$ **where** $\langle\sigma,r\rangle{\in}\vartheta\ r{\in}G\ val(P,G,\sigma){=}x$ **by** *blast*
moreover from *this* **and** ‹$p{\in}G$› ‹$M\_generic(G)$›
obtain q **where** $q{\in}G\ q{\preceq}p\ q{\preceq}r$ **by** *force*
moreover from *this* **and** ‹$p{\in}G$› ‹$M\_generic(G)$›
have $q{\in}P\ p{\in}P$ **by** *blast+*
moreover from *calculation* **and** ‹$M\_generic(G)$›
have $forces\_mem(q,\sigma,\vartheta)$
using *forces_memI* **by** *blast*
moreover
note ‹$forces\_eq(p,\tau,\vartheta)$›
ultimately
have $forces\_mem(q,\sigma,\tau)$
using *def_forces_eq* **by** *blast*
with ‹$q{\in}P$› ‹$q{\in}G$› $IH[of\ q\ \sigma]$ ‹$\langle\sigma,r\rangle{\in}\vartheta$› ‹$val(P,G,\sigma) = x$›
show $x{\in}val(P,G,\tau)$ **by** (*blast*)
qed

lemma *IV240a_eq*:
assumes
$M\_generic(G)\ p{\in}G\ forces\_eq(p,\tau,\vartheta)$
and
IH:$\bigwedge q\ \sigma.\ q{\in}P \Longrightarrow q{\in}G \Longrightarrow \sigma{\in}domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(forces\_mem(q,\sigma,\tau) \longrightarrow val(P,G,\sigma) \in val(P,G,\tau)) \wedge$
$(forces\_mem(q,\sigma,\vartheta) \longrightarrow val(P,G,\sigma) \in val(P,G,\vartheta))$
shows
$val(P,G,\tau) = val(P,G,\vartheta)$
using *IV240a_eq_1st_incl*[*OF assms*] *IV240a_eq_2nd_incl*[*OF assms*] *IH* **by** *blast*

19.10 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 -
  {
    fix ft p τ ϑ
    have Transset(eclose({τ,ϑ})) (is Transset(?e))
      using Transset_eclose by simp
    have τ ∈ ?e ϑ ∈ ?e
      using arg_into_eclose by simp_all
    moreover
    assume ft ∈ 2 p ∈ P
    ultimately
    have ⟨ft,τ,ϑ,p⟩∈ 2×?e×?e×P (is ?a∈2×?e×?e×P) by simp
    then
    have Q(ftype(?a), name1(?a), name2(?a), cond_of(?a))
     using core_induction_aux[of ?e P Q ?a,OF ‹Transset(?e)› assms(1,2) ‹?a∈_›]

      by (clarify) (blast)
    then have Q(ft,τ,ϑ,p) by (simp add:components_simp)
  }
  then show ?thesis using assms by simp
qed

lemma forces_induction_with_conds:
  assumes
    ⋀τ ϑ p. p ∈ P ⟹ ⟦⋀q σ. ⟦q∈P ; σ∈domain(ϑ)⟧ ⟹ Q(q,τ,σ)⟧ ⟹ R(p,τ,ϑ)
    ⋀τ ϑ p. p ∈ P ⟹ ⟦⋀q σ. ⟦q∈P ; σ∈domain(τ) ∪ domain(ϑ)⟧ ⟹ R(q,σ,τ)
∧ R(q,σ,ϑ)⟧ ⟹ Q(p,τ,ϑ)
    p ∈ P
  shows
    Q(p,τ,ϑ) ∧ R(p,τ,ϑ)
proof -
  let ?Q=λft τ ϑ p. (ft = 0 ⟶ Q(p,τ,ϑ)) ∧ (ft = 1 ⟶ R(p,τ,ϑ))
  from assms(1)
  have ⋀τ ϑ p. p ∈ P ⟹ ⟦⋀q σ. ⟦q∈P ; σ∈domain(ϑ)⟧ ⟹ ?Q(0,τ,σ,q)⟧ ⟹
?Q(1,τ,ϑ,p)
    by simp
  moreover from assms(2)
  have ⋀τ ϑ p. p ∈ P ⟹ ⟦⋀q σ. ⟦q∈P ; σ∈domain(τ) ∪ domain(ϑ)⟧ ⟹
?Q(1,σ,τ,q) ∧ ?Q(1,σ,ϑ,q)⟧ ⟹ ?Q(0,τ,ϑ,p)
    by simp
  moreover
  note ‹p∈P›
  ultimately
  have ?Q(ft,τ,ϑ,p) if ft∈2 for ft
    by (rule core_induction[OF _ _ that, of ?Q])
  then
  show ?thesis by auto
qed

lemma forces_induction:
```

```
  assumes
    ⋀τ ϑ. ⟦⋀σ. σ∈domain(ϑ) ⟹ Q(τ,σ)⟧ ⟹ R(τ,ϑ)
    ⋀τ ϑ. ⟦⋀σ. σ∈domain(τ) ∪ domain(ϑ) ⟹ R(σ,τ) ∧ R(σ,ϑ)⟧ ⟹ Q(τ,ϑ)
  shows
    Q(τ,ϑ) ∧ R(τ,ϑ)
proof (intro forces_induction_with_conds[OF _ _ one_in_P ])
  fix τ ϑ p
  assume q ∈ P ⟹ σ ∈ domain(ϑ) ⟹ Q(τ, σ) for q σ
  with assms(1)
  show R(τ,ϑ)
    using one_in_P by simp
next
  fix τ ϑ p
   assume q ∈ P ⟹ σ ∈ domain(τ) ∪ domain(ϑ) ⟹ R(σ,τ) ∧ R(σ,ϑ) for q σ
   with assms(2)
   show Q(τ,ϑ)
     using one_in_P by simp
qed
```

19.11 Lemma IV.2.40(a), in full

```
lemma IV240a:
  assumes
    M_generic(G)
  shows
    (τ∈M ⟶ ϑ∈M ⟶ (∀p∈G. forces_eq(p,τ,ϑ) ⟶ val(P,G,τ) = val(P,G,ϑ)))
∧
   (τ∈M ⟶ ϑ∈M ⟶ (∀p∈G. forces_mem(p,τ,ϑ) ⟶ val(P,G,τ) ∈ val(P,G,ϑ)))
    (is ?Q(τ,ϑ) ∧ ?R(τ,ϑ))
proof (intro forces_induction[of ?Q ?R] impI)
  fix τ ϑ
  assume τ∈M ϑ∈M  σ∈domain(ϑ) ⟹ ?Q(τ,σ) for σ
  moreover from this
  have σ∈domain(ϑ) ⟹ forces_eq(q, τ, σ) ⟹ val(P,G, τ) = val(P,G, σ)
    if q∈P q∈G for q σ
    using that domain_closed[of ϑ] transitivity by auto
  moreover
  note assms
  ultimately
  show ∀p∈G. forces_mem(p,τ,ϑ) ⟶ val(P,G,τ) ∈ val(P,G,ϑ)
    using IV240a_mem domain_closed transitivity by (simp)
next
  fix τ ϑ
  assume τ∈M ϑ∈M σ ∈ domain(τ) ∪ domain(ϑ) ⟹ ?R(σ,τ) ∧ ?R(σ,ϑ) for σ
  moreover from this
  have IH':σ ∈ domain(τ) ∪ domain(ϑ) ⟹ q∈G ⟹
          (forces_mem(q, σ, τ) ⟶ val(P,G, σ) ∈ val(P,G, τ)) ∧
          (forces_mem(q, σ, ϑ) ⟶ val(P,G, σ) ∈ val(P,G, ϑ)) for q σ
    by (auto intro:  transitivity[OF _ domain_closed[simplified]])
```

ultimately
show $\forall p{\in}G.\ forces\_eq(p,\tau,\vartheta) \longrightarrow val(P,G,\tau) = val(P,G,\vartheta)$
using *IV240a_eq[OF assms(1) _ _ IH']* **by** *(simp)*
qed

19.12 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
$IH{:}\bigwedge\sigma.\ \sigma{\in}domain(\tau) \Longrightarrow val(P,G,\pi) = val(P,G,\sigma) \Longrightarrow$
$\exists p{\in}G.\ forces\_eq(p,\pi,\sigma)$
shows
$\exists p{\in}G.\ forces\_mem(p,\pi,\tau)$
proof -
from ‹$val(P,G,\pi){\in}val(P,G,\tau)$›
obtain σ r **where** $r{\in}G\ \langle\sigma,r\rangle{\in}\tau\ val(P,G,\pi) = val(P,G,\sigma)$ **by** *auto*
moreover from *this* **and** *IH*
obtain p' **where** $p'{\in}G\ forces\_eq(p',\pi,\sigma)$ **by** *blast*
moreover
note ‹$M\_generic(G)$›
ultimately
obtain p **where** $p{\preceq}r\ p{\in}G\ forces\_eq(p,\pi,\sigma)$
using *M_generic_compatD strengthening_eq[of p']* **by** *blast*
moreover
note ‹$M\_generic(G)$›
moreover from *calculation*
have $forces\_eq(q,\pi,\sigma)$ **if** $q{\in}P\ q{\preceq}p$ **for** q
using *that strengthening_eq* **by** *blast*
moreover
note ‹$\langle\sigma,r\rangle{\in}\tau$› ‹$r{\in}G$›
ultimately
have $r{\in}P \wedge \langle\sigma,r\rangle \in \tau \wedge q{\preceq}r \wedge forces\_eq(q,\pi,\sigma)$ **if** $q{\in}P\ q{\preceq}p$ **for** q
using *that leq_transD[of _ p r]* **by** *blast*
then
have $dense\_below(\{q{\in}P.\ \exists s\ r.\ r{\in}P \wedge \langle s,r\rangle \in \tau \wedge q{\preceq}r \wedge forces\_eq(q,\pi,s)\},p)$
using *refl_leq* **by** *blast*
moreover
note ‹$M\_generic(G)$› ‹$p{\in}G$›
moreover from *calculation*
have $forces\_mem(p,\pi,\tau)$
using *forces_mem_iff_dense_below* **by** *blast*
ultimately
show *?thesis* **by** *blast*
qed

end

lemma *Collect_forces_eq_in_M*:
assumes $\tau \in M$ $\vartheta \in M$
shows $\{p{\in}P.\ forces\_eq(p,\tau,\vartheta)\} \in M$
using *assms Collect_in_M_4p*[*of forces_eq_fm*(*1*,*2*,*0*,*3*,*4*) *P leq* τ ϑ
$\lambda A\ x\ p\ l\ t1\ t2.\ is\_forces\_eq(x,t1,t2)$
$\lambda\ x\ p\ l\ t1\ t2.\ forces\_eq(x,t1,t2)$ *P*]
arity_forces_eq_fm P_in_M leq_in_M sats_forces_eq_fm forces_eq_abs forces_eq_fm_type

by (*simp add*: *nat_union_abs1 Un_commute*)

lemma *IV240b_eq_Collects*:
assumes $\tau \in M$ $\vartheta \in M$
shows $\{p{\in}P.\ \exists\sigma{\in}domain(\tau) \cup domain(\vartheta).\ forces\_mem(p,\sigma,\tau) \wedge forces\_nmem(p,\sigma,\vartheta)\}{\in}M$
and
$\{p{\in}P.\ \exists\sigma{\in}domain(\tau) \cup domain(\vartheta).\ forces\_nmem(p,\sigma,\tau) \wedge forces\_mem(p,\sigma,\vartheta)\}{\in}M$
proof -
let *?rel_pred*=$\lambda M\ x\ a1\ a2\ a3\ a4.$
$\exists\sigma[M].\ \exists u[M].\ \exists da3[M].\ \exists da4[M].\ is\_domain(M,a3,da3) \wedge is\_domain(M,a4,da4)$
$\wedge$
$union(M,da3,da4,u) \wedge \sigma{\in}u \wedge is\_forces\_mem'(M,a1,a2,x,\sigma,a3) \wedge$
$is\_forces\_nmem'(M,a1,a2,x,\sigma,a4)$
let *?φ=Exists*(*Exists*(*Exists*(*Exists*(*And*(*domain_fm*(*7*,*1*),*And*(*domain_fm*(*8*,*0*),
And(*union_fm*(*1*,*0*,*2*),*And*(*Member*(*3*,*2*),*And*(*forces_mem_fm*(*5*,*6*,*4*,*3*,*7*),
forces_nmem_fm(*5*,*6*,*4*,*3*,*8*))))))))))
have *1*:$\sigma{\in}M$ **if** $\langle\sigma,y\rangle{\in}\delta$ $\delta{\in}M$ **for** σ δ y
using *that pair_in_M_iff transitivity*[*of* $\langle\sigma,y\rangle$ δ] **by** *simp*
have *abs1*:$?rel\_pred(\#\#M,p,P,leq,\tau,\vartheta) \longleftrightarrow$
$(\exists\sigma{\in}domain(\tau) \cup domain(\vartheta).\ forces\_mem'(P,leq,p,\sigma,\tau) \wedge forces\_nmem'(P,leq,p,\sigma,\vartheta))$

if $p{\in}M$ **for** p
unfolding *forces_mem_def forces_nmem_def*
using *assms that forces_mem'_abs forces_nmem'_abs P_in_M leq_in_M*
domain_closed Un_closed
by (*auto simp add*:*1*[*of* _ _ τ] *1*[*of* _ _ ϑ])
have *abs2*:$?rel\_pred(\#\#M,p,P,leq,\vartheta,\tau) \longleftrightarrow (\exists\sigma{\in}domain(\tau) \cup domain(\vartheta).$
$forces\_nmem'(P,leq,p,\sigma,\tau) \wedge forces\_mem'(P,leq,p,\sigma,\vartheta))$ **if** $p{\in}M$ **for** p
unfolding *forces_mem_def forces_nmem_def*
using *assms that forces_mem'_abs forces_nmem'_abs P_in_M leq_in_M*
domain_closed Un_closed
by (*auto simp add*:*1*[*of* _ _ τ] *1*[*of* _ _ ϑ])
have *fsats1*:$(M,[p,P,leq,\tau,\vartheta] \models ?\varphi) \longleftrightarrow ?rel\_pred(\#\#M,p,P,leq,\tau,\vartheta)$ **if** $p{\in}M$
for p
using *that assms sats_forces_mem'_fm sats_forces_nmem'_fm P_in_M leq_in_M*
domain_closed Un_closed **by** *simp*
have *fsats2*:$(M,[p,P,leq,\vartheta,\tau] \models ?\varphi) \longleftrightarrow ?rel\_pred(\#\#M,p,P,leq,\vartheta,\tau)$ **if** $p{\in}M$
for p
using *that assms sats_forces_mem'_fm sats_forces_nmem'_fm P_in_M leq_in_M*
domain_closed Un_closed **by** *simp*
have *fty*:$?\varphi{\in}formula$ **by** *simp*

have *farit*:*arity*(*?φ*)=*5*
unfolding *forces_nmem_fm_def domain_fm_def pair_fm_def upair_fm_def union_fm_def*
using *arity_forces_mem_fm* **by** (*simp add*:*nat_simp_union Un_commute*)
show
{*p* ∈ *P* . ∃σ∈*domain*(τ) ∪ *domain*(ϑ). *forces_mem*(*p*, σ, τ) ∧ *forces_nmem*(*p*, σ, ϑ)} ∈ *M*
and {*p* ∈ *P* . ∃σ∈*domain*(τ) ∪ *domain*(ϑ). *forces_nmem*(*p*, σ, τ) ∧ *forces_mem*(*p*, σ, ϑ)} ∈ *M*
unfolding *forces_mem_def*
using *abs1 fty fsats1 farit P_in_M leq_in_M assms forces_nmem*
Collect_in_M_4p[*of ?φ _ _ _ _ _*
λ*x p l a1 a2*. (∃σ∈*domain*(*a1*) ∪ *domain*(*a2*). *forces_mem'*(*p*,*l*,*x*,σ,*a1*) ∧
forces_nmem'(*p*,*l*,*x*,σ,*a2*))]
using *abs2 fty fsats2 farit P_in_M leq_in_M assms forces_nmem domain_closed Un_closed*
Collect_in_M_4p[*of ?φ P leq* ϑ τ *?rel_pred*
λ*x p l a2 a1*. (∃σ∈*domain*(*a1*) ∪ *domain*(*a2*). *forces_nmem'*(*p*,*l*,*x*,σ,*a1*) ∧
forces_mem'(*p*,*l*,*x*,σ,*a2*)) *P*]
by *simp_all*
qed

lemma *IV240b_eq*:
assumes
M_generic(*G*) *val*(*P*,*G*,τ) = *val*(*P*,*G*,ϑ) τ∈*M* ϑ∈*M*
and
IH:⋀σ. σ∈*domain*(τ)∪*domain*(ϑ) ⟹
(*val*(*P*,*G*,σ)∈*val*(*P*,*G*,τ) ⟶ (∃ *q*∈*G*. *forces_mem*(*q*,σ,τ))) ∧
(*val*(*P*,*G*,σ)∈*val*(*P*,*G*,ϑ) ⟶ (∃ *q*∈*G*. *forces_mem*(*q*,σ,ϑ)))

shows
∃ *p*∈*G*. *forces_eq*(*p*,τ,ϑ)
proof -
let *?D1*={*p*∈*P*. *forces_eq*(*p*,τ,ϑ)}
let *?D2*={*p*∈*P*. ∃σ∈*domain*(τ) ∪ *domain*(ϑ). *forces_mem*(*p*,σ,τ) ∧ *forces_nmem*(*p*,σ,ϑ)}
let *?D3*={*p*∈*P*. ∃σ∈*domain*(τ) ∪ *domain*(ϑ). *forces_nmem*(*p*,σ,τ) ∧ *forces_mem*(*p*,σ,ϑ)}
let *?D*=*?D1* ∪ *?D2* ∪ *?D3*
note *assms*
moreover from *this*
have *domain*(τ) ∪ *domain*(ϑ)∈*M* (**is** *?B*∈*M*) **using** *domain_closed Un_closed* **by** *auto*
moreover from *calculation*
have *?D2*∈*M* **and** *?D3*∈*M* **using** *IV240b_eq_Collects* **by** *simp_all*
ultimately
have *?D*∈*M* **using** *Collect_forces_eq_in_M Un_closed* **by** *auto*
moreover
have *dense*(*?D*)
proof

fix *p*
assume $p \in P$
have $\exists\, d \in P.\ (forces\_eq(d,\ \tau,\ \vartheta) \vee$
$(\exists\, \sigma \in domain(\tau) \cup domain(\vartheta).\ forces\_mem(d,\ \sigma,\ \tau) \wedge forces\_nmem(d,\ \sigma,$
$\vartheta)) \vee$
$(\exists\, \sigma \in domain(\tau) \cup domain(\vartheta).\ forces\_nmem(d,\ \sigma,\ \tau) \wedge forces\_mem(d,\ \sigma,$
$\vartheta))) \wedge$
$d \preceq p$
proof $(cases\ forces\_eq(p,\ \tau,\ \vartheta))$
case *True*
with $\langle p \in P \rangle$
show *?thesis* **using** *refl_leq* **by** *blast*
next
case *False*
moreover note $\langle p \in P \rangle$
moreover from *calculation*
obtain $\sigma\ q$ **where** $\sigma \in domain(\tau) \cup domain(\vartheta)\ q \in P\ q \preceq p$
$(forces\_mem(q,\ \sigma,\ \tau) \wedge \neg\ forces\_mem(q,\ \sigma,\ \vartheta)) \vee$
$(\neg\ forces\_mem(q,\ \sigma,\ \tau) \wedge forces\_mem(q,\ \sigma,\ \vartheta))$
using *def_forces_eq* **by** *blast*
moreover from *this*
obtain *r* **where** $r \preceq q\ r \in P$
$(forces\_mem(r,\ \sigma,\ \tau) \wedge forces\_nmem(r,\ \sigma,\ \vartheta)) \vee$
$(forces\_nmem(r,\ \sigma,\ \tau) \wedge forces\_mem(r,\ \sigma,\ \vartheta))$
using *not_forces_nmem strengthening_mem* **by** *blast*
ultimately
show *?thesis* **using** *leq_transD* **by** *blast*
qed
then
show $\exists\, d \in ?D1 \cup ?D2 \cup ?D3.\ d \preceq p$ **by** *blast*
qed
moreover
have $?D \subseteq P$
by *auto*
moreover
note $\langle M\_generic(G) \rangle$
ultimately
obtain *p* **where** $p \in G\ p \in ?D$
unfolding *M_generic_def* **by** *blast*
then
consider
$(1)\ forces\_eq(p,\tau,\vartheta)\ |$
$(2)\ \exists\, \sigma \in domain(\tau) \cup domain(\vartheta).\ forces\_mem(p,\sigma,\tau) \wedge forces\_nmem(p,\sigma,\vartheta)\ |$
$(3)\ \exists\, \sigma \in domain(\tau) \cup domain(\vartheta).\ forces\_nmem(p,\sigma,\tau) \wedge forces\_mem(p,\sigma,\vartheta)$
by *blast*
then
show *?thesis*
proof (*cases*)
case *1*

```
  with ‹p∈G›
  show ?thesis by blast
next
  case 2
  then
 obtain σ where σ∈domain(τ) ∪ domain(ϑ) forces_mem(p,σ,τ) forces_nmem(p,σ,ϑ)

    by blast
  moreover from this and ‹p∈G› and assms
  have val(P,G,σ)∈val(P,G,τ)
    using IV240a[of G σ τ] transitivity[OF _ domain_closed[simplified]] by blast
  moreover note IH ‹val(P,G,τ) = _›
  ultimately
  obtain q where q∈G forces_mem(q, σ, ϑ) by auto
  moreover from this and ‹p∈G› ‹M_generic(G)›
  obtain r where r∈P r≼p r≼q
    by blast
  moreover
  note ‹M_generic(G)›
  ultimately
  have forces_mem(r, σ, ϑ)
    using strengthening_mem by blast
  with ‹r≼p› ‹forces_nmem(p,σ,ϑ)› ‹r∈P›
  have False
    unfolding forces_nmem_def by blast
  then
  show ?thesis by simp
next
  case 3
  then
 obtain σ where σ∈domain(τ) ∪ domain(ϑ) forces_mem(p,σ,ϑ) forces_nmem(p,σ,τ)

    by blast
  moreover from this and ‹p∈G› and assms
  have val(P,G,σ)∈val(P,G,ϑ)
    using IV240a[of G σ ϑ] transitivity[OF _ domain_closed[simplified]] by blast
  moreover note IH ‹val(P,G,τ) = _›
  ultimately
  obtain q where q∈G forces_mem(q, σ, τ) by auto
  moreover from this and ‹p∈G› ‹M_generic(G)›
  obtain r where r∈P r≼p r≼q
    by blast
  moreover
  note ‹M_generic(G)›
  ultimately
  have forces_mem(r, σ, τ)
    using strengthening_mem by blast
  with ‹r≼p› ‹forces_nmem(p,σ,τ)› ‹r∈P›
  have False
```

    unfolding *forces_nmem_def* **by** *blast*
  then
  show *?thesis* **by** *simp*
 qed
qed

lemma *IV240b*:
 assumes
  M_generic(*G*)
 shows
  ($\tau \in M \longrightarrow \vartheta \in M \longrightarrow val(P,G,\tau) = val(P,G,\vartheta) \longrightarrow (\exists p \in G.\ forces\_eq(p,\tau,\vartheta))$) $\wedge$
  ($\tau \in M \longrightarrow \vartheta \in M \longrightarrow val(P,G,\tau) \in val(P,G,\vartheta) \longrightarrow (\exists p \in G.\ forces\_mem(p,\tau,\vartheta))$)

  (**is** *?Q*(τ,ϑ) $\wedge$ *?R*(τ,ϑ))
proof (*intro forces_induction*)
 fix τ ϑ *p*
 assume $\sigma \in domain(\vartheta) \Longrightarrow$ *?Q*(τ, σ) **for** σ
 with *assms*
 show *?R*(τ, ϑ)
  using *IV240b_mem domain_closed transitivity* **by** (*simp*)
next
 fix τ ϑ *p*
 assume $\sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow$ *?R*(σ,τ) $\wedge$ *?R*(σ,ϑ) **for** σ
 moreover from *this*
 have *IH'*:$\tau \in M \Longrightarrow \vartheta \in M \Longrightarrow \sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow$
    ($val(P,G, \sigma) \in val(P,G, \tau) \longrightarrow (\exists q \in G.\ forces\_mem(q, \sigma, \tau))$) $\wedge$
    ($val(P,G, \sigma) \in val(P,G, \vartheta) \longrightarrow (\exists q \in G.\ forces\_mem(q, \sigma, \vartheta))$) **for** σ
  by (*blast intro*:*left_in_M*)
 ultimately
 show *?Q*(τ,ϑ)
  using *IV240b_eq*[*OF assms*(*1*)] **by** (*auto*)
qed

lemma *map_val_in_MG*:
 assumes
  $env \in list(M)$
 shows
  $map(val(P,G),env) \in list(M[G])$
 unfolding *GenExt_def* **using** *assms map_type2* **by** *simp*

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
  ($\exists p \in G.\ p \Vdash Member(n,m)\ env$) $\longleftrightarrow$ $M[G],\ map(val(P,G),env) \models Member(n,m)$
 using *assms IV240a*[*OF assms*(*2*), *of nth*(*n*,*env*) *nth*(*m*,*env*)]

IV240b[*OF assms*(*2*), *of nth*(*n*,*env*) *nth*(*m*,*env*)]
P_in_M leq_in_M one_in_M
Forces_Member[*of _ nth*(*n*,*env*) *nth*(*m*,*env*) *env n m*] *map_val_in_MG*
by (*auto*)

lemma *truth_lemma_eq*:
assumes
env∈*list*(*M*) *M_generic*(*G*)
n∈*nat m*∈*nat n*<*length*(*env*) *m*<*length*(*env*)
shows
(∃ *p*∈*G*. *p* ⊩ *Equal*(*n*,*m*) *env*) ⟷ *M*[*G*], *map*(*val*(*P*,*G*),*env*) ⊨ *Equal*(*n*,*m*)
using *assms IV240a*(*1*)[*OF assms*(*2*), *of nth*(*n*,*env*) *nth*(*m*,*env*)]
IV240b(*1*)[*OF assms*(*2*), *of nth*(*n*,*env*) *nth*(*m*,*env*)]
P_in_M leq_in_M one_in_M
Forces_Equal[*of _ nth*(*n*,*env*) *nth*(*m*,*env*) *env n m*] *map_val_in_MG*
by (*auto*)

lemma *arities_at_aux*:
assumes
n ∈ *nat m* ∈ *nat env* ∈ *list*(*M*) *succ*(*n*) ∪ *succ*(*m*) ≤ *length*(*env*)
shows
n < *length*(*env*) *m* < *length*(*env*)
using *assms succ_leE*[*OF Un_leD1*, *of n succ*(*m*) *length*(*env*)]
succ_leE[*OF Un_leD2*, *of succ*(*n*) *m length*(*env*)] **by** *auto*

19.13 The Strenghtening Lemma

lemma *strengthening_lemma*:
assumes
p∈*P* *φ*∈*formula r*∈*P r*≼*p*
shows
⋀*env*. *env*∈*list*(*M*) ⟹ *arity*(*φ*)≤*length*(*env*) ⟹ *p* ⊩ *φ env* ⟹ *r* ⊩ *φ env*
using *assms*(*2*)
proof (*induct*)
case (*Member n m*)
then
have *n*<*length*(*env*) *m*<*length*(*env*)
using *arities_at_aux* **by** *simp_all*
moreover
assume *env*∈*list*(*M*)
moreover
note *assms Member*
ultimately
show *?case*
using *Forces_Member*[*of _ nth*(*n*,*env*) *nth*(*m*,*env*) *env n m*]
strengthening_mem[*of p r nth*(*n*,*env*) *nth*(*m*,*env*)] **by** *simp*
next
case (*Equal n m*)
then

have *n*<*length*(*env*) *m*<*length*(*env*)
using *arities_at_aux* **by** *simp_all*
moreover
assume *env*∈*list*(*M*)
moreover
note *assms Equal*
ultimately
show *?case*
using *Forces_Equal*[*of* _ *nth*(*n*,*env*) *nth*(*m*,*env*) *env n m*]
strengthening_eq[*of p r nth*(*n*,*env*) *nth*(*m*,*env*)] **by** *simp*
next
case (*Nand* φ ψ)
with *assms*
show *?case*
using *Forces_Nand transitivity*[*OF* _ *P_in_M*] *pair_in_M_iff*
transitivity[*OF* _ *leq_in_M*] *leq_transD* **by** *auto*
next
case (*Forall* φ)
with *assms*
have *p* ⊩ φ ([*x*] @ *env*) **if** *x*∈*M* **for** *x*
using *that Forces_Forall* **by** *simp*
with *Forall*
have *r* ⊩ φ ([*x*] @ *env*) **if** *x*∈*M* **for** *x*
using *that pred_le2* **by** (*simp*)
with *assms Forall*
show *?case*
using *Forces_Forall* **by** *simp*
qed

19.14 The Density Lemma

lemma *arity_Nand_le*:
assumes $\varphi \in$ *formula* $\psi \in$ *formula arity*(*Nand*(φ, ψ)) ≤ *length*(*env*) *env*∈*list*(*A*)
shows *arity*(φ) ≤ *length*(*env*) *arity*(ψ) ≤ *length*(*env*)
using *assms*
by (*rule_tac Un_leD1*, *rule_tac* [5] *Un_leD2*, *auto*)

lemma *dense_below_imp_forces*:
assumes
p∈*P* φ∈*formula*
shows
⋀*env*. *env*∈*list*(*M*) ⟹ *arity*(φ)≤*length*(*env*) ⟹
dense_below({*q*∈*P*. (*q* ⊩ φ *env*)},*p*) ⟹ (*p* ⊩ φ *env*)
using *assms*(*2*)
proof (*induct*)
case (*Member n m*)
then
have *n*<*length*(*env*) *m*<*length*(*env*)
using *arities_at_aux* **by** *simp_all*

```
  moreover
  assume env∈list(M)
  moreover
  note assms Member
  ultimately
  show ?case
    using Forces_Member[of _ nth(n,env) nth(m,env) env n m]
      density_mem[of p nth(n,env) nth(m,env)] by simp
next
  case (Equal n m)
  then
  have n<length(env) m<length(env)
    using arities_at_aux by simp_all
  moreover
  assume env∈list(M)
  moreover
  note assms Equal
  ultimately
  show ?case
    using Forces_Equal[of _ nth(n,env) nth(m,env) env n m]
      density_eq[of p nth(n,env) nth(m,env)] by simp
next
case (Nand φ ψ)
  {
    fix q
    assume q∈M q∈P q≼ p q ⊩ φ env
    moreover
    note Nand
    moreover from calculation
    obtain d where d∈P d ⊩ Nand(φ, ψ) env d≼ q
      using dense_belowI by auto
    moreover from calculation
    have ¬(d⊩ ψ env) if d ⊩ φ env
      using that Forces_Nand refl_leq transitivity[OF _ P_in_M, of d] by auto
    moreover
    note arity_Nand_le[of φ ψ]
    moreover from calculation
    have d ⊩ φ env
       using strengthening_lemma[of q φ d env] Un_leD1 by auto
    ultimately
    have ¬ (q ⊩ ψ env)
      using strengthening_lemma[of q ψ d env] by auto
  }
  with ‹p∈P›
  show ?case
    using Forces_Nand[symmetric, OF _ Nand(5,1,3)] by blast
next
  case (Forall φ)
  have dense_below({q∈P. q ⊩ φ ([a]@env)},p) if a∈M for a
```

```
  proof
    fix r
    assume r∈P r≼p
    with ‹dense_below(_,p)›
    obtain q where q∈P q≼r q ⊩ Forall(φ) env
      by blast
    moreover
    note Forall ‹a∈M›
    moreover from calculation
    have q ⊩ φ ([a]@env)
      using Forces_Forall by simp
    ultimately
    show ∃ d ∈ {q∈P. q ⊩ φ ([a]@env)}. d ∈ P ∧ d≼r
      by auto
  qed
  moreover
  note Forall(2)[of Cons(_,env)] Forall(1,3-5)
  ultimately
  have p ⊩ φ ([a]@env) if a∈M for a
    using that pred_le2 by simp
  with assms Forall
  show ?case using Forces_Forall by simp
qed

lemma density_lemma:
  assumes
    p∈P φ∈formula env∈list(M) arity(φ)≤length(env)
  shows
    p ⊩ φ env   ⟷   dense_below({q∈P. (q ⊩ φ env)},p)
proof
  assume dense_below({q∈P. (q ⊩ φ env)},p)
  with assms
  show  (p ⊩ φ env)
    using dense_below_imp_forces by simp
next
  assume p ⊩ φ env
  with assms
  show dense_below({q∈P. q ⊩ φ env},p)
    using strengthening_lemma refl_leq by auto
qed
```

19.15 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
  assume $p \Vdash And(\varphi, \psi)\ env$
  with *assms*
  have $dense\_below(\{r \in P\ .\ (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)\}, p)$
    using *Forces_And_iff_dense_below* **by** *simp*
  then
  have $dense\_below(\{r \in P\ .\ (r \Vdash \varphi\ env)\}, p)\ dense\_below(\{r \in P\ .\ (r \Vdash \psi\ env)\}, p)$
    by *blast+*
  with *assms*
  show $(p \Vdash \varphi\ env) \wedge (p \Vdash \psi\ env)$
    using *density_lemma[symmetric]* **by** *simp*
next
  assume $(p \Vdash \varphi\ env) \wedge (p \Vdash \psi\ env)$
  have $dense\_below(\{r \in P\ .\ (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)\}, p)$
  proof (*intro dense_belowI bexI conjI*, *assumption*)
    fix q
    assume $q \in P\ q \preceq p$
    with *assms* ‹$(p \Vdash \varphi\ env) \wedge (p \Vdash \psi\ env)$›
    show $q \in \{r \in P\ .\ (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)\}\ q \preceq q$
      using *strengthening_lemma refl_leq* **by** *auto*
  qed
  with *assms*
  show $p \Vdash And(\varphi,\psi)\ env$
    using *Forces_And_iff_dense_below* **by** *simp*
qed

lemma *Forces_Nand_alt*:
  assumes
    $p \in P\ env \in list(M)\ \varphi \in formula\ \psi \in formula$
    $arity(\varphi) \le length(env)\ arity(\psi) \le length(env)$
  shows
    $(p \Vdash Nand(\varphi,\psi)\ env) \longleftrightarrow (p \Vdash Neg(And(\varphi,\psi))\ env)$
  using *assms Forces_Nand Forces_And Forces_Neg* **by** *auto*

lemma *truth_lemma_Neg*:
  assumes
    $\varphi \in formula\ M\_generic(G)\ env \in list(M)\ arity(\varphi) \le length(env)$ **and**
    IH: $(\exists p \in G.\ p \Vdash \varphi\ env) \longleftrightarrow M[G],\ map(val(P,G),env) \models \varphi$
  shows
    $(\exists p \in G.\ p \Vdash Neg(\varphi)\ env) \longleftrightarrow M[G],\ map(val(P,G),env) \models Neg(\varphi)$
proof (*intro iffI*, *elim bexE*, *rule ccontr*)

  fix p
  assume $p \in G\ p \Vdash Neg(\varphi)\ env\ \neg(M[G],map(val(P,G),env) \models Neg(\varphi))$
  moreover
  note *assms*
  moreover from *calculation*
  have $M[G],\ map(val(P,G),env) \models \varphi$

```
    using map_val_in_MG by simp
  with IH
  obtain r where r ⊩ φ env r∈G by blast
  moreover from this and ‹M_generic(G)› ‹p∈G›
  obtain q where q≼p q≼r q∈G
    by blast
  moreover from calculation
  have q ⊩ φ env
    using strengthening_lemma[where φ=φ] by blast
  ultimately
  show False
    using Forces_Neg[where φ=φ] transitivity[OF _ P_in_M] by blast
next
  assume M[G], map(val(P,G),env) ⊨ Neg(φ)
  with assms
  have ¬ (M[G], map(val(P,G),env) ⊨ φ)
    using map_val_in_MG by simp
  let ?D={p∈P. (p ⊩ φ env) ∨ (p ⊩ Neg(φ) env)}
  have separation(##M,λp. (p ⊩ φ env))
     using separation_ax arity_forces assms P_in_M leq_in_M one_in_M arity_forces_le
    by simp
  moreover
  have separation(##M,λp. (p ⊩ Neg(φ) env))
     using separation_ax arity_forces assms P_in_M leq_in_M one_in_M arity_forces_le
    by simp
  ultimately
  have separation(##M,λp. (p ⊩ φ env) ∨ (p ⊩ Neg(φ) env))
    using separation_disj by simp
  then
  have ?D ∈ M
    using separation_closed P_in_M by simp
  moreover
  have ?D ⊆ P by auto
  moreover
  have dense(?D)
  proof
    fix q
    assume q∈P
    show ∃ d∈{p ∈ P . (p ⊩ φ env) ∨ (p ⊩ Neg(φ) env)}. d≼ q
    proof (cases q ⊩ Neg(φ) env)
      case True
      with ‹q∈P›
      show ?thesis using refl_leq by blast
    next
      case False
      with ‹q∈P› and assms
      show ?thesis using Forces_Neg by auto
    qed
  qed
```

```
  moreover
  note ‹M_generic(G)›
  ultimately
  obtain p where p∈G (p ⊩ φ env) ∨ (p ⊩ Neg(φ) env)
    by blast
  then
  consider (1) p ⊩ φ env | (2) p ⊩ Neg(φ) env by blast
  then
  show ∃p∈G. (p ⊩ Neg(φ) env)
  proof (cases)
    case 1
    with ‹¬ (M[G],map(val(P,G),env) ⊨ φ)› ‹p∈G› IH
    show ?thesis
      by blast
  next
    case 2
    with ‹p∈G›
    show ?thesis by blast
  qed
qed

lemma truth_lemma_And:
  assumes
    env∈list(M) φ∈formula ψ∈formula
    arity(φ)≤length(env) arity(ψ) ≤ length(env) M_generic(G)
    and
    IH: (∃p∈G. p ⊩ φ env) ⟷ M[G], map(val(P,G),env) ⊨ φ
        (∃p∈G. p ⊩ ψ env) ⟷ M[G], map(val(P,G),env) ⊨ ψ
  shows
    (∃p∈G. (p ⊩ And(φ,ψ) env)) ⟷ M[G] , map(val(P,G),env) ⊨ And(φ,ψ)
  using assms map_val_in_MG Forces_And[OF M_genericD assms(1-5)]
proof (intro iffI, elim bexE)
  fix p
  assume p∈G p ⊩ And(φ,ψ) env
  with assms
  show M[G], map(val(P,G),env) ⊨ And(φ,ψ)
    using Forces_And[OF M_genericD, of _ _ _ φ ψ] map_val_in_MG by auto
next
  assume M[G], map(val(P,G),env) ⊨ And(φ,ψ)
  moreover
  note assms
  moreover from calculation
  obtain q r where q ⊩ φ env r ⊩ ψ env q∈G r∈G
    using map_val_in_MG Forces_And[OF M_genericD assms(1-5)] by auto
  moreover from calculation
  obtain p where p≼q p≼r p∈G
    by blast
  moreover from calculation
  have (p ⊩ φ env) ∧ (p ⊩ ψ env)
```

```
    using strengthening_lemma by (blast)
  ultimately
  show ∃p∈G. (p ⊩ And(φ,ψ) env)
    using Forces_And[OF M_genericD assms(1-5)] by auto
qed

definition
  ren_truth_lemma :: i⇒i where
  ren_truth_lemma(φ) ≡
    Exists(Exists(Exists(Exists(Exists(
   And(Equal(0,5),And(Equal(1,8),And(Equal(2,9),And(Equal(3,10),And(Equal(4,6),
    iterates(λp. incr_bv(p)`5 , 6, φ))))))))))

lemma ren_truth_lemma_type[TC] :
  φ∈formula ⟹ ren_truth_lemma(φ) ∈formula
  unfolding ren_truth_lemma_def
  by simp

lemma arity_ren_truth :
  assumes φ∈formula
  shows arity(ren_truth_lemma(φ)) ≤ 6 ∪ succ(arity(φ))
proof -
  consider (lt) 5 <arity(φ) | (ge) ¬ 5 < arity(φ)
    by auto
  then
  show ?thesis
  proof cases
    case lt
    consider (a) 5<arity(φ)#+5  | (b) arity(φ)#+5 ≤ 5
      using not_lt_iff_le ‹φ∈_› by force
    then
    show ?thesis
    proof cases
      case a
      with ‹φ∈_› lt
     have 5 < succ(arity(φ)) 5<arity(φ)#+2  5<arity(φ)#+3  5<arity(φ)#+4
        using succ_ltI by auto
       with ‹φ∈_›
     have c:arity(iterates(λp. incr_bv(p)`5,5,φ)) = 5#+arity(φ) (is arity(?φ') =
_)
        using arity_incr_bv_lemma lt a
        by simp
      with ‹φ∈_›
      have arity(incr_bv(?φ')`5) = 6#+arity(φ)
        using arity_incr_bv_lemma[of ?φ' 5] a by auto
      with ‹φ∈_›
      show ?thesis
        unfolding ren_truth_lemma_def
      using pred_Un_distrib nat_union_abs1 Un_assoc[symmetric] a c nat_union_abs2
```

```
        by simp
    next
      case b
      with ‹φ∈_› lt
      have 5 < succ(arity(φ)) 5<arity(φ)#+2  5<arity(φ)#+3  5<arity(φ)#+4
5<arity(φ)#+5
        using succ_ltI by auto
      with ‹φ∈_›
      have arity(iterates(λp. incr_bv(p)`5,6,φ)) = 6#+arity(φ) (is arity(?φ') =
_)
        using arity_incr_bv_lemma lt
        by simp
      with ‹φ∈_›
      show ?thesis
        unfolding ren_truth_lemma_def
        using pred_Un_distrib nat_union_abs1 Un_assoc[symmetric]  nat_union_abs2
        by simp
    qed
  next
    case ge
    with ‹φ∈_›
    have arity(φ) ≤ 5 pred^5(arity(φ)) ≤ 5
      using not_lt_iff_le le_trans[OF le_pred]
      by auto
    with ‹φ∈_›
    have arity(iterates(λp. incr_bv(p)`5,6,φ)) = arity(φ) arity(φ)≤6 pred^5(arity(φ))
≤ 6
     using arity_incr_bv_lemma ge le_trans[OF ‹arity(φ)≤5›] le_trans[OF ‹pred^5(arity(φ))≤5›]
      by auto
    with ‹arity(φ) ≤ 5› ‹φ∈_› ‹pred^5(_) ≤ 5›
    show ?thesis
      unfolding ren_truth_lemma_def
      using  pred_Un_distrib nat_union_abs1 Un_assoc[symmetric] nat_union_abs2
      by simp
  qed
qed

lemma sats_ren_truth_lemma:
  [q,b,d,a1,a2,a3] @ env ∈ list(M) ⟹ φ ∈ formula ⟹
   (M, [q,b,d,a1,a2,a3] @ env ⊨ ren_truth_lemma(φ) ) ⟷
   (M, [q,a1,a2,a3,b] @ env ⊨ φ)
  unfolding ren_truth_lemma_def
  by (insert sats_incr_bv_iff  [of _ _ M _ [q,a1,a2,a3,b]], simp)

lemma truth_lemma' :
  assumes
    φ∈formula env∈list(M) arity(φ) ≤ succ(length(env))
  shows
    separation(##M,λd. ∃ b∈M. ∀ q∈P. q≼d ⟶ ¬(q ⊩ φ ([b]@env)))
```

proof -
  let *?rel_pred*=λM *x a1 a2 a3*. $\exists$ *b*$\in$*M*. $\forall$ *q*$\in$*M*. *q*$\in$*a1* $\wedge$ *is_leq*(##*M*,*a2*,*q*,*x*) $\longrightarrow$
        $\neg$(*M*, [*q*,*a1*,*a2*,*a3*,*b*] @ *env* $\models$ *forces*(φ))
  let *?*ψ=*Exists*(*Forall*(*Implies*(*And*(*Member*(*0*,*3*),*leq_fm*(*4*,*0*,*2*)),
      Neg(*ren_truth_lemma*(*forces*(φ))))))
  have *q*$\in$*M* **if** *q*$\in$*P* **for** *q* **using** *that transitivity*[*OF* _ *P_in_M*] **by** *simp*
  then
  have *1*:$\forall$ *q*$\in$*M*. *q*$\in$*P* $\wedge$ *R*(*q*) $\longrightarrow$ *Q*(*q*) $\Longrightarrow$ ($\forall$ *q*$\in$*P*. *R*(*q*) $\longrightarrow$ *Q*(*q*)) **for** *R Q*
    by *auto*
  then
  have ⟦*b* $\in$ *M*; $\forall$ *q*$\in$*M*. *q* $\in$ *P* $\wedge$ *q* $\preceq$ *d* $\longrightarrow$ $\neg$(*q* ⊩ φ ([*b*]@*env*))⟧ $\Longrightarrow$
    $\exists$ *c*$\in$*M*. $\forall$ *q*$\in$*P*. *q* $\preceq$ *d* $\longrightarrow$ $\neg$(*q* ⊩ φ ([*c*]@*env*)) **for** *b d*
    by (*rule bexI*,*simp_all*)
  then
  have *?rel_pred*(*M*,*d*,*P*,*leq*,*one*) $\longleftrightarrow$ ($\exists$ *b*$\in$*M*. $\forall$ *q*$\in$*P*. *q*$\preceq$*d* $\longrightarrow$ $\neg$(*q* ⊩ φ ([*b*]@*env*)))
if *d*$\in$*M* **for** *d*
    using *that leq_abs leq_in_M P_in_M one_in_M assms*
    by *auto*
  moreover
  have *?*$\psi$$\in$*formula* **using** *assms* **by** *simp*
  moreover
  have (*M*, [*d*,*P*,*leq*,*one*]@*env* $\models$ *?*ψ) $\longleftrightarrow$ *?rel_pred*(*M*,*d*,*P*,*leq*,*one*) **if** *d*$\in$*M* **for**
d
    using *assms that P_in_M leq_in_M one_in_M sats_leq_fm sats_ren_truth_lemma*
    by *simp*
  moreover
  have *arity*(*?*ψ) $\leq$ *4*#+*length*(*env*)
  proof -
    have *eq*:*arity*(*leq_fm*(*4*, *0*, *2*)) = *5*
      using *arity_leq_fm succ_Un_distrib nat_simp_union*
      by *simp*
    with ⟨$\varphi$$\in$_⟩
    have *arity*(*?*ψ) = *3* $\cup$ (*pred*^*2*(*arity*(*ren_truth_lemma*(*forces*(φ)))))
      using *nat_union_abs1 pred_Un_distrib* **by** *simp*
    moreover
    have ... $\leq$ *3* $\cup$ (*pred*(*pred*(*6* $\cup$ *succ*(*arity*(*forces*(φ)))))) (**is** _ $\leq$ *?r*)
      using ⟨$\varphi$$\in$_⟩ *Un_le_compat*[*OF le_refl*[*of 3*]]
            le_imp_subset arity_ren_truth[*of forces*(φ)]
            pred_mono
      by *auto*
    finally
    have *arity*(*?*ψ) $\leq$ *?r* **by** *simp*
    have *i*:*?r* $\leq$ *4* $\cup$ *pred*(*arity*(*forces*(φ)))
     using *pred_Un_distrib pred_succ_eq* ⟨$\varphi$$\in$_⟩ *Un_assoc*[*symmetric*] *nat_union_abs1*
by *simp*
    have *h*:*4* $\cup$ *pred*(*arity*(*forces*(φ))) $\leq$ *4* $\cup$ (*4*#+*length*(*env*))
      using ⟨*env*$\in$_⟩ *add_commute* ⟨$\varphi$$\in$_⟩
            Un_le_compat[*of 4 4*,*OF* _ *pred_mono*[*OF* _ *arity_forces_le*[*OF* _ _
⟨*arity*(φ)$\leq$_⟩]]]

‹env∈_› **by** *auto*
with ‹φ∈_› ‹env∈_›
show *?thesis*
using *le_trans*[*OF* ‹*arity*(*?ψ*) ≤ *?r*› *le_trans*[*OF i h*]] *nat_simp_union* **by** *simp*
qed
ultimately
show *?thesis* **using** *assms P_in_M leq_in_M one_in_M*
separation_ax[*of ?ψ* [*P*,*leq*,*one*]@*env*]
separation_cong[*of ##M λy.* (*M*, [*y*,*P*,*leq*,*one*]@*env* ⊨*?ψ*)]
by *simp*
qed

lemma *truth_lemma*:
assumes
φ∈*formula M_generic*(*G*)
shows
⋀*env. env*∈*list*(*M*) ⟹ *arity*(*φ*)≤*length*(*env*) ⟹
(∃*p*∈*G. p* ⊩ *φ env*) ⟷ *M*[*G*], *map*(*val*(*P*,*G*),*env*) ⊨ *φ*
using *assms*(*1*)
proof (*induct*)
case (*Member x y*)
then
show *?case*
using *assms truth_lemma_mem*[*OF* ‹*env*∈*list*(*M*)› *assms*(*2*) ‹*x*∈*nat*› ‹*y*∈*nat*›]
arities_at_aux **by** *simp*
next
case (*Equal x y*)
then
show *?case*
using *assms truth_lemma_eq*[*OF* ‹*env*∈*list*(*M*)› *assms*(*2*) ‹*x*∈*nat*› ‹*y*∈*nat*›]
arities_at_aux **by** *simp*
next
case (*Nand φ ψ*)
moreover
note ‹*M_generic*(*G*)›
ultimately
show *?case*
using *truth_lemma_And truth_lemma_Neg Forces_Nand_alt*
M_genericD map_val_in_MG arity_Nand_le[*of φ ψ*] **by** *auto*
next
case (*Forall φ*)
with ‹*M_generic*(*G*)›
show *?case*
proof (*intro iffI*)
assume ∃*p*∈*G.* (*p* ⊩ *Forall*(*φ*) *env*)
with ‹*M_generic*(*G*)›
obtain *p* **where** *p*∈*G p*∈*M p*∈*P p* ⊩ *Forall*(*φ*) *env*

```
      using transitivity[OF _ P_in_M] by auto
    with ‹env∈list(M)› ‹φ∈formula›
    have p ⊩ φ ([x]@env) if x∈M for x
      using that Forces_Forall by simp
    with ‹p∈G› ‹φ∈formula› ‹env∈_› ‹arity(Forall(φ)) ≤ length(env)›
      Forall(2)[of Cons(_,env)]
    show M[G], map(val(P,G),env) ⊨  Forall(φ)
      using pred_le2 map_val_in_MG
      by (auto iff:GenExtD)
  next
    assume M[G], map(val(P,G),env) ⊨ Forall(φ)
    let ?D1={d∈P. (d ⊩ Forall(φ) env)}
    let ?D2={d∈P. ∃b∈M. ∀q∈P. q≼d ⟶ ¬(q ⊩ φ ([b]@env))}
    define D where D ≡ ?D1 ∪ ?D2
    have arφ:arity(φ)≤succ(length(env))
        using assms ‹arity(Forall(φ)) ≤ length(env)› ‹φ∈formula› ‹env∈list(M)›
pred_le2
      by simp
    then
    have arity(Forall(φ)) ≤ length(env)
      using pred_le ‹φ∈formula› ‹env∈list(M)› by simp
    then
    have ?D1∈M using Collect_forces arφ ‹φ∈formula› ‹env∈list(M)› by simp
    moreover
   have ?D2∈M using ‹env∈list(M)› ‹φ∈formula›  truth_lemma' separation_closed
arφ
                      P_in_M
      by simp
    ultimately
    have D∈M unfolding D_def using Un_closed by simp
    moreover
    have D ⊆ P unfolding D_def by auto
    moreover
    have dense(D)
    proof
      fix p
      assume p∈P
      show ∃d∈D. d≼ p
      proof (cases p ⊩ Forall(φ) env)
        case True
        with ‹p∈P›
        show ?thesis unfolding D_def using refl_leq by blast
      next
        case False
        with Forall ‹p∈P›
        obtain b where b∈M ¬(p ⊩ φ ([b]@env))
          using Forces_Forall by blast
        moreover from this ‹p∈P› Forall
        have ¬dense_below({q∈P. q ⊩ φ ([b]@env)},p)
```

```
        using density_lemma pred_le2  by auto
      moreover from this
      obtain d where d≼p ∀q∈P. q≼d ⟶ ¬(q ⊩ φ ([b] @ env))
        d∈P by blast
      ultimately
      show ?thesis unfolding D_def by auto
    qed
  qed
  moreover
  note ‹M_generic(G)›
  ultimately
  obtain d where d ∈ D  d ∈ G by blast
  then
  consider (1) d∈?D1 | (2) d∈?D2 unfolding D_def by blast
  then
  show ∃p∈G. (p ⊩ Forall(φ) env)
  proof (cases)
    case 1
    with ‹d∈G›
    show ?thesis by blast
  next
    case 2
    then
    obtain b where b∈M ∀q∈P. q≼d ⟶¬(q ⊩ φ ([b] @ env))
      by blast
    moreover from this(1) and  ‹M[G], _ ⊨  Forall(φ)› and
      Forall(2)[of Cons(b,env)] Forall(1,3-4) ‹M_generic(G)›
    obtain p where p∈G p∈P p ⊩ φ ([b] @ env)
      using pred_le2 using map_val_in_MG by (auto iff:GenExtD)
    moreover
    note ‹d∈G› ‹M_generic(G)›
    ultimately
    obtain q where q∈G q∈P q≼d q≼p by blast
    moreover from this and  ‹p ⊩ φ ([b] @ env)›
      Forall ‹b∈M› ‹p∈P›
    have q ⊩ φ ([b] @ env)
      using pred_le2 strengthening_lemma by simp
    moreover
    note ‹∀q∈P. q≼d ⟶¬(q ⊩ φ ([b] @ env))›
    ultimately
    show ?thesis by simp
  qed
 qed
qed
```

19.16 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 (intro iffI allI impI, elim conjE)
  fix G
  assume (p ⊩ φ env) M_generic(G) p ∈ G
  with assms
  show M[G], map(val(P,G),env) ⊨ φ
    using truth_lemma by blast
next
  assume 1: ∀ G.(M_generic(G)∧ p∈G)⟶ M[G] , map(val(P,G),env) ⊨ φ
  {
    fix r
    assume 2: r∈P r≼p
    then
    obtain G where r∈G M_generic(G)
      using generic_filter_existence by auto
    moreover from calculation 2 ‹p∈P›
    have p∈G
      unfolding M_generic_def using filter_leqD by simp
    moreover note 1
    ultimately
    have M[G], map(val(P,G),env) ⊨ φ
      by simp
    with assms ‹M_generic(G)›
    obtain s where s∈G (s ⊩ φ env)
      using truth_lemma by blast
    moreover from this and  ‹M_generic(G)› ‹r∈G›
    obtain q where q∈G q≼s q≼r
      by blast
    moreover from calculation ‹s∈G› ‹M_generic(G)›
    have s∈P q∈P
      unfolding M_generic_def filter_def by auto
    moreover
    note assms
    ultimately
    have ∃ q∈P. q≼r ∧ (q ⊩ φ env)
      using strengthening_lemma by blast
  }
  then
  have dense_below({q∈P. (q ⊩ φ env)},p)
    unfolding dense_below_def by blast
  with assms
  show (p ⊩ φ env)
    using density_lemma by blast
qed

lemmas definability = forces_type
```

end

end

20 Auxiliary renamings for Separation

theory *Separation_Rename*
 imports *Interface Renaming*
begin

lemmas *apply_fun* = *apply_iff* [*THEN iffD1*]

lemma *nth_concat* : [*p*,*t*] ∈ *list*(*A*) ⟹ *env*∈ *list*(*A*) ⟹ *nth*(*1* #+ *length*(*env*),[*p*]@ *env* @ [*t*]) = *t*
 by(*auto simp add*:*nth_append*)

lemma *nth_concat2* : *env*∈ *list*(*A*) ⟹ *nth*(*length*(*env*),*env* @ [*p*,*t*]) = *p*
 by(*auto simp add*:*nth_append*)

lemma *nth_concat3* : *env*∈ *list*(*A*) ⟹ *u* = *nth*(*succ*(*length*(*env*)), *env* @ [*pi*, *u*])
 by(*auto simp add*:*nth_append*)

definition
 sep_var :: *i* ⇒ *i* **where**
 sep_var(*n*) ≡ {⟨*0*,*1*⟩,⟨*1*,*3*⟩,⟨*2*,*4*⟩,⟨*3*,*5*⟩,⟨*4*,*0*⟩,⟨*5*#+*n*,*6*⟩,⟨*6*#+*n*,*2*⟩}

definition
 sep_env :: *i* ⇒ *i* **where**
 sep_env(*n*) ≡ λ *i* ∈ (*5*#+*n*)-*5* . *i*#+*2*

definition *weak* :: [*i*, *i*] ⇒ *i* **where**
 weak(*n*,*m*) ≡ {*i*#+*m* . *i* ∈ *n*}

lemma *weakD* :
 assumes *n* ∈ *nat k*∈*nat x* ∈ *weak*(*n*,*k*)
 shows ∃ *i* ∈ *n* . *x* = *i*#+*k*
 using *assms* **unfolding** *weak_def* **by** *blast*

lemma *weak_equal* :
 assumes *n*∈*nat m*∈*nat*
 shows *weak*(*n*,*m*) = (*m*#+*n*) - *m*
proof -
 have *weak*(*n*,*m*)⊆(*m*#+*n*)-*m*
 proof(*intro subsetI*)
  fix *x*
  assume *x*∈*weak*(*n*,*m*)
  with *assms*
  obtain *i* **where**
   i∈*n x*=*i*#+*m*

```
    using weakD by blast
  then
  have m≤i#+m i<n
    using add_le_self2[of m i] ‹m∈nat› ‹n∈nat› ltI[OF ‹i∈n›] by simp_all
  then
  have ¬i#+m<m
    using not_lt_iff_le in_n_in_nat[OF ‹n∈nat› ‹i∈n›] ‹m∈nat› by simp
  with ‹x=i#+m›
  have x∉m
    using ltI ‹m∈nat› by auto
  moreover
  from assms ‹x=i#+m› ‹i<n›
  have x<m#+n
    using add_lt_mono1[OF ‹i<n› ‹n∈nat›] by simp
  ultimately
  show x∈(m#+n)-m
    using ltD DiffI by simp
qed
moreover
have (m#+n)-m⊆weak(n,m)
proof (intro subsetI)
  fix x
  assume x∈(m#+n)-m
  then
  have x∈m#+n x∉m
    using DiffD1[of x n#+m m] DiffD2[of x n#+m m] by simp_all
  then
  have x<m#+n x∈nat
    using ltI in_n_in_nat[OF add_type[of m n]] by simp_all
  then
  obtain i where
    m#+n = succ(x#+i)
    using less_iff_succ_add[OF ‹x∈nat›,of m#+n] add_type by auto
  then
  have x#+i<m#+n using succ_le_iff by simp
  with ‹x∉m›
  have ¬x<m using ltD by blast
  with ‹m∈nat› ‹x∈nat›
  have m≤x using not_lt_iff_le by simp
  with ‹x<m#+n› ‹n∈nat›
  have x#-m<m#+n#-m
    using diff_mono[OF ‹x∈nat› _ ‹m∈nat›] by simp
  have m#+n#-m = n using diff_cancel2 ‹m∈nat› ‹n∈nat› by simp
  with ‹x#-m<m#+n#-m› ‹x∈nat›
  have x#-m ∈ n x=x#-m#+m
    using ltD add_diff_inverse2[OF ‹m≤x›] by simp_all
  then
  show x∈weak(n,m)
    unfolding weak_def by auto
```

```
  qed
  ultimately
  show ?thesis by auto
qed

lemma weak_zero:
  shows weak(0,n) = 0
  unfolding weak_def by simp

lemma weakening_diff :
  assumes n ∈ nat
  shows weak(n,7) - weak(n,5) ⊆ {5#+n, 6#+n}
  unfolding weak_def using assms
proof(auto)
  {
    fix i
    assume i∈n succ(succ(natify(i)))≠n ∀ w∈n. succ(succ(natify(i))) ≠ natify(w)
    then
    have i<n
      using ltI ‹n∈nat› by simp
    from ‹n∈nat› ‹i∈n› ‹succ(succ(natify(i)))≠n›
    have i∈nat succ(succ(i))≠n using in_n_in_nat by simp_all
    from ‹i<n›
    have succ(i)≤n using succ_leI by simp
    with ‹n∈nat›
    consider (a) succ(i) = n | (b) succ(i) < n
      using leD by auto
    then have succ(i) = n
    proof cases
      case a
      then show ?thesis .
    next
      case b
      then
      have succ(succ(i))≤n using succ_leI by simp
      with ‹n∈nat›
      consider (a) succ(succ(i)) = n | (b) succ(succ(i)) < n
        using leD by auto
      then have succ(i) = n
      proof cases
        case a
        with ‹succ(succ(i))≠n› show ?thesis by blast
      next
        case b
        then
        have succ(succ(i))∈n using ltD by simp
        with ‹i∈nat›
        have succ(succ(natify(i))) ≠ natify(succ(succ(i)))
          using  ‹∀ w∈n. succ(succ(natify(i))) ≠ natify(w)› by auto
```

```
      then
      have False using ‹i∈nat› by auto
      then show ?thesis by blast
    qed
    then show ?thesis .
  qed
  with ‹i∈nat› have succ(natify(i)) = n by simp
}
then
show n ∈ nat ⟹
  succ(succ(natify(y))) ≠ n ⟹
  ∀x∈n. succ(succ(natify(y))) ≠ natify(x) ⟹
  y ∈ n ⟹ succ(natify(y)) = n for y
  by blast
qed

lemma in_add_del :
  assumes x∈j#+n n∈nat j∈nat
  shows x < j ∨ x ∈ weak(n,j)
proof (cases x<j)
  case True
  then show ?thesis ..
next
  case False
  have x∈nat j#+n∈nat
    using in_n_in_nat[OF _ ‹x∈j#+n›] assms by simp_all
  then
  have j ≤ x x < j#+n
    using not_lt_iff_le False ‹j∈nat› ‹n∈nat› ltI[OF ‹x∈j#+n›] by auto
  then
  have x#-j < (j #+ n) #- j x = j #+ (x #-j)
    using diff_mono ‹x∈nat› ‹j#+n∈nat› ‹j∈nat› ‹n∈nat›
      add_diff_inverse[OF ‹j≤x›] by simp_all
  then
  have x#-j < n x = (x #-j ) #+ j
    using diff_add_inverse ‹n∈nat› add_commute by simp_all
  then
  have x#-j ∈n using ltD by simp
  then
  have x ∈ weak(n,j)
    unfolding weak_def
    using ‹x= (x#-j) #+j› RepFunI[OF ‹x#-j∈n›] add_commute by force
  then show ?thesis ..
qed


lemma sep_env_action:
  assumes
    [t,p,u,P,leq,o,pi] ∈ list(M)
```

env ∈ *list*(*M*)
shows ∀ *i* . *i* ∈ *weak*(*length*(*env*),*5*) ⟶
nth(*sep_env*(*length*(*env*))‘*i*,[*t*,*p*,*u*,*P*,*leq*,*o*,*pi*]@*env*) = *nth*(*i*,[*p*,*P*,*leq*,*o*,*t*] @ *env* @ [*pi*,*u*])
proof -
from *assms*
have *A*: *5*#+*length*(*env*)∈*nat* [*p*, *P*, *leq*, *o*, *t*] ∈*list*(*M*)
by *simp_all*
let *?f*=*sep_env*(*length*(*env*))
have *EQ*: *weak*(*length*(*env*),*5*) = *5*#+*length*(*env*) - *5*
using *weak_equal length_type*[*OF* ‹*env*∈*list*(*M*)›] **by** *simp*
let *?tgt*=[*t*,*p*,*u*,*P*,*leq*,*o*,*pi*]@*env*
let *?src*=[*p*,*P*,*leq*,*o*,*t*] @ *env* @ [*pi*,*u*]
have *nth*(*?f*‘*i*,[*t*,*p*,*u*,*P*,*leq*,*o*,*pi*]@*env*) = *nth*(*i*,[*p*,*P*,*leq*,*o*,*t*] @ *env* @ [*pi*,*u*])
if *i* ∈ (*5*#+*length*(*env*)-*5*) **for** *i*
proof -
from *that*
have *2*: *i* ∈ *5*#+*length*(*env*) *i* ∉ *5* *i* ∈ *nat* *i*#-*5*∈*nat* *i*#+*2*∈*nat*
using *in_n_in_nat*[*OF* ‹*5*#+*length*(*env*)∈*nat*›] **by** *simp_all*
then
have *3*: ¬ *i* < *5* **using** *ltD* **by** *force*
then
have *5* ≤ *i* *2* ≤ *5*
using *not_lt_iff_le* ‹*i*∈*nat*› **by** *simp_all*
then have *2* ≤ *i* **using** *le_trans*[*OF* ‹*2*≤*5*›] **by** *simp*
from *A* ‹*i* ∈ *5*#+*length*(*env*)›
have *i* < *5*#+*length*(*env*) **using** *ltI* **by** *simp*
with ‹*i*∈*nat*› ‹*2*≤*i*› *A*
have *C*:*i*#+*2* < *7*#+*length*(*env*) **by** *simp*
with *that*
have *B*: *?f*‘*i* = *i*#+*2* **unfolding** *sep_env_def* **by** *simp*
from *3* *assms*(*1*) ‹*i*∈*nat*›
have ¬ *i*#+*2* < *7* **using** *not_lt_iff_le add_le_mono* **by** *simp*
from ‹*i* < *5*#+*length*(*env*)› *3* ‹*i*∈*nat*›
have *i*#-*5* < *5*#+*length*(*env*) #- *5*
using *diff_mono*[*of i 5*#+*length*(*env*) *5*,*OF* _ _ _ ‹*i* < *5*#+*length*(*env*)›]
not_lt_iff_le[*THEN iffD1*] **by** *force*
with *assms*(*2*)
have *i*#-*5* < *length*(*env*) **using** *diff_add_inverse length_type* **by** *simp*
have *nth*(*i*,*?src*) =*nth*(*i*#-*5*,*env*@[*pi*,*u*])
using *nth_append*[*OF* *A*(*2*) ‹*i*∈*nat*›] *3* **by** *simp*
also
have ... = *nth*(*i*#-*5*, *env*)
using *nth_append*[*OF* ‹*env* ∈*list*(*M*)› ‹*i*#-*5*∈*nat*›] ‹*i*#-*5* < *length*(*env*)› **by** *simp*
also
have ... = *nth*(*i*#+*2*, *?tgt*)
using *nth_append*[*OF* *assms*(*1*) ‹*i*#+*2*∈*nat*›] ‹¬ *i*#+*2* <*7*› **by** *simp*
ultimately

```
    have nth(i,?src) = nth(?f`i,?tgt)
      using B by simp
    then show ?thesis using that by simp
  qed
  then show ?thesis using EQ by force
qed

lemma sep_env_type :
  assumes n ∈ nat
  shows sep_env(n) : (5#+n)-5 → (7#+n)-7
proof -
  let ?h=sep_env(n)
  from ‹n∈nat›
  have (5#+n)#+2 = 7#+n 7#+n∈nat 5#+n∈nat by simp_all
  have
    D: sep_env(n)`x ∈ (7#+n)-7 if x ∈ (5#+n)-5 for x
  proof -
    from ‹x∈5#+n-5›
    have ?h`x = x#+2 x<5#+n x∈nat
      unfolding sep_env_def using ltI in_n_in_nat[OF ‹5#+n∈nat›] by simp_all
    then
    have x#+2 < 7#+n by simp
    then
    have x#+2 ∈ 7#+n using ltD by simp
    from ‹x∈5#+n-5›
    have x∉5 by simp
    then have ¬x<5 using ltD by blast
    then have 5≤x using not_lt_iff_le ‹x∈nat› by simp
    then have 7≤x#+2 using add_le_mono ‹x∈nat› by simp
    then have ¬x#+2<7 using not_lt_iff_le ‹x∈nat› by simp
    then have x#+2 ∉ 7 using ltI ‹x∈nat› by force
    with ‹x#+2 ∈ 7#+n› show ?thesis using ‹?h`x = x#+2› DiffI by simp
  qed
  then show ?thesis unfolding sep_env_def using lam_type by simp
qed

lemma sep_var_fin_type :
  assumes n ∈ nat
  shows sep_var(n) : 7#+n  -||> 7#+n
  unfolding sep_var_def
  using consI ltD emptyI by force

lemma sep_var_domain :
  assumes n ∈ nat
  shows domain(sep_var(n)) =  7#+n - weak(n,5)
proof -
  let ?A=weak(n,5)
  have A:domain(sep_var(n)) ⊆ (7#+n)
    unfolding sep_var_def
```

by(*auto simp add*: *le_natE*)
have *C*: *x*=*5*#+*n* ∨ *x*=*6*#+*n* ∨ *x* ≤ *4* if *x*∈*domain*(*sep_var*(*n*)) for *x*
using *that* unfolding *sep_var_def* by *auto*
have *D* : *x*<*n*#+*7* if *x*∈*7*#+*n* for *x*
using *that* ‹*n*∈*nat*› *ltI* by *simp*
have ¬ *5*#+*n* < *5*#+*n* using ‹*n*∈*nat*› *lt_irrefl*[*of* _ *False*] by *force*
have ¬ *6*#+*n* < *5*#+*n* using ‹*n*∈*nat*› by *force*
have *R*: *x* < *5*#+*n* if *x*∈*?A* for *x*
proof -
from *that*
obtain *i* where
i<*n* *x*=*5*#+*i*
unfolding *weak_def*
using *ltI* ‹*n*∈*nat*› *RepFun_iff* by *force*
with ‹*n*∈*nat*›
have *5*#+*i* < *5*#+*n* using *add_lt_mono2* by *simp*
with ‹*x*=*5*#+*i*›
show *x* < *5*#+*n* by *simp*
qed
then
have *1*:*x*∉*?A* if ¬*x* <*5*#+*n* for *x* using *that* by *blast*
have *5*#+*n* ∉ *?A* *6*#+*n*∉*?A*
proof -
show *5*#+*n* ∉ *?A* using *1* ‹¬*5*#+*n*<*5*#+*n*› by *blast*
with *1* show *6*#+*n* ∉ *?A* using ‹¬*6*#+*n*<*5*#+*n*› by *blast*
qed
then
have *E*:*x*∉*?A* if *x*∈*domain*(*sep_var*(*n*)) for *x*
unfolding *weak_def*
using *C* *that* by *force*
then
have *F*: *domain*(*sep_var*(*n*)) ⊆ *7*#+*n* - *?A* using *A* by *auto*
from *assms*
have *x*<*7* ∨ *x*∈*weak*(*n*,*7*) if *x*∈*7*#+*n* for *x*
using *in_add_del*[*OF* ‹*x*∈*7*#+*n*›] by *simp*
moreover
{
fix *x*
assume *asm*:*x*∈*7*#+*n* *x*∉*?A* *x*∈*weak*(*n*,*7*)
then
have *x*∈*domain*(*sep_var*(*n*))
proof -
from ‹*n*∈*nat*›
have *weak*(*n*,*7*)-*weak*(*n*,*5*)⊆{*n*#+*5*,*n*#+*6*}
using *weakening_diff* by *simp*
with ‹*x*∉*?A*› *asm*
have *x*∈{*n*#+*5*,*n*#+*6*} using *subsetD* *DiffI* by *blast*
then
show *?thesis* unfolding *sep_var_def* by *simp*

```
  qed
}
moreover
{
  fix x
  assume asm:x∈7#+n  x∉?A x<7
  then have x∈domain(sep_var(n))
  proof (cases 2 ≤ n)
    case True
    moreover
    have 0<n using  leD[OF ‹n∈nat› ‹2≤n›] lt_imp_0_lt by auto
    ultimately
    have x<5
      using ‹x<7› ‹x∉?A› ‹n∈nat› in_n_in_nat
      unfolding weak_def
      by (clarsimp simp add:not_lt_iff_le, auto simp add:lt_def)
    then
    show ?thesis unfolding sep_var_def
      by (clarsimp simp add:not_lt_iff_le, auto simp add:lt_def)
  next
    case False
    then
    show ?thesis
    proof (cases n=0)
      case True
      then show ?thesis
        unfolding sep_var_def using ltD asm ‹n∈nat› by auto
    next
      case False
      then
      have n < 2 using  ‹n∈nat› not_lt_iff_le ‹¬ 2 ≤ n›  by force
      then
      have ¬ n <1 using ‹n≠0› by simp
      then
      have n=1 using not_lt_iff_le ‹n<2› le_iff by auto
      then show ?thesis
        using ‹x∉?A›
        unfolding weak_def sep_var_def
        using ltD asm ‹n∈nat› by force
    qed
  qed
}
ultimately
have w∈domain(sep_var(n)) if w∈ 7#+n - ?A for w
  using that by blast
then
have 7#+n - ?A ⊆ domain(sep_var(n)) by blast
with F
show ?thesis by auto
```

qed

lemma *sep_var_type* :
  assumes *n* ∈ *nat*
  shows *sep_var*(*n*) : (*7#+n*)-*weak*(*n*,*5*) → *7#+n*
  using *FiniteFun_is_fun*[*OF sep_var_fin_type*[*OF* ‹*n*∈*nat*›]]
    sep_var_domain[*OF* ‹*n*∈*nat*›] **by** *simp*

lemma *sep_var_action* :
  assumes
    [*t*,*p*,*u*,*P*,*leq*,*o*,*pi*] ∈ *list*(*M*)
    env ∈ *list*(*M*)
  shows ∀ *i* . *i* ∈ (*7#+length*(*env*)) - *weak*(*length*(*env*),*5*) ⟶
    nth(*sep_var*(*length*(*env*))`*i*,[*t*,*p*,*u*,*P*,*leq*,*o*,*pi*]@*env*) = *nth*(*i*,[*p*,*P*,*leq*,*o*,*t*] @ *env* @ [*pi*,*u*])
  using *assms*
proof (*subst sep_var_domain*[*OF length_type*[*OF* ‹*env*∈*list*(*M*)›],*symmetric*],*auto*)
  fix *i y*
  assume ⟨*i*, *y*⟩ ∈ *sep_var*(*length*(*env*))
  with *assms*
  show *nth*(*sep_var*(*length*(*env*)) ` *i*,
          Cons(*t*, *Cons*(*p*, *Cons*(*u*, *Cons*(*P*, *Cons*(*leq*, *Cons*(*o*, *Cons*(*pi*, *env*)))))))) =
      nth(*i*, *Cons*(*p*, *Cons*(*P*, *Cons*(*leq*, *Cons*(*o*, *Cons*(*t*, *env* @ [*pi*, *u*]))))))
    using *apply_fun*[*OF sep_var_type*] *assms*
      unfolding *sep_var_def*
    using *nth_concat2*[*OF* ‹*env*∈*list*(*M*)›] *nth_concat3*[*OF* ‹*env*∈*list*(*M*)›,*symmetric*]
      by *force*
  qed

definition
  rensep :: *i* ⇒ *i* **where**
  rensep(*n*) ≡ *union_fun*(*sep_var*(*n*),*sep_env*(*n*),*7#+n*-*weak*(*n*,*5*),*weak*(*n*,*5*))

lemma *rensep_aux* :
  assumes *n*∈*nat*
  shows (*7#+n*-*weak*(*n*,*5*)) ∪ *weak*(*n*,*5*) = *7#+n 7#+n* ∪ (*7* #+ *n* - *7*) = *7#+n*
proof -
  from ‹*n*∈*nat*›
  have *weak*(*n*,*5*) = *n#+5-5*
    using *weak_equal* **by** *simp*
  with ‹*n*∈*nat*›
  show (*7#+n*-*weak*(*n*,*5*)) ∪ *weak*(*n*,*5*) = *7#+n 7#+n* ∪ (*7* #+ *n* - *7*) = *7#+n*
    using *Diff_partition le_imp_subset* **by** *auto*
qed

lemma *rensep_type* :

assumes *n∈nat*
shows *rensep(n) ∈ 7#+n → 7#+n*
proof -
from ‹*n∈nat*›
have *rensep(n) ∈ (7#+n-weak(n,5)) ∪ weak(n,5) → 7#+n ∪ (7#+n - 7)*
unfolding *rensep_def*
using *union_fun_type sep_var_type* ‹*n∈nat*› *sep_env_type weak_equal*
by *force*
then
show *?thesis* **using** *rensep_aux* ‹*n∈nat*› **by** *auto*
qed

lemma *rensep_action* :
assumes *[t,p,u,P,leq,o,pi] @ env ∈ list(M)*
shows ∀ *i . i < 7#+length(env) ⟶ nth(rensep(length(env))‘i,[t,p,u,P,leq,o,pi]@env) = nth(i,[p,P,leq,o,t] @ env @ [pi,u])*
proof -
let *?tgt=[t,p,u,P,leq,o,pi]@env*
let *?src=[p,P,leq,o,t] @ env @ [pi,u]*
let *?m=7 #+ length(env) - weak(length(env),5)*
let *?p=weak(length(env),5)*
let *?f=sep_var(length(env))*
let *?g=sep_env(length(env))*
let *?n=length(env)*
from *assms*
have *1 : [t,p,u,P,leq,o,pi] ∈ list(M) env ∈ list(M)*
?src ∈ list(M) ?tgt ∈ list(M)
7#+?n = (7#+?n-weak(?n,5)) ∪ weak(?n,5)
length(?src) = (7#+?n-weak(?n,5)) ∪ weak(?n,5)
using *Diff_partition le_imp_subset rensep_aux* **by** *auto*
then
have *nth(i, ?src) = nth(union_fun(?f, ?g, ?m, ?p) ‘ i, ?tgt)* **if** *i < 7#+length(env)* **for** *i*
proof -
from ‹*i<7#+?n*›
have *i ∈ (7#+?n-weak(?n,5)) ∪ weak(?n,5)*
using *ltD* **by** *simp*
then show *?thesis*
unfolding *rensep_def* **using**
union_fun_action[OF ‹*?src∈list(M)*› ‹*?tgt∈list(M)*› ‹*length(?src) = (7#+?n-weak(?n,5)) ∪ weak(?n,5)*›
sep_var_action[OF ‹*[t,p,u,P,leq,o,pi] ∈ list(M)*› ‹*env∈list(M)*›*]*
sep_env_action[OF ‹*[t,p,u,P,leq,o,pi] ∈ list(M)*› ‹*env∈list(M)*›*]*
] that
by *simp*
qed
then show *?thesis* **unfolding** *rensep_def* **by** *simp*
qed

definition *sep_ren* :: [*i*,*i*] ⇒ *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*)
unfolding *sep_ren_def*
using *arity_ren rensep_type assms*
by *simp*

lemma *type_rensep* [*TC*]:
assumes φ∈*formula env*∈*list*(*M*)
shows *sep_ren*(*length*(*env*),φ) ∈ *formula*
unfolding *sep_ren_def*
using *ren_tc rensep_type assms*
by *simp*

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 -
from *assms*
have *1*: [*t*, *p*, *u*, *P*, *leq*, *o*, *pi*] @ *env* ∈ *list*(*M*)
[*P*,*leq*,*o*,*p*,*t*] ∈ *list*(*M*)
[*pi*,*u*] ∈ *list*(*M*)
by *simp_all*
then
have *2*: [*p*,*P*,*leq*,*o*,*t*] @ *env* @ [*pi*,*u*] ∈ *list*(*M*) **using** *app_type* **by** *simp*
show *?thesis*
unfolding *sep_ren_def*
using *sats_iff_sats_ren*[*OF* ‹φ∈*formula*›
add_type[*of 7 length*(*env*)]
add_type[*of 7 length*(*env*)]
2 1(*1*)
rensep_type[*OF length_type*[*OF* ‹*env*∈*list*(*M*)›]]
‹*arity*(φ) ≤ *7* #+ *length*(*env*)›]
rensep_action[*OF 1*(*1*),*rule_format*,*symmetric*]
by *simp*
qed

end

21 The Axiom of Separation in $M[G]$

theory *Separation_Axiom*

```
  imports Forcing_Theorems Separation_Rename
begin

context G_generic
begin

lemma map_val :
  assumes env∈list(M[G])
  shows ∃ nenv∈list(M). env = map(val(P,G),nenv)
  using assms
  proof(induct env)
    case Nil
    have map(val(P,G),Nil) = Nil by simp
    then show ?case by force
  next
    case (Cons a l)
    then obtain a' l' where
      l' ∈ list(M) l=map(val(P,G),l') a = val(P,G,a')
      Cons(a,l) = map(val(P,G),Cons(a',l')) Cons(a',l') ∈ list(M)
      using ‹a∈M[G]› GenExtD
      by force
    then show ?case by force
qed


lemma Collect_sats_in_MG :
  assumes
    c∈M[G]
    φ ∈ formula env∈list(M[G]) arity(φ) ≤ 1 #+ length(env)
  shows
    {x∈c. (M[G], [x] @ env ⊨ φ)}∈ M[G]
proof -
  from ‹c∈M[G]›
  obtain π where π ∈ M val(P,G, π) = c
    using GenExt_def by auto
  let ?χ=And(Member(0,1 #+ length(env)),φ) and ?Pl1=[P,leq,one]
  let ?new_form=sep_ren(length(env),forces(?χ))
  let ?ψ=Exists(Exists(And(pair_fm(0,1,2),?new_form)))
  note phi = ‹φ∈formula› ‹arity(φ) ≤ 1 #+ length(env)›
  then
  have ?χ∈formula by simp
  with ‹env∈_› phi
  have arity(?χ) ≤ 2#+length(env)
    using nat_simp_union leI by simp
  with ‹env∈list(_)› phi
  have arity(forces(?χ)) ≤ 6 #+ length(env)
    using  arity_forces_le by simp
  then
  have arity(forces(?χ)) ≤ 7 #+ length(env)
```

using *nat_simp_union arity_forces leI* **by** *simp*
with ‹*arity*(*forces*(*?χ*)) ≤*7* #+ _› ‹*env* ∈ _› ‹*φ* ∈ *formula*›
have *arity*(*?new_form*) ≤ *7* #+ *length*(*env*) *?new_form* ∈ *formula*
using *arity_rensep*[*OF definability*[*of ?χ*]] *definability*[*of ?χ*] *type_rensep*
by *auto*
then
have *pred*(*pred*(*arity*(*?new_form*))) ≤ *5* #+ *length*(*env*) *?ψ*∈*formula*
unfolding *pair_fm_def upair_fm_def*
using *nat_simp_union length_type*[*OF* ‹*env*∈*list*(*M*[*G*])›]
pred_mono[*OF* _ *pred_mono*[*OF* _ ‹*arity*(*?new_form*) ≤ _›]]
by *auto*
with ‹*arity*(*?new_form*) ≤ _› ‹*?new_form* ∈ *formula*›
have *arity*(*?ψ*) ≤ *5* #+ *length*(*env*)
unfolding *pair_fm_def upair_fm_def*
using *nat_simp_union arity_forces*
by *auto*
from ‹*φ*∈*formula*›
have *forces*(*?χ*) ∈ *formula*
using *definability* **by** *simp*
from ‹*π*∈*M*› *P_in_M*
have *domain*(*π*)∈*M* *domain*(*π*) × *P* ∈ *M*
by (*simp_all flip*:*setclass_iff*)
from ‹*env* ∈ _›
obtain *nenv* **where** *nenv*∈*list*(*M*) *env* = *map*(*val*(*P*,*G*),*nenv*) *length*(*nenv*) = *length*(*env*)
using *map_val* **by** *auto*
from ‹*arity*(*φ*) ≤ _› ‹*env*∈_› ‹*φ*∈_›
have *arity*(*φ*) ≤ *2*#+ *length*(*env*)
using *le_trans*[*OF* ‹*arity*(*φ*)≤_›] *add_le_mono*[*of 1 2*,*OF* _ *le_refl*]
by *auto*
with ‹*nenv*∈_› ‹*env*∈_› ‹*π*∈*M*› ‹*φ*∈_› ‹*length*(*nenv*) = *length*(*env*)›
have *arity*(*?χ*) ≤ *length*([*ϑ*] @ *nenv* @ [*π*]) **for** *ϑ*
using *nat_union_abs2*[*OF* _ _ ‹*arity*(*φ*) ≤ *2*#+ _›] *nat_simp_union*
by *simp*
note *in_M* = ‹*π*∈*M*› ‹*domain*(*π*) × *P* ∈ *M*› *P_in_M one_in_M leq_in_M*
{
fix *u*
assume *u* ∈ *domain*(*π*) × *P* *u* ∈ *M*
with *in_M* ‹*?new_form* ∈ *formula*› ‹*?ψ*∈*formula*› ‹*nenv* ∈ _›
have *Eq1*: (*M*, [*u*] @ *?Pl1* @ [*π*] @ *nenv* ⊨ *?ψ*) ⟷
(∃ *ϑ*∈*M*. ∃ *p*∈*P*. *u* =⟨*ϑ*,*p*⟩ ∧
M, [*ϑ*,*p*,*u*]@*?Pl1*@[*π*] @ *nenv* ⊨ *?new_form*)
by (*auto simp add*: *transitivity*)
have *Eq3*: *ϑ*∈*M* ⟹ *p*∈*P* ⟹
(*M*, [*ϑ*,*p*,*u*]@*?Pl1*@[*π*]@*nenv* ⊨ *?new_form*) ⟷
(∀ *F*. *M_generic*(*F*) ∧ *p* ∈ *F* ⟶ (*M*[*F*], *map*(*val*(*P*,*F*), [*ϑ*] @ *nenv*@[*π*]) ⊨ *?χ*))
for *ϑ p*
proof -

fix *p ϑ*
assume $\vartheta \in M$ $p{\in}P$
then
have $p{\in}M$ **using** *P_in_M* **by** (*simp add*: *transitivity*)
note *in_M′* = *in_M* ‹$\vartheta \in M$› ‹$p{\in}M$› ‹$u \in domain(\pi) \times P$› ‹$u \in M$› ‹*nenv*∈_›
then
have $[\vartheta,u] \in list(M)$ **by** *simp*
let *?env*=[*p*]@*?Pl1*@[ϑ] @ *nenv* @ [π,*u*]
let *?new_env*= [ϑ,*p*,*u*,*P*,*leq*,*one*,π] @ *nenv*
let *?ψ*=*Exists*(*Exists*(*And*(*pair_fm*(*0*,*1*,*2*),*?new_form*)))
have [ϑ, *p*, *u*, π, *leq*, *one*, π] ∈ *list*(*M*)
  using *in_M′* **by** *simp*
have *?χ* ∈ *formula* *forces*(*?χ*)∈ *formula*
  using *phi* **by** *simp_all*
from *in_M′*
have *?Pl1* ∈ *list*(*M*) **by** *simp*
from *in_M′* **have** *?env* ∈ *list*(*M*) **by** *simp*
have *Eq1′*: *?new_env* ∈ *list*(*M*) **using** *in_M′* **by** *simp*
then
have (*M*, [ϑ,*p*,*u*]@*?Pl1*@[π] @ *nenv* ⊨ *?new_form*) ⟷ (*M*, *?new_env* ⊨ *?new_form*)
  by *simp*
from *in_M′* ‹*env* ∈ _› *Eq1′* ‹*length*(*nenv*) = *length*(*env*)›
  ‹*arity*(*forces*(*?χ*)) ≤ *7* #+ *length*(*env*)› ‹*forces*(*?χ*)∈ *formula*›
  ‹[ϑ, *p*, *u*, π, *leq*, *one*, π] ∈ *list*(*M*)›
have ... ⟷ *M*, *?env* ⊨ *forces*(*?χ*)
  using *sepren_action*[*of forces*(*?χ*) *nenv*,*OF* _ _ ‹*nenv*∈*list*(*M*)›]
  by *simp*
also from *in_M′*
have ... ⟷ *M*, ([*p*,*P*, *leq*, *one*,ϑ]@*nenv*@ [π])@[*u*] ⊨ *forces*(*?χ*)
  using *app_assoc* **by** *simp*
also
from *in_M′* ‹*env*∈_› *phi* ‹*length*(*nenv*) = *length*(*env*)›
  ‹*arity*(*forces*(*?χ*)) ≤ *6* #+ *length*(*env*)› ‹*forces*(*?χ*)∈*formula*›
have ... ⟷ *M*, [*p*,*P*, *leq*, *one*,ϑ]@ *nenv* @ [π] ⊨ *forces*(*?χ*)
  by (*rule_tac arity_sats_iff*,*auto*)
also
from ‹*arity*(*forces*(*?χ*)) ≤ *6* #+ *length*(*env*)› ‹*forces*(*?χ*)∈*formula*› *in_M′* *phi*
have ... ⟷ (∀ *F*. *M_generic*(*F*) ∧ *p* ∈ *F* ⟶
    M[*F*], *map*(*val*(*P*,*F*), [ϑ] @ *nenv* @ [π]) ⊨ *?χ*)
  using *definition_of_forcing*
proof (*intro iffI*)
  assume *a1*: *M*, [*p*,*P*, *leq*, *one*,ϑ] @ *nenv* @ [π] ⊨ *forces*(*?χ*)
  note *definition_of_forcing* ‹*arity*(φ)≤ *1*#+_›
  with ‹*nenv*∈_› ‹*arity*(*?χ*) ≤ *length*([ϑ] @ *nenv* @ [π])› ‹*env*∈_›
  have *p* ∈ *P* ⟹ *?χ*∈*formula* ⟹ [ϑ,π] ∈ *list*(*M*) ⟹
      M, [*p*,*P*, *leq*, *one*] @ [ϑ]@ *nenv*@[π] ⊨ *forces*(*?χ*) ⟹
    ∀ *G*. *M_generic*(*G*) ∧ *p* ∈ *G* ⟶ *M*[*G*], *map*(*val*(*P*,*G*), [ϑ] @ *nenv*

@[π]) ⊨ ?χ
by *auto*
then
show ∀ F. M_generic(F) ∧ p ∈ F ⟶
M[F], map(val(P,F), [ϑ] @ nenv @ [π]) ⊨ ?χ
using ‹?χ∈formula› ‹p∈P› a1 ‹ϑ∈M› ‹π∈M› **by** *simp*
next
assume ∀ F. M_generic(F) ∧ p ∈ F ⟶
M[F], map(val(P,F), [ϑ] @ nenv @[π]) ⊨ ?χ
with definition_of_forcing [THEN iffD2] ‹arity(?χ) ≤ length([ϑ] @ nenv @ [π])›
show M, [p, P, leq, one,ϑ] @ nenv @ [π] ⊨ forces(?χ)
using ‹?χ∈formula› ‹p∈P› in_M′
by *auto*
qed
finally
show (M, [ϑ,p,u]@?Pl1@[π]@nenv ⊨ ?new_form) ⟷ (∀ F. M_generic(F) ∧ p ∈ F ⟶
M[F], map(val(P,F), [ϑ] @ nenv @ [π]) ⊨ ?χ)
by *simp*
qed
with Eq1
have (M, [u] @ ?Pl1 @ [π] @ nenv ⊨ ?ψ) ⟷
(∃ϑ∈M. ∃p∈P. u =⟨ϑ,p⟩ ∧
(∀ F. M_generic(F) ∧ p ∈ F ⟶ M[F], map(val(P,F), [ϑ] @ nenv @ [π]) ⊨ ?χ))
by *auto*
}
then
have Equivalence: u∈ domain(π) × P ⟹ u ∈ M ⟹
(M, [u] @ ?Pl1 @ [π] @ nenv ⊨ ?ψ) ⟷
(∃ϑ∈M. ∃p∈P. u =⟨ϑ,p⟩ ∧
(∀ F. M_generic(F) ∧ p ∈ F ⟶ M[F], map(val(P,F), [ϑ] @ nenv @[π]) ⊨ ?χ))
for u
by *simp*
moreover from ‹env = _› ‹π∈M› ‹nenv∈list(M)›
have map_nenv:map(val(P,G), nenv@[π]) = env @ [val(P,G,π)]
using map_app_distrib append1_eq_iff **by** *auto*
ultimately
have aux:(∃ϑ∈M. ∃p∈P. u =⟨ϑ,p⟩ ∧ (p∈G ⟶ M[G], [val(P,G,ϑ)] @ env @ [val(P,G,π)] ⊨ ?χ))
(**is** (∃ϑ∈M. ∃p∈P. _ (_ ⟶ _, ?vals(ϑ) ⊨ _)))
if u ∈ domain(π) × P u ∈ M M, [u]@ ?Pl1 @[π] @ nenv ⊨ ?ψ **for** u
using Equivalence[THEN iffD1, OF that] generic **by** *force*
moreover
have ϑ∈M ⟹ val(P,G,ϑ)∈M[G] **for** ϑ
using GenExt_def **by** *auto*
moreover

have $\vartheta\in M \Longrightarrow [val(P,G, \vartheta)]$ @ *env* @ $[val(P,G, \pi)] \in list(M[G])$ **for** ϑ
proof -
from ‹$\pi\in M$›
have $val(P,G,\pi)\in M[G]$ **using** *GenExtI* **by** *simp*
moreover
assume $\vartheta \in M$
moreover
note ‹$env \in list(M[G])$›
ultimately
show *?thesis*
using *GenExtI* **by** *simp*
qed
ultimately
have $(\exists\vartheta\in M. \exists p\in P. u=\langle\vartheta,p\rangle \wedge (p\in G \longrightarrow val(P,G,\vartheta)\in nth(1 \#+ length(env),[val(P,G,$ $\vartheta)]$ @ *env* @ $[val(P,G, \pi)])$
$\wedge\ M[G], \ ?vals(\vartheta) \models \ \varphi))$
if $u \in domain(\pi) \times P$ $u \in M$ M, $[u]$ @ *?Pl1* @$[\pi]$ @ *nenv* $\models$ $?\psi$ **for** u
using *aux*[*OF that*] **by** *simp*
moreover from ‹$env \in$ _› ‹$\pi\in M$›
have *nth*:$nth(1 \#+ length(env),[val(P,G, \vartheta)]$ @ *env* @ $[val(P,G, \pi)]) = val(P,G,\pi)$

if $\vartheta\in M$ **for** ϑ
using *nth_concat*[*of* $val(P,G,\vartheta)$ $val(P,G,\pi)$ $M[G]$] **using** *that GenExtI* **by** *simp*
ultimately
have $(\exists\vartheta\in M. \exists p\in P. u=\langle\vartheta,p\rangle \wedge (p\in G \longrightarrow val(P,G,\vartheta)\in val(P,G,\pi) \wedge M[G],$ $?vals(\vartheta) \models \ \varphi))$
if $u \in domain(\pi) \times P$ $u \in M$ M, $[u]$ @ *?Pl1* @$[\pi]$ @ *nenv* $\models$ $?\psi$ **for** u
using *that* ‹$\pi\in M$› ‹$env \in$ _› **by** *simp*
with ‹$domain(\pi)\times P\in M$›
have $\forall u\in domain(\pi)\times P$. $(M$, $[u]$ @ *?Pl1* @$[\pi]$ @ *nenv* $\models$ $?\psi) \longrightarrow (\exists\vartheta\in M.$ $\exists p\in P. u =\langle\vartheta,p\rangle \wedge$
$(p \in G \longrightarrow val(P,G, \vartheta)\in val(P,G, \pi) \wedge M[G], \ ?vals(\vartheta) \models \ \varphi))$
by (*simp add*:*transitivity*)
then
have $\{u\in domain(\pi)\times P$. $(M,[u]$ @ *?Pl1* @$[\pi]$ @ *nenv* $\models$ $?\psi) \} \subseteq$
$\{u\in domain(\pi)\times P$. $\exists\vartheta\in M. \exists p\in P. u =\langle\vartheta,p\rangle \wedge$
$(p \in G \longrightarrow val(P,G, \vartheta)\in val(P,G, \pi) \wedge (M[G],\ ?vals(\vartheta) \models \varphi))\}$
(**is** $?n\subseteq ?m$)
by *auto*
with *val_mono*
have *first_incl*: $val(P,G,?n) \subseteq val(P,G,?m)$
by *simp*
note ‹$val(P,G,\pi) = c$›
with ‹$?\psi\in formula$› ‹$arity(?\psi) \leq$ _› *in_M* ‹$nenv \in$ _› ‹$env \in$ _› ‹$length(nenv) =$ _›
have $?n\in M$
using *separation_ax leI separation_iff* **by** *auto*
from *generic*
have $filter(G)$ $G\subseteq P$

```
    unfolding M_generic_def filter_def by simp_all
  from ‹val(P,G,π) = c›
  have val(P,G,?m) =
              {val(P,G,t) .. t∈domain(π) , ∃q∈P .
                   (∃ϑ∈M. ∃p∈P. ⟨t,q⟩ = ⟨ϑ, p⟩ ∧
            (p ∈ G ⟶ val(P,G, ϑ) ∈ c ∧ (M[G],  [val(P,G, ϑ)] @ env @ [c] ⊨
φ)) ∧ q ∈ G)}
    using val_of_name by auto
  also
  have ... =  {val(P,G,t) .. t∈domain(π) , ∃q∈P.
                   val(P,G, t) ∈ c ∧ (M[G],  [val(P,G, t)] @ env @ [c] ⊨  φ) ∧ q ∈
G}
  proof -

    have t∈M ⟹
      (∃q∈P. (∃ϑ∈M. ∃p∈P. ⟨t,q⟩ = ⟨ϑ, p⟩ ∧
              (p ∈ G ⟶ val(P,G, ϑ) ∈ c ∧ (M[G],  [val(P,G, ϑ)] @ env @ [c] ⊨
φ)) ∧ q ∈ G))
      ⟷
      (∃q∈P. val(P,G, t) ∈ c ∧ ( M[G], [val(P,G, t)]@env@[c]⊨ φ ) ∧ q ∈ G)
for t
      by auto
    then show ?thesis using ‹domain(π)∈M› by (auto simp add:transitivity)
  qed
  also
  have ... =  {x .. x∈c , ∃q∈P. x ∈ c ∧ (M[G],  [x] @ env @ [c] ⊨  φ) ∧ q ∈ G}
  proof

    show ... ⊆ {x .. x∈c , ∃q∈P. x ∈ c ∧ (M[G],  [x] @ env @ [c] ⊨  φ) ∧ q ∈
G}
      by auto
  next

    {
      fix x
      assume x∈{x .. x∈c , ∃q∈P. x ∈ c ∧ (M[G],  [x] @ env @ [c] ⊨  φ) ∧ q ∈
G}
      then
      have ∃q∈P. x ∈ c ∧ (M[G],  [x] @ env @ [c] ⊨  φ) ∧ q ∈ G
        by simp
      with ‹val(P,G,π) = c›
      have ∃q∈P. ∃t∈domain(π). val(P,G,t) =x ∧ (M[G],  [val(P,G,t)] @ env @
[c] ⊨  φ) ∧ q ∈ G
        using Sep_and_Replace elem_of_val by auto
    }
    then
    show  {x .. x∈c , ∃q∈P. x ∈ c ∧ (M[G],  [x] @ env @ [c] ⊨  φ) ∧ q ∈ G} ⊆
...
      using SepReplace_iff by force
```

qed
also
have ... = {x∈c. (M[G], [x] @ env @ [c] ⊨ φ)}
using ‹G⊆P› *G_nonempty* **by** *force*
finally
have *val_m*: *val*(P,G,?m) = {x∈c. (M[G], [x] @ env @ [c] ⊨ φ)} **by** *simp*
have *val*(P,G,?m) ⊆ *val*(P,G,?n)
proof
fix x
assume x ∈ *val*(P,G,?m)
with *val_m*
have *Eq4*: x ∈ {x∈c. (M[G], [x] @ env @ [c] ⊨ φ)} **by** *simp*
with ‹*val*(P,G,π) = c›
have x ∈ *val*(P,G,π) **by** *simp*
then
have ∃ϑ. ∃q∈G. ⟨ϑ,q⟩∈π ∧ *val*(P,G,ϑ) =x
using *elem_of_val_pair* **by** *auto*
then obtain ϑ q **where**
⟨ϑ,q⟩∈π q∈G *val*(P,G,ϑ)=x **by** *auto*
from ‹⟨ϑ,q⟩∈π›
have ϑ∈M
using *domain_trans*[OF *trans_M* ‹π∈_›] **by** *auto*
with ‹π∈M› ‹nenv ∈ _› ‹env = _›
have [*val*(P,G,ϑ), *val*(P,G,π)] @ env ∈*list*(M[G])
using *GenExt_def* **by** *auto*
with *Eq4* ‹*val*(P,G,ϑ)=x› ‹*val*(P,G,π) = c› ‹x ∈ *val*(P,G,π)› *nth* ‹ϑ∈M›
have *Eq5*: M[G], [*val*(P,G,ϑ)] @ env @[*val*(P,G,π)] ⊨ *And*(*Member*(0,1 #+ *length*(env)),φ)
by *auto*

with ‹ϑ∈M› ‹π∈M› *Eq5* ‹*M_generic*(G)› ‹φ∈*formula*› ‹nenv ∈ _ › ‹env = _ › *map_nenv*
‹*arity*(?χ) ≤ *length*([ϑ] @ nenv @ [π])›
have (∃r∈G. M, [r,P,leq,one,ϑ] @ nenv @[π] ⊨ *forces*(?χ))
using *truth_lemma*
by *auto*
then obtain r **where**
r∈G M, [r,P,leq,one,ϑ] @ nenv @ [π] ⊨ *forces*(?χ) **by** *auto*
with ‹*filter*(G)› **and** ‹q∈G› **obtain** p **where**
p∈G p⪯q p⪯r
unfolding *filter_def compat_in_def* **by** *force*
with ‹r∈G› ‹q∈G› ‹G⊆P›
have p∈P r∈P q∈P p∈M
using *P_in_M* **by** (*auto simp add*:*transitivity*)
with ‹φ∈*formula*› ‹ϑ∈M› ‹π∈M› ‹p⪯r› ‹nenv ∈ _› ‹*arity*(?χ) ≤ *length*([ϑ] @ nenv @ [π])›
‹M, [r,P,leq,one,ϑ] @ nenv @ [π] ⊨ *forces*(?χ)› ‹env∈_›
have M, [p,P,leq,one,ϑ] @ nenv @ [π] ⊨ *forces*(?χ)
using *strengthening_lemma*

by *simp*
with ‹$p\in P$› ‹$\varphi\in formula$› ‹$\vartheta\in M$› ‹$\pi\in M$› ‹$nenv \in \_$› ‹$arity(?\chi) \le length([\vartheta]$ @ $nenv$ @ $[\pi])$›
have $\forall F.\ M\_generic(F) \wedge p \in F \longrightarrow$
$M[F],\quad map(val(P,F), [\vartheta]$ @ $nenv$ @$[\pi]) \models\ ?\chi$
using *definition_of_forcing*
by *simp*
with ‹$p\in P$› ‹$\vartheta\in M$›
have *Eq6*: $\exists\,\vartheta'\in M.\ \exists\,p'\in P.\ \langle\vartheta,p\rangle = <\vartheta',p'> \wedge (\forall\,F.\ M\_generic(F) \wedge p' \in F \longrightarrow$
$M[F],\quad map(val(P,F), [\vartheta']$ @ $nenv$ @ $[\pi]) \models\ ?\chi)$ **by** *auto*
from ‹$\pi\in M$› ‹$\langle\vartheta,q\rangle\in\pi$›
have $\langle\vartheta,q\rangle \in M$ **by** (*simp add:transitivity*)
from ‹$\langle\vartheta,q\rangle\in\pi$› ‹$\vartheta\in M$› ‹$p\in P$› ‹$p\in M$›
have $\langle\vartheta,p\rangle\in M$ $\langle\vartheta,p\rangle\in domain(\pi)\times P$
using *tuples_in_M* **by** *auto*
with ‹$\vartheta\in M$› *Eq6* ‹$p\in P$›
have M, $[\langle\vartheta,p\rangle]$ @ $?Pl1$ @ $[\pi]$ @ $nenv \models ?\psi$
using *Equivalence* **by** *auto*
with ‹$\langle\vartheta,p\rangle\in domain(\pi)\times P$›
have $\langle\vartheta,p\rangle\in ?n$ **by** *simp*
with ‹$p\in G$› ‹$p\in P$›
have $val(P,G,\vartheta)\in val(P,G,?n)$
using *val_of_elem*[*of* ϑ *p*] **by** *simp*
with ‹$val(P,G,\vartheta)=x$›
show $x\in val(P,G,?n)$ **by** *simp*
qed
with *val_m first_incl*
have $val(P,G,?n) = \{x\in c.\ (M[G], [x]$ @ env @ $[c] \models \varphi)\}$ **by** *auto*
also
have ... $= \{x\in c.\ (M[G], [x]$ @ $env \models \varphi)\}$
proof -
{
fix *x*
assume $x\in c$
moreover from *assms*
have $c\in M[G]$
unfolding *GenExt_def* **by** *auto*
moreover from *this* **and** ‹$x\in c$›
have $x\in M[G]$
using *transitivity_MG*
by *simp*
ultimately
have $(M[G],\ ([x]$ @ $env)$ @$[c] \models\ \varphi) \longleftrightarrow (M[G],\ [x]$ @ $env \models\ \varphi)$
using *phi* ‹$env \in \_$› **by** (*rule_tac arity_sats_iff*, *simp_all*)
}
then show *?thesis* **by** *auto*
qed
finally

show {x∈c. (M[G], [x] @ env ⊨ φ)}∈ M[G]
using ‹?n∈M› GenExt_def by force
qed

theorem *separation_in_MG*:
assumes
φ∈formula **and** arity(φ) ≤ 1 #+ length(env) **and** env∈list(M[G])
shows
separation(##M[G],λx. (M[G], [x] @ env ⊨ φ))
proof -
{
fix c
assume c∈M[G]
moreover from ‹env ∈ _›
obtain nenv **where** nenv∈list(M)
env = map(val(P,G),nenv) length(env) = length(nenv)
using GenExt_def map_val[of env] **by** auto
moreover note ‹φ ∈ _› ‹arity(φ) ≤ _› ‹env ∈ _›
ultimately
have Eq1: {x∈c. (M[G], [x] @ env ⊨ φ)} ∈ M[G]
using Collect_sats_in_MG **by** auto
}
then
show ?thesis
using separation_iff rev_bexI **unfolding** is_Collect_def **by** force
qed

end

end

22 The Axiom of Pairing in $M[G]$

theory *Pairing_Axiom* **imports** *Names* **begin**

context *forcing_data*
begin

lemma *val_Upair* :
one ∈ G ⟹ val(P,G,{⟨τ,one⟩,⟨ϱ,one⟩}) = {val(P,G,τ),val(P,G,ϱ)}
by (insert one_in_P, rule trans, subst def_val,auto simp add: Sep_and_Replace)

lemma *pairing_in_MG* :
assumes M_generic(G)
shows upair_ax(##M[G])
proof -
{
fix x y
have one∈G **using** assms one_in_G **by** simp

from *assms*
have $G \subseteq P$ **unfolding** *M_generic_def* **and** *filter_def* **by** *simp*
with ‹$one \in G$›
have $one \in P$ **using** *subsetD* **by** *simp*
then
have $one \in M$ **using** *transitivity*[*OF* _ *P_in_M*] **by** *simp*
assume $x \in M[G]$ $y \in M[G]$
then
obtain τ ϱ **where**
0 : $val(P,G,\tau) = x$ $val(P,G,\varrho) = y$ $\varrho \in M$ $\tau \in M$
using *GenExtD* **by** *blast*
with ‹$one \in M$›
have $\langle \tau, one \rangle \in M$ $\langle \varrho, one \rangle \in M$ **using** *pair_in_M_iff* **by** *auto*
then
have *1*: $\{\langle \tau, one \rangle, \langle \varrho, one \rangle\} \in M$ (**is** $?\sigma \in$ _) **using** *upair_in_M_iff* **by** *simp*
then
have $val(P,G,?\sigma) \in M[G]$ **using** *GenExtI* **by** *simp*
with *1*
have $\{val(P,G,\tau), val(P,G,\varrho)\} \in M[G]$ **using** *val_Upair assms one_in_G* **by** *simp*
with *0*
have $\{x,y\} \in M[G]$ **by** *simp*
}
then show *?thesis* **unfolding** *upair_ax_def upair_def* **by** *auto*
qed

end
end

23 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 $\equiv$
Exists(
Exists(*And*(*pair_fm*(*1*,*0*,*2*),

```
  Exists (
  Exists (And(Member(0,7),
    Exists (And(And(pair_fm(2,1,0),Member(0,6)),
      Exists (And(Member(0,9),
       Exists (And(And(pair_fm(6,1,0),Member(0,4)),
        Exists (And(And(pair_fm(6,2,0),Member(0,10)),
        Exists (And(pair_fm(7,5,0),Member(0,11)))))))))))))))

lemma Union_name_fm_type [TC]:
  Union_name_fm ∈formula
  unfolding Union_name_fm_def by simp


lemma arity_Union_name_fm :
  arity(Union_name_fm) = 4
  unfolding Union_name_fm_def upair_fm_def pair_fm_def
  by(auto simp add: nat_simp_union)

lemma sats_Union_name_fm :
  ⟦ env ∈ list(M); P' ∈ M ; p ∈ M ; ϑ ∈ M ; τ ∈ M ; leq' ∈ M ⟧ ⟹
    sats(M,Union_name_fm,[⟨ϑ,p⟩,τ,leq',P']@env) ⟷
    Union_name_body(P',leq',τ,⟨ϑ,p⟩)
  unfolding Union_name_fm_def Union_name_body_def tuples_in_M
  by (subgoal_tac ⟨ϑ,p⟩ ∈ M, auto simp add : tuples_in_M)

definition Union_name :: i ⇒ i where
  Union_name(τ) ≡
    {u ∈ domain(⋃(domain(τ))) × P . Union_name_body(P,leq,τ,u)}

lemma Union_name_M : assumes τ ∈ M
  shows Union_name(τ) ∈ M
proof -
  let ?P=λ x . sats(M,Union_name_fm,[x,τ,leq,P])
  let ?Q=λ x . Union_name_body(P,leq,τ,x)
  from ‹τ∈M›
  have domain(⋃(domain(τ)))∈M (is ?d ∈ _) using domain_closed Union_closed
by simp
  then
  have ?d × P ∈ M using cartprod_closed P_in_M by simp
  have arity(Union_name_fm)≤4 using arity_Union_name_fm by simp
  with ‹τ∈M› P_in_M leq_in_M
  have separation(##M,?P)
    using separation_ax by simp
  with ‹?d × P ∈ M›
  have A:{ u ∈ ?d × P . ?P(u) } ∈ M
    using separation_iff by force
  have ?P(x)⟷ ?Q(x) if x∈?d×P for x
  proof -
    from ‹x∈?d×P›
```

```
    have x = ⟨fst(x),snd(x)⟩ using Pair_fst_snd_eq by simp
    with ‹x∈?d×P› ‹?d∈M›
    have fst(x) ∈ M snd(x) ∈ M
      using transitivity fst_type snd_type P_in_M by auto
    then
    have ?P(⟨fst(x),snd(x)⟩) ⟷ ?Q(⟨fst(x),snd(x)⟩)
      using P_in_M sats_Union_name_fm P_in_M ‹τ∈M› leq_in_M by simp
    with ‹x = ⟨fst(x),snd(x)⟩›
    show ?P(x) ⟷ ?Q(x) using ‹x∈_› by simp
  qed
  then show ?thesis using Collect_cong A unfolding Union_name_def by simp
qed

lemma Union_MG_Eq :
  assumes a ∈ M[G] and a = val(P,G,τ) and filter(G) and τ ∈ M
  shows ⋃ a = val(P,G,Union_name(τ))
proof -
  {
    fix x
    assume x ∈ ⋃ (val(P,G,τ))
    then obtain i where i ∈ val(P,G,τ) x ∈ i by blast
    with ‹τ ∈ M› obtain σ q where
      q ∈ G ⟨σ,q⟩ ∈ τ val(P,G,σ) = i σ ∈ M
      using elem_of_val_pair domain_trans[OF trans_M] by blast
    with ‹x ∈ i› obtain ϑ r where
      r ∈ G ⟨ϑ,r⟩ ∈ σ val(P,G,ϑ) = x ϑ ∈ M
      using elem_of_val_pair domain_trans[OF trans_M] by blast
    with ‹⟨σ,q⟩∈τ› have ϑ ∈ domain(⋃(domain(τ))) by auto
    with ‹filter(G)› ‹q∈G› ‹r∈G› obtain p where
      A: p ∈ G ⟨p,r⟩ ∈ leq ⟨p,q⟩ ∈ leq p ∈ P r ∈ P q ∈ P
      using low_bound_filter filterD  by blast
    then
    have p ∈ M q∈M r∈M
      using P_in_M by (auto dest:transM)
    with A ‹⟨ϑ,r⟩ ∈ σ› ‹⟨σ,q⟩ ∈ τ› ‹ϑ ∈ M› ‹ϑ ∈ domain(⋃(domain(τ)))›  ‹σ∈M›
    have ⟨ϑ,p⟩ ∈ Union_name(τ)
      unfolding Union_name_def Union_name_body_def
      by auto
    with ‹p∈P› ‹p∈G›
    have val(P,G,ϑ) ∈ val(P,G,Union_name(τ))
      using val_of_elem by simp
    with ‹val(P,G,ϑ)=x›
    have x ∈ val(P,G,Union_name(τ)) by simp
  }
  with ‹a=val(P,G,τ)›
  have 1: x ∈ ⋃ a ⟹ x ∈ val(P,G,Union_name(τ)) for x by simp
  {
    fix x
    assume x ∈ (val(P,G,Union_name(τ)))
```

```
    then obtain ϑ p where
      p ∈ G ⟨ϑ,p⟩ ∈ Union_name(τ) val(P,G,ϑ) = x
      using elem_of_val_pair by blast
    with ‹filter(G)› have p∈P using filterD by simp
    from ‹⟨ϑ,p⟩ ∈ Union_name(τ)› obtain σ q r where
      σ ∈ domain(τ)  ⟨σ,q⟩ ∈ τ  ⟨ϑ,r⟩ ∈ σ r∈P q∈P ⟨p,r⟩ ∈ leq ⟨p,q⟩ ∈ leq
      unfolding Union_name_def Union_name_body_def by force
    with ‹p∈G› ‹filter(G)› have r ∈ G q ∈ G
      using filter_leqD by auto
    with ‹⟨ϑ,r⟩ ∈ σ› ‹⟨σ,q⟩∈τ› ‹q∈P› ‹r∈P› have
      val(P,G,σ) ∈ val(P,G,τ) val(P,G,ϑ) ∈ val(P,G,σ)
      using val_of_elem by simp+
    then have val(P,G,ϑ) ∈ ⋃ val(P,G,τ) by blast
    with ‹val(P,G,ϑ)=x› ‹a=val(P,G,τ)› have
      x ∈ ⋃ a by simp
  }
  with ‹a=val(P,G,τ)›
  have x ∈ val(P,G,Union_name(τ)) ⟹ x ∈ ⋃ a for x by blast
  then
  show ?thesis using 1 by blast
qed

lemma union_in_MG : assumes filter(G)
  shows Union_ax(##M[G])
proof -
  { fix a
    assume a ∈ M[G]
    then
    interpret mgtrans : M_trans ##M[G]
      using transitivity_MG by (unfold_locales; auto)
    from ‹a∈_› obtain τ where τ ∈ M a=val(P,G,τ) using GenExtD by blast
    then
     have Union_name(τ) ∈ M (is ?π ∈ _) using Union_name_M unfolding
Union_name_def by simp
    then
    have val(P,G,?π) ∈ M[G] (is ?U ∈ _) using GenExtI by simp
    with ‹a∈_›
    have (##M[G])(a) (##M[G])(?U) by auto
    with ‹τ ∈ M› ‹filter(G)› ‹?U ∈ M[G]› ‹a=val(P,G,τ)›
    have big_union(##M[G],a,?U)
      using Union_MG_Eq Union_abs by simp
    with ‹?U ∈ M[G]›
    have ∃z[##M[G]]. big_union(##M[G],a,z) by auto
  }
  then
  show ?thesis unfolding Union_ax_def by simp
qed

theorem Union_MG : M_generic(G) ⟹ Union_ax(##M[G])
```

```
  by (simp add:M_generic_def union_in_MG)

end
end
```

24 The Powerset Axiom in $M[G]$

```
theory Powerset_Axiom
  imports Renaming_Auto Separation_Axiom Pairing_Axiom Union_Axiom
begin

simple_rename perm_pow src [ss,p,l,o,fs,χ] tgt [fs,ss,sp,p,l,o,χ]

lemma Collect_inter_Transset:
  assumes
    Transset(M) b ∈ M
  shows
    {x∈b . P(x)} = {x∈b . P(x)} ∩ M
  using assms unfolding Transset_def
  by (auto)

context G_generic  begin

lemma name_components_in_M:
  assumes <σ,p>∈ϑ ϑ ∈ M
  shows   σ∈M p∈M
proof -
  from assms obtain a where
    σ ∈ a p ∈ a a∈<σ,p>
    unfolding Pair_def by auto
  moreover from assms
  have <σ,p>∈M
    using transitivity by simp
  moreover from calculation
  have a∈M
    using transitivity by simp
  ultimately
  show σ∈M p∈M
    using transitivity by simp_all
qed

lemma sats_fst_snd_in_M:
  assumes
    A∈M B∈M φ ∈ formula p∈M l∈M o∈M χ∈M
    arity(φ) ≤ 6
  shows
    {<s,q> ∈A×B . sats(M,φ,[q,p,l,o,s,χ])} ∈ M
    (is ?ϑ ∈ M)
proof -
```

```
have 6∈nat 7∈nat by simp_all
let ?φ' = ren(φ)`6`7`perm_pow_fn
from ‹A∈M› ‹B∈M› have
  A×B ∈ M
  using cartprod_closed by simp
from ‹arity(φ) ≤ 6› ‹φ∈ formula› ‹6∈_› ‹7∈_›
have ?φ' ∈ formula arity(?φ')≤7
  unfolding perm_pow_fn_def
  using perm_pow_thm arity_ren ren_tc Nil_type
  by auto
with ‹?φ' ∈ formula›
have 1: arity(Exists(Exists(And(pair_fm(0,1,2),?φ'))))≤5     (is arity(?ψ)≤5)
  unfolding pair_fm_def upair_fm_def
  using nat_simp_union pred_le arity_type by auto
{
  fix sp
  note ‹A×B ∈ M›
  moreover
  assume sp ∈ A×B
  moreover from calculation
  have fst(sp) ∈ A snd(sp) ∈ B
    using fst_type snd_type by simp_all
  ultimately
  have sp ∈ M fst(sp) ∈ M snd(sp) ∈ M
    using ‹A∈M› ‹B∈M› transitivity
    by simp_all
  note inM = ‹A∈M› ‹B∈M› ‹p∈M› ‹l∈M› ‹o∈M› ‹χ∈M›
    ‹sp∈M› ‹fst(sp)∈M› ‹snd(sp)∈M›
  with 1 ‹sp ∈ M› ‹?φ' ∈ formula›
  have M, [sp,p,l,o,χ]@[p] ⊨ ?ψ ⟷ M,[sp,p,l,o,χ] ⊨ ?ψ (is M,?env0@ _⊨_
⟷ _)
    using arity_sats_iff[of ?ψ [p] M ?env0] by auto
  also from inM ‹sp ∈ A×B›
  have ... ⟷ sats(M,?φ',[fst(sp),snd(sp),sp,p,l,o,χ])
    by auto
  also from inM ‹φ ∈ formula› ‹arity(φ) ≤ 6›
  have ... ⟷ sats(M,φ,[snd(sp),p,l,o,fst(sp),χ])
    (is sats(_,_,?env1) ⟷ sats(_,_,?env2))
    using sats_iff_sats_ren[of φ 6 7 ?env2 M ?env1 perm_pow_fn] perm_pow_thm
    unfolding perm_pow_fn_def by simp
  finally
  have sats(M,?ψ,[sp,p,l,o,χ,p]) ⟷ sats(M,φ,[snd(sp),p,l,o,fst(sp),χ])
    by simp
}
then have
  ?ϑ = {sp∈A×B . sats(M,?ψ,[sp,p,l,o,χ,p])}
  by auto
also from assms ‹A×B∈M› have
  ... ∈ M
```

```
  proof -
    from 1
    have arity(?ψ) ≤ 6
      using leI by simp
    moreover from ‹?φ' ∈ formula›
    have ?ψ ∈ formula
      by simp
    moreover note assms ‹A×B∈M›
    ultimately
    show {x ∈ A×B . sats(M, ?ψ, [x, p, l, o, χ, p])} ∈ M
      using separation_ax separation_iff
      by simp
  qed
  finally show ?thesis .
qed

lemma Pow_inter_MG:
  assumes
    a∈M[G]
  shows
    Pow(a) ∩ M[G] ∈ M[G]
proof -
  from assms obtain τ where τ ∈ M val(P,G, τ) = a
    using GenExtD by auto
  let ?Q=Pow(domain(τ)×P) ∩ M
  from ‹τ∈M›
  have domain(τ)×P ∈ M domain(τ) ∈ M
    using domain_closed cartprod_closed P_in_M
    by simp_all
  then
  have ?Q ∈ M
  proof -
    from power_ax ‹domain(τ)×P ∈ M› obtain Q where
      powerset(##M,domain(τ)×P,Q) Q ∈ M
      unfolding power_ax_def by auto
    moreover from calculation
    have z∈Q ⟹ z∈M for z
      using transitivity by blast
    ultimately
    have Q = {a∈Pow(domain(τ)×P) . a∈M}
      using ‹domain(τ)×P ∈ M› powerset_abs[of domain(τ)×P Q]
      by (simp flip: setclass_iff)
    also
    have ... = ?Q
      by auto
    finally
    show ?thesis using ‹Q∈M› by simp
  qed
  let ?π=?Q×{one}
```

```
let ?b=val(P,G,?π)
from ‹?Q∈M›
have ?π∈M
  using one_in_P P_in_M transitivity
  by (simp flip: setclass_iff)
then
have ?b ∈ M[G]
  using GenExtI by simp
have Pow(a) ∩ M[G] ⊆ ?b
proof
  fix c
  assume c ∈ Pow(a) ∩ M[G]
  then obtain χ where c∈M[G] χ ∈ M val(P,G,χ) = c
    using GenExtD by auto
  let ?ϑ={<σ,p> ∈domain(τ)×P . p ⊩ (Member(0,1)) [σ,χ] }
  have arity(forces(Member(0,1))) = 6
    using arity_forces_at by auto
  with ‹domain(τ) ∈ M› ‹χ ∈ M›
  have ?ϑ ∈ M
    using P_in_M one_in_M leq_in_M sats_fst_snd_in_M
    by simp
  then
  have ?ϑ ∈ ?Q by auto
  then
  have val(P,G,?ϑ) ∈ ?b
    using one_in_G one_in_P generic val_of_elem [of ?ϑ one ?π G]
    by auto
  have val(P,G,?ϑ) = c
  proof(intro equalityI subsetI)
    fix x
    assume x ∈ val(P,G,?ϑ)
    then obtain σ p where
      1: <σ,p>∈?ϑ p∈G val(P,G,σ) =  x
      using elem_of_val_pair
      by blast
    moreover from ‹<σ,p>∈?ϑ› ‹?ϑ ∈ M›
    have σ∈M
      using name_components_in_M[of _ _ ?ϑ] by auto
    moreover from 1
    have (p ⊩ (Member(0,1)) [σ,χ]) p∈P
      by simp_all
    moreover
    note ‹val(P,G,χ) = c›
    ultimately
    have sats(M[G],Member(0,1),[x,c])
      using ‹χ ∈ M› generic definition_of_forcing nat_simp_union
      by auto
    moreover
    have x∈M[G]
```

```
    using ‹val(P,G,σ) =  x› ‹σ∈M›  ‹χ∈M› GenExtI by blast
  ultimately
  show x∈c
    using ‹c∈M[G]› by simp
next
  fix x
  assume x ∈ c
  with ‹c ∈ Pow(a) ∩ M[G]›
  have x ∈ a c∈M[G] x∈M[G]
    using transitivity_MG by auto
  with ‹val(P,G, τ) = a›
  obtain σ where σ∈domain(τ) val(P,G,σ) = x
    using elem_of_val by blast
  moreover note ‹x∈c› ‹val(P,G,χ) = c›
  moreover from calculation
  have val(P,G,σ) ∈ val(P,G,χ)
    by simp
  moreover note ‹c∈M[G]› ‹x∈M[G]›
  moreover from calculation
  have sats(M[G],Member(0,1),[x,c])
    by simp
  moreover
  have σ∈M
  proof -
    from ‹σ∈domain(τ)›
    obtain p where <σ,p> ∈ τ
      by auto
    with ‹τ∈M›
    show ?thesis
      using name_components_in_M by blast
  qed
  moreover
  note ‹χ ∈ M›
  ultimately
  obtain p where p∈G (p ⊩ Member(0,1) [σ,χ])
    using generic truth_lemma[of Member(0,1) G [σ,χ] ] nat_simp_union
    by auto
  moreover from ‹p∈G›
  have p∈P
    using generic by blast
  ultimately
  have <σ,p>∈?ϑ
    using ‹σ∈domain(τ)› by simp
  with ‹val(P,G,σ) =  x› ‹p∈G›
  show x∈val(P,G,?ϑ)
    using val_of_elem [of _ _ ?ϑ] by auto
qed
with ‹val(P,G,?ϑ) ∈ ?b›
show c∈?b by simp
```

```
  qed
  then
  have Pow(a) ∩ M[G] = {x∈?b . x⊆a ∧ x∈M[G]}
    by auto
  also from ‹a∈M[G]›
  have ... = {x∈?b . (M[G], [x,a] ⊨ subset_fm(0,1)) ∧ x∈M[G]}
    using Transset_MG by force
  also
  have ... = {x∈?b . (M[G], [x,a] ⊨ subset_fm(0,1))} ∩ M[G]
    by auto
  also from ‹?b∈M[G]›
  have ... = {x∈?b . (M[G], [x,a] ⊨ subset_fm(0,1))}
    using Collect_inter_Transset Transset_MG
    by simp
  also from ‹?b∈M[G]› ‹a∈M[G]›
  have ... ∈ M[G]
    using Collect_sats_in_MG GenExtI nat_simp_union by simp
  finally show ?thesis .
qed
end


context G_generic begin

interpretation mgtriv: M_trivial ##M[G]
  using generic Union_MG pairing_in_MG zero_in_MG transitivity_MG
  unfolding M_trivial_def M_trans_def M_trivial_axioms_def by (simp; blast)


theorem power_in_MG : power_ax(##(M[G]))
  unfolding power_ax_def
proof (intro rallI, simp only:setclass_iff rex_setclass_is_bex)

  fix a
  assume a ∈ M[G]
  then
  have (##M[G])(a) by simp
  have {x∈Pow(a) . x ∈ M[G]} = Pow(a) ∩ M[G]
    by auto
  also from ‹a∈M[G]›
  have ... ∈ M[G]
    using Pow_inter_MG by simp
  finally
  have {x∈Pow(a) . x ∈ M[G]} ∈ M[G] .
  moreover from ‹a∈M[G]› ‹{x∈Pow(a) . x ∈ M[G]} ∈ _›
  have powerset(##M[G], a, {x∈Pow(a) . x ∈ M[G]})
    using mgtriv.powerset_abs[OF ‹(##M[G])(a)›]
    by simp
  ultimately
```

show ∃ x∈$M[G]$. *powerset*(##$M[G]$, a, x)
by *auto*
qed
end
end

25 The Axiom of Extensionality in $M[G]$

theory *Extensionality_Axiom*
imports
Names
begin

context *forcing_data*
begin

lemma *extensionality_in_MG* : *extensionality*(##($M[G]$))
proof -
{
fix x y z
assume
asms: $x \in M[G]$ $y \in M[G]$ $(\forall w \in M[G] . w \in x \longleftrightarrow w \in y)$
from ⟨$x \in M[G]$⟩ **have**
$z \in x \longleftrightarrow z \in M[G] \wedge z \in x$
using *transitivity_MG* **by** *auto*
also have
... $\longleftrightarrow z \in y$
using *asms transitivity_MG* **by** *auto*
finally have
$z \in x \longleftrightarrow z \in y$ **.**
}
then have
$\forall x \in M[G] . \forall y \in M[G] . (\forall z \in M[G] . z \in x \longleftrightarrow z \in y) \longrightarrow x = y$
by *blast*
then show *?thesis* **unfolding** *extensionality_def* **by** *simp*
qed

end
end

26 The Axiom of Foundation in $M[G]$

theory *Foundation_Axiom*
imports
Names
begin

context *forcing_data*

begin

lemma *foundation_in_MG* : *foundation_ax*(##(*M*[*G*]))
unfolding *foundation_ax_def*
by (*rule rallI*, *cut_tac A*=*x* **in** *foundation*, *auto intro*: *transitivity_MG*)

lemma *foundation_ax*(##(*M*[*G*]))
proof -
{
fix *x*
assume $x \in M[G]$ $\exists\, y \in M[G]\ .\ y \in x$
then
have $\exists\, y \in M[G]\ .\ y \in x \cap M[G]$ **by** *simp*
then
obtain *y* **where** $y \in x \cap M[G]$ $\forall\, z \in y.\ z \notin x \cap M[G]$
using *foundation*[*of* $x \cap M[G]$] **by** *blast*
then
have $\exists\, y \in M[G]\ .\ y \in x \wedge (\forall\, z \in M[G]\ .\ z \notin x \vee z \notin y)$**by** *auto*
}
then show *?thesis*
unfolding *foundation_ax_def* **by** *auto*
qed

end
end

27 The binder *Least*

theory *Least*
imports
Forcing_Data — only for a result to be moved below
Internalizations

begin

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

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

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

show *?thesis* **using** *Least_Ord* **by** *simp*
qed

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

definition
least_fm :: $[i,i] \Rightarrow i$ **where**
least_fm$(q,i) \equiv$ *And*(*ordinal_fm*(i),
Or(*And*(*empty_fm*(i),*Forall*(*Implies*(*ordinal_fm*(0),*Neg*(q)))),
And(*Exists*(*And*(q,*Equal*$(0$,*succ*$(i))$)),
Forall(*Implies*(*And*(*ordinal_fm*(0),*Member*$(0$,*succ*$(i))$),*Neg*(q))))))

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

lemmas *basic_fm_simps* = *sats_subset_fm′ sats_transset_fm′ sats_ordinal_fm′*

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

lemma *least_iff_sats*:
assumes *is_Q_iff_sats*:
$\bigwedge a.\ a \in A \Longrightarrow is\_Q(a) \longleftrightarrow sats(A,\ q,\ Cons(a,env))$
shows
$[\![nth(j,env) = y;\ j \in nat;\ env \in list(A);\ 0{\in}A]\!]$
$\Longrightarrow least(\#\#A,\ is\_Q,\ y) \longleftrightarrow sats(A,\ least\_fm(q,j),\ env)$
using *sats_least_fm* [*OF is_Q_iff_sats*, *of j* , *symmetric*]
by *simp*

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

— FIXME: Better to have this in *M_basic* or similar. And perhaps to have it disciplined
lemma (**in** *M_ctm*) *unique_least*: $a{\in}M \Longrightarrow b{\in}M \Longrightarrow least(\#\#M,Q,a) \Longrightarrow least(\#\#M,Q,b) \Longrightarrow a{=}b$

unfolding *least_def*
by (*auto*, *erule_tac* $i{=}a$ **and** $j{=}b$ **in** *Ord_linear_lt*; (*drule ltD* | *simp*); *auto intro*:*Ord_in_Ord*)

context *M_trivial*
begin

27.1 Absoluteness and closure under *Least*

lemma *least_abs*:
assumes $\bigwedge x.\ Q(x) \Longrightarrow Ord(x) \Longrightarrow \exists y[M].\ Q(y) \wedge Ord(y)$ $M(a)$
shows $least(M,Q,a) \longleftrightarrow a = (\mu\ x.\ Q(x))$
unfolding *least_def*
proof (*cases* $\forall b[M].\ Ord(b) \longrightarrow \neg\ Q(b)$; *intro iffI*; *simp add*:*assms*)
case *True*
with *assms*
have $\neg\ (\exists i.\ Ord(i) \wedge Q(i))$ **by** *blast*
then
show $0 = (\mu\ x.\ Q(x))$ **using** *Least_0* **by** *simp*
then
show $ordinal(M, \mu\ x.\ Q(x)) \wedge (empty(M, Least(Q)) \vee Q(Least(Q)))$
by *simp*
next
assume $\exists b[M].\ Ord(b) \wedge Q(b)$
then
obtain *i* **where** $M(i)$ $Ord(i)$ $Q(i)$ **by** *blast*
assume $a = (\mu\ x.\ Q(x))$
moreover
note ‹$M(a)$›
moreover from ‹$Q(i)$› ‹$Ord(i)$›
have $Q(\mu\ x.\ Q(x))$ (**is** *?G*)
by (*blast intro*:*LeastI*)
moreover
have $(\forall b[M].\ Ord(b) \wedge b \in (\mu\ x.\ Q(x)) \longrightarrow \neg\ Q(b))$ (**is** *?H*)
using *less_LeastE*[*of Q _ False*]
by (*auto*, *drule_tac ltI*, *simp*, *blast*)
ultimately
show $ordinal(M, \mu\ x.\ Q(x)) \wedge (empty(M, \mu\ x.\ Q(x)) \wedge (\forall b[M].\ Ord(b) \longrightarrow \neg\ Q(b)) \vee ?G \wedge ?H)$
by *simp*
next
assume *1*:$\exists b[M].\ Ord(b) \wedge Q(b)$
then
obtain *i* **where** $M(i)$ $Ord(i)$ $Q(i)$ **by** *blast*
assume $Ord(a) \wedge (a = 0 \wedge (\forall b[M].\ Ord(b) \longrightarrow \neg\ Q(b)) \vee Q(a) \wedge (\forall b[M].\ Ord(b) \wedge b \in a \longrightarrow \neg\ Q(b)))$
with *1*
have $Ord(a)$ $Q(a)$ $\forall b[M].\ Ord(b) \wedge b \in a \longrightarrow \neg\ Q(b)$
by *blast+*

moreover from *this* **and** *assms*
have $Ord(b) \Longrightarrow b \in a \Longrightarrow \neg\ Q(b)$ **for** b
by (*auto dest*:*transM*)
moreover from *this* **and** ‹$Ord(a)$›
have $b < a \Longrightarrow \neg\ Q(b)$ **for** b
unfolding *lt_def* **using** *Ord_in_Ord* **by** *blast*
ultimately
show $a = (\mu\ x.\ Q(x))$
using *Least_equality* **by** *simp*
qed

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

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

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

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

end

end

28 The Axiom of Replacement in $M[G]$

theory *Replacement_Axiom*
imports
Least Relative_Univ Separation_Axiom Renaming_Auto
begin

rename *renrep1* **src** [p,P,*leq*,o,ϱ,τ] **tgt** [V,τ,ϱ,p,α,P,*leq*,o]

definition *renrep_fn* :: $i \Rightarrow i$ **where**
renrep_fn(*env*) $\equiv$ *sum*(*renrep1_fn*,*id*(*length*(*env*)),*6*,*8*,*length*(*env*))

definition
renrep :: $[i,i] \Rightarrow i$ **where**
renrep(φ,*env*) = *ren*(φ)‘(*6*#+*length*(*env*))‘(*8*#+*length*(*env*))‘*renrep_fn*(*env*)

lemma *renrep_type* [*TC*]:

assumes φ∈*formula* *env* ∈ *list*(M)
shows *renrep*(φ,*env*) ∈ *formula*
unfolding *renrep_def renrep_fn_def renrep1_fn_def*
using *assms renrep1_thm*(*1*) *ren_tc*
by *simp*

lemma *arity_renrep*:
assumes φ∈*formula* *arity*(φ)≤ *6*#+*length*(*env*) *env* ∈ *list*(M)
shows *arity*(*renrep*(φ,*env*)) ≤ *8*#+*length*(*env*)
unfolding *renrep_def renrep_fn_def renrep1_fn_def*
using *assms renrep1_thm*(*1*) *arity_ren*
by *simp*

lemma *renrep_sats* :
assumes *arity*(φ) ≤ *6* #+ *length*(*env*)
[P,*leq*,o,p,ϱ,τ] @ *env* ∈ *list*(M)
V ∈ M α ∈ M
φ∈*formula*
shows *sats*(M, φ, [p,P,*leq*,o,ϱ,τ] @ *env*) ⟷ *sats*(M, *renrep*(φ,*env*), [V,τ,ϱ,p,α,P,*leq*,o] @ *env*)
unfolding *renrep_def renrep_fn_def renrep1_fn_def*
by (*rule sats_iff_sats_ren*,*insert assms*, *auto simp add*:*renrep1_thm*(*1*)[*of* _ M,*simplified*] *renrep1_thm*(*2*)[*simplified*,**where** p=p **and** α=α])

rename *renpbdy1* **src** [ϱ,p,α,P,*leq*,o] **tgt** [ϱ,p,x,α,P,*leq*,o]

definition *renpbdy_fn* :: i ⇒ i **where**
renpbdy_fn(*env*) ≡ *sum*(*renpbdy1_fn*,*id*(*length*(*env*)),*6*,*7*,*length*(*env*))

definition
renpbdy :: [i,i] ⇒ i **where**
renpbdy(φ,*env*) = *ren*(φ)‘(*6*#+*length*(*env*))‘(*7*#+*length*(*env*))‘*renpbdy_fn*(*env*)

lemma
renpbdy_type [TC]: φ∈*formula* ⟹ *env*∈*list*(M) ⟹ *renpbdy*(φ,*env*) ∈ *formula*
unfolding *renpbdy_def renpbdy_fn_def renpbdy1_fn_def*
using *renpbdy1_thm*(*1*) *ren_tc*
by *simp*

lemma *arity_renpbdy*: φ∈*formula* ⟹ *arity*(φ) ≤ *6* #+ *length*(*env*) ⟹ *env*∈*list*(M) ⟹ *arity*(*renpbdy*(φ,*env*)) ≤ *7* #+ *length*(*env*)
unfolding *renpbdy_def renpbdy_fn_def renpbdy1_fn_def*
using *renpbdy1_thm*(*1*) *arity_ren*
by *simp*

lemma
sats_renpbdy: *arity*(φ) ≤ *6* #+ *length*(*nenv*) ⟹ [ϱ,p,x,α,P,*leq*,o,π] @ *nenv* ∈ *list*(M) ⟹ φ∈*formula* ⟹

$sats(M, \varphi, [\varrho,p,\alpha,P,leq,o] @ nenv) \longleftrightarrow sats(M, renpbdy(\varphi,nenv), [\varrho,p,x,\alpha,P,leq,o] @ nenv)$
unfolding *renpbdy_def renpbdy_fn_def renpbdy1_fn_def*
by (*rule sats_iff_sats_ren*,*auto simp add*: *renpbdy1_thm*(*1*)[*of _ M*,*simplified*] *renpbdy1_thm*(*2*)[*simplified*,**where** α=α **and** *x*=*x*])

rename *renbody1* **src** [*x*,α,*P*,*leq*,*o*] **tgt** [α,*x*,*m*,*P*,*leq*,*o*]

definition *renbody_fn* :: *i* $\Rightarrow$ *i* **where**
renbody_fn(*env*) $\equiv$ *sum*(*renbody1_fn*,*id*(*length*(*env*)),*5*,*6*,*length*(*env*))

definition
renbody :: [*i*,*i*] $\Rightarrow$ *i* **where**
renbody(φ,*env*) = *ren*(φ)‘(*5*#+*length*(*env*))‘(*6*#+*length*(*env*))‘*renbody_fn*(*env*)

lemma
renbody_type [*TC*]: φ∈*formula* $\Longrightarrow$ *env*∈*list*(*M*) $\Longrightarrow$ *renbody*(φ,*env*) ∈ *formula*
unfolding *renbody_def renbody_fn_def renbody1_fn_def*
using *renbody1_thm*(*1*) *ren_tc*
by *simp*

lemma *arity_renbody*: φ∈*formula* $\Longrightarrow$ *arity*(φ) ≤ *5* #+ *length*(*env*) $\Longrightarrow$ *env*∈*list*(*M*) $\Longrightarrow$
arity(*renbody*(φ,*env*)) ≤ *6* #+ *length*(*env*)
unfolding *renbody_def renbody_fn_def renbody1_fn_def*
using *renbody1_thm*(*1*) *arity_ren*
by *simp*

lemma
sats_renbody: *arity*(φ) ≤ *5* #+ *length*(*nenv*) $\Longrightarrow$ [α,*x*,*m*,*P*,*leq*,*o*] @ *nenv* ∈ *list*(*M*) $\Longrightarrow$ φ∈*formula* $\Longrightarrow$
$sats(M, \varphi, [x,\alpha,P,leq,o] @ nenv) \longleftrightarrow sats(M, renbody(\varphi,nenv), [\alpha,x,m,P,leq,o] @ nenv)$
unfolding *renbody_def renbody_fn_def renbody1_fn_def*
by (*rule sats_iff_sats_ren*, *auto simp add*:*renbody1_thm*(*1*)[*of _ M*,*simplified*] *renbody1_thm*(*2*)[**where** α=α **and** *m*=*m*,*simplified*])

context *G_generic*
begin

lemma *pow_inter_M*:
assumes
x∈*M* *y*∈*M*
shows
powerset(##*M*,*x*,*y*) $\longleftrightarrow$ *y* = *Pow*(*x*) ∩ *M*
using *assms* **by** *auto*

```
schematic_goal sats_prebody_fm_auto:
  assumes
   φ∈formula [P,leq,one,p,ϱ,π] @ nenv ∈list(M)  α∈M arity(φ) ≤ 2 #+ length(nenv)
  shows
   (∃τ∈M. ∃ V∈M. is_Vset(##M,α,V) ∧ τ∈V ∧ sats(M,forces(φ),[p,P,leq,one,ϱ,τ]
@ nenv))
   ⟷ sats(M,?prebody_fm,[ϱ,p,α,P,leq,one] @ nenv)
 apply (insert assms; (rule sep_rules is_Vset_iff_sats[OF _ _ _ _ _ nonempty[simplified]]
| simp))
  apply (rule sep_rules is_Vset_iff_sats is_Vset_iff_sats[OF _ _ _ _ _ nonempty[simplified]]
| simp)+
      apply (rule nonempty[simplified])
     apply (simp_all)
    apply (rule length_type[THEN nat_into_Ord], blast)+
  apply ((rule sep_rules | simp))
   apply ((rule sep_rules | simp))
    apply ((rule sep_rules | simp))
     apply ((rule sep_rules | simp))
    apply ((rule sep_rules | simp))
   apply ((rule sep_rules | simp))
  apply ((rule sep_rules | simp))
  apply (rule renrep_sats[simplified])
     apply (insert assms)
     apply(auto simp add: renrep_type definability)
proof -
  from assms
  have nenv∈list(M) by simp
  with ‹arity(φ)≤_› ‹φ∈_›
  show arity(forces(φ)) ≤ succ(succ(succ(succ(succ(succ(length(nenv)))))))
   using arity_forces_le by simp
qed


synthesize_notc prebody_fm from_schematic sats_prebody_fm_auto

lemma prebody_fm_type [TC]:
  assumes φ∈formula
   env ∈ list(M)
  shows prebody_fm(φ,env)∈formula
proof -
  from ‹φ∈formula›
  have forces(φ)∈formula by simp
  then
  have renrep(forces(φ),env)∈formula
   using ‹env∈list(M)› by simp
  then show ?thesis unfolding prebody_fm_def by simp
qed
```

declare *is_eclose_fm_def[fm_definitions]*
is_eclose_fm_def[fm_definitions]
mem_eclose_fm_def[fm_definitions]
eclose_n_fm_def[fm_definitions]

lemma *sats_prebody_fm*:
assumes
$[P,leq,one,p,\varrho]$ @ $nenv \in list(M)$ $\varphi \in formula$ $\alpha \in M$ $arity(\varphi) \leq 2$ #+ $length(nenv)$
shows
$sats(M,prebody\_fm(\varphi,nenv),[\varrho,p,\alpha,P,leq,one]$ @ $nenv) \longleftrightarrow$
$(\exists \tau \in M.\ \exists V \in M.\ is\_Vset(\#\#M,\alpha,V) \wedge \tau \in V \wedge sats(M,forces(\varphi),[p,P,leq,one,\varrho,\tau]$
@ $nenv))$
unfolding *prebody_fm_def* **using** *assms sats_prebody_fm_auto* **by** *force*

lemma *arity_prebody_fm*:
assumes
$\varphi \in formula$ $\alpha \in M$ $env \in list(M)$ $arity(\varphi) \leq 2$ #+ $length(env)$
shows
$arity(prebody\_fm(\varphi,env)) \leq 6$ #+ $length(env)$
unfolding *prebody_fm_def is_HVfrom_fm_def is_powapply_fm_def*
using *assms fm_definitions nat_simp_union*
arity_renrep[of forces(φ)] arity_forces_le[simplified] pred_le **by** *auto*

definition
body_fm' :: $[i,i] \Rightarrow i$ **where**
$body\_fm'(\varphi,env) \equiv Exists(Exists(And(pair\_fm(0,1,2),renpbdy(prebody\_fm(\varphi,env),env))))$

lemma *body_fm'_type[TC]*: $\varphi \in formula \Longrightarrow env \in list(M) \Longrightarrow body\_fm'(\varphi,env) \in formula$
unfolding *body_fm'_def* **using** *prebody_fm_type*
by *simp*

lemma *arity_body_fm'*:
assumes
$\varphi \in formula$ $\alpha \in M$ $env \in list(M)$ $arity(\varphi) \leq 2$ #+ $length(env)$
shows
$arity(body\_fm'(\varphi,env)) \leq 5$ #+ $length(env)$
unfolding *body_fm'_def*
using *assms fm_definitions nat_simp_union arity_prebody_fm pred_le arity_renpbdy[of*
prebody_fm(φ,env)]
by *auto*

lemma *sats_body_fm'*:
assumes
$\exists t\ p.\ x=\langle t,p \rangle$ $x \in M$ $[\alpha,P,leq,one,p,\varrho]$ @ $nenv \in list(M)$ $\varphi \in formula$ $arity(\varphi) \leq$
2 #+ $length(nenv)$
shows
$sats(M,body\_fm'(\varphi,nenv),[x,\alpha,P,leq,one]$ @ $nenv) \longleftrightarrow$

$sats(M,renpbdy(prebody\_fm(\varphi,nenv),nenv),[fst(x),snd(x),x,\alpha,P,leq,one] @ nenv)$
using *assms fst_snd_closed*[*OF* ‹x∈M›] **unfolding** *body_fm′_def*
by (*auto*)

definition
body_fm :: [*i*,*i*]⇒*i* **where**
$body\_fm(\varphi,env) \equiv renbody(body\_fm'(\varphi,env),env)$

lemma *body_fm_type* [*TC*]: $env \in list(M) \Longrightarrow \varphi \in formula \Longrightarrow body\_fm(\varphi,env) \in formula$
unfolding *body_fm_def* **by** *simp*

lemma *sats_body_fm*:
assumes
$\exists t\ p.\ x=\langle t,p\rangle$ $[\alpha,x,m,P,leq,one]$ @ $nenv \in list(M)$
$\varphi \in formula$ $arity(\varphi) \le 2$ #+ $length(nenv)$
shows
$sats(M,body\_fm(\varphi,nenv),[\alpha,x,m,P,leq,one]$ @ $nenv) \longleftrightarrow$
$sats(M,renpbdy(prebody\_fm(\varphi,nenv),nenv),[fst(x),snd(x),x,\alpha,P,leq,one]$ @ $nenv)$
using *assms sats_body_fm′ sats_renbody*[*OF* _ *assms*(2), *symmetric*] *arity_body_fm′*
unfolding *body_fm_def*
by *auto*

lemma *sats_renpbdy_prebody_fm*:
assumes
$\exists t\ p.\ x=\langle t,p\rangle$ $x \in M$ $[\alpha,m,P,leq,one]$ @ $nenv \in list(M)$
$\varphi \in formula$ $arity(\varphi) \le 2$ #+ $length(nenv)$
shows
$sats(M,renpbdy(prebody\_fm(\varphi,nenv),nenv),[fst(x),snd(x),x,\alpha,P,leq,one]$ @ $nenv)$
$\longleftrightarrow$
$sats(M,prebody\_fm(\varphi,nenv),[fst(x),snd(x),\alpha,P,leq,one]$ @ $nenv)$
using *assms fst_snd_closed*[*OF* ‹x∈M›]
sats_renpbdy[*OF arity_prebody_fm* _ *prebody_fm_type*, *of concl*:*M*, *symmetric*]
by *force*

lemma *body_lemma*:
assumes
$\exists t\ p.\ x=\langle t,p\rangle$ $x \in M$ $[x,\alpha,m,P,leq,one]$ @ $nenv \in list(M)$
$\varphi \in formula$ $arity(\varphi) \le 2$ #+ $length(nenv)$
shows
$sats(M,body\_fm(\varphi,nenv),[\alpha,x,m,P,leq,one]$ @ $nenv) \longleftrightarrow$
$(\exists \tau \in M.\ \exists V \in M.\ is\_Vset(\lambda a.\ (\#\#M)(a),\alpha,V) \wedge \tau \in V \wedge (snd(x) \Vdash \varphi\ ([fst(x),\tau]@nenv)))$
using *assms sats_body_fm*[*of x α m nenv*] *sats_renpbdy_prebody_fm*[*of x α*]
sats_prebody_fm[*of snd*(*x*) *fst*(*x*)] *fst_snd_closed*[*OF* ‹x∈M›]
by (*simp*, *simp flip*: *setclass_iff*,*simp*)

lemma *Replace_sats_in_MG*:
assumes
$c \in M[G]$ $env \in list(M[G])$
$\varphi \in formula$ $arity(\varphi) \le 2$ #+ $length(env)$

univalent(##*M*[*G*], *c*, λ*x* *v*. (*M*[*G*] , [*x*,*v*]@*env* ⊨ φ))
shows
{*v*. *x*∈*c*, *v*∈*M*[*G*] ∧ (*M*[*G*] , [*x*,*v*]@*env* ⊨ φ)} ∈ *M*[*G*]
proof -
let *?R* = λ *x* *v* . *v*∈*M*[*G*] ∧ (*M*[*G*] , [*x*,*v*]@*env* ⊨ φ)
from ‹*c*∈*M*[*G*]›
obtain π′ **where** *val*(*P*,*G*, π′) = *c* π′ ∈ *M*
using *GenExt_def* **by** *auto*
then
have *domain*(π′)×*P*∈*M* (**is** *?π*∈*M*)
using *cartprod_closed P_in_M domain_closed* **by** *simp*
from ‹*val*(*P*,*G*, π′) = *c*›
have *c* ⊆ *val*(*P*,*G*,*?π*)
using *def_val*[*of G ?π*] *one_in_P one_in_G*[*OF generic*] *elem_of_val domain_of_prod*[*OF one_in_P*, *of domain*(π′)] **by** *force*
from ‹*env* ∈ _›
obtain *nenv* **where** *nenv*∈*list*(*M*) *env* = *map*(*val*(*P*,*G*),*nenv*)
using *map_val* **by** *auto*
then
have *length*(*nenv*) = *length*(*env*) **by** *simp*
define *f* **where** *f*(ϱ*p*) ≡ μ α. α∈*M* ∧ (∃τ∈*M*. τ ∈ *Vset*(α) ∧ (*snd*(ϱ*p*) ⊩ φ ([*fst*(ϱ*p*),τ] @ *nenv*))) (**is** _ ≡ μ α. *?P*(ϱ*p*,α)) **for** ϱ*p*
have *f*(ϱ*p*) = (μ α. α∈*M* ∧ (∃τ∈*M*. ∃ *V*∈*M*. *is_Vset*(##*M*,α,*V*) ∧ τ∈*V* ∧ (*snd*(ϱ*p*) ⊩ φ ([*fst*(ϱ*p*),τ] @ *nenv*)))) (**is** _ = (μ α. α∈*M* ∧ *?Q*(ϱ*p*,α))) **for** ϱ*p*
unfolding *f_def* **using** *Vset_abs Vset_closed Ord_Least_cong*[*of ?P*(ϱ*p*) λ α. α∈*M* ∧ *?Q*(ϱ*p*,α)]
by (*simp*, *simp del*:*setclass_iff*)
moreover
have *f*(ϱ*p*) ∈ *M* **for** ϱ*p*
unfolding *f_def* **using** *Least_closed′*[*of ?P*(ϱ*p*)] **by** *simp*
ultimately
have *1*:*least*(##*M*,λα. *?Q*(ϱ*p*,α),*f*(ϱ*p*)) **for** ϱ*p*
using *least_abs′*[*of* λα. α∈*M* ∧ *?Q*(ϱ*p*,α) *f*(ϱ*p*)] *least_conj*
by (*simp flip*: *setclass_iff*)
have *Ord*(*f*(ϱ*p*)) **for** ϱ*p* **unfolding** *f_def* **by** *simp*
define *QQ* **where** *QQ*≡*?Q*
from *1*
have *least*(##*M*,λα. *QQ*(ϱ*p*,α),*f*(ϱ*p*)) **for** ϱ*p*
unfolding *QQ_def* **.**
from ‹*arity*(φ) ≤ _› ‹*length*(*nenv*) = _›
have *arity*(φ) ≤ *2* #+ *length*(*nenv*)
by *simp*
moreover
note *assms* ‹*nenv*∈*list*(*M*)› ‹*?π*∈*M*›
moreover
have ϱ*p*∈*?π* ⟹ ∃ *t* *p*. ϱ*p*=⟨*t*,*p*⟩ **for** ϱ*p*
by *auto*
ultimately

```
  have body:M , [α,ϱp,m,P,leq,one] @ nenv ⊨ body_fm(φ,nenv) ⟷ ?Q(ϱp,α)
    if ϱp∈?π ϱp∈M m∈M α∈M for α ϱp m
    using that P_in_M leq_in_M one_in_M body_lemma[of ϱp α m nenv φ] by simp
  let ?f_fm=least_fm(body_fm(φ,nenv),1)
  {
    fix ϱp m
    assume asm: ϱp∈M ϱp∈?π m∈M
    note inM = this P_in_M leq_in_M one_in_M ‹nenv∈list(M)›
    with body
     have body':⋀α. α ∈ M ⟹ (∃τ∈M. ∃V∈M. is_Vset(λa. (##M)(a), α, V)
∧ τ ∈ V ∧
          (snd(ϱp) ⊩ φ ([fst(ϱp),τ] @ nenv))) ⟷
          M, Cons(α, [ϱp, m, P, leq, one] @ nenv) ⊨ body_fm(φ,nenv) by simp
    from inM
    have M , [ϱp,m,P,leq,one] @ nenv ⊨ ?f_fm ⟷ least(##M, QQ(ϱp), m)
      using sats_least_fm[OF body', of 1] unfolding QQ_def
      by (simp, simp flip: setclass_iff)
  }
  then
  have M, [ϱp,m,P,leq,one] @ nenv ⊨ ?f_fm ⟷ least(##M, QQ(ϱp), m)
    if ϱp∈M ϱp∈?π m∈M for ϱp m using that by simp
  then
  have univalent(##M, ?π, λϱp m. M , [ϱp,m] @ ([P,leq,one] @ nenv) ⊨ ?f_fm)
    unfolding univalent_def by (auto intro:unique_least)
  moreover from ‹length(_) = _› ‹env ∈ _›
  have length([P,leq,one] @ nenv) = 3 #+ length(env) by simp
  moreover from ‹arity(_) ≤ 2 #+ length(nenv)›
    ‹length(_) = length(_)›[symmetric] ‹nenv∈_› ‹φ∈_›
  have arity(?f_fm) ≤ 5 #+ length(env)
    unfolding body_fm_def fm_definitions least_fm_def
    using arity_forces arity_renrep arity_renbody arity_body_fm' nonempty
   by (simp add: pred_Un Un_assoc, simp add: Un_assoc[symmetric] nat_union_abs1
pred_Un)
      (auto simp add: nat_simp_union, rule pred_le, auto intro:leI)
  moreover from ‹φ∈formula› ‹nenv∈list(M)›
  have ?f_fm∈formula by simp
  moreover
  note inM = P_in_M leq_in_M one_in_M ‹nenv∈list(M)› ‹?π∈M›
  ultimately
  obtain Y where Y∈M
    ∀m∈M. m ∈ Y ⟷ (∃ϱp∈M. ϱp ∈ ?π ∧ M, [ϱp,m] @ ([P,leq,one] @ nenv)
⊨ ?f_fm)
    using replacement_ax[of ?f_fm [P,leq,one] @ nenv]
    unfolding strong_replacement_def by auto
  with ‹least(_,QQ(_),f(_))› ‹f(_) ∈ M› ‹?π∈M›
    ‹_ ⟹ _ ⟹ _ ⟹ M,_ ⊨ ?f_fm ⟷ least(_,_,_)›
  have f(ϱp)∈Y if ϱp∈?π for ϱp
    using that transitivity[OF _ ‹?π∈M›]
    by (clarsimp, rule_tac x=⟨x,y⟩ in bexI, auto)
```

```
moreover
have {y∈Y. Ord(y)} ∈ M
  using ‹Y∈M› separation_ax sats_ordinal_fm trans_M
    separation_cong[of ##M λy. sats(M,ordinal_fm(0),[y]) Ord]
    separation_closed by simp
then
have ⋃ {y∈Y. Ord(y)} ∈ M (is ?sup ∈ M)
  using Union_closed by simp
then
have {x∈Vset(?sup). x ∈ M} ∈ M
  using Vset_closed by simp
moreover
have {one} ∈ M
  using one_in_M singletonM by simp
ultimately
have {x∈Vset(?sup). x ∈ M} × {one} ∈ M (is ?big_name ∈ M)
  using cartprod_closed by simp
then
have val(P,G,?big_name) ∈ M[G]
  by (blast intro:GenExtI)
{
  fix v x
  assume x∈c
  moreover
  note ‹val(P,G,π')=c› ‹π'∈M›
  moreover
  from calculation
  obtain ϱ p where ⟨ϱ,p⟩∈π'  val(P,G,ϱ) = x p∈G ϱ∈M
    using elem_of_val_pair'[of π' x G] by blast
  moreover
  assume v∈M[G]
  then
  obtain σ where val(P,G,σ) = v σ∈M
    using GenExtD by auto
  moreover
  assume sats(M[G], φ, [x,v] @ env)
  moreover
  note ‹φ∈_› ‹nenv∈_› ‹env = _› ‹arity(φ)≤ 2 #+ length(env)›
  ultimately
  obtain q where q∈G q ⊩ φ ([ϱ,σ]@nenv)
    using truth_lemma[OF ‹φ∈_› generic, symmetric, of [ϱ,σ] @ nenv]
    by auto
  with ‹⟨ϱ,p⟩∈π'› ‹⟨ϱ,q⟩∈?π ⟹ f(⟨ϱ,q⟩)∈Y›
  have f(⟨ϱ,q⟩)∈Y
    using generic unfolding M_generic_def filter_def by blast
  let ?α=succ(rank(σ))
  note ‹σ∈M›
  moreover from this
  have ?α ∈ M
```

using *rank_closed cons_closed* **by** (*simp flip*: *setclass_iff*)
moreover
have $\sigma \in$ *Vset*(*?*α)
using *Vset_Ord_rank_iff* **by** *auto*
moreover
note ‹$q \Vdash \varphi$ ([ϱ,σ] @ *nenv*)›
ultimately
have *?P*(⟨ϱ,q⟩,*?*α) **by** (*auto simp del*: *Vset_rank_iff*)
moreover
have (μ α. *?P*(⟨ϱ,q⟩,α)) = f(⟨ϱ,q⟩)
unfolding *f_def* **by** *simp*
ultimately
obtain τ **where** $\tau\in M$ $\tau \in$ *Vset*(f(⟨ϱ,q⟩)) $q \Vdash \varphi$ ([ϱ,τ] @ *nenv*)
using *LeastI*[*of* λ α. *?P*(⟨ϱ,q⟩,α) *?*α] **by** *auto*
with ‹$q\in G$› ‹$\varrho\in M$› ‹*nenv*∈_› ‹*arity*(φ)≤ *2* #+ *length*(*nenv*)›
have $M[G]$, *map*(*val*(P,G),[ϱ,τ] @ *nenv*) $\models \varphi$
using *truth_lemma*[*OF* ‹φ∈_› *generic*, *of* [ϱ,τ] @ *nenv*] **by** *auto*
moreover from ‹$x\in c$› ‹$c\in M[G]$›
have $x\in M[G]$ **using** *transitivity_MG* **by** *simp*
moreover
note ‹$M[G]$,[x,v] @ *env*$\models \varphi$› ‹*env* = *map*(*val*(P,G),*nenv*)› ‹$\tau\in M$› ‹*val*(P,G,ϱ)=x›
‹*univalent*(##$M[G]$,_,_)› ‹$x\in c$› ‹$v\in M[G]$›
ultimately
have v=*val*(P,G,τ)
using *GenExtI*[*of* τ G] **unfolding** *univalent_def* **by** (*auto*)
from ‹$\tau \in$ *Vset*(f(⟨ϱ,q⟩))› ‹*Ord*(f(_))› ‹f(⟨ϱ,q⟩)∈Y›
have $\tau \in$ *Vset*(*?sup*)
using *Vset_Ord_rank_iff lt_Union_iff*[*of* _ *rank*(τ)] **by** *auto*
with ‹$\tau\in M$›
have *val*(P,G,τ) ∈ *val*(P,G,*?big_name*)
using *domain_of_prod*[*of one* {*one*} {x∈*Vset*(*?sup*). $x \in M$}] *def_val*[*of G*
?big_name]
one_in_G[*OF generic*] *one_in_P* **by** (*auto simp del*: *Vset_rank_iff*)
with ‹v=*val*(P,G,τ)›
have $v \in$ *val*(P,G,{x∈*Vset*(*?sup*). $x \in M$} × {*one*})
by *simp*
}
then
have {v. $x\in c$, *?R*(x,v)} ⊆ *val*(P,G,*?big_name*) (**is** *?repl*⊆*?big*)
by *blast*
with ‹*?big_name*∈M›
have *?repl* = {v∈*?big*. ∃x∈c. *sats*($M[G]$, φ, [x,v] @ *env*)} (**is** _ = *?rhs*)
proof(*intro equalityI subsetI*)
fix v
assume v∈*?repl*
with ‹*?repl*⊆*?big*›
obtain x **where** $x\in c$ $M[G]$, [x, v] @ *env* $\models \varphi$ v∈*?big*
using *subsetD* **by** *auto*
with ‹*univalent*(##$M[G]$,_,_)› ‹$c\in M[G]$›

```
    show v ∈ ?rhs
      unfolding univalent_def
      using transitivity_MG ReplaceI[of λ x v. ∃x∈c. M[G], [x, v] @ env ⊨ φ] by
blast
  next
    fix v
    assume v∈?rhs
    then
    obtain x where
      v∈val(P,G, ?big_name) M[G], [x, v] @ env ⊨ φ x∈c
      by blast
    moreover from this ‹c∈M[G]›
    have v∈M[G] x∈M[G]
      using transitivity_MG GenExtI[OF ‹?big_name∈_›,of G] by auto
    moreover from calculation ‹univalent(##M[G],_,_)›
    have ?R(x,y) ⟹ y = v for y
      unfolding univalent_def by auto
    ultimately
    show v∈?repl
      using ReplaceI[of ?R x v c]
      by blast
  qed
  moreover
  let ?ψ = Exists(And(Member(0,2#+length(env)),φ))
  have v∈M[G] ⟹ (∃x∈c. M[G], [x,v] @ env ⊨ φ) ⟷ M[G], [v] @ env @ [c]
⊨ ?ψ
    arity(?ψ) ≤ 2 #+ length(env) ?ψ∈formula
    for v
  proof -
    fix v
    assume v∈M[G]
    with ‹c∈M[G]›
    have nth(length(env)#+1,[v]@env@[c]) = c
      using ‹env∈_›nth_concat[of v c M[G] env]
      by auto
    note inMG= ‹nth(length(env)#+1,[v]@env@[c]) = c› ‹c∈M[G]› ‹v∈M[G]›
‹env∈_›
    show (∃x∈c. M[G], [x,v] @ env ⊨ φ) ⟷ M[G], [v] @ env @ [c] ⊨ ?ψ
    proof
      assume ∃x∈c. M[G], [x, v] @ env ⊨ φ
      then obtain x where
        x∈c M[G], [x, v] @ env ⊨ φ x∈M[G]
        using transitivity_MG[OF _ ‹c∈M[G]›]
        by auto
      with ‹φ∈_› ‹arity(φ)≤2#+length(env)› inMG
      show M[G], [v] @ env @ [c] ⊨ Exists(And(Member(0, 2 #+ length(env)),
φ))
        using arity_sats_iff[of φ [c] _ [x,v]@env]
        by auto
```

```
    next
     assume M[G], [v] @ env @ [c] ⊨ Exists(And(Member(0, 2 #+ length(env)),
φ))
      with inMG
      obtain x where
        x∈M[G] x∈c M[G], [x,v]@env@[c] ⊨ φ
        by auto
      with ‹φ∈_› ‹arity(φ)≤2#+length(env)› inMG
      show ∃x∈c. M[G], [x, v] @ env⊨ φ
        using arity_sats_iff[of φ [c] _ [x,v]@env]
        by auto
    qed
  next
    from ‹env∈_› ‹φ∈_›
    show arity(?ψ)≤2#+length(env)
      using pred_mono[OF _ ‹arity(φ)≤2#+length(env)›] lt_trans[OF _ le_refl]
      by (auto simp add:nat_simp_union)
  next
    from ‹φ∈_›
    show ?ψ∈formula by simp
  qed
  moreover from this
  have {v∈?big. ∃x∈c. M[G], [x,v] @ env ⊨ φ} = {v∈?big. M[G], [v] @ env @
[c] ⊨  ?ψ}
    using transitivity_MG[OF _ GenExtI, OF _ ‹?big_name∈M›]
    by simp
  moreover from calculation and ‹env∈_› ‹c∈_› ‹?big∈M[G]›
  have {v∈?big. M[G] , [v] @ env @ [c] ⊨ ?ψ} ∈ M[G]
    using Collect_sats_in_MG by auto
  ultimately
  show ?thesis by simp
qed

theorem strong_replacement_in_MG:
  assumes
    φ∈formula and arity(φ) ≤ 2 #+ length(env) env ∈ list(M[G])
  shows
    strong_replacement(##M[G],λx v. sats(M[G],φ,[x,v] @ env))
proof -
  let ?R=λx y . M[G], [x, y] @ env ⊨ φ
  {
    fix A
    let ?Y={v . x ∈ A, v∈M[G] ∧ ?R(x,v)}
    assume 1: (##M[G])(A)
      ∀x[##M[G]]. x ∈ A ⟶ (∀y[##M[G]]. ∀z[##M[G]]. ?R(x,y) ∧ ?R(x,z)
⟶ y = z)
    then
    have univalent(##M[G], A, ?R) A∈M[G]
      unfolding univalent_def by simp_all
```

```
    with assms ‹A∈_›
    have (##M[G])(?Y)
      using Replace_sats_in_MG by auto
    have b ∈ ?Y ⟷ (∃x[##M[G]]. x ∈ A ∧ ?R(x,b)) if (##M[G])(b) for b
    proof(rule)
      from ‹A∈_›
      show ∃x[##M[G]]. x ∈ A ∧ ?R(x,b) if b ∈ ?Y
        using that transitivity_MG by auto
    next
      show b ∈ ?Y if ∃x[##M[G]]. x ∈ A ∧ ?R(x,b)
      proof -
        from ‹(##M[G])(b)›
        have b∈M[G] by simp
        with that
        obtain x where (##M[G])(x) x∈A b∈M[G] ∧ ?R(x,b)
          by blast
        moreover from this 1 ‹(##M[G])(b)›
        have x∈M[G] z∈M[G] ∧ ?R(x,z) ⟹ b = z for z
          by auto
        ultimately
        show ?thesis
          using ReplaceI[of λ x y. y∈M[G] ∧ ?R(x,y)] by auto
      qed
    qed
    then
    have ∀b[##M[G]]. b ∈ ?Y ⟷ (∃x[##M[G]]. x ∈ A ∧ ?R(x,b))
      by simp
    with ‹(##M[G])(?Y)›
     have  (∃Y[##M[G]]. ∀b[##M[G]]. b ∈ Y ⟷ (∃x[##M[G]]. x ∈ A ∧
?R(x,b)))
      by auto
  }
  then show ?thesis unfolding strong_replacement_def univalent_def
    by auto
qed

end

end
```

29 The Axiom of Infinity in $M[G]$

```
theory Infinity_Axiom
  imports Pairing_Axiom Union_Axiom Separation_Axiom
begin

context G_generic begin

interpretation mg_triv: M_trivial##M[G]
```

using *transitivity_MG zero_in_MG generic Union_MG pairing_in_MG*
by *unfold_locales auto*

lemma *infinity_in_MG* : *infinity_ax*(##*M*[*G*])
proof -
from *infinity_ax* **obtain** *I* **where**
Eq1: *I*∈*M 0* ∈ *I* ∀ *y*∈*M*. *y* ∈ *I* ⟶ *succ*(*y*) ∈ *I*
unfolding *infinity_ax_def* **by** *auto*
then
have *check*(*I*) ∈ *M*
using *check_in_M* **by** *simp*
then
have *I*∈ *M*[*G*]
using *valcheck generic one_in_G one_in_P GenExtI*[*of check*(*I*) *G*] **by** *simp*
with ⟨*0*∈*I*⟩
have *0*∈*M*[*G*] **using** *transitivity_MG* **by** *simp*
with ⟨*I*∈*M*⟩
have *y* ∈ *M* **if** *y* ∈ *I* **for** *y*
using *transitivity*[*OF* _ ⟨*I*∈*M*⟩] *that* **by** *simp*
with ⟨*I*∈*M*[*G*]⟩
have *succ*(*y*) ∈ *I* ∩ *M*[*G*] **if** *y* ∈ *I* **for** *y*
using *that Eq1 transitivity_MG* **by** *blast*
with *Eq1* ⟨*I*∈*M*[*G*]⟩ ⟨*0*∈*M*[*G*]⟩
show *?thesis*
unfolding *infinity_ax_def* **by** *auto*
qed

end
end

30 The Axiom of Choice in $M[G]$

theory *Choice_Axiom*
imports *Powerset_Axiom Pairing_Axiom Union_Axiom Extensionality_Axiom Foundation_Axiom Powerset_Axiom Separation_Axiom Replacement_Axiom Interface Infinity_Axiom Relativization*
begin

definition
induced_surj :: *i*⇒*i*⇒*i*⇒*i* **where**
induced_surj(*f*,*a*,*e*) ≡ *f*-‘‘(*range*(*f*)-*a*)×{*e*} ∪ *restrict*(*f*,*f*-‘‘*a*)

lemma *domain_induced_surj*: *domain*(*induced_surj*(*f*,*a*,*e*)) = *domain*(*f*)
unfolding *induced_surj_def* **using** *domain_restrict domain_of_prod* **by** *auto*

lemma *range_restrict_vimage*:
assumes *function*(*f*)
shows *range*(*restrict*(*f*,*f*-‘‘*a*)) ⊆ *a*
proof

from *assms*
have *function*(*restrict*(*f*,*f*-‘‘*a*))
  using *function_restrictI* **by** *simp*
fix *y*
assume $y \in$ *range*(*restrict*(*f*,*f*-‘‘*a*))
then
obtain *x* **where** $\langle x,y\rangle \in$ *restrict*(*f*,*f*-‘‘*a*) $x \in$ *f*-‘‘*a* $x\in$*domain*(*f*)
  using *domain_restrict domainI*[*of* _ _ *restrict*(*f*,*f*-‘‘*a*)] **by** *auto*
moreover
note ‹*function*(*restrict*(*f*,*f*-‘‘*a*))›
ultimately
have *y* = *restrict*(*f*,*f*-‘‘*a*)‘*x*
  using *function_apply_equality* **by** *blast*
also from ‹$x \in$ *f*-‘‘*a*›
have *restrict*(*f*,*f*-‘‘*a*)‘*x* = *f*‘*x*
  by *simp*
finally
have *y*=*f*‘*x* **.**
moreover from *assms* ‹$x\in$*domain*(*f*)›
have $\langle$*x*,*f*‘*x*$\rangle \in f$
  using *function_apply_Pair* **by** *auto*
moreover
note *assms* ‹$x \in$ *f*-‘‘*a*›
ultimately
show $y\in a$
  using *function_image_vimage*[*of f a*] **by** *auto*
qed

lemma *induced_surj_type*:
assumes
  function(*f*)
shows
  induced_surj(*f*,*a*,*e*): *domain*(*f*) $\to \{e\} \cup a$
  and
  $x \in$ *f*-‘‘*a* $\Longrightarrow$ *induced_surj*(*f*,*a*,*e*)‘*x* = *f*‘*x*
proof -
let *?f1*=*f*-‘‘(*range*(*f*)-*a*) $\times \{e\}$ **and** *?f2*=*restrict*(*f*, *f*-‘‘*a*)
have *domain*(*?f2*) = *domain*(*f*) $\cap$ *f*-‘‘*a*
  using *domain_restrict* **by** *simp*
moreover from *assms*
have *1*: *domain*(*?f1*) = *f*-‘‘(*range*(*f*))-*f*-‘‘*a*
  using *domain_of_prod function_vimage_Diff* **by** *simp*
ultimately
have *domain*(*?f1*) $\cap$ *domain*(*?f2*) = *0*
  by *auto*
moreover
have *function*(*?f1*) *relation*(*?f1*) *range*(*?f1*) $\subseteq \{e\}$
  unfolding *function_def relation_def range_def* **by** *auto*
moreover from *this* **and** *assms*

```
  have ?f1: domain(?f1) → range(?f1)
    using function_imp_Pi by simp
  moreover from assms
  have ?f2: domain(?f2) → range(?f2)
    using function_imp_Pi[of restrict(f, f -`` a)] function_restrictI by simp
  moreover from assms
  have range(?f2) ⊆ a
    using range_restrict_vimage by simp
  ultimately
  have induced_surj(f,a,e): domain(?f1) ∪ domain(?f2) → {e} ∪ a
   unfolding induced_surj_def using fun_is_function fun_disjoint_Un fun_weaken_type
by simp
  moreover
  have domain(?f1) ∪ domain(?f2) = domain(f)
    using domain_restrict domain_of_prod by auto
  ultimately
  show induced_surj(f,a,e): domain(f) → {e} ∪ a
    by simp
  assume x ∈ f-``a
  then
  have ?f2`x = f`x
    using restrict by simp
  moreover from ‹x ∈ f-``a› and 1
  have x ∉ domain(?f1)
    by simp
  ultimately
  show induced_surj(f,a,e)`x = f`x
    unfolding induced_surj_def using fun_disjoint_apply2[of x ?f1 ?f2] by simp
qed

lemma induced_surj_is_surj :
  assumes
    e∈a function(f) domain(f) = α ⋀y. y ∈ a ⟹ ∃x∈α. f ` x = y
  shows
    induced_surj(f,a,e) ∈ surj(α,a)
  unfolding surj_def
proof (intro CollectI ballI)
  from assms
  show induced_surj(f,a,e): α → a
    using induced_surj_type[of f a e] cons_eq cons_absorb by simp
  fix y
  assume y ∈ a
  with assms
  have ∃x∈α. f ` x = y
    by simp
  then
  obtain x where x∈α f ` x = y by auto
  with ‹y∈a› assms
  have x∈f-``a
```

using *vimage_iff function_apply_Pair*[*of f x*] **by** *auto*
with ‹*f* ‘ *x* = *y*› *assms*
have *induced_surj*(*f*, *a*, *e*) ‘ *x* = *y*
using *induced_surj_type* **by** *simp*
with ‹*x*∈α› **show**
∃ *x*∈α. *induced_surj*(*f*, *a*, *e*) ‘ *x* = *y* **by** *auto*
qed

context *G_generic*
begin

definition
upair_name :: *i* ⇒ *i* ⇒ *i* **where**
upair_name(τ,ϱ) ≡ *Upair*(⟨τ,*one*⟩,⟨ϱ,*one*⟩)

lemma *Upair_simp* : *Upair*(*a*,*b*) = {*a*,*b*}
by *auto*

relativize *upair_name is_upair_name*

lemma *upair_name_abs* :
assumes *x*∈*M* *y*∈*M* *z*∈*M*
shows *is_upair_name*(##*M*,*x*,*y*,*z*) ⟷ *z* = *upair_name*(*x*,*y*)
unfolding *is_upair_name_def upair_name_def*
using *assms zero_in_M one_in_M empty_abs pair_abs pair_in_M_iff upair_in_M_iff*
upair_abs
Upair_simp
by *simp*

definition
opair_name :: *i* ⇒ *i* ⇒ *i* **where**
opair_name(τ,ϱ) ≡ *upair_name*(*upair_name*(τ,τ),*upair_name*(τ,ϱ))

relativize *opair_name is_opair_name*

lemma *upair_name_closed* :
⟦ *x*∈*M*; *y*∈*M* ⟧ ⟹ *upair_name*(*x*,*y*)∈*M*
unfolding *upair_name_def* **using** *upair_in_M_iff pair_in_M_iff one_in_M upair_in_M_iff*
Upair_simp
by *simp*

definition
upair_name_fm :: [*i*,*i*,*i*,*i*] ⇒ *i* **where**
upair_name_fm(*x*,*y*,*o*,*z*) ≡ *Exists*(*Exists*(*And*(*pair_fm*(*x*#+*2*,*o*#+*2*,*1*),
And(*pair_fm*(*y*#+*2*,*o*#+*2*,*0*),*upair_fm*(*1*,*0*,*z*#+*2*)))))

lemma *upair_name_fm_type*[*TC*] :
⟦ *s*∈*nat*;*x*∈*nat*;*y*∈*nat*;*o*∈*nat*⟧ ⟹ *upair_name_fm*(*s*,*x*,*y*,*o*)∈*formula*
unfolding *upair_name_fm_def* **by** *simp*

lemma *sats_upair_name_fm* :
assumes *x∈nat y∈nat z∈nat o∈nat env∈list(M) nth(o,env)=one*
shows
sats(M,upair_name_fm(x,y,o,z),env) ⟷ is_upair_name(##M,nth(x,env),nth(y,env),nth(z,env))
unfolding *upair_name_fm_def is_upair_name_def*
using *assms zero_in_M empty_abs pair_abs one_in_M pair_in_M_iff Upair_simp upair_in_M_iff*
by *auto*

lemma *opair_name_abs* :
assumes *x∈M y∈M z∈M*
shows *is_opair_name(##M,x,y,z) ⟷ z = opair_name(x,y)*
unfolding *is_opair_name_def opair_name_def* **using** *assms upair_name_abs upair_name_closed*
by *simp*

lemma *opair_name_closed* :
⟦ *x∈M*; *y∈M* ⟧ ⟹ *opair_name(x,y)∈M*
unfolding *opair_name_def* **using** *upair_name_closed* **by** *simp*

definition
opair_name_fm :: [*i*,*i*,*i*,*i*] ⇒ *i* **where**
opair_name_fm(x,y,o,z) ≡ Exists(Exists(And(upair_name_fm(x#+2,x#+2,o#+2,1),
And(upair_name_fm(x#+2,y#+2,o#+2,0),upair_name_fm(1,0,o#+2,z#+2)))))

lemma *opair_name_fm_type*[*TC*] :
⟦ *s∈nat;x∈nat;y∈nat;o∈nat*⟧ ⟹ *opair_name_fm(s,x,y,o)∈formula*
unfolding *opair_name_fm_def* **by** *simp*

lemma *sats_opair_name_fm* :
assumes *x∈nat y∈nat z∈nat o∈nat env∈list(M) nth(o,env)=one*
shows
sats(M,opair_name_fm(x,y,o,z),env) ⟷ is_opair_name(##M,nth(x,env),nth(y,env),nth(z,env))
unfolding *opair_name_fm_def is_opair_name_def*
using *assms sats_upair_name_fm*
by *auto*

lemma *val_upair_name* : *val(P,G,upair_name(τ,ϱ)) = {val(P,G,τ),val(P,G,ϱ)}*
unfolding *upair_name_def* **using** *val_Upair Upair_simp generic one_in_G one_in_P*
by *simp*

lemma *val_opair_name* : *val(P,G,opair_name(τ,ϱ)) = ⟨val(P,G,τ),val(P,G,ϱ)⟩*
unfolding *opair_name_def Pair_def* **using** *val_upair_name* **by** *simp*

lemma *val_RepFun_one*: *val(P,G,{⟨f(x),one⟩ . x∈a}) = {val(P,G,f(x)) . x∈a}*
proof -
let *?A = {f(x) . x ∈ a}*
let *?Q = λ⟨x,p⟩ . p = one*
have *one ∈ P∩G* **using** *generic one_in_G one_in_P* **by** *simp*

have {⟨f(x),one⟩ . x ∈ a} = {t ∈ ?A × P . ?Q(t)}
using *one_in_P* **by** *force*
then
have *val*(P,G,{⟨f(x),one⟩ . x ∈ a}) = *val*(P,G,{t ∈ ?A × P . ?Q(t)})
by *simp*
also
have ... = {*val*(P,G,t) .. t ∈ ?A , ∃ p∈P∩G . ?Q(⟨t,p⟩)}
using *val_of_name_alt* **by** *simp*
also
have ... = {*val*(P,G,t) . t ∈ ?A }
using ‹*one*∈P∩G› **by** *force*
also
have ... = {*val*(P,G,f(x)) . x ∈ a}
by *auto*
finally show *?thesis* **by** *simp*
qed

30.1 $M[G]$ is a transitive model of ZF

interpretation *mgzf*: *M_ZF_trans M*[G]
using *Transset_MG generic pairing_in_MG Union_MG*
extensionality_in_MG power_in_MG foundation_in_MG
strong_replacement_in_MG separation_in_MG infinity_in_MG
by *unfold_locales simp_all*

definition
is_opname_check :: [i,i,i] ⇒ 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*
unfolding *opname_check_fm_def* **by** *simp*

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))
unfolding *opname_check_fm_def is_opname_check_def*
using *assms sats_check_fm sats_opair_name_fm one_in_M* **by** *simp*

lemma *opname_check_abs* :
  assumes *s*∈*M* *x*∈*M* *y*∈*M*
  shows *is_opname_check*(*s*,*x*,*y*) ⟷ *y* = *opair_name*(*check*(*x*),*s*‘*x*)
  unfolding *is_opname_check_def*
  using *assms check_abs check_in_M opair_name_abs apply_abs apply_closed* **by** *simp*

lemma *repl_opname_check* :
  assumes
    A∈*M* *f*∈*M*
  shows
    {*opair_name*(*check*(*x*),*f*‘*x*). *x*∈*A*}∈*M*
proof -
  have *arity*(*opname_check_fm*(*3*,*0*,*1*,*2*))= *4*
   unfolding *fm_definitions opname_check_fm_def opair_name_fm_def upair_name_fm_def*
    by (*simp add*:*nat_simp_union*)
  moreover
  have *x*∈*A* ⟹ *opair_name*(*check*(*x*), *f* ‘ *x*)∈*M* **for** *x*
    using *assms opair_name_closed apply_closed transitivity check_in_M*
    by *simp*
  ultimately
  show *?thesis* **using** *assms opname_check_abs*[*of f*] *sats_opname_check_fm*
      one_in_M
      Repl_in_M[*of opname_check_fm*(*3*,*0*,*1*,*2*) [*one*,*f*] *is_opname_check*(*f*)
          λ*x*. *opair_name*(*check*(*x*),*f*‘*x*)]
    by *simp*
qed

theorem *choice_in_MG*:
  assumes *choice_ax*(##*M*)
  shows *choice_ax*(##*M*[*G*])
proof -
  {
    fix *a*
    assume *a*∈*M*[*G*]
    then
    obtain τ **where** τ∈*M* *val*(*P*,*G*,τ) = *a*
      using *GenExt_def* **by** *auto*
    with ‹τ∈*M*›
    have *domain*(τ)∈*M*
      using *domain_closed* **by** *simp*
    then
    obtain *s* α **where** *s*∈*surj*(α,*domain*(τ)) *Ord*(α) *s*∈*M* α∈*M*
      using *assms choice_ax_abs* **by** *auto*
    then
    have α∈*M*[*G*]
      using *M_subset_MG generic one_in_G subsetD* **by** *blast*
    let *?A*=*domain*(τ)×*P*

let *?g* = {*opair_name*(*check*(β),*s*‘β). β∈α}
have *?g* ∈ *M* **using** ‹*s*∈*M*› ‹α∈*M*› *repl_opname_check* **by** *simp*
let *?f_dot*={⟨*opair_name*(*check*(β),*s*‘β),*one*⟩. β∈α}
have *?f_dot* = *?g* × {*one*} **by** *blast*
from *one_in_M* **have** {*one*} ∈ *M* **using** *singletonM* **by** *simp*
define *f* **where**
  f ≡ *val*(*P*,*G*,*?f_dot*)
from ‹{*one*}∈*M*› ‹*?g*∈*M*› ‹*?f_dot* = *?g*×{*one*}›
have *?f_dot*∈*M*
  using *cartprod_closed* **by** *simp*
then
have *f* ∈ *M*[*G*]
  unfolding *f_def* **by** (*blast intro*:*GenExtI*)
have *f* = {*val*(*P*,*G*,*opair_name*(*check*(β),*s*‘β)) . β∈α}
  unfolding *f_def* **using** *val_RepFun_one* **by** *simp*
also
have ... = {⟨β,*val*(*P*,*G*,*s*‘β)⟩ . β∈α}
  using *val_opair_name valcheck generic one_in_G one_in_P* **by** *simp*
finally
have *f* = {⟨β,*val*(*P*,*G*,*s*‘β)⟩ . β∈α} **.**
then
have *1*: *domain*(*f*) = α *function*(*f*)
  unfolding *function_def* **by** *auto*
have *2*: *y* ∈ *a* ⟹ ∃*x*∈α. *f* ‘ *x* = *y* **for** *y*
proof -
  fix *y*
  assume
    y ∈ *a*
  with ‹*val*(*P*,*G*,τ) = *a*›
  obtain σ **where** σ∈*domain*(τ) *val*(*P*,*G*,σ) = *y*
    using *elem_of_val*[*of y _* τ] **by** *blast*
  with ‹*s*∈*surj*(α,*domain*(τ))›
  obtain β **where** β∈α *s*‘β = σ
    unfolding *surj_def* **by** *auto*
  with ‹*val*(*P*,*G*,σ) = *y*›
  have *val*(*P*,*G*,*s*‘β) = *y*
    by *simp*
  with ‹*f* = {⟨β,*val*(*P*,*G*,*s*‘β)⟩ . β∈α}› ‹β∈α›
  have ⟨β,*y*⟩∈*f*
    by *auto*
  with ‹*function*(*f*)›
  have *f*‘β = *y*
    using *function_apply_equality* **by** *simp*
  with ‹β∈α› **show**
    ∃β∈α. *f* ‘ β = *y*
    by *auto*
qed
then
have ∃α∈(*M*[*G*]). ∃*f'*∈(*M*[*G*]). *Ord*(α) ∧ *f'* ∈ *surj*(α,*a*)

```
    proof (cases a=0)
      case True
      then
      show ?thesis
        unfolding surj_def using zero_in_MG by auto
    next
      case False
      with ‹a∈M[G]›
      obtain e where e∈a e∈M[G]
        using transitivity_MG by blast
      with 1 and 2
      have induced_surj(f,a,e) ∈ surj(α,a)
        using induced_surj_is_surj by simp
      moreover from ‹f∈M[G]› ‹a∈M[G]› ‹e∈M[G]›
      have induced_surj(f,a,e) ∈ M[G]
        unfolding induced_surj_def
        by (simp flip: setclass_iff)
      moreover note
        ‹α∈M[G]› ‹Ord(α)›
      ultimately show ?thesis by auto
    qed
  }
  then
  show ?thesis using mgzf.choice_ax_abs by simp
qed

end

end
```

31 Ordinals in generic extensions

```
theory Ordinals_In_MG
  imports
    Forcing_Theorems Relative_Univ
begin

context G_generic
begin

lemma rank_val: rank(val(P,G,x)) ≤ rank(x) (is ?Q(x))
proof (induct rule:ed_induction[of ?Q])
  case (1 x)
  have val(P,G,x) = {val(P,G,u). u∈{t∈domain(x). ∃p∈P . ⟨t,p⟩∈x ∧ p ∈ G
}}
    using def_val unfolding Sep_and_Replace by blast
  then
  have rank(val(P,G,x)) = (⋃u∈{t∈domain(x). ∃p∈P . ⟨t,p⟩∈x ∧ p ∈ G }.
succ(rank(val(P,G,u))))
```

```
    using rank[of val(P,G,x)] by simp
  moreover
  have succ(rank(val(P,G, y))) ≤ rank(x) if ed(y, x) for y
    using 1[OF that] rank_ed[OF that] by (auto intro:lt_trans1)
  moreover from this
  have (⋃u∈{t∈domain(x). ∃p∈P . ⟨t,p⟩∈x ∧ p ∈ G }. succ(rank(val(P,G,u))))
≤ rank(x)
    by (rule_tac UN_least_le) (auto)
  ultimately
  show ?case by simp
qed

lemma Ord_MG_iff:
  assumes Ord(α)
  shows α ∈ M ⟷ α ∈ M[G]
proof
  show α ∈ M ⟹ α ∈ M[G]
    using generic[THEN one_in_G, THEN M_subset_MG] ..
next
  assume α ∈ M[G]
  then
  obtain x where x∈M val(P,G,x) = α
    using GenExtD by auto
  then
  have rank(α) ≤ rank(x)
    using rank_val by blast
  with assms
  have α ≤ rank(x)
    using rank_of_Ord by simp
  then
  have α ∈ succ(rank(x)) using ltD by simp
  with ‹x∈M›
  show α ∈ M
    using cons_closed transitivity[of α succ(rank(x))]
      rank_closed unfolding succ_def by simp
qed

end

end
```

32 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∈P ⟹ ∃ q∈P. ∃ r∈P. q ≼ p ∧ r ≼ p ∧ q ⊥ r
begin
```

For separative preorders, the complement of every filter is dense. Hence an M-generic filter can't belong to the ground model.

```
lemma filter_complement_dense:
  assumes filter(G) shows dense(P - G)
proof
  fix p
  assume p∈P
  show ∃ d∈P - G. d ≼ p
  proof (cases p∈G)
    case True
    note ‹p∈P› assms
    moreover
    obtain q r where q ≼ p r ≼ p q ⊥ r q∈P r∈P
      using separative[OF ‹p∈P›]
      by force
    with ‹filter(G)›
    obtain s where s ≼ p s ∉ G s ∈ P
      using filter_imp_compat[of G q r]
      by auto
    then
    show ?thesis by blast
  next
    case False
    with ‹p∈P›
    show ?thesis using refl_leq unfolding Diff_def by auto
  qed
qed

end

locale ctm_separative = forcing_data + separative_notion
begin

lemma generic_not_in_M: assumes M_generic(G)  shows G ∉ M
proof
  assume G∈M
  then
  have P - G ∈ M
    using P_in_M Diff_closed by simp
  moreover
  have ¬(∃ q∈G. q ∈ P - G) (P - G) ⊆ P
    unfolding Diff_def by auto
  moreover
```

note *assms*
ultimately
show *False*
using *filter_complement_dense*[*of G*] *M_generic_denseD*[*of G P-G*]
M_generic_def **by** *simp* — need to put generic ==¿ filter in claset
qed

theorem *proper_extension*: **assumes** *M_generic*(G) **shows** $M \neq M[G]$
using *assms G_in_Gen_Ext*[*of G*] *one_in_G*[*of G*] *generic_not_in_M*
by *force*

end

end

33 A poset of successions

theory *Succession_Poset*
imports
Arities
Proper_Extension
Synthetic_Definition
Names
begin

33.1 The set of finite binary sequences

notation *nat* (‹ω›) — MOVE THIS to an appropriate place

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*]: $n\in\alpha \Longrightarrow f{:}n\to B \Longrightarrow f\in B^{<\alpha}$
unfolding *seqspace_def* **by** *blast*

lemma *seqspaceD*[*dest*]: $f\in B^{<\alpha} \Longrightarrow \exists\, n\in\alpha.\ f{:}n\to B$
unfolding *seqspace_def* **by** *blast*

— FIXME: Now this is too particular (only for ω-sequences. A relative definition for *seqspace* would be appropriate.
schematic_goal *seqspace_fm_auto*:
assumes
$nth(i,env) = n$ $nth(j,env) = z$ $nth(h,env) = B$
$i \in nat$ $j \in nat$ $h\in nat$ $env \in list(A)$
shows

(∃ *om*∈*A*. *omega*(##*A*,*om*) ∧ *n* ∈ *om* ∧ *is_funspace*(##*A*, *n*, *B*, *z*)) ⟷ (*A*, *env* ⊨ (*?sqsprp*(*i*,*j*,*h*)))
unfolding *is_funspace_def*
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

synthesize *seqspace_rep_fm* **from_schematic** *seqspace_fm_auto*

locale *M_seqspace* = *M_trancl* +
assumes
seqspace_replacement: *M*(*B*) ⟹ *strong_replacement*(*M*,λ*n z*. *n*∈*nat* ∧ *is_funspace*(*M*,*n*,*B*,*z*))
begin

lemma *seqspace_closed*:
$M(B) \Longrightarrow M(B^{<\omega})$
unfolding *seqspace_def* **using** *seqspace_replacement*[*of B*] *RepFun_closed2*
by *simp*

end

sublocale *M_ctm* ⊆ *M_seqspace* ##*M*
proof (*unfold_locales*, *simp*)
fix *B*
have *arity*(*seqspace_rep_fm*(*0*,*1*,*2*)) ≤ *3* *seqspace_rep_fm*(*0*,*1*,*2*)∈*formula*
unfolding *seqspace_rep_fm_def*
using *arity_pair_fm arity_omega_fm arity_typed_function_fm nat_simp_union*
by *auto*
moreover
assume *B*∈*M*
ultimately
have *strong_replacement*(##*M*, λ*x y*. *M*, [*x*, *y*, *B*] ⊨ *seqspace_rep_fm*(*0*, *1*, *2*))
using *replacement_ax*[*of seqspace_rep_fm*(*0*,*1*,*2*)]
by *simp*
moreover
note ‹*B*∈*M*›
moreover from *this*
have *univalent*(##*M*, *A*, λ*x y*. *M*, [*x*, *y*, *B*] ⊨ *seqspace_rep_fm*(*0*, *1*, *2*))
if *A*∈*M* **for** *A*
using *that* **unfolding** *univalent_def seqspace_rep_fm_def*
by (*auto*, *blast dest*:*transitivity*)
ultimately
have *strong_replacement*(##*M*, λ*n z*. ∃ *om*[##*M*]. *omega*(##*M*,*om*) ∧ *n* ∈ *om* ∧ *is_funspace*(##*M*, *n*, *B*, *z*))
using *seqspace_fm_auto*[*of 0* [_,_,*B*] _ *1* _ *2 B M*] **unfolding** *seqspace_rep_fm_def strong_replacement_def*
by *simp*
with ‹*B*∈*M*›
show *strong_replacement*(##*M*, λ*n z*. *n* ∈ *nat* ∧ *is_funspace*(##*M*, *n*, *B*, *z*))
using *M_nat* **by** *simp*

```
qed

definition seq_upd :: i ⇒ i ⇒ i where
  seq_upd(f,a) ≡ λ j ∈ succ(domain(f)) . if j < domain(f) then f`j else a

lemma seq_upd_succ_type :
  assumes n∈nat f∈n→A a∈A
  shows seq_upd(f,a)∈ succ(n) → A
proof -
  from assms
  have equ: domain(f) = n using domain_of_fun by simp
  {
    fix j
    assume j∈succ(domain(f))
    with equ ‹n∈_›
    have j≤n using ltI by auto
    with ‹n∈_›
    consider (lt) j<n | (eq) j=n using leD by auto
    then
    have (if j < n then f`j else a) ∈ A
    proof cases
      case lt
      with ‹f∈_›
      show ?thesis using apply_type ltD[OF lt] by simp
    next
      case eq
      with ‹a∈_›
      show ?thesis by auto
    qed
  }
  with equ
  show ?thesis
    unfolding seq_upd_def
    using lam_type[of succ(domain(f))]
    by auto
qed

lemma seq_upd_type :
  assumes f∈A<ω a∈A
  shows seq_upd(f,a) ∈ A<ω
proof -
  from ‹f∈_›
  obtain y where y∈nat f∈y→A
    unfolding seqspace_def by blast
  with ‹a∈A›
  have seq_upd(f,a)∈succ(y)→A
    using seq_upd_succ_type by simp
  with ‹y∈_›
  show ?thesis
```

unfolding *seqspace_def* **by** *auto*
qed

lemma *seq_upd_apply_domain* [*simp*]:
assumes $f{:}n{\rightarrow}A$ $n{\in}nat$
shows $seq\_upd(f,a)\text{`}n = a$
unfolding *seq_upd_def* **using** *assms domain_of_fun* **by** *auto*

lemma *zero_in_seqspace* :
shows $0 \in A^{<\omega}$
unfolding *seqspace_def*
by *force*

definition
seqleR :: $i \Rightarrow i \Rightarrow o$ **where**
$seqleR(f,g) \equiv g \subseteq f$

definition
seqlerel :: $i \Rightarrow i$ **where**
$seqlerel(A) \equiv Rrel(\lambda x\ y.\ y \subseteq x, A^{<\omega})$

definition
seqle :: i **where**
$seqle \equiv seqlerel(2)$

lemma *seqleI*[*intro*!]:
$\langle f,g\rangle \in 2^{<\omega}\times 2^{<\omega} \Longrightarrow g \subseteq f \Longrightarrow \langle f,g\rangle \in seqle$
unfolding *seqspace_def seqle_def seqlerel_def Rrel_def*
by *blast*

lemma *seqleD*[*dest*!]:
$z \in seqle \Longrightarrow \exists\, x\ y.\ \langle x,y\rangle \in 2^{<\omega}\times 2^{<\omega} \wedge y \subseteq x \wedge z = \langle x,y\rangle$
unfolding *seqle_def seqlerel_def Rrel_def*
by *blast*

lemma *upd_leI* :
assumes $f{\in}2^{<\omega}$ $a{\in}2$
shows $\langle seq\_upd(f,a),f\rangle{\in}seqle$ (**is** $\langle ?f,\_\rangle{\in}\_$)
proof
show $\langle ?f, f\rangle \in 2^{<\omega} \times 2^{<\omega}$
using *assms seq_upd_type* **by** *auto*
next
show $f \subseteq seq\_upd(f,a)$
proof
fix x
assume $x \in f$
moreover from $\langle f \in 2^{<\omega}\rangle$
obtain n **where** $n{\in}nat$ $f : n \rightarrow 2$
by *blast*

```
    moreover from calculation
    obtain y where y∈n x=⟨y,f`y⟩ using Pi_memberD[of f n λ_ . 2]
      by blast
    moreover from ‹f:n→2›
    have domain(f) = n using domain_of_fun by simp
    ultimately
    show x ∈ seq_upd(f,a)
      unfolding seq_upd_def lam_def
      by (auto intro:ltI)
  qed
qed

lemma preorder_on_seqle: preorder_on(2^<ω,seqle)
  unfolding preorder_on_def refl_def trans_on_def by blast

lemma zero_seqle_max: x∈2^<ω ⟹ ⟨x,0⟩ ∈ seqle
  using zero_in_seqspace
  by auto

interpretation sp:forcing_notion 2^<ω seqle 0
  using preorder_on_seqle zero_seqle_max zero_in_seqspace
  by unfold_locales simp_all

notation sp.Leq (infixl ≼s 50)
notation sp.Incompatible (infixl ⊥s 50)

lemma seqspace_separative:
  assumes f∈2^<ω
  shows seq_upd(f,0) ⊥s seq_upd(f,1) (is ?f ⊥s ?g)
proof
  assume sp.compat(?f, ?g)
  then
  obtain h where h ∈ 2^<ω ?f ⊆ h ?g ⊆ h
    by blast
  moreover from ‹f∈_›
  obtain y where y∈nat f:y→2 by blast
  moreover from this
  have ?f: succ(y) → 2 ?g: succ(y) → 2
    using seq_upd_succ_type by blast+
  moreover from this
  have ⟨y,?f`y⟩ ∈ ?f ⟨y,?g`y⟩ ∈ ?g using apply_Pair by auto
  ultimately
  have ⟨y,0⟩ ∈ h ⟨y,1⟩ ∈ h by auto
  moreover from ‹h ∈ 2^<ω›
  obtain n where n∈nat h:n→2 by blast
  ultimately
  show False
    using fun_is_function[of h n λ_. 2]
    unfolding seqspace_def function_def by auto
```

qed

definition *is_seqleR* :: [*i*⇒*o*,*i*,*i*] ⇒ *o* **where**
is_seqleR(*Q*,*f*,*g*) ≡ *g* ⊆ *f*

definition *seqleR_fm* :: *i* ⇒ *i* **where**
seqleR_fm(*fg*) ≡ *Exists*(*Exists*(*And*(*pair_fm*(*0*,*1*,*fg*#+*2*),*subset_fm*(*1*,*0*))))

lemma *type_seqleR_fm* :
fg ∈ *nat* ⟹ *seqleR_fm*(*fg*) ∈ *formula*
unfolding *seqleR_fm_def*
by *simp*

lemma *arity_seqleR_fm* :
fg ∈ *nat* ⟹ *arity*(*seqleR_fm*(*fg*)) = *succ*(*fg*)
unfolding *seqleR_fm_def*
using *arity_pair_fm arity_subset_fm nat_simp_union* **by** *simp*

lemma (**in** *M_basic*) *seqleR_abs*:
assumes *M*(*f*) *M*(*g*)
shows *seqleR*(*f*,*g*) ⟷ *is_seqleR*(*M*,*f*,*g*)
unfolding *seqleR_def is_seqleR_def*
using *assms apply_abs domain_abs domain_closed*[*OF* ‹*M*(*f*)›] *domain_closed*[*OF* ‹*M*(*g*)›]
by *auto*

definition
relP :: [*i*⇒*o*,[*i*⇒*o*,*i*,*i*]⇒*o*,*i*] ⇒ *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*))
unfolding *seqleR_fm_def is_seqleR_def relP_def*
using *assms trans_M sats_subset_fm pair_iff_sats*
by *auto*

lemma (**in** *M_basic*) *is_related_abs* :
assumes ⋀ *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*))
unfolding *relP_def* **using** *pair_in_M_iff assms* **by** *auto*

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 -
  from ‹M(A)›
  have M(z) if z∈A×A for z
    using cartprod_closed transM[of z A×A] that by simp
  then
  have A:relP(M, is_rel, z) ⟷ (∃ x y. z = ⟨x, y⟩ ∧ rel(x, y)) M(z) if z∈A×A
for z
    using that is_related_abs[of rel is_rel,OF assms(3)] by auto
  then
  have Collect(A×A,relP(M,is_rel)) = Collect(A×A,λz. (∃ x y. z = ⟨x,y⟩ ∧ rel(x,y)))
   using Collect_cong[of A×A A×A relP(M,is_rel),OF _ A(1)] assms(1) assms(2)
    by auto
  with assms
  show ?thesis unfolding is_RRel_def Rrel_def using cartprod_closed
    by auto
qed

definition
  is_seqlerel :: [i⇒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)
  unfolding is_seqlerel_def
  using is_Rrel_abs[OF ‹M(A)› ‹M(r)›,of seqleR is_seqleR] seqleR_abs
  by auto

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

lemma Rrel_eq : RrelP(R,A) = Rrel(R,A)
  unfolding Rrel_def RrelP_def by auto

context M_ctm
begin

lemma Rrel_closed:
  assumes A∈M
    ⋀ a. a ∈ nat ⟹ rel_fm(a)∈formula
    ⋀ f g . (##M)(f) ⟹ (##M)(g) ⟹ rel(f,g) ⟷ is_rel(##M,f,g)
    arity(rel_fm(0)) = 1
    ⋀ a . a ∈ M ⟹ sats(M,rel_fm(0),[a]) ⟷ relP(##M,is_rel,a)
  shows (##M)(Rrel(rel,A))
proof -
  have z∈ M ⟹ relP(##M, is_rel, z) ⟷ (∃ x y. z = ⟨x, y⟩ ∧ rel(x, y)) for z
    using assms(3) is_related_abs[of rel is_rel]
```

by *auto*
with *assms*
have *Collect*($A \times A$,λz. ($\exists\, x\ y.\ z = \langle x,y\rangle \wedge rel(x,y)$)) $\in M$
using *Collect_in_M_0p*[*of rel_fm*(*0*) λ *A z* . *relP*(*A*,*is_rel*,*z*) λ *z*.$\exists\, x\ y.\ z = \langle x,$
$y\rangle \wedge rel(x,\ y)$]
cartprod_closed
by *simp*
then show *?thesis*
unfolding *Rrel_def* **by** *simp*
qed

lemma *seqle_in_M*: *seqle* $\in M$
using *Rrel_closed seqspace_closed*
transitivity[*OF _ nat_in_M*] *type_seqleR_fm*[*of 0*] *arity_seqleR_fm*[*of 0*]
seqleR_fm_sats[*of 0*] *seqleR_abs seqlerel_abs*
unfolding *seqle_def seqlerel_def seqleR_def*
by *auto*

33.2 Cohen extension is proper

interpretation *ctm_separative* $2^{<\omega}$ *seqle 0*
proof (*unfold_locales*)
fix *f*
let *?q=seq_upd*(*f*,*0*) **and** *?r=seq_upd*(*f*,*1*)
assume $f \in 2^{<\omega}$
then
have *?q* $\preceq$*s f* $\wedge$ *?r* $\preceq$*s f* $\wedge$ *?q* $\perp$*s ?r*
using *upd_leI seqspace_separative* **by** *auto*
moreover from *calculation*
have *?q* $\in 2^{<\omega}$ *?r* $\in 2^{<\omega}$
using *seq_upd_type*[*of f 2*] **by** *auto*
ultimately
show $\exists\, q \in 2^{<\omega}.\ \exists\, r \in 2^{<\omega}.\ q \preceq s\ f \wedge r \preceq s\ f \wedge q \perp s\ r$
by (*rule_tac bexI*)+ — why the heck auto-tools don't solve this?
next
show $2^{<\omega} \in M$ **using** *nat_into_M seqspace_closed* **by** *simp*
next
show *seqle* $\in M$ **using** *seqle_in_M* **.**
qed

lemma *cohen_extension_is_proper*: $\exists\, G.\ M\_generic(G) \wedge M \neq M^{2^{<\omega}}[G]$
using *proper_extension generic_filter_existence zero_in_seqspace*
by *force*

end

end

34 The main theorem

theory *Forcing_Main*
  imports
  Internal_ZFC_Axioms
  Choice_Axiom
  Ordinals_In_MG
  Succession_Poset

begin

34.1 The generic extension is countable

definition
  minimum :: *i* ⇒ *i* ⇒ *i* **where**
  minimum(*r*,*B*) ≡ *THE b. first*(*b*,*B*,*r*)

lemma *minimum_in*: ⟦ *well_ord*(*A*,*r*); *B*⊆*A*; *B*≠*0* ⟧ ⟹ *minimum*(*r*,*B*) ∈ *B*
  using *the_first_in* **unfolding** *minimum_def* **by** *simp*

lemma *well_ord_surj_imp_lepoll*:
  assumes *well_ord*(*A*,*r*) *h* ∈ *surj*(*A*,*B*)
  shows *B* ≲ *A*
proof -
  let *?f*=λ*b*∈*B*. *minimum*(*r*, {*a*∈*A*. *h*‘*a*=*b*})
  have *minimum*(*r*, {*a* ∈ *A* . *h* ‘ *a* = *b*}) ∈ {*a*∈*A*. *h*‘*a*=*b*} **if** *b*∈*B* **for** *b*
  proof -
    from ‹*h* ∈ *surj*(*A*,*B*)› *that*
    have {*a*∈*A*. *h*‘*a*=*b*} ≠ *0*
      unfolding *surj_def* **by** *blast*
    with ‹*well_ord*(*A*,*r*)›
    show *minimum*(*r*,{*a*∈*A*. *h*‘*a*=*b*}) ∈ {*a*∈*A*. *h*‘*a*=*b*}
      using *minimum_in* **by** *blast*
  qed
  moreover from *this*
  have *?f* : *B* → *A*
      using *lam_type*[*of B* _ λ_.*A*] **by** *simp*
  moreover
  have *?f* ‘ *w* = *?f* ‘ *x* ⟹ *w* = *x* **if** *w*∈*B* *x*∈*B* **for** *w x*
  proof -
    from *calculation that*
    have *w* = *h* ‘ *minimum*(*r*,{*a*∈*A*. *h*‘*a*=*w*})
        x = *h* ‘ *minimum*(*r*,{*a*∈*A*. *h*‘*a*=*x*})
      by *simp_all*
    moreover
    assume *?f* ‘ *w* = *?f* ‘ *x*
    moreover from *this* **and** *that*
    have *minimum*(*r*, {*a* ∈ *A* . *h* ‘ *a* = *w*}) = *minimum*(*r*, {*a* ∈ *A* . *h* ‘ *a* = *x*})
      unfolding *minimum_def* **by** *simp_all*
    moreover from *calculation*(*1*,*2*,*4*)

```
    show w=x by simp
    qed
  ultimately
  show ?thesis
  unfolding lepoll_def inj_def by blast
qed

lemma (in forcing_data) surj_nat_MG :
  ∃f. f ∈ surj(ω,M[G])
proof -
  let ?f=λn∈ω. val(P,G,enum`n)
  have x ∈ ω ⟹ val(P,G, enum ` x)∈ M[G] for x
    using GenExtD[THEN iffD2, of _ G] bij_is_fun[OF M_countable] by force
  then
  have ?f: ω → M[G]
    using lam_type[of ω λn. val(P,G,enum`n) λ_.M[G]] by simp
  moreover
  have ∃n∈ω. ?f`n = x if x∈M[G] for x
    using that GenExtD[of _ G] bij_is_surj[OF M_countable]
    unfolding surj_def by auto
  ultimately
  show ?thesis
    unfolding surj_def by blast
qed

lemma (in G_generic) MG_eqpoll_nat: M[G] ≈ ω
proof -
  interpret MG: M_ZF_trans M[G]
    using Transset_MG generic pairing_in_MG
      Union_MG extensionality_in_MG power_in_MG
      foundation_in_MG strong_replacement_in_MG[simplified]
      separation_in_MG[simplified] infinity_in_MG
    by unfold_locales simp_all
  obtain f where f ∈ surj(ω,M[G])
    using surj_nat_MG by blast
  then
  have M[G] ≲ ω
    using well_ord_surj_imp_lepoll well_ord_Memrel[of ω]
    by simp
  moreover
  have ω ≲ M[G]
    using MG.nat_into_M subset_imp_lepoll by auto
  ultimately
  show ?thesis using eqpollI
    by simp
qed
```

34.2 The main result

```
theorem extensions_of_ctms:
  assumes
    M ≈ ω Transset(M) M ⊨ ZF
  shows
    ∃N.
      M ⊆ N ∧ N ≈ ω ∧ Transset(N) ∧ N ⊨ ZF ∧ M≠N ∧
      (∀α. Ord(α) ⟶ (α ∈ M ⟷ α ∈ N)) ∧
      (M, []⊨ AC ⟶ N ⊨ ZFC)
proof -
  from ‹M ⊨ ZF›
  interpret M_ZF M
    using M_ZF_iff_M_satT
    by simp
  from ‹Transset(M)›
  interpret M_ZF_trans M
    using M_ZF_iff_M_satT
    by unfold_locales
  from ‹M ≈ ω›
  obtain enum where enum ∈ bij(ω,M)
    using eqpoll_sym unfolding eqpoll_def by blast
  then
  interpret M_ctm M enum by unfold_locales
  interpret forcing_data 2^<ω seqle 0 M enum
    using nat_into_M seqspace_closed seqle_in_M
    by unfold_locales simp
  obtain G where M_generic(G) M ≠ M^(2^<ω)[G] (is M≠?N)
    using cohen_extension_is_proper
    by blast
  then
  interpret G_generic 2^<ω seqle 0 _ enum G by unfold_locales
  interpret MG: M_ZF ?N
    using generic pairing_in_MG
      Union_MG  extensionality_in_MG power_in_MG
      foundation_in_MG  strong_replacement_in_MG[simplified]
      separation_in_MG[simplified] infinity_in_MG
    by unfold_locales simp_all
  have ?N ⊨ ZF
    using M_ZF_iff_M_satT[of ?N] MG.M_ZF_axioms by simp
  moreover
  have M, []⊨ AC ⟹ ?N ⊨ ZFC
  proof -
    assume M, [] ⊨ AC
    then
    have choice_ax(##M)
      unfolding ZF_choice_fm_def using ZF_choice_auto by simp
    then
    have choice_ax(##?N) using choice_in_MG by simp
```

with ‹*?N* ⊨ *ZF*›
show *?N* ⊨ *ZFC*
using *ZF_choice_auto sats_ZFC_iff_sats_ZF_AC*
unfolding *ZF_choice_fm_def* **by** *simp*
qed
moreover
note ‹*M* ≠ *?N*›
moreover
have *Transset*(*?N*) **using** *Transset_MG* **.**
moreover
have *M* ⊆ *?N* **using** *M_subset_MG*[*OF one_in_G*] *generic* **by** *simp*
ultimately
show *?thesis*
using *Ord_MG_iff MG_eqpoll_nat*
by (*rule_tac x*=*?N* **in** *exI*, *simp*)
qed

end

35 Main definitions of the development

theory *Definitions_Main*
imports *Forcing_Main*

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 35.3. 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 35.1) and in the ZF-Constructible library (Section 35.2) on which our definitions are built.

declare [[*show_question_marks*=*false*]]

35.1 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*) ≡
{*f* ∈ *A* → *B* . ∀ *w*∈*A*. ∀ *x*∈*A*. *f* ‘ *w* = *f* ‘ *x* ⟶ *w* = *x*} ∩

$\{f \in A \to B . \forall y \in B. \exists x \in A. f ` x = y\}$

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*

$i < j \equiv i \in j \wedge Ord(j)$

The set of natural numbers ω 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*]

$Ord(\omega) \wedge 0 < \omega \wedge (\forall y. y < \omega \longrightarrow succ(y) < \omega)$
$Ord(i) \wedge 0 < i \wedge (\forall y. y < i \longrightarrow succ(y) < i) \Longrightarrow \omega \le i$

hide_const (**open**) *Order.pred*
thm *add_0_right add_succ_right pred_0 pred_succ_eq*

$m \#+ succ(n) = succ(m \#+ n)$
$m \in \omega \Longrightarrow m \#+ 0 = m$
$pred(0) = 0$
$pred(succ(y)) = y$

Lists

thm *Nil Cons list.induct*

$[] \in list(A)$
$[\![a \in A; l \in list(A)]\!] \Longrightarrow Cons(a, l) \in list(A)$
$[\![x \in list(A); P([]); \bigwedge a\ l. [\![a \in A; l \in list(A); P(l)]\!] \Longrightarrow P(Cons(a, l))]\!]$
$\Longrightarrow P(x)$

thm *length.simps app.simps nth_0 nth_Cons*

$length([]) = 0$
$length(Cons(a, l)) = succ(length(l))$
$[] @ ys = ys$
$Cons(a, l) @ ys = Cons(a, l @ ys)$
$nth(0, Cons(a, l)) = a$
$n \in \omega \Longrightarrow nth(succ(n), Cons(a, l)) = nth(n, l)$

Relative quantifications

lemma $\forall x[M]. P(x) \equiv \forall x. M(x) \longrightarrow P(x)$
$\exists x[M]. P(x) \equiv \exists x. M(x) \wedge P(x)$
unfolding *rall_def rex_def* **.**

thm *setclass_iff*

$(\#\#A)(x) \longleftrightarrow x \in A$

35.2 ZF-Constructible

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 *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 *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 *upair_ax_def*

$upair\_ax(M) \equiv \forall x[M]. \forall y[M]. \exists z[M]. upair(M, x, y, z)$

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 *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 *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*

$strong\_replacement(M, P) \equiv$
$\forall A[M].$
$univalent(M, A, P) \longrightarrow (\exists Y[M]. \forall b[M]. b \in Y \longleftrightarrow (\exists x[M]. x \in A \wedge P(x,$
$b)))$

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*

$fun\_apply(M, f, x, y) \equiv$
$\exists xs[M].$
$\quad \exists fxs[M]. upair(M, x, x, xs) \wedge image(M, f, xs, fxs) \wedge big\_union(M, fxs, y)$

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 *surjection_def*

$surjection(M, A, B, f) \equiv$
$typed\_function(M, A, B, f) \wedge$
$(\forall y[M]. \ y \in B \longrightarrow (\exists x[M]. \ x \in A \wedge fun\_apply(M, f, x, y)))$

Internalized formulas

thm *Member Equal Nand Forall formula.induct*

$[\![x \in \omega; \ y \in \omega]\!] \Longrightarrow Member(x, y) \in formula$
$[\![x \in \omega; \ y \in \omega]\!] \Longrightarrow Equal(x, y) \in formula$
$[\![p \in formula; \ q \in formula]\!] \Longrightarrow Nand(p, q) \in formula$
$p \in formula \Longrightarrow Forall(p) \in formula$
$[\![x \in formula; \ \bigwedge x \ y. \ [\![x \in \omega; \ y \in \omega]\!] \Longrightarrow P(Member(x, y));$
$\bigwedge x \ y. \ [\![x \in \omega; \ y \in \omega]\!] \Longrightarrow P(Equal(x, y));$
$\bigwedge p \ q. \ [\![p \in formula; \ P(p); \ q \in formula; \ P(q)]\!] \Longrightarrow P(Nand(p, q));$
$\bigwedge p. \ [\![p \in formula; \ P(p)]\!] \Longrightarrow P(Forall(p))]\!]$
$\Longrightarrow P(x)$

thm *arity.simps*

$arity(Member(x, y)) = succ(x) \cup succ(y)$
$arity(Equal(x, y)) = succ(x) \cup succ(y)$
$arity(Nand(p, q)) = arity(p) \cup arity(q)$
$arity(Forall(p)) = pred(arity(p))$

thm *mem_iff_sats equal_iff_sats sats_Nand_iff sats_Forall_iff*

$[\![nth(i, env) = x; \ nth(j, env) = y; \ env \in list(A)]\!]$
$\Longrightarrow x \in y \longleftrightarrow A, env \models Member(i, j)$
$[\![nth(i, env) = x; \ nth(j, env) = y; \ env \in list(A)]\!]$
$\Longrightarrow x = y \longleftrightarrow A, env \models Equal(i, j)$
$env \in list(A) \Longrightarrow A, env \models Nand(p, q) \longleftrightarrow \neg (A, env \models p \wedge A, env \models q)$
$env \in list(A) \Longrightarrow A, env \models Forall(p) \longleftrightarrow (\forall x \in A. \ A, Cons(x, env) \models p)$

35.3 Forcing

thm *infinity_ax_def*

$infinity\_ax(M) \equiv$
$\exists I[M].$
$(\exists z[M]. \ empty(M, z) \wedge z \in I) \wedge$
$(\forall y[M]. \ y \in I \longrightarrow (\exists sy[M]. \ successor(M, y, sy) \wedge sy \in I))$

thm *choice_ax_def*

$choice\_ax(M) \equiv \forall x[M]. \exists a[M]. \exists f[M]. ordinal(M, a) \wedge surjection(M, a, x, f)$

thm *ZF_union_fm_iff_sats ZF_power_fm_iff_sats ZF_pairing_fm_iff_sats ZF_foundation_fm_iff_sats ZF_extensionality_fm_iff_sats ZF_infinity_fm_iff_sats sats_ZF_separation_fm_iff sats_ZF_replacement_fm_iff ZF_choice_fm_iff_sats*

$Union\_ax(\#\#A) \longleftrightarrow A, [] \models ZF\_union\_fm$
$power\_ax(\#\#A) \longleftrightarrow A, [] \models ZF\_power\_fm$
$upair\_ax(\#\#A) \longleftrightarrow A, [] \models ZF\_pairing\_fm$
$foundation\_ax(\#\#A) \longleftrightarrow A, [] \models ZF\_foundation\_fm$
$extensionality(\#\#A) \longleftrightarrow A, [] \models ZF\_extensionality\_fm$
$infinity\_ax(\#\#A) \longleftrightarrow A, [] \models ZF\_infinity\_fm$
$\varphi \in formula \Longrightarrow$
$M, [] \models ZF\_separation\_fm(\varphi) \longleftrightarrow$
$(\forall env \in list(M).$
$\quad arity(\varphi) \leq 1 \#+ length(env) \longrightarrow separation(\#\#M, \lambda x. M, [x] @ env \models \varphi))$
$\varphi \in formula \Longrightarrow$
$M, [] \models ZF\_replacement\_fm(\varphi) \longleftrightarrow$
$(\forall env \in list(M).$
$\quad arity(\varphi) \leq 2 \#+ length(env) \longrightarrow$
$\quad strong\_replacement(\#\#M, \lambda x\ y. M, [x, y] @ env \models \varphi))$
$choice\_ax(\#\#A) \longleftrightarrow A, [] \models ZF\_choice\_fm$

thm *ZF_fin_def ZF_inf_def ZF_def ZFC_fin_def ZFC_def*

$ZF\_fin \equiv$
$\{ZF\_extensionality\_fm, ZF\_foundation\_fm, ZF\_pairing\_fm, ZF\_union\_fm,$
$ZF\_infinity\_fm, ZF\_power\_fm\}$
$ZF\_inf \equiv$
$\{ZF\_separation\_fm(p) . p \in formula\} \cup \{ZF\_replacement\_fm(p) . p \in formula\}$
$ZF \equiv ZF\_inf \cup ZF\_fin$
$ZFC\_fin \equiv ZF\_fin \cup \{ZF\_choice\_fm\}$
$ZFC \equiv ZF\_inf \cup ZFC\_fin$

thm *satT_def*

$A \models \Phi \equiv \forall \varphi \in \Phi. A, [] \models \varphi$

thm *extensions_of_ctms*

$[\![M \approx \omega;\ \mathit{Transset}(M);\ M \models \mathit{ZF}]\!]$
$\Longrightarrow \exists N.\ M \subseteq N \wedge$
$N \approx \omega \wedge$
$\mathit{Transset}(N) \wedge$
$N \models \mathit{ZF} \wedge$
$M \neq N \wedge$
$(\forall \alpha.\ \mathit{Ord}(\alpha) \longrightarrow \alpha \in M \longleftrightarrow \alpha \in N) \wedge (M, [] \models \mathit{ZF\_choice\_fm} \longrightarrow N \models \mathit{ZFC})$

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).