Formalization of Forcing in Isabelle/ZF

Emmanuel Gunther* Miguel Pagano* Pedro Sánchez Terraf*†

March 17, 2025

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*

definition
  antichain :: *i*⇒*i*⇒*i*⇒*o* **where**
  antichain(*P*,*leq*,*A*) ≡ *A*⊆*P* ∧ (∀ *p*∈*A*.∀ *q*∈*A*.(¬ *compat_in*(*P*,*leq*,*p*,*q*)))

definition
  ccc :: *i* ⇒ *i* ⇒ *o* **where**
  ccc(*P*,*leq*) ≡ ∀ *A*. *antichain*(*P*,*leq*,*A*) ⟶ |*A*| ≤ *nat*

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 *leq_reflI*: $p{\in}P \Longrightarrow p{\preceq}p$
using *leq_preord* **unfolding** *preorder_on_def refl_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 leq_reflI* **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 leq_reflI* **by** *blast*

definition
$antichain :: i{\Rightarrow}o$ **where**
$antichain(A) \equiv A{\subseteq}P \wedge (\forall\, p{\in}A.\forall\, q{\in}A.(\neg compat(p,q)))$

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

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

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

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

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

lemma *low_bound_filter*: — says the compatibility is attained inside G
assumes $filter(G)$ **and** $p{\in}G$ **and** $q{\in}G$

shows $\exists\, r{\in}G.\ r{\preceq}p \wedge r{\preceq}q$
using *assms*
unfolding *compat_in_def filter_def* **by** *blast*

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

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

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

lemma *upclosureE* [*elim*] :
$p{\in}upclosure(A) \Longrightarrow (\bigwedge x\ a.\ x{\in}P \Longrightarrow a{\in}A \Longrightarrow a{\preceq}x \Longrightarrow R) \Longrightarrow R$
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∈A b∈A a≼p b≼q p∈P q∈P
      using upclosureD[OF ‹p∈?UA›] upclosureD[OF ‹q∈?UA›] by auto
    with assms(2)
    obtain d where d∈A d≼a d≼b
      unfolding compat_in_def by auto
    with 1
    have 2:d≼p d≼q d∈?UA
      using A_sub_upclosure[THEN subsetD] ‹A⊆P›
        leq_transD'[of A d a] leq_transD'[of A d b] by auto
    then
    show ?thesis unfolding compat_in_def by auto
  qed
qed

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

lemma decr_succ_decr:
  assumes f ∈ nat → P preorder_on(P,leq)
    ∀ n∈nat. ⟨f ‘ succ(n), f ‘ n⟩ ∈ leq
    m∈nat
  shows n∈nat ⟹ n≤m ⟹ ⟨f ‘ m, f ‘ n⟩ ∈ leq
  using ‹m∈_›
proof(induct m)
  case 0
  then show ?case using assms leq_reflI 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 leq_reflI by simp
  qed
qed

lemma decr_seq_linear:
  assumes refl(P,leq) f ∈ nat → P
    ∀ n∈nat. ⟨f ‘ succ(n), f ‘ n⟩ ∈ leq
    trans[P](leq)
```

```
  shows linear(f `` nat, leq)
proof -
  have preorder_on(P,leq)
    unfolding preorder_on_def using assms by simp
  {
    fix n m
    assume n∈nat m∈nat
    then
    have f`m ≼ f`n ∨ f`n ≼ f`m
    proof(cases m≤n)
      case True
      with ‹n∈_› ‹m∈_›
      show ?thesis
        using decr_succ_decr[of f n m] assms leI ‹preorder_on(P,leq)› by simp
    next
      case False
      with ‹n∈_› ‹m∈_›
      show ?thesis
      using decr_succ_decr[of f m n] assms leI not_le_iff_lt ‹preorder_on(P,leq)›
by simp
    qed
  }
  then
  show ?thesis
    unfolding linear_def using ball_image_simp assms by auto
qed

end
```

2.2 Towards Rasiowa-Sikorski Lemma (RSL)

```
locale countable_generic = forcing_notion +
  fixes 𝒟
  assumes countable_subs_of_P:  𝒟 ∈ nat→Pow(P)
    and     seq_of_denses:          ∀ n ∈ nat. dense(𝒟`n)

begin

definition
  D_generic :: i⇒o where
  D_generic(G) ≡ filter(G) ∧ (∀ n∈nat.(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)≼ f ` n ∧ f ` succ(n) ∈ 𝒟 ` n
  shows
    ∃ G. p∈G ∧ D_generic(G)
```

proof -
note *assms*
moreover from *this*
have $f``nat \subseteq P$
by (*simp add*:*subset_fun_image*)
moreover from *calculation*
have *refl*($f``nat$, *leq*) $\wedge$ *trans*[*P*](*leq*)
using *leq_preord* **unfolding** *preorder_on_def* **by** (*blast intro*:*refl_monot_domain*)
moreover from *calculation*
have $\forall n{\in}nat.\ \ f\ `\ succ(n) \preceq f\ `\ n$ **by** (*simp*)
moreover from *calculation*
have *linear*($f``nat$, *leq*)
using *leq_preord* **and** *decr_seq_linear* **unfolding** *preorder_on_def* **by** (*blast*)
moreover from *calculation*
have $(\forall p{\in}f``nat.\forall q{\in}f``nat.\ compat\_in(f``nat,leq,p,q))$
using *chain_compat* **by** (*auto*)
ultimately
have *filter*(*upclosure*($f``nat$)) (**is** *filter*(*?G*))
using *closure_compat_filter* **by** *simp*
moreover
have $\forall n{\in}nat.\ \mathcal{D}\ `\ n \cap ?G \neq 0$
proof
fix *n*
assume $n{\in}nat$
with *assms*
have $f`succ(n) \in ?G \wedge f`succ(n) \in \mathcal{D}\ `\ n$
using *aux_RS1* **by** *simp*
then
show $\mathcal{D}\ `\ n \cap ?G \neq 0$ **by** *blast*
qed
moreover from *assms*
have $p \in ?G$
using *aux_RS1* **by** *auto*
ultimately
show *?thesis* **unfolding** *D_generic_def* **by** *auto*
qed

end

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

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

context *countable_generic*
begin

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

lemma *countable_RS_sequence_aux*:
fixes *p enum*
defines $f(n) \equiv RS\_seq(n,P,leq,p,enum,\mathcal{D})$
and $Q(q,k,m) \equiv enum`m \preceq q \wedge enum`m \in \mathcal{D}\ `\ k$
assumes $n \in nat$ $p \in P$ $P \subseteq range(enum)$ $enum : nat \to M$
$\bigwedge x\ k.\ x \in P \Longrightarrow k \in nat \Longrightarrow\ \exists\, q \in P.\ q \preceq x \wedge q \in \mathcal{D}\ `\ k$
shows
$f(succ(n)) \in P \wedge f(succ(n)) \preceq f(n) \wedge f(succ(n)) \in \mathcal{D}\ `\ n$
using ‹$n \in nat$›
proof (*induct*)
case *0*
from *assms*
obtain q **where** $q \in P$ $q \preceq p$ $q \in \mathcal{D}\ `\ 0$ **by** *blast*
moreover from *this* **and** ‹$P \subseteq range(enum)$›
obtain m **where** $m \in nat$ $enum`m = q$
using *preimage_rangeD*[*OF* ‹$enum : nat \to M$›] **by** *blast*
moreover
have $\mathcal{D}`0 \subseteq P$
using *apply_funtype*[*OF countable_subs_of_P*] **by** *simp*
moreover note ‹$p \in P$›
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** $q \in P$ $q \preceq f(succ(n))$ $q \in \mathcal{D}\ `\ succ(n)$ **by** *blast*
moreover from *this* **and** ‹$P \subseteq range(enum)$›
obtain m **where** $m \in nat$ $enum`m \preceq f(succ(n))$ $enum`m \in \mathcal{D}\ `\ succ(n)$
using *preimage_rangeD*[*OF* ‹$enum : nat \to M$›] **by** *blast*
moreover note *succ*
moreover from *calculation*
have $\mathcal{D}`succ(n) \subseteq P$
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 $f \equiv \lambda n \in nat.\ RS\_seq(n,P,leq,p,enum,\mathcal{D})$

and $Q(q,k,m) \equiv enum`m \preceq q \wedge enum`m \in \mathcal{D}\ `\ k$
assumes $n{\in}nat\ p{\in}P\ P \subseteq range(enum)\ enum{:}nat{\to}M$
shows
$f`0 = p\ f`succ(n) \preceq f`n \wedge f`succ(n) \in \mathcal{D}\ `\ n\ f`succ(n) \in P$
proof -
from *assms*
show $f`0 = p$ **by** *simp*
{
fix $x\ k$
assume $x{\in}P\ k{\in}nat$
then
have $\exists\, q{\in}P.\ q \preceq x \wedge q \in \mathcal{D}\ `\ k$
using *seq_of_denses apply_funtype*[*OF countable_subs_of_P*]
unfolding *dense_def* **by** *blast*
}
with *assms*
show $f`succ(n) \preceq f`n \wedge f`succ(n) \in \mathcal{D}\ `\ n\ f`succ(n){\in}P$
unfolding *f_def* **using** *countable_RS_sequence_aux* **by** *simp_all*
qed

lemma *RS_seq_type*:
assumes $n \in nat\ p{\in}P\ P \subseteq range(enum)\ enum{:}nat{\to}M$
shows $RS\_seq(n,P,leq,p,enum,\mathcal{D}) \in P$
using *assms countable_RS_sequence*(*1*,*3*)
by (*induct*;*simp*)

lemma *RS_seq_funtype*:
assumes $p{\in}P\ P \subseteq range(enum)\ enum{:}nat{\to}M$
shows $(\lambda n{\in}nat.\ RS\_seq(n,P,leq,p,enum,\mathcal{D})){:}\ nat \to P$
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 $dc\_witness :: i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow i$
primrec
$wit0 \ : dc\_witness(0,A,a,s,R) = a$
$witrec\ : dc\_witness(succ(n),A,a,s,R) = s`\{x{\in}A.\ \langle dc\_witness(n,A,a,s,R),x\rangle{\in}R\ \}$

lemma *witness_into_A* [*TC*]:
assumes $a{\in}A$
$(\forall X\ .\ X{\neq}0 \wedge X{\subseteq}A \longrightarrow s`X{\in}X)$
$\forall y{\in}A.\ \{x{\in}A.\ \langle y,x\rangle{\in}R\ \} \neq 0\ n{\in}nat$
shows $dc\_witness(n,\ A,\ a,\ s,\ R){\in}A$
using ‹$n{\in}nat$›
proof(*induct n*)
case *0*
then show *?case* **using** ‹$a{\in}A$› **by** *simp*
next
case (*succ x*)
then
show *?case* **using** *assms* **by** *auto*
qed

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

lemma *witness_funtype*:
assumes $a{\in}A$
$(\forall X\ .\ X{\neq}0 \wedge X{\subseteq}A \longrightarrow s`X{\in}X)$
$\forall y{\in}A.\ \{x{\in}A.\ \langle y,x\rangle{\in}R\ \} \neq 0$
shows $(\lambda n{\in}nat.\ dc\_witness(n,\ A,\ a,\ s,\ R)) \in nat \to A$ (**is** *?f* ∈ _ → _)
proof -
have *?f* ∈ *nat* → {$dc\_witness(n,\ A,\ a,\ s,\ R).\ n{\in}nat$} (**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** $a{\in}A$
$(\forall X\ .\ X{\neq}0 \wedge X{\subseteq}A \longrightarrow s`X{\in}X)$
$\forall y{\in}A.\ \{x{\in}A.\ \langle y,x\rangle{\in}R\ \} \neq 0$
shows $\exists f \in nat{\to}A.\ \forall n{\in}nat.\ f`n = dc\_witness(n,A,a,s,R)$

using *assms bexI*[*of* _ $\lambda n{\in}nat.$ *dc_witness*(n,A,a,s,R)] *witness_funtype*
by *simp*

theorem *pointed_DC* :
assumes $(\forall\, x{\in}A.\ \exists\, y{\in}A.\ \langle x,y\rangle{\in}\ R)$
shows $\forall\, a{\in}A.\ (\exists f \in nat{\to}A.\ f`0 = a \wedge (\forall\, n \in nat.\ \langle f`n,f`succ(n)\rangle{\in}R))$
proof -
have *0*:$\forall\, y{\in}A.\ \{x \in A\ .\ \langle y,\ x\rangle \in R\} \neq 0$
using *assms* **by** *auto*
from *AC_func_Pow*[*of A*]
obtain *g*
where *1*: $g \in Pow(A)$ - $\{0\} \to A$
$\forall\, X.\ X \neq 0 \wedge X \subseteq A \longrightarrow g\ `\ X \in X$
by *auto*
let *?f* $=\lambda a.\lambda n{\in}nat.$ *dc_witness*(n,A,a,g,R)
{
fix *a*
assume $a{\in}A$
from ‹$a{\in}A$›
have *f0*: *?f*$(a)`0 = a$ **by** *simp*
with ‹$a{\in}A$›
have $\langle$*?f*$(a)\ `\ n,$ *?f*$(a)\ `\ succ(n)\rangle \in R$ **if** $n{\in}nat$ **for** *n*
using *witness_related*[*OF* ‹$a{\in}A$› *1*(*2*) *0*] *beta that* **by** *simp*
then
have $\exists f{\in}nat \to A.\ f\ `\ 0 = a \wedge (\forall\, n{\in}nat.\ \langle f\ `\ n,\ f\ `\ succ(n)\rangle \in R)$ (**is** $\exists\, x{\in}\_$.*?P*(x))
using *f0 witness_funtype 0 1* ‹$a{\in}\_$› **by** *blast*
}
then show *?thesis* **by** *auto*
qed

lemma *aux_DC_on_AxNat2* : $\forall\, x{\in}A{\times}nat.\ \exists\, y{\in}A.\ \langle x,\langle y,succ(snd(x))\rangle\rangle \in R \Longrightarrow$
$\forall\, x{\in}A{\times}nat.\ \exists\, y{\in}A{\times}nat.\ \langle x,y\rangle \in \{\langle a,b\rangle{\in}R.\ snd(b) = succ(snd(a))\}$
by (*rule ballI*, *erule_tac* $x{=}x$ **in** *ballE*, *simp_all*)

lemma *infer_snd* : $c{\in}\ A{\times}B \Longrightarrow snd(c) = k \Longrightarrow c{=}\langle fst(c),k\rangle$
by *auto*

corollary *DC_on_A_x_nat* :
assumes $(\forall\, x{\in}A{\times}nat.\ \exists\, y{\in}A.\ \langle x,\langle y,succ(snd(x))\rangle\rangle \in R)\ a{\in}A$
shows $\exists f \in nat{\to}A.\ f`0 = a \wedge (\forall\, n \in nat.\ \langle\langle f`n,n\rangle,\langle f`succ(n),succ(n)\rangle\rangle{\in}R)$ (**is** $\exists\, x{\in}\_.$*?P*(x))
proof -
let *?R'*$=\{\langle a,b\rangle{\in}R.\ snd(b) = succ(snd(a))\}$
from *assms*(*1*)
have $\forall\, x{\in}A{\times}nat.\ \exists\, y{\in}A{\times}nat.\ \langle x,y\rangle \in$ *?R'*
using *aux_DC_on_AxNat2* **by** *simp*
with ‹$a{\in}\_$›
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 $p{\in}P$ $n{\in}nat$
shows $\exists\, y{\in}P.\ \langle p,y\rangle \in (\lambda m{\in}nat.\ \{\langle x,y\rangle{\in}P{\times}P.\ y{\preceq}x \wedge y{\in}\mathcal{D}\text{`}(pred(m))\})\text{`}n$
proof -
from *seq_of_denses* ‹$n{\in}nat$›
have $dense(\mathcal{D}\ \text{`}\ pred(n))$ **by** *simp*
with ‹$p{\in}P$›
have $\exists\, d{\in}\mathcal{D}\ \text{`}\ Arith.pred(n).\ d{\preceq}\ p$
unfolding *dense_def* **by** *simp*
then obtain d **where** *3*: $d \in \mathcal{D}\ \text{`}\ Arith.pred(n) \wedge d{\preceq}\ p$
by *blast*
from *countable_subs_of_P* ‹$n{\in}nat$›
have $\mathcal{D}\ \text{`}\ Arith.pred(n) \in Pow(P)$
by (*blast dest*:*apply_funtype intro*:*pred_type*)
then
have $\mathcal{D}\ \text{`}\ Arith.pred(n) \subseteq P$
by (*rule PowD*)
with *3*
have $d \in P \wedge d{\preceq}\ p \wedge d \in \mathcal{D}\ \text{`}\ Arith.pred(n)$
by *auto*
with ‹$p{\in}P$› ‹$n{\in}nat$›
show *?thesis* **by** *auto*
qed

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

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

end

end

5 Auxiliary results on arithmetic

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *pred2_Un*:
assumes $j \in nat\ m \leq j\ n \leq j$
shows $pred(pred(m \cup n)) \leq pred(pred(j))$
using *assms pred_mono*[*of j*] *le_in_nat Un_least_lt pred_mono* **by** *simp*

lemma *nat_union_abs1* :
$[\![\ Ord(i)\ ;\ Ord(j)\ ;\ i \leq j\ ]\!] \Longrightarrow i \cup j = j$
by (*rule Un_absorb1*,*erule le_imp_subset*)

lemma *nat_union_abs2* :
$[\![\ Ord(i)\ ;\ Ord(j)\ ;\ i \leq j\ ]\!] \Longrightarrow j \cup i = j$
by (*rule Un_absorb2*,*erule le_imp_subset*)

lemma *nat_un_max* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow i \cup j = max(i,j)$
using *max_def nat_union_abs1 not_lt_iff_le leI nat_union_abs2*
by *auto*

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

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

by *simp*

lemma *Un_le_compat* : $o \leq p \Longrightarrow q \leq r \Longrightarrow Ord(o) \Longrightarrow Ord(p) \Longrightarrow Ord(q) \Longrightarrow Ord(r) \Longrightarrow o \cup q \leq p \cup r$
using *le_trans*[*of q r p∪r*,*OF _ Un_upper2_le*] *le_trans*[*of o p p∪r*,*OF _ Un_upper1_le*]
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 $m\#-p \in nat$ $m\#-p \#+ p = m$
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*≤*n*›,*of j*<*n* | *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) ≤ *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) ≤ *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*≤*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 ∨ 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 $n ** \alpha ++ x < n ** \beta ++ y$
proof -
consider (0) $\beta=0$ | (s) γ **where** $Ord(\gamma)$ $\beta = succ(\gamma)$ | (l) $Limit(\beta)$
using *Ord_cases*[*OF* ‹$Ord(\beta)$›,*of ?thesis*] **by** *force*
then show *?thesis*
proof *cases*
case *0*
then show *?thesis* **using** ‹$\alpha<\beta$› **by** *auto*
next
case *s*
then
have $\alpha\leq\gamma$ **using** ‹$\alpha<\beta$› **using** *leI* **by** *auto*
then
have $n ** \alpha \leq n ** \gamma$ **using** *omult_le_mono*[*OF* _ ‹$\alpha\leq\gamma$›] ‹$Ord(n)$› **by** *simp*
then
have $n ** \alpha ++ x < n ** \gamma ++ n$ **using** *oadd_lt_mono*[*OF* _ ‹$x<n$›] **by** *simp*
also
have $... = n ** \beta$ **using** ‹$\beta=succ(\_)$› *omult_succ* ‹$Ord(\beta)$› ‹$Ord(n)$› **by** *simp*
finally
have $n ** \alpha ++ x < n ** \beta$ **by** *auto*
then
show *?thesis* **using** *oadd_le_self* ‹$Ord(\beta)$› *lt_trans2* ‹$Ord(n)$› **by** *auto*
next
case *l*
have $Ord(x)$ **using** ‹$x<n$› *lt_Ord* **by** *simp*
with *l*
have $succ(\alpha) < \beta$ **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* _ ‹$Ord(\alpha)$›]] ‹$x<n$›] ‹$Ord(n)$› **by** *simp*
also
have $... = n ** succ(\alpha)$ **using** *omult_succ* ‹$Ord(\alpha)$› ‹$Ord(n)$› **by** *simp*
finally
have $n ** \alpha ++ x < n ** succ(\alpha)$ **by** *simp*
with ‹$succ(\alpha) < \beta$›
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* ‹$Ord(\beta)$› *lt_trans2* ‹$Ord(n)$› **by** *auto*
qed
qed
end

6 Aids to internalize formulas

theory *Internalizations*
imports
ZF-Constructible.DPow_absolute
begin

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

lemma *nth_closed* :
  assumes *0∈A env∈list(A)*
  shows *nth(n,env)∈A*
  using *assms(2,1)* **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

end

7 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 $R^{\widehat{}}*$

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) ⟹ r-‘‘{a} ⊆ tr_down(r,a)
  by(simp add: tr_down_def vimage_mono r_subset_trancl)

lemma tr_down_mono : relation(r) ⟹ x ∈ r-‘‘{a} ⟹ tr_down(r,x) ⊆ 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} ⊆ B and a ∈ B
  shows r-‘‘{a} = (r∩B×B)-‘‘{a}
proof (intro equalityI subsetI)
  fix x
  assume x ∈ r-‘‘{a}
  then
  have x ∈ B using assms by (simp add: subsetD)
  from ‹x∈ r-‘‘{a}›
  have ⟨x,a⟩ ∈ r using underD by simp
  then
  show x ∈ (r∩B×B)-‘‘{a} using ‹x∈B› ‹a∈B› underI by simp
next
  from assms
  show x ∈ r -‘‘ {a} if  x ∈ (r ∩ B×B) -‘‘ {a} for x
    using vimage_mono that by auto
qed

lemma wfrec_restr_eq : r' = r ∩ A×A ⟹ 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 ∈ A ⟹ tr_down(r,a) ⊆ A ⟹ 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*} ⊆ *tr_down*(*r*, *a*) **.**
with *1*
have *r*-‘‘{*a*} ⊆ *A* **by** (*force simp add*: *subset_trans*)
{
fix *x*
assume *x_a* : *x* ∈ *r*-‘‘{*a*}
with ‹*r*-‘‘{*a*} ⊆ *A*›
have *x* ∈ *A* **..**
from *pred_down rr*
have *b* : *r* -‘‘{*x*} ⊆ *tr_down*(*r*,*x*) **.**
then
have *tr_down*(*r*,*x*) ⊆ *tr_down*(*r*,*a*)
using *tr_down_mono x_a rr* **by** *simp*
with *1*
have *tr_down*(*r*,*x*) ⊆ *A* **using** *subset_trans* **by** *force*
have ⟨*x*,*a*⟩ ∈ *r* **using** *x_a underD* **by** *simp*
with *1* ‹*tr_down*(*r*,*x*) ⊆ *A*› ‹*x* ∈ *A*›
have *wfrec*(*r*,*x*,*H*) = *wfrec*[*A*](*r*,*x*,*H*) **by** *simp*
}
then
have *x*∈ *r*-‘‘{*a*} ⟹ *wfrec*(*r*,*x*,*H*) = *wfrec*[*A*](*r*,*x*,*H*) **for** *x* **.**
then
have *Eq1* :(λ *x* ∈ *r*-‘‘{*a*} . *wfrec*(*r*,*x*,*H*)) = (λ *x* ∈ *r*-‘‘{*a*} . *wfrec*[*A*](*r*,*x*,*H*))
using *lam_cong* **by** *simp*

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

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

lemma *wfrec_trans_restr* : *relation*(*r*) ⟹ *wf*(*r*) ⟹ *trans*(*r*) ⟹ *r*-‘‘{*a*}⊆*A* ⟹
a ∈ *A* ⟹

wfrec(*r*, *a*, *H*) = *wfrec*[*A*](*r*, *a*, *H*)
by(*subgoal_tac tr_down*(*r*,*a*) ⊆ *A*,*auto simp add* : *wfrec_restr tr_down_def trancl_eq_r*)

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

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

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

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

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

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

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

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

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

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

lemma *restrict_trancl_Rrel*:
assumes *R*(*w*,*y*)
shows *restrict*(*f*,*Rrel*(*R*,*d*)-‘‘{*y*})‘*w*
= *restrict*(*f*,(*Rrel*(*R*,*d*)^+)-‘‘{*y*})‘*w*
proof (*cases y*∈*d*)
let *?r*=*Rrel*(*R*,*d*) **and** *?s*=(*Rrel*(*R*,*d*))^+
case *True*
show *?thesis*
proof (*cases w*∈*d*)
case *True*
with ‹*y*∈*d*› *assms*
have ⟨*w*,*y*⟩∈*?r*
unfolding *Rrel_def* **by** *blast*
then

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

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

lemma wf_eq_trancl:
  assumes ⋀ f y . H(y,restrict(f,R-‘‘{y})) = H(y,restrict(f,R^+-‘‘{y}))
  shows  wfrec(R, x, H) = wfrec(R^+, x, H) (is wfrec(?r,_,_) = wfrec(?r',_,_))
```

proof -
  have *wfrec*(*R*, *x*, *H*) = *wftrec*(*?r*^+, *x*, λ*y f*. *H*(*y*, *restrict*(*f*,*?r*-‘‘{*y*}))))
    unfolding *wfrec_def* **..**
  also
  have ... = *wftrec*(*?r*^+, *x*, λ*y 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

8 Relativization of the cumulative hierarchy

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

begin

lemma (**in** *M_trivial*) *powerset_abs'* [*simp*]:
  assumes
    M(*x*) *M*(*y*)
  shows
    powerset(*M*,*x*,*y*) ⟷ *y* = {*a*∈*Pow*(*x*) . *M*(*a*)}
  using *powerset_abs assms* **by** *simp*

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*)

lemma (**in** *M_trivial*) *family_union_closed*: ⟦*strong_replacement*(*M*, λ*x y*. *y* = *f*(*x*)); *M*(*A*); ∀ *x*∈*A*. *M*(*f*(*x*))⟧
      ⟹ *M*(⋃*x*∈*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**
$is\_Vset(M,i,V) \equiv \exists\, z[M].\ empty(M,z) \wedge is\_Vfrom(M,z,i,V)$

8.1 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 is_Collect_def powerset_def subset_def*
using *nth_closed assms*
by (*simp*) (*rule sep_rules* | *simp*)+

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

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

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

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

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

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

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

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

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

8.2 Absoluteness results

locale *M_eclose_pow* = *M_eclose* +
assumes
power_ax : *power_ax*(M) **and**
powapply_replacement : $M(f) \Longrightarrow strong\_replacement(M,is\_powapply(M,f))$
and
HVfrom_replacement : $[\![\ M(i)\ ;\ M(A)\ ]\!] \Longrightarrow$
$transrec\_replacement(M,is\_HVfrom(M,A),i)$ **and**
PHrank_replacement : $M(f) \Longrightarrow strong\_replacement(M,PHrank(M,f))$ **and**
is_Hrank_replacement : $M(x) \Longrightarrow wfrec\_replacement(M,is\_Hrank(M),rrank(x))$

begin

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

lemma $[\![M(A);\ M(x);\ M(f);\ M(h)\ ]\!] \Longrightarrow$

is_HVfrom(*M*,*A*,*x*,*f*,*h*) ⟷
(∃ *R*[*M*]. *h* = *A* ∪ ⋃ *R* ∧ *is_Replace*(*M*, *x*,λ*x 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*) ⟷ *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*) ⟧ ⟹ *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*) ⟧ ⟹ *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*,λ*y*.{*x*∈*Pow*(*f*‘*y*). *M*(*x*)})
proof -
have {*y* . *x* ∈ *A*, *M*(*y*) ∧ *y* = {*x* ∈ *Pow*(*f* ‘ *x*) . *M*(*x*)}} = {*y* . *x* ∈ *A*, *y* = {*x* ∈ *Pow*(*f* ‘ *x*) . *M*(*x*)}}
using *assms powapply_closed transM*[*of* _ *A*] **by** *blast*
also
have ... = {{*x* ∈ *Pow*(*f* ‘ *y*) . *M*(*x*)} . *y* ∈ *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*∈*A* ; *is_powapply*(*M*,*f*,*x*,*y*) ⟧ ⟹ *M*(*y*) **for** *x y*

using *assms* **unfolding** *is_powapply_def* **by** *simp*
ultimately
show *?thesis* **using** *assms powapply_replacement* **by** *simp*
qed

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

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

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

lemma *transrec_HVfrom*:
assumes $M(A)$
shows $Ord(i) \Longrightarrow \{x{\in}Vfrom(A,i).\ M(x)\} = transrec(i,HVfrom(M,A))$
proof (*induct rule*:*trans_induct*)
case (*step i*)
have $Vfrom(A,i) = A \cup (\bigcup y{\in}i.\ Pow((\lambda x{\in}i.\ Vfrom(A,\ x))\ `\ y))$
using *def_transrec*[*OF Vfrom_def*, *of A i*] **by** *simp*
then
have $Vfrom(A,i) = A \cup (\bigcup y{\in}i.\ Pow(Vfrom(A,\ y)))$
by *simp*
then
have $\{x{\in}Vfrom(A,i).\ M(x)\} = \{x{\in}A.\ M(x)\} \cup (\bigcup y{\in}i.\ \{x{\in}Pow(Vfrom(A,\ y)).\ M(x)\})$
by *auto*
with ‹$M(A)$›
have $\{x{\in}Vfrom(A,i).\ M(x)\} = A \cup (\bigcup y{\in}i.\ \{x{\in}Pow(Vfrom(A,\ y)).\ M(x)\})$
by (*auto intro*:*transM*)
also
have $\ldots = A \cup (\bigcup y{\in}i.\ \{x{\in}Pow(\{z{\in}Vfrom(A,y).\ M(z)\}).\ M(x)\})$
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 ⋀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(⋃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 $a \in \{x \in Vfrom(0, ?i).\ M(x)\}$ (**is** $a \in ?V$)
using *Vset_Ord_rank_iff* **by** *simp*
moreover from *assms*
have $M(?i)$
using *rank_closed* **by** *simp*
moreover
note ‹$M(a)$›
moreover from *calculation*
have $M(?V)$
using *Vfrom_closed* **by** *simp*
moreover from *calculation*
have $ordinal(M, ?i) \wedge is\_Vfrom(M,\ 0,\ ?i,\ ?V) \wedge a \in ?V$
using *Ord_rank Vfrom_abs* **by** *simp*
ultimately
show *?thesis* **by** *blast*
qed

end
end

9 Automatic synthesis of formulas

theory *Synthetic_Definition*
imports *Utils*
keywords *synthesize* :: *thy_decl* % *ML*
and *synthesize_notc* :: *thy_decl* % *ML*
and *from_schematic*
begin

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

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

fun print_theorem pos (*thms*, *lthy*) =
(*Proof_Display.print_theorem pos lthy thms*; *lthy*)

fun prove_tc_form goal thms ctxt =
Goal.prove ctxt [] [] *goal*
(*fn* {*context* = *ctxt′*, ...} =>
rewrite_goal_tac ctxt′ thms 1
THEN TypeCheck.typecheck_tac ctxt′)

fun prove_sats goal thms thm_auto ctxt =
Goal.prove ctxt [] [] *goal*
(*fn* {*context* = *ctxt′*, ...} =>
let val ctxt″ = *ctxt′* |> *Simplifier.add_simp* (*thm_auto* |> *hd*) *in*
rewrite_goal_tac ctxt″ thms 1

```
      THEN PARALLEL_ALLGOALS (asm_simp_tac ctxt'')
      THEN TypeCheck.typecheck_tac ctxt''
    end)

fun is_mem Const_ ‹mem for _ _› = 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 for Const‹satisfies for set r_tm› env›
    val rhs = Const‹IFOL.eq Type‹i› for sats Const‹succ for Const‹zero›››
    val concl = Const‹iff for 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 |> print_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 for 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 |> print_theorem pos
 end


fun synthetic_def def_name thmref pos tc auto thy =
  let
    val (thm_ref,_) = thmref |>> Facts.ref_name
    val thm = Proof_Context.get_thm thy thm_ref;
    val thm_vars = rev (Term.add_vars (Thm.full_prop_of thm) []);
    val (((_,inst),thm_tms),_) = Variable.import true [thm] thy
```

```
    val vars = map (fn v => (v, the (Vars.lookup inst v))) thm_vars;
    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 relevant ts Const_ ‹mem for t _› = not (Term.is_Free t) orelse
          member (op =) ts (t |> Term.dest_Free |> #1)
      | relevant _ _ = false
    val t_vars = sort_strings (Term.add_free_names t [])
    val vs = filter (member (op =) t_vars o #1 o #1 o #1) vars
    val at = fold_rev (lambda o Thm.term_of o #2) vs t
    val hyps' = filter (relevant t_vars o Utils.dest_trueprop) hyps
  in
    Local_Theory.define ((Binding.name def_name, NoSyn),
                         ((Binding.name (def_name ^ _def), []), 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
synthetic 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

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
    Synthetic_Definition
begin

syntax
  sats :: [*i*, *i*, *i*] ⇒ *o* (‹(, _ ⊨ _)› [*36*,*36*,*36*] *60*)
syntax_consts
  _sats ⇌ *sats*
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_trans* =
fixes *M*
assumes
upair_ax: *upair_ax*(##*M*)
and *Union_ax*: *Union_ax*(##*M*)
and *power_ax*: *power_ax*(##*M*)
and *extensionality*: *extensionality*(##*M*)
and *foundation_ax*: *foundation_ax*(##*M*)
and *infinity_ax*: *infinity_ax*(##*M*)
and *separation_ax*: *φ*∈*formula* ⟹ *env*∈*list*(*M*) ⟹ *arity*(*φ*) ≤ *1* #+ *length*(*env*) ⟹
separation(##*M*,λ*x*. *sats*(*M*,*φ*,[*x*] @ *env*))
and *replacement_ax*: *φ*∈*formula* ⟹ *env*∈*list*(*M*) ⟹ *arity*(*φ*) ≤ *2* #+ *length*(*env*) ⟹
strong_replacement(##*M*,λ*x y*. *sats*(*M*,*φ*,[*x*,*y*] @ *env*))
and *trans_M*: *Transset*(*M*)
begin

lemma *TranssetI* :
(⋀*y* *x*. *y*∈*x* ⟹ *x*∈*M* ⟹ *y*∈*M*) ⟹ *Transset*(*M*)
by (*auto simp add*: *Transset_def*)

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*
with *trans_M* **have** *z=0*
by (*simp add*: *empty_def*, *blast intro*: *Transset_intf*)
with *zm* **show** *?thesis*
by *simp*
qed

10.1 Interface with *M_trivial*

lemma *mtrans* :
M_trans(##M)
using *Transset_intf*[*OF trans_M*] *zero_in_M exI*[*of* λx. x∈M]
by *unfold_locales auto*

lemma *mtriv* :
M_trivial(##M)
using *trans_M M_trivial.intro mtrans M_trivial_axioms.intro upair_ax Union_ax*
by *simp*

end

sublocale *M_ZF_trans* ⊆ *M_trivial* ##M
by (*rule mtriv*)

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 $x\notin b \longleftrightarrow sats(M,dfm(0,1),[x,b])$
if $b\in M$ $x\in M$ **for** b x
using *that fmsats*[*of* $[x,b]$] **by** *simp*
ultimately
have $\forall\, b\in M.\ separation(\#\#M,\ \lambda x\ .\ x\notin b)$
unfolding *separation_def* **by** *simp*
with ‹$B\in M$› **show** *?thesis* **by** *simp*
qed

schematic_goal *cprod_fm_auto*:
assumes
$nth(i,env) = z$ $nth(j,env) = B$ $nth(h,env) = C$
$i \in nat$ $j \in nat$ $h \in nat$ $env \in list(A)$
shows
$(\exists\, x\in A.\ x\in B \wedge (\exists\, y\in A.\ y\in C \wedge pair(\#\#A,x,y,z))) \longleftrightarrow sats(A,?cpfm(i,j,h),env)$
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *cartprod_sep_intf* :
assumes
$A\in M$
and
$B\in M$
shows
$separation(\#\#M,\lambda z.\ \exists\, x\in M.\ x\in A \wedge (\exists\, y\in M.\ y\in B \wedge pair(\#\#M,x,y,z)))$
proof -
obtain *cpfm* **where**
fmsats:$\bigwedge env.\ env\in list(M) \Longrightarrow$
$(\exists\, x\in M.\ x\in nth(1,env) \wedge (\exists\, y\in M.\ y\in nth(2,env) \wedge pair(\#\#M,x,y,nth(0,env))))$
$\longleftrightarrow sats(M,cpfm(0,1,2),env)$
and
$cpfm(0,1,2) \in formula$
and
$arity(cpfm(0,1,2)) = 3$
using *cprod_fm_auto* **by** (*simp del*:*FOL_sats_iff add*: *fm_defs nat_simp_union*)
then
have $\forall\, a\in M.\ \forall\, b\in M.\ separation(\#\#M,\ \lambda z.\ sats(M,cpfm(0,1,2)\ ,\ [z,\ a,\ b]))$
using *separation_ax* **by** *simp*
moreover
have $(\exists\, x\in M.\ x\in a \wedge (\exists\, y\in M.\ y\in b \wedge pair(\#\#M,x,y,z))) \longleftrightarrow sats(M,cpfm(0,1,2),[z,a,b])$
if $a\in M$ $b\in M$ $z\in M$ **for** a b z
using *that fmsats*[*of* $[z,a,b]$] **by** *simp*
ultimately
have $\forall\, a\in M.\ \forall\, b\in M.\ separation(\#\#M,\ \lambda z\ .\ (\exists\, x\in M.\ x\in a\ \wedge\ (\exists\, y\in M.\ y\in b\ \wedge$
$pair(\#\#M,x,y,z))))$
unfolding *separation_def* **by** *simp*
with ‹$A\in M$› ‹$B\in M$› **show** *?thesis* **by** *simp*
qed

schematic_goal *im_fm_auto*:
assumes
nth(*i*,*env*) = *y* *nth*(*j*,*env*) = *r* *nth*(*h*,*env*) = *B*
i ∈ *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_defs 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,λz. ∃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_defs 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:$\bigwedge$*env*. *env*∈*list*(*M*) $\Longrightarrow$
(∃ *x*∈*M*. *x*∈*nth*(1,*env*) & (∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*nth*(0,*env*))))
$\longleftrightarrow$ *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_defs 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*))) $\longleftrightarrow$
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*)
$\longleftrightarrow$ *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:$\bigwedge$*env*. *env*∈*list*(*M*) $\Longrightarrow$
(∃ *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*))

$\longleftrightarrow$ *sats*(*M*,*cfm*(*0*,*1*,*2*),*env*)
and
cfm(*0*,*1*,*2*) $\in$ *formula*
and
arity(*cfm*(*0*,*1*,*2*)) = *3*
using *comp_fm_auto* **by** (*simp del*:*FOL_sats_iff pair_abs add*: *fm_defs nat_simp_union*)
then
have $\forall$ *r*$\in$*M*. $\forall$ *s*$\in$*M*. *separation*(*##M*, λ*y*. *sats*(*M*,*cfm*(*0*,*1*,*2*) , [*y*,*s*,*r*]))
using *separation_ax* **by** *simp*
moreover
have ($\exists$ *x*$\in$*M*. $\exists$ *y*$\in$*M*. $\exists$ *z*$\in$*M*. $\exists$ *xy*$\in$*M*. $\exists$ *yz*$\in$*M*.
pair(*##M*,*x*,*z*,*xz*) & *pair*(*##M*,*x*,*y*,*xy*) & *pair*(*##M*,*y*,*z*,*yz*) & *xy*$\in$*s* & *yz*$\in$*r*)
$\longleftrightarrow$ *sats*(*M*,*cfm*(*0*,*1*,*2*) , [*xz*,*s*,*r*])
if *xz*$\in$*M* *s*$\in$*M* *r*$\in$*M* **for** *xz s r*
using *that fmsats*[*of* [*xz*,*s*,*r*]] **by** *simp*
ultimately
have $\forall$ *s*$\in$*M*. $\forall$ *r*$\in$*M*. *separation*(*##M*, λ*xz* . $\exists$ *x*$\in$*M*. $\exists$ *y*$\in$*M*. $\exists$ *z*$\in$*M*. $\exists$ *xy*$\in$*M*. $\exists$ *yz*$\in$*M*.
pair(*##M*,*x*,*z*,*xz*) & *pair*(*##M*,*x*,*y*,*xy*) & *pair*(*##M*,*y*,*z*,*yz*) & *xy*$\in$*s* & *yz*$\in$*r*)
unfolding *separation_def* **by** *simp*
with ‹*S*$\in$*M*› ‹*R*$\in$*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 $\in$ *nat* *j* $\in$ *nat* *h* $\in$ *nat* *env* $\in$ *list*(*A*)
shows
($\exists$ *p*$\in$*A*. *p*$\in$*R* & *pair*(*##A*,*y*,*X*,*p*)) $\longleftrightarrow$ *sats*(*A*,*?pfm*(*i*,*j*,*h*),*env*)
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *pred_sep_intf*:
assumes
R$\in$*M*
and
X$\in$*M*
shows
separation(*##M*, λ*y*. $\exists$ *p*$\in$*M*. *p*$\in$*R* & *pair*(*##M*,*y*,*X*,*p*))
proof -
obtain *pfm* **where**
fmsats:$\bigwedge$*env*. *env*$\in$*list*(*M*) $\Longrightarrow$
($\exists$ *p*$\in$*M*. *p*$\in$*nth*(*1*,*env*) & *pair*(*##M*,*nth*(*0*,*env*),*nth*(*2*,*env*),*p*)) $\longleftrightarrow$ *sats*(*M*,*pfm*(*0*,*1*,*2*),*env*)
and
pfm(*0*,*1*,*2*) $\in$ *formula*

```
    and
    arity(pfm(0,1,2)) = 3
   using pred_fm_auto by (simp del:FOL_sats_iff pair_abs add: fm_defs nat_simp_union)
  then
  have ∀ x∈M. ∀ r∈M. separation(##M, λy. sats(M,pfm(0,1,2) , [y,r,x]))
    using separation_ax by simp
  moreover
  have (∃ p∈M. p∈r & pair(##M,y,x,p))
        ⟷ sats(M,pfm(0,1,2) , [y,r,x])
    if y∈M r∈M x∈M for y x r
    using that fmsats[of [y,r,x]] by simp
  ultimately
  have ∀ x∈M. ∀ r∈M. separation(##M, λ y . ∃ p∈M. p∈r & pair(##M,y,x,p))
    unfolding separation_def by simp
  with ‹X∈M› ‹R∈M› show ?thesis by simp
qed


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

lemma memrel_sep_intf:
  separation(##M, λz. ∃ x∈M. ∃ y∈M. pair(##M,x,y,z) & x ∈ y)
proof -
  obtain mfm where
    fmsats:⋀env. env∈list(M) ⟹
   (∃ x∈M. ∃ y∈M. pair(##M,x,y,nth(0,env)) & x ∈ y) ⟷ sats(M,mfm(0),env)
    and
    mfm(0) ∈ formula
    and
    arity(mfm(0)) = 1
   using mem_fm_auto by (simp del:FOL_sats_iff pair_abs add: fm_defs 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_defs 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 $\forall\, a1{\in}M.\ \forall\, a2{\in}M.\ \forall\, a3{\in}M.\ \forall\, a4{\in}M.\ \forall\, a5{\in}M.\ separation(\#\#M,\ \lambda\ x\ .$
$\exists\, xa{\in}M.\ \exists\, xb{\in}M.\ pair(\#\#M,x,a4,xa)\ \&\ xa \in a1\ \&\ pair(\#\#M,x,a5,xb)$
$\&\ xb \in a1\ \&$
$(\exists\, fx{\in}M.\ \exists\, gx{\in}M.\ fun\_apply(\#\#M,a2,x,fx)\ \&\ fun\_apply(\#\#M,a3,x,gx)$
$\&\ fx \neq gx))$
unfolding *separation_def* **by** *simp*
with ‹$r{\in}M$› ‹$f{\in}M$› ‹$g{\in}M$› ‹$a{\in}M$› ‹$b{\in}M$› **show** *?thesis* **by** *simp*
qed

schematic_goal *funsp_fm_auto*:
assumes
$nth(i,env) = p\ nth(j,env) = z\ nth(h,env) = n$
$i \in nat\ j \in nat\ h \in nat\ env \in list(A)$
shows
$(\exists\, f{\in}A.\ \exists\, b{\in}A.\ \exists\, nb{\in}A.\ \exists\, cnbf{\in}A.\ pair(\#\#A,f,b,p)\ \&\ pair(\#\#A,n,b,nb)\ \&$
$is\_cons(\#\#A,nb,f,cnbf)\ \&$
$upair(\#\#A,cnbf,cnbf,z)) \longleftrightarrow sats(A,?fsfm(i,j,h),env)$
by (*insert assms* ; (*rule sep_rules* | *simp*)+)

lemma *funspace_succ_rep_intf* :
assumes
$n{\in}M$
shows
$strong\_replacement(\#\#M,$
$\lambda p\ z.\ \exists\, f{\in}M.\ \exists\, b{\in}M.\ \exists\, nb{\in}M.\ \exists\, cnbf{\in}M.$
$pair(\#\#M,f,b,p)\ \&\ pair(\#\#M,n,b,nb)\ \&\ is\_cons(\#\#M,nb,f,cnbf)$
$\&$
$upair(\#\#M,cnbf,cnbf,z))$
proof -
obtain *fsfm* **where**
$fmsats{:}env{\in}list(M) \Longrightarrow$
$(\exists\, f{\in}M.\ \exists\, b{\in}M.\ \exists\, nb{\in}M.\ \exists\, cnbf{\in}M.\ pair(\#\#M,f,b,nth(0,env))\ \&\ pair(\#\#M,nth(2,env),b,nb)$
$\&\ is\_cons(\#\#M,nb,f,cnbf)\ \&\ upair(\#\#M,cnbf,cnbf,nth(1,env)))$
$\longleftrightarrow sats(M,fsfm(0,1,2),env)$
and $fsfm(0,1,2) \in formula$ **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_defs nat_simp_union)
then
have $\forall\, n0{\in}M.\ strong\_replacement(\#\#M,\ \lambda p\ z.\ sats(M,fsfm(0,1,2)\ ,\ [p,z,n0]))$
using *replacement_ax* **by** *simp*
moreover
have $(\exists\, f{\in}M.\ \exists\, b{\in}M.\ \exists\, nb{\in}M.\ \exists\, cnbf{\in}M.\ pair(\#\#M,f,b,p)\ \&\ pair(\#\#M,n0,b,nb)$
$\&$
$is\_cons(\#\#M,nb,f,cnbf)\ \&\ upair(\#\#M,cnbf,cnbf,z))$
$\longleftrightarrow sats(M,fsfm(0,1,2)\ ,\ [p,z,n0])$

if $p{\in}M$ $z{\in}M$ $n0{\in}M$ **for** p z $n0$
using *that fmsats*[*of* [p,z,$n0$]] **by** *simp*
ultimately
have $\forall\, n0{\in}M.$ *strong_replacement*($\#\#M$, λ p z.
$\exists\, f{\in}M.\ \exists\, b{\in}M.\ \exists\, nb{\in}M.\ \exists\, cnbf{\in}M.$ *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 ‹$n{\in}M$› **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

lemma *mbasic* : *M_basic*($\#\#M$)
using *trans_M zero_in_M power_ax M_basic_sep_instances funspace_succ_rep_intf mtriv*
by *unfold_locales auto*

end

sublocale *M_ZF_trans* $\subseteq$ *M_basic* $\#\#M$
by (*rule mbasic*)

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
$i \in nat$ $j \in nat$ $k \in nat$ *env* $\in$ *list*(A)
shows
rtran_closure_mem($\#\#A$,B,r,p) $\longleftrightarrow$ *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
$r{\in}M$
and
$A{\in}M$
shows
separation ($\#\#M$, *rtran_closure_mem*($\#\#M$,A,r))
proof -

obtain *rcfm* **where**
fmsats:$\bigwedge$*env*. *env*∈*list*(*M*) $\Longrightarrow$
(*rtran_closure_mem*(##*M*,*nth*(2,*env*),*nth*(1,*env*),*nth*(0,*env*))) $\longleftrightarrow$ *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_defs nat_simp_union*)
then
have ∀ *x*∈*M*. ∀ *a*∈*M*. *separation*(##*M*, λ*y*. *sats*(*M*,*rcfm*(0,1,2) , [*y*,*x*,*a*]))
using *separation_ax* **by** *simp*
moreover
have (*rtran_closure_mem*(##*M*,*a*,*x*,*y*))
$\longleftrightarrow$ *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*) $\longleftrightarrow$ *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*) $\longleftrightarrow$ *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*) $\longleftrightarrow$ *sats*(*A*,*?wtf*(*i*,*j*,*k*),*env*)

unfolding *wellfounded_trancl_def*
by (*insert assms* ; (*rule sep_rules trans_closure_fm_iff_sats* | *simp*)+)

lemma (**in** *M_ZF_trans*) *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_defs trans_closure_fm_def*
by (*simp del*:*FOL_sats_iff pair_abs add*: *fm_defs 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

lemma (**in** *M_ZF_trans*) *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 (**in** *M_ZF_trans*) *nat_subset_I'* :
⟦ *I*∈*M* ; *0*∈*I* ; ⋀*x*. *x*∈*I* ⟹ *succ*(*x*)∈*I* ⟧ ⟹ *nat* ⊆ *I*
by (*rule subsetI*,*induct_tac x*,*simp*+)

lemma (**in** *M_ZF_trans*) *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 *Transset_intf*[*OF trans_M*] **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 (**in** *M_ZF_trans*) *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

lemma (**in** *M_ZF_trans*) *mtrancl* : *M_trancl*(##*M*)
using *mbasic rtrancl_separation_intf wftrancl_separation_intf nat_in_M*
wellfounded_trancl_def
by *unfold_locales auto*

sublocale *M_ZF_trans* ⊆ *M_trancl* ##*M*

by (*rule mtrancl*)

10.4 Interface with *M_eclose*

lemma *repl_sats*:
assumes
sat:$\bigwedge x\ z.\ x{\in}M \Longrightarrow z{\in}M \Longrightarrow sats(M,\varphi,Cons(x,Cons(z,env))) \longleftrightarrow P(x,z)$
shows
strong_replacement$(\#\#M,\lambda x\ z.\ sats(M,\varphi,Cons(x,Cons(z,env)))) \longleftrightarrow$
strong_replacement$(\#\#M,P)$
by (*rule strong_replacement_cong*,*simp add*:*sat*)

lemma (**in** *M_ZF_trans*) *nat_trans_M* :
$n{\in}M$ **if** $n{\in}nat$ **for** n
using *that nat_in_M Transset_intf*[*OF trans_M*] **by** *simp*

lemma (**in** *M_ZF_trans*) *list_repl1_intf*:
assumes
$A{\in}M$
shows
iterates_replacement$(\#\#M,\ is\_list\_functor(\#\#M,A),\ 0)$
proof -
{
fix n
assume $n{\in}nat$
have $succ(n){\in}M$
using ‹$n{\in}nat$› *nat_trans_M* **by** *simp*
then have 1:$Memrel(succ(n)){\in}M$
using ‹$n{\in}nat$› *Memrel_closed* **by** *simp*
have $0{\in}M$
using *nat_0I nat_trans_M* **by** *simp*
then have *is_list_functor*$(\#\#M,\ A,\ a,\ b)$
$\longleftrightarrow sats(M,\ list\_functor\_fm(13,1,0),\ [b,a,c,d,a0,a1,a2,a3,a4,y,x,z,Memrel(succ(n)),A,0])$
if $a{\in}M\ b{\in}M\ c{\in}M\ d{\in}M\ a0{\in}M\ a1{\in}M\ a2{\in}M\ a3{\in}M\ a4{\in}M\ y{\in}M\ x{\in}M\ z{\in}M$
for *a b c d a0 a1 a2 a3 a4 y x z*
using *that* 1 ‹$A{\in}M$› *list_functor_iff_sats* **by** *simp*
then have *sats*$(M,\ iterates\_MH\_fm(list\_functor\_fm(13,1,0),10,2,1,0),\ [a0,a1,a2,a3,a4,y,x,z,Memrel(su$
$\longleftrightarrow iterates\_MH(\#\#M,is\_list\_functor(\#\#M,A),0,a2,\ a1,\ a0)$
if $a0{\in}M\ a1{\in}M\ a2{\in}M\ a3{\in}M\ a4{\in}M\ y{\in}M\ x{\in}M\ z{\in}M$
for *a0 a1 a2 a3 a4 y x z*
using *that sats_iterates_MH_fm*[*of M is_list_functor*$(\#\#M,A)$ _] 1 ‹$0{\in}M$›
‹$A{\in}M$› **by** *simp*
then have 2:*sats*$(M,\ is\_wfrec\_fm(iterates\_MH\_fm(list\_functor\_fm(13,1,0),10,2,1,0),3,1,0),$
$[y,x,z,Memrel(succ(n)),A,0])$
$\longleftrightarrow$
$is\_wfrec(\#\#M,\ iterates\_MH(\#\#M,is\_list\_functor(\#\#M,A),0)\ ,\ Mem$-
$rel(succ(n)),\ x,\ y)$
if $y{\in}M\ x{\in}M\ z{\in}M$ **for** $y\ x\ z$
using *that sats_is_wfrec_fm* 1 ‹$0{\in}M$› ‹$A{\in}M$› **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 iterates_MH_fm_def is_wfrec_fm_def is_recfun_fm_def is_nat_case_fm_def
       restriction_fm_def list_functor_fm_def number1_fm_def cartprod_fm_def
       sum_fm_def quasinat_fm_def pre_image_fm_def fm_defs
      by (simp add:nat_simp_union)
    then
    have strong_replacement(##M,λx 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,λx z.
       ∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, iterates_MH(##M,is_list_functor(##M,A),0)
,
                Memrel(succ(n)), x, y))
     using repl_sats[of M ?f [Memrel(succ(n)),A,0]]  satsf by (simp del:pair_abs)
  }
  then
   show ?thesis unfolding iterates_replacement_def wfrec_replacement_def by
simp
qed




lemma (in M_ZF_trans) iterates_repl_intf :
  assumes
    v∈M and
    isfm:is_F_fm ∈ formula and
    arty:arity(is_F_fm)=2 and
    satsf: ⋀a b env'. ⟦ a∈M ; b∈M ; env'∈list(M) ⟧
           ⟹ is_F(a,b) ⟷ sats(M, is_F_fm, [b,a]@env')
  shows
    iterates_replacement(##M,is_F,v)
proof -
  {
    fix n
    assume n∈nat
    have succ(n)∈M
      using ‹n∈nat› nat_trans_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 iterates_MH_fm_def is_wfrec_fm_def is_recfun_fm_def is_nat_case_fm_def
     restriction_fm_def pre_image_fm_def quasinat_fm_def fm_defs
    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_trans_M* **by** *simp*
have *1*:*arity*(*formula_functor_fm*(*1*,*0*)) = *2*
unfolding *formula_functor_fm_def fm_defs sum_fm_def cartprod_fm_def number1_fm_def*
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*,λ*l'* *t*. *is_tl*(*##M*,*l'*,*t*),*l*)
proof -
have *1*:*arity*(*tl_fm*(*1*,*0*)) = *2*
unfolding *tl_fm_def fm_defs quasilist_fm_def Cons_fm_def Nil_fm_def Inr_fm_def number1_fm_def*
Inl_fm_def **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 *big_union_fm_def fm_defs* **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_trans_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 *is_iterates_fm_def restriction_fm_def list_functor_fm_def number1_fm_def Memrel_fm_def*
cartprod_fm_def sum_fm_def quasinat_fm_def pre_image_fm_def fm_defs is_wfrec_fm_def
is_recfun_fm_def iterates_MH_fm_def is_nat_case_fm_def
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_trans_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 is_iterates_fm_def formula_functor_fm_def fm_defs sum_fm_def
quasinat_fm_def
      cartprod_fm_def number1_fm_def Memrel_fm_def ordinal_fm_def trans-
set_fm_def
    is_wfrec_fm_def is_recfun_fm_def iterates_MH_fm_def is_nat_case_fm_def
subset_fm_def
     pre_image_fm_def restriction_fm_def
    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 is_iterates_fm_def formula_functor_fm_def fm_defs sum_fm_def
quasinat_fm_def
       cartprod_fm_def number1_fm_def Memrel_fm_def ordinal_fm_def trans-
set_fm_def
    is_wfrec_fm_def is_recfun_fm_def iterates_MH_fm_def is_nat_case_fm_def
subset_fm_def
      pre_image_fm_def restriction_fm_def
    by (simp add:nat_simp_union)
  then
  have strong_replacement(##M,λn 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

lemma (in M_ZF_trans) mdatatypes : M_datatypes(##M)
  using  mtrancl list_repl1_intf list_repl2_intf formula_repl1_intf
    formula_repl2_intf nth_repl_intf
  by unfold_locales auto

sublocale M_ZF_trans ⊆ M_datatypes ##M
  by (rule mdatatypes)

lemma (in M_ZF_trans) meclose : M_eclose(##M)
  using mdatatypes eclose_repl1_intf eclose_repl2_intf
  by unfold_locales auto

sublocale M_ZF_trans ⊆ M_eclose ##M
  by (rule meclose)




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*

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 is_Collect_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_defs nat_simp_union*)
then
have ∀*f0*∈*M*. *strong_replacement*(##*M*, λ*p 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* [*simp*]:
⟦ *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_defs nat_simp_union*)
then
have ∀*f0*∈*M*. *strong_replacement*(##*M*, λ*p 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*
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* [*simp*]:
⟦ *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*M*)⟧

$\Longrightarrow$ *sats*(*M*,*is_Hrank_fm*(*x*,*y*,*z*),*env*) $\longleftrightarrow$
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*

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*) $\longleftrightarrow$
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*
then
have
1:*sats*(*M*, *is_wfrec_fm*(*is_Hrank_fm*(*2*,*1*,*0*),*3*,*1*,*0*),[*y*,*x*,*z*,*rrank*(*X*)])
$\longleftrightarrow$ *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*
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*)])
$\longleftrightarrow$ ($\exists$ *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 *is_wfrec_fm_def is_recfun_fm_def is_nat_case_fm_def is_Hrank_fm_def PHrank_fm_def*
restriction_fm_def list_functor_fm_def number1_fm_def cartprod_fm_def
sum_fm_def quasinat_fm_def pre_image_fm_def fm_defs
by (*simp add*:*nat_simp_union*)
then
have *strong_replacement*(*##M*,λ*x z*. *sats*(*M*,*?f*,[*x*,*z*,*rrank*(*X*)]))
using *replacement_ax 1* ‹*X*∈*M*› *rrank_in_M* **by** *simp*
then
have *strong_replacement*(*##M*,λ*x z*.
$\exists$ *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 :: $[i,i,i,i] \Rightarrow i$ **where**
is_HVfrom_fm$(A,x,f,h) \equiv$ *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)))))

lemma *is_HVfrom_type* [*TC*]:
$[\![\ A{\in}nat;\ x \in nat;\ f \in nat;\ h \in nat\ ]\!] \Longrightarrow$ *is_HVfrom_fm*$(A,x,f,h) \in$ *formula*
by (*simp add*:*is_HVfrom_fm_def*)

lemma *sats_is_HVfrom_fm* :
$[\![\ a{\in}nat;\ x \in nat;\ f \in nat;\ h \in nat;\ env \in list(A);\ 0{\in}A]\!]$
$\Longrightarrow$ *sats*$(A,$*is_HVfrom_fm*$(a,x,f,h),env) \longleftrightarrow$
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*$(h,env) = hh$
$a{\in}nat\ x{\in}nat\ f{\in}nat\ h{\in}nat\ env{\in}list(A)\ 0{\in}A$
shows
is_HVfrom$(\#\#A,aa,xx,ff,hh) \longleftrightarrow$ *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
$i{\in}nat\ v{\in}nat\ env{\in}list(A)\ 0{\in}A$
$i <$ *length*$(env)\ v <$ *length*(env)
shows
is_Vset$(\#\#A,$*nth*$(i,$ $env),$*nth*$(v,$ $env))$
$\longleftrightarrow$ *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*$(v,env) = vv$
$i{\in}nat\ v{\in}nat\ env{\in}list(A)\ 0{\in}A$
$i <$ *length*$(env)\ v <$ *length*(env)
shows
is_Vset$(\#\#A,ii,vv) \longleftrightarrow$ *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 is_HVfrom_fm_def is_wfrec_fm_def is_recfun_fm_def is_nat_case_fm_def
       restriction_fm_def list_functor_fm_def number1_fm_def cartprod_fm_def
          is_powapply_fm_def sum_fm_def quasinat_fm_def pre_image_fm_def
fm_defs
      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

lemma (**in** *M_ZF_trans*) *meclose_pow* : *M_eclose_pow*(*##M*)
using *meclose power_ax powapply_repl phrank_repl trans_repl_HVFrom wfrec_rank*
by *unfold_locales auto*

sublocale *M_ZF_trans* ⊆ *M_eclose_pow ##M*
by (*rule meclose_pow*)

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*, λ*x 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*, λ*x 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* λ*x 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*) λ*x 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*,λ*x*. *sats*(*M*,φ,[*x*] @ *env*))
using *assms separation_ax* **by** *simp*
then show *?thesis* **using**
‹*A*∈*M*› *satsQ trans_M*
separation_cong[*of* ##*M* λ*y*. *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_ZF* =
fixes *M*
assumes
upair_ax: *upair_ax*(##*M*)
and *Union_ax*: *Union_ax*(##*M*)
and *power_ax*: *power_ax*(##*M*)
and *extensionality*: *extensionality*(##*M*)
and *foundation_ax*: *foundation_ax*(##*M*)
and *infinity_ax*: *infinity_ax*(##*M*)
and *separation_ax*: φ∈*formula* ⟹ *env*∈*list*(*M*) ⟹ *arity*(φ) ≤ *1* #+ *length*(*env*) ⟹
separation(##*M*,λ*x*. *sats*(*M*,φ,[*x*] @ *env*))
and *replacement_ax*: φ∈*formula* ⟹ *env*∈*list*(*M*) ⟹ *arity*(φ) ≤ *2* #+ *length*(*env*) ⟹
strong_replacement(##*M*,λ*x y*. *sats*(*M*,φ,[*x*,*y*] @ *env*))

locale *M_ctm* = *M_ZF* +
fixes *enum*
assumes *M_countable*: *enum*∈*bij*(*nat*,*M*)
and *trans_M*: *Transset*(*M*)

begin
interpretation *intf*: *M_ZF_trans M*
  using *M_ZF_trans.intro*
    trans_M upair_ax Union_ax power_ax extensionality
    foundation_ax infinity_ax separation_ax[*simplified*]
    replacement_ax[*simplified*]
  by *simp*

lemmas *transitivity* = *Transset_intf*[*OF trans_M*]

lemma *zero_in_M*: *0* ∈ *M*
  by (*rule intf.zero_in_M*)

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

lemma *nat_in_M* : *nat* ∈ *M*
  by (*rule intf.nat_in_M*)

lemma *n_in_M* : *n*∈*nat* ⟹ *n*∈*M*
  using *nat_in_M transitivity* **by** *simp*

lemma *mtriv*: *M_trivial*(##*M*)
  by (*rule intf.mtriv*)

lemma *mtrans*: *M_trans*(##*M*)
  by (*rule intf.mtrans*)

lemma *mbasic*: *M_basic*(##*M*)
  by (*rule intf.mbasic*)

lemma *mtrancl*: *M_trancl*(##*M*)
  by (*rule intf.mtrancl*)

lemma *mdatatypes*: *M_datatypes*(##*M*)
  by (*rule intf.mdatatypes*)

lemma *meclose*: *M_eclose*(##*M*)
  by (*rule intf.meclose*)

lemma *meclose_pow*: *M_eclose_pow*(##*M*)
  by (*rule intf.meclose_pow*)

end

sublocale *M_ctm* ⊆ *M_trivial* ##*M*
by (*rule mtriv*)

sublocale *M_ctm* ⊆ *M_trans* ##*M*
by (*rule mtrans*)

sublocale *M_ctm* ⊆ *M_basic* ##*M*
by (*rule mbasic*)

sublocale *M_ctm* ⊆ *M_trancl* ##*M*
by (*rule mtrancl*)

sublocale *M_ctm* ⊆ *M_datatypes* ##*M*
by (*rule mdatatypes*)

sublocale *M_ctm* ⊆ *M_eclose* ##*M*
by (*rule meclose*)

sublocale *M_ctm* ⊆ *M_eclose_pow* ##*M*
by (*rule meclose_pow*)

context *M_ctm*
begin

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*[*of z A*] **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* λ*y*. *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,λx. Q(x,y,z))∈M
proof -
  have z∈A ⟹ z∈M for z
    using ‹A∈M› transitivity[of z A] by simp
  then
  have 1:Collect(A,λx. is_Q(##M,x,y,z)) = Collect(A,λx. Q(x,y,z))
    using Qabs Collect_cong[of A A λx. is_Q(##M,x,y,z) λx. Q(x,y,z)] by simp
  have separation(##M,λx. is_Q(##M,x,y,z))
    using separation_ax Qsats Qarty Qfm params_M
      separation_cong[of ##M λx. sats(M,Q_fm,[x,y,z]) λx. is_Q(##M,x,y,z)]
    by simp
  then
  have Collect(A,λx. 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,λx. Q(x,a1,a2,a3,a4))∈M
proof -
  have z∈A ⟹ z∈M for z
    using ‹A∈M› transitivity[of z A] by simp
  then
 have 1:Collect(A,λx. is_Q(##M,x,a1,a2,a3,a4)) = Collect(A,λx. 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*,λ*x*. *is_Q*(*##M*,*x*,*a1*,*a2*,*a3*,*a4*))
using *separation_ax Qsats Qarty Qfm params_M*
separation_cong[*of ##M* λ*x*. *sats*(*M*,*Q_fm*,[*x*,*a1*,*a2*,*a3*,*a4*])
λ*x*. *is_Q*(*##M*,*x*,*a1*,*a2*,*a3*,*a4*)]
by *simp*
then
have *Collect*(*A*,λ*x*. *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*, λ*x 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*, λ*x y*. *y* = *f*(*x*))
using *fsats fabs*
strong_replacement_cong[*of ##M* λ*x y*. *sats*(*M*,*f_fm*,[*x*,*y*]@*env*) λ*x 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* : $Transset(M) \Longrightarrow y \in M \Longrightarrow y \subseteq M$
by (*unfold Transset_def*, *blast*)

lemmas $P\_sub\_M = transD[OF\ trans\_M\ P\_in\_M]$

definition
$M\_generic :: i \Rightarrow o$ **where**
$M\_generic(G) \equiv filter(G) \wedge (\forall\, D{\in}M.\ D{\subseteq}P \wedge dense(D) \longrightarrow D \cap G \neq 0)$

lemma *M_genericD* [*dest*]: $M\_generic(G) \Longrightarrow x{\in}G \Longrightarrow x{\in}P$
unfolding *M_generic_def* **by** (*blast dest*:*filterD*)

lemma *M_generic_leqD* [*dest*]: $M\_generic(G) \Longrightarrow p{\in}G \Longrightarrow q{\in}P \Longrightarrow p \preceq q \Longrightarrow q{\in}G$
unfolding *M_generic_def* **by** (*blast dest*:*filter_leqD*)

lemma *M_generic_compatD* [*dest*]: $M\_generic(G) \Longrightarrow p{\in}G \Longrightarrow r{\in}G \Longrightarrow \exists\, q{\in}G.\ q \preceq p \wedge q \preceq r$
unfolding *M_generic_def* **by** (*blast dest*:*low_bound_filter*)

lemma *M_generic_denseD* [*dest*]: $M\_generic(G) \Longrightarrow dense(D) \Longrightarrow D{\subseteq}P \Longrightarrow D{\in}M \Longrightarrow \exists\, q{\in}G.\ q{\in}D$
unfolding *M_generic_def* **by** *blast*

lemma *G_nonempty*: $M\_generic(G) \Longrightarrow G{\neq}0$
proof -
have $P{\subseteq}P$ **..**
assume
$M\_generic(G)$
with *P_in_M P_dense* ‹$P{\subseteq}P$› **show**
$G \neq 0$
unfolding *M_generic_def* **by** *auto*
qed

lemma *one_in_G* :
assumes $M\_generic(G)$
shows $one \in G$
proof -
from *assms* **have** $G{\subseteq}P$
unfolding *M_generic_def* **and** *filter_def* **by** *simp*
from ‹$M\_generic(G)$› **have** $increasing(G)$
unfolding *M_generic_def* **and** *filter_def* **by** *simp*
with ‹$G{\subseteq}P$› **and** ‹$M\_generic(G)$›
show *?thesis*
using *G_nonempty* **and** *one_in_P* **and** *one_max*

```
    unfolding increasing_def by blast
qed

lemma G_subset_M: M_generic(G) ⟹ 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


lemma compat_in_abs :
  assumes
    A∈M r∈M p∈M q∈M
  shows
    is_compat_in(##M,A,r,p,q) ⟷ compat_in(A,r,p,q)
proof -
  have d∈A ⟹ d∈M for d
    using transitivity ‹A∈M› by simp
  moreover from this
  have d∈A ⟹ ⟨d, t⟩ ∈ M if t∈M for t d
    using that pair_in_M_iff by simp
  ultimately
  show ?thesis
    unfolding is_compat_in_def compat_in_def
    using assms pair_in_M_iff transitivity by auto
qed

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*:
$Union\_ax(\#\#A) \longleftrightarrow (A, [] \models ?zfunion)$
unfolding *Union_ax_def*
by ((*rule sep_rules* | *simp*)+)

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

schematic_goal *ZF_power_auto*:
$power\_ax(\#\#A) \longleftrightarrow (A, [] \models ?zfpow)$
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*:
$upair\_ax(\#\#A) \longleftrightarrow (A, [] \models ?zfpair)$
unfolding *upair_ax_def*
by ((*rule sep_rules* | *simp*)+)

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

schematic_goal *ZF_foundation_auto*:
$foundation\_ax(\#\#A) \longleftrightarrow (A, [] \models ?zfpow)$
unfolding *foundation_ax_def*
by ((*rule sep_rules* | *simp*)+)

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

schematic_goal *ZF_extensionality_auto*:
$extensionality(\#\#A) \longleftrightarrow (A, [] \models ?zfpow)$
unfolding *extensionality_def*
by ((*rule sep_rules* | *simp*)+)

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

schematic_goal *ZF_infinity_auto*:
$infinity\_ax(\#\#A) \longleftrightarrow (A, [] \models (?\varphi(i,j,h)))$

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*›)
syntax_consts
  _choice ⇌ *ZF_choice_fm*
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
$n\in nat \Longrightarrow ms \in list(M) \Longrightarrow$
$M,\ ms \models (Forall^{\frown}n(\varphi)) \longleftrightarrow$
$(\forall\, rest \in list(M).\ length(rest) = n \longrightarrow M,\ rest\ @\ ms \models \varphi)$
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∈nat* **for** *n P*
using *that last_init_eq* **by** *force*
from *this[of _ λrest _. (M, rest @ ms ⊨ φ)] ‹n∈nat›*
have $(\forall\, rest\in list(M).\ length(rest) = succ(n) \longrightarrow M,\ rest\ @\ ms \models \varphi) \longleftrightarrow$
$(\forall\, t\in M.\ \forall\, res\in list(M).\ length(res) = n \longrightarrow\ M,\ (res\ @\ [t])\ @\ ms \models \varphi)$
by *simp*
with *assms succ(1,3) succ(2)[of Cons(_,ms)]*
show *?case*
using *arity_sats_iff[of φ _ M Cons(_, ms @ _)] app_assoc*
by (*simp*)
qed

definition
sep_body_fm :: $i \Rightarrow i$ **where**
sep_body_fm(p) ≡ *Forall(Exists(Forall(*
Iff(Member(0,1),And(Member(0,2),
incr_bv1^2(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**
$ZF\_separation\_fm(p) \equiv Forall^\frown(pred(arity(p)))(sep\_body\_fm(p))$

lemma *ZF_separation_fm_type* [*TC*]: $p \in formula \Longrightarrow ZF\_separation\_fm(p) \in formula$
by (*simp add*: *ZF_separation_fm_def*)

lemma *sats_ZF_separation_fm_iff*:
assumes
$\varphi{\in}formula$
shows
$(M, [] \models (ZF\_separation\_fm(\varphi)))$
$\longleftrightarrow$
$(\forall\, env{\in}list(M).\ arity(\varphi) \le 1\ \#{+}\ length(env) \longrightarrow$
$separation(\#\#M{,}\lambda x.\ M, [x]\ @\ env \models \varphi))$
proof (*intro iffI ballI impI*)
let *?n*=*Arith.pred*(*arity*(φ))
fix *env*
assume $M, [] \models ZF\_separation\_fm(\varphi)$
assume $arity(\varphi) \le 1\ \#{+}\ length(env)$ $env{\in}list(M)$
moreover from *this*
have $arity(\varphi) \le succ(length(env))$ **by** *simp*
then
obtain *some rest* **where** $some{\in}list(M)$ $rest{\in}list(M)$
$env = some\ @\ rest$ $length(some) = Arith.pred(arity(\varphi))$
using *list_split*[*OF* ‹$arity(\varphi) \le succ(\_)$› ‹$env{\in}\_$›] **by** *force*
moreover from ‹$\varphi{\in}\_$›
have $arity(\varphi) \le succ(Arith.pred(arity(\varphi)))$
using *succpred_leI* **by** *simp*
moreover
note *assms*
moreover
assume $M, [] \models ZF\_separation\_fm(\varphi)$
moreover from *calculation*
have $M, some \models sep\_body\_fm(\varphi)$
using *sats_nForall*[*of sep_body_fm*(φ) *?n*]
unfolding *ZF_separation_fm_def* **by** *simp*
ultimately
show $separation(\#\#M, \lambda x.\ M, [x]\ @\ env \models \varphi)$
unfolding *ZF_separation_fm_def*

```
    using sats_sep_body_fm[of φ [] M some]
      arity_sats_iff[of φ rest M [_] @ some]
      separation_cong[of ##M λx. M, Cons(x, some @ rest) ⊨ φ _ ]
    by simp
next — almost equal to the previous implication
  let ?n=Arith.pred(arity(φ))
  assume asm:∀ env∈list(M). arity(φ) ≤ 1 #+ length(env) ⟶
    separation(##M, λx. M, [x] @ env ⊨ φ)
  {
    fix some
    assume some∈list(M) length(some) = Arith.pred(arity(φ))
    moreover
    note ‹φ∈_›
    moreover from calculation
    have arity(φ) ≤ 1 #+ length(some)
      using le_trans[OF succpred_leI] succpred_leI by simp
    moreover from calculation and asm
    have separation(##M, λx. M, [x] @ some ⊨ φ) by blast
    ultimately
    have M, some ⊨ sep_body_fm(φ)
    using sats_sep_body_fm[of φ [] M some]
      arity_sats_iff[of φ _ M [_,_] @ some]
     strong_replacement_cong[of ##M λx y. M, Cons(x, Cons(y, some @ _)) ⊨
φ _ ]
    by simp
  }
  with ‹φ∈_›
  show M, [] ⊨ ZF_separation_fm(φ)
    using sats_nForall[of sep_body_fm(φ) ?n]
    unfolding ZF_separation_fm_def
    by simp
qed
```

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*) ⟷ (*A*,*Cons*(*z*,*Cons*(*y*,*Cons*(*x*,*env*))) ⊨ *Q2_fm*)
and
asms: *nth*(*i*,*env*) = *B* *i* ∈ *nat* *env* ∈ *list*(*A*)
shows
A,*env* ⊨ *univalent_fm*(*Q1_fm*,*Q2_fm*,*i*) ⟷ *univalent*(##*A*,*B*,*Q*)
unfolding *univalent_fm_def* **using** *asms sats_univalent_fm_auto*[*OF Q_iff_sats*]
by *simp*

definition
swap_vars :: *i*⇒*i* **where**
swap_vars(*φ*) ≡
Exists(*Exists*(*And*(*Equal*(*0*,*3*),*And*(*Equal*(*1*,*2*),*iterates*(λ*p*. *incr_bv*(*p*)‘*2* , *2*,
φ)))))

lemma *swap_vars_type*[*TC*] :
φ∈*formula* ⟹ *swap_vars*(*φ*) ∈*formula*
unfolding *swap_vars_def* **by** *simp*

lemma *sats_swap_vars* :
[*x*,*y*] @ *env* ∈ *list*(*M*) ⟹ *φ*∈*formula* ⟹
M, [*x*,*y*] @ *env* ⊨ *swap_vars*(*φ*)⟷ *M*,[*y*,*x*] @ *env* ⊨ *φ*
unfolding *swap_vars_def*
using *sats_incr_bv_iff* [*of* _ _ *M* _ [*y*,*x*]] **by** *simp*

definition
univalent_Q1 :: *i* ⇒ *i* **where**
univalent_Q1(*φ*) ≡ *incr_bv1*(*swap_vars*(*φ*))

definition
univalent_Q2 :: *i* ⇒ *i* **where**
univalent_Q2(*φ*) ≡ *incr_bv*(*swap_vars*(*φ*))‘*0*

lemma *univalent_Qs_type* [*TC*]:
assumes *φ*∈*formula*
shows *univalent_Q1*(*φ*) ∈ *formula* *univalent_Q2*(*φ*) ∈ *formula*
unfolding *univalent_Q1_def univalent_Q2_def* **using** *assms* **by** *simp_all*

lemma *sats_univalent_fm_assm*:

assumes
x ∈ A y ∈ A z∈A env∈ list(A) φ ∈ formula
shows
(A, ([x,z] @ env) ⊨ φ) ⟷ (A, Cons(z,Cons(y,Cons(x,env))) ⊨ (univalent_Q1(φ)))
(A, ([x,y] @ env) ⊨ φ) ⟷ (A, Cons(z,Cons(y,Cons(x,env))) ⊨ (univalent_Q2(φ)))
unfolding *univalent_Q1_def univalent_Q2_def*
using
sats_incr_bv_iff[of _ _ A _ []] — simplifies iterates of *λx. 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* $\models$ *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 $\lambda x\ y.\ M$, *Cons*(x, *Cons*(y, *some* @ _)) $\models$ φ _]
by *simp*
}
with ‹φ∈_›
show M, [] $\models$ *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) . $p \in$ *formula* } ∪ {*ZF_replacement_fm*(p) . $p \in$ *formula* }

lemma *Un_subset_formula*: A⊆*formula* ∧ B⊆*formula* ⟹ A∪$B \subseteq$ *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**
$A \models \Phi \equiv \forall \varphi \in \Phi.\ (A,[] \models \varphi)$

lemma *satTI* [*intro*!]:
assumes ⋀φ. φ∈Φ ⟹ A,[] ⊨ φ
shows $A \models \Phi$

using *assms* **unfolding** *satT_def* **by** *simp*

lemma *satTD* [*dest*] :$A \models \Phi \Longrightarrow\ \varphi{\in}\Phi \Longrightarrow A{,}[] \models \varphi$
unfolding *satT_def* **by** *simp*

lemma *sats_ZFC_iff_sats_ZF_AC*:
$(N \models ZFC) \longleftrightarrow (N \models ZF) \wedge (N, [] \models AC)$
unfolding *ZFC_def ZFC_fin_def ZF_def* **by** *auto*

lemma *M_ZF_iff_M_satT*: $M\_ZF(M) \longleftrightarrow (M \models ZF)$
proof
assume $M \models ZF$
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 $\varphi \in$ *formula env*∈*list*(*M*)
moreover from ‹$M \models ZF$›
have ∀ *p*∈*formula*. (M, [] ⊨ (*ZF_separation_fm*(*p*)))
∀ *p*∈*formula*. (M, [] ⊨ (*ZF_replacement_fm*(*p*)))
unfolding *ZF_def ZF_inf_def* **by** *auto*
moreover from *calculation*
have *arity*(φ) ≤ *succ*(*length*(*env*)) ⟹ *separation*(##*M*, λ*x*. (M, *Cons*(*x*, *env*) ⊨ φ))
arity(φ) ≤ *succ*(*succ*(*length*(*env*))) ⟹ *strong_replacement*(##*M*,λ*x y*. *sats*(*M*,φ,*Cons*(*x*,*Cons*(*y*, *env*))))
using *sats_ZF_separation_fm_iff sats_ZF_replacement_fm_iff* **by** *simp_all*

}
with *fin*
show *M_ZF*(M)
unfolding *M_ZF_def* **by** *simp*
next
assume ‹*M_ZF*(M)›
then
have $M \models ZF\_fin$
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 $M \models ZF$
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∈nat› by simp

lemma id_fn_action:
  assumes n ∈ nat env∈list(M)
  shows ⋀ j . j < n ⟹ nth(j,env) = nth(id(n)‘j,env)
proof -
  show nth(j,env) = nth(id(n)‘j,env) if j < n for j using that ‹n∈nat› ltD by
simp
qed


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

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

lemma sum_inr:
  assumes m ∈ nat n∈nat p∈nat
    g∈p→q m ≤ x x < m#+p
```

```
  shows sum(f,g,m,n,p)`x = g`(x#-m)#+n
proof -
  from assms
  have x∈nat
    using in_n_in_nat[of m#+p] ltD
    by simp
  with assms
  have ¬ x<m
    using not_lt_iff_le[THEN iffD2] by simp
  from assms
  have x∈m#+p
    using ltD by simp
  with ‹¬ x<m›
  show ?thesis unfolding 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 ∈ *nat* ⟹ *q* ∈ *nat* ⟹ *f* ∈ *p* → *q* ⟹ *sum_id*(*p*,*f*) ∈ *1*#+*p* → *1*#+*q*
by (*unfold sum_id_def*,*rule sum_type*,*simp_all*)

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

13.2 Renaming of formulas

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

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

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

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

lemma *arity_meml* : *l* ∈ *nat* ⟹ *Member*(*x*,*y*) ∈ *formula* ⟹ *arity*(*Member*(*x*,*y*)) ≤ *l* ⟹ *x* ∈ *l*
by(*simp*,*rule subsetD*,*rule le_imp_subset*,*assumption*,*simp*)
lemma *arity_memr* : *l* ∈ *nat* ⟹ *Member*(*x*,*y*) ∈ *formula* ⟹ *arity*(*Member*(*x*,*y*)) ≤ *l* ⟹ *y* ∈ *l*
by(*simp*,*rule subsetD*,*rule le_imp_subset*,*assumption*,*simp*)
lemma *arity_eql* : *l* ∈ *nat* ⟹ *Equal*(*x*,*y*) ∈ *formula* ⟹ *arity*(*Equal*(*x*,*y*)) ≤ *l* ⟹ *x* ∈ *l*
by(*simp*,*rule subsetD*,*rule le_imp_subset*,*assumption*,*simp*)

lemma *arity_eqr* : $l \in nat \Longrightarrow Equal(x,y) \in formula \Longrightarrow arity(Equal(x,y)) \leq l \Longrightarrow y \in l$
by(*simp*,*rule subsetD*,*rule le_imp_subset*,*assumption*,*simp*)
lemma *nand_ar1* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(p) \leq arity(Nand(p,q))$
by (*simp*,*rule Un_upper1_le*,*simp*+)
lemma *nand_ar2* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(q) \leq arity(Nand(p,q))$
by (*simp*,*rule Un_upper2_le*,*simp*+)

lemma *nand_ar1D* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(Nand(p,q)) \leq n \Longrightarrow arity(p) \leq n$
by(*auto simp add*: *le_trans*[*OF Un_upper1_le*[*of arity*(*p*) *arity*(*q*)]])
lemma *nand_ar2D* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(Nand(p,q)) \leq n \Longrightarrow arity(q) \leq n$
by(*auto simp add*: *le_trans*[*OF Un_upper2_le*[*of arity*(*p*) *arity*(*q*)]])

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

lemma *arity_ren* :
fixes *p*
assumes $p \in formula$
shows $\bigwedge n\ m\ f\ .\ n \in nat \Longrightarrow m \in nat \Longrightarrow f \in n \rightarrow m \Longrightarrow arity(p) \leq n \Longrightarrow arity(ren(p)`n`m`f) \leq m$
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*)∈*nat succ*(*m*)∈*nat* **by** *auto*
from *Forall* **have** *2*: *sum_id*(*n*,*f*) ∈ *succ*(*n*)→*succ*(*m*) **by** (*simp add*:*sum_id_tc*)
from *Forall* **have** *3*:*arity*(*p*) ≤ *succ*(*n*) **by** (*rule_tac n*=*arity*(*p*) **in** *natE*,*simp*+)
then have *arity*(*ren*(*p*)‘*succ*(*n*)‘*succ*(*m*)‘*sum_id*(*n*,*f*))≤*succ*(*m*) **using**
Forall ‹*succ*(*n*)∈*nat*› ‹*succ*(*m*)∈*nat*› *2* **by** *force*
then show *?case* **using** *Forall 2 3 ren_tc arity_type pred_le* **by** *auto*
qed

lemma *arity_forallE* : *p* ∈ *formula* ⟹ *m* ∈ *nat* ⟹ *arity*(*Forall*(*p*)) ≤ *m* ⟹ *arity*(*p*) ≤ *succ*(*m*)
by(*rule_tac n*=*arity*(*p*) **in** *natE*,*erule arity_type*,*simp*+)

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

lemma *sats_iff_sats_ren* :

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_meml* **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) ∈ succ(n) → succ(m) (is ?g ∈ _) using sum_id_tc Forall
by simp
  then have 2: arity(p) ≤ succ(n)
    using Forall le_trans[of _ succ(pred(arity(p)))] succpred_leI by simp
  have succ(n)∈nat succ(m)∈nat using Forall by auto
  then have A:⋀ j .j < succ(n) ⟹ nth(j, Cons(a, ϱ)) = nth(?g`j, Cons(a, ϱ′))
if a∈M for a
    using that env_coincidence_sum_id Forall ltD by force
  have
   sats(M,p,Cons(a,ϱ)) ⟷ sats(M,ren(p)`succ(n)`succ(m)`?g,Cons(a,ϱ′)) if a∈M
for a
  proof -
    have C:Cons(a,ϱ) ∈ list(M) Cons(a,ϱ′)∈list(M) using Forall that by auto
    have sats(M,p,Cons(a,ϱ)) ⟷ sats(M,ren(p)`succ(n)`succ(m)`?g,Cons(a,ϱ′))
        using Forall(2)[OF ‹succ(n)∈nat› ‹succ(m)∈nat› C(1) C(2) 1 2 A[OF
‹a∈M›]] by simp
    then show ?thesis .
  qed
  then show ?case using Forall 0 1 2 by simp
qed

end
theory Renaming_Auto
  imports
    Renaming
    Utils
    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 ‹renaming.ML›
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) =
      (Renaming.fix_vars tc_lemma fvs ctxt, Renaming.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 syn-
thetic definitions
   (ren_parser >> (fn ((name,(from,to)),_) => renaming_def Renaming.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 Renaming.ren_thm
name from to ))

in
end
›
end
```

14 Names and generic extensions

```
theory Names
  imports
    Forcing_Data
    Interface
    Recursion_Thms
    Synthetic_Definition
begin

definition
```

SepReplace :: $[i, i \Rightarrow i, i \Rightarrow o] \Rightarrow i$ **where**
SepReplace$(A,b,Q) \equiv \{y\ .\ x{\in}A,\ y{=}b(x) \wedge Q(x)\}$

syntax
_SepReplace :: $[i,$ *pttrn*, $i,\ o] \Rightarrow i$ $(\langle(1\{\_\ ../\ \_ \in \_,\ \_\})\rangle)$
syntax_consts
_SepReplace == *SepReplace*
translations
$\{b\ ..\ x{\in}A,\ Q\} \Rightarrow$ *CONST SepReplace*$(A, \lambda x.\ b, \lambda x.\ Q)$

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

lemma *SepReplace_subset* : $A{\subseteq}A' \Longrightarrow \{b\ ..\ x{\in}A,\ Q\}{\subseteq}\{b\ ..\ x{\in}A',\ Q\}$
by (*auto simp add*:*SepReplace_def*)

lemma *SepReplace_iff* [*simp*]: $y{\in}\{b(x)\ ..\ x{\in}A,\ P(x)\} \longleftrightarrow (\exists\, x{\in}A.\ y{=}b(x)\ \&\ P(x))$
by (*auto simp add*:*SepReplace_def*)

lemma *SepReplace_dom_implies* :
$(\bigwedge x\ .\ x \in A \Longrightarrow b(x) = b'(x)) \Longrightarrow \{b(x)\ ..\ x{\in}A,\ Q(x)\}{=}\{b'(x)\ ..\ x{\in}A,\ Q(x)\}$
by (*simp add*:*SepReplace_def*)

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

14.1 The well-founded relation *ed*

lemma *eclose_sing* : $x \in eclose(a) \Longrightarrow x \in eclose(\{a\})$
by(*rule subsetD*[*OF mem_eclose_subset*],*simp+*)

lemma *ecloseE* :
assumes $x \in eclose(A)$
shows $x \in A \vee (\exists\ B \in A\ .\ x \in eclose(B))$
using *assms*
proof (*induct rule*:*eclose_induct_down*)
case (*1 y*)
then
show *?case*
using *arg_into_eclose* **by** *auto*
next
case (*2 y z*)
from $\langle y \in A \vee (\exists\, B{\in}A.\ y \in eclose(B))\rangle$
consider (*inA*) $y \in A$ | (*exB*) $(\exists\, B{\in}A.\ y \in eclose(B))$
by *auto*
then show *?case*
proof (*cases*)

case *inA*
then
show *?thesis* **using** *2 arg_into_eclose* **by** *auto*
next
case *exB*
then obtain B **where** $y \in eclose(B)$ $B \in A$
by *auto*
then
show *?thesis* **using** *2 ecloseD*[*of y B z*] **by** *auto*
qed
qed

lemma *eclose_singE* : $x \in eclose(\{a\}) \Longrightarrow x = a \vee x \in eclose(a)$
by(*blast dest*: *ecloseE*)

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

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

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

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

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

$edrel(A) \equiv Rrel(ed,A)$

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

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

lemma *rank_ed*:
assumes $ed(y,x)$
shows $succ(rank(y)) \leq rank(x)$
proof
from *assms*
obtain p **where** $\langle y,p\rangle{\in}x$ **by** *auto*
moreover
obtain s **where** $y{\in}s$ $s{\in}\langle y,p\rangle$ **unfolding** *Pair_def* **by** *auto*
ultimately
have $rank(y) < rank(s)$ $rank(s) < rank(\langle y,p\rangle)$ $rank(\langle y,p\rangle) < rank(x)$
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∈edrel(A)
  then obtain x y where
    Eq1:   x∈A y∈A z=⟨x,y⟩ x∈domain(y)
    using edrelD
    by blast
  then obtain u v where
    Eq2:   x∈u u∈v v∈y
    unfolding domain_def Pair_def by auto
  with Eq1 have
    Eq3:   x∈eclose(A) y∈eclose(A) u∈eclose(A) v∈eclose(A)
   by (auto, rule_tac [3-4] ecloseD, rule_tac [3] ecloseD, simp_all add:arg_into_eclose)
  let
    ?r=trancl(Memrel(eclose(A)))
  from Eq2 and Eq3 have
    ⟨x,u⟩∈?r ⟨u,v⟩∈?r ⟨v,y⟩∈?r
    by (auto simp add: r_into_trancl)
  then  have
    ⟨x,y⟩∈?r
    by (rule_tac trancl_trans, rule_tac [2] trancl_trans, simp)
  with Eq1 show z∈?r by simp
qed

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

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

lemma dom_under_edrel_eclose: edrel(eclose({x})) -`` {x} = domain(x)
proof
  show edrel(eclose({x})) -`` {x} ⊆ domain(x)
    unfolding edrel_def Rrel_def ed_def
```

```
    by auto
next
  show domain(x) ⊆ edrel(eclose({x})) -‘‘ {x}
    unfolding edrel_def Rrel_def
    using in_dom_in_eclose eclose_sing arg_into_eclose
    by blast
qed

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

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


lemma restrict_edrel_eq :
  assumes z ∈ domain(x)
  shows edrel(eclose({x})) ∩ eclose({z})×eclose({z}) = edrel(eclose({z}))
proof(intro equalityI subsetI)
  let ?ec=λ y . edrel(eclose({y}))
  let ?ez=eclose({z})
  let ?rr=?ec(x) ∩ ?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 $y \in ?rr$ **by** *simp*
qed

lemma *tr_edrel_subset* :
assumes $z \in domain(x)$
shows $tr\_down(edrel(eclose(\{x\})),z) \subseteq eclose(\{z\})$
proof(*intro subsetI*)
let $?r=\lambda\ x\ .\ edrel(eclose(\{x\}))$
fix y
assume $y \in tr\_down(?r(x),z)$
then
have $\langle y,z\rangle \in ?r(x)^{+}$ **using** *tr_downD* **by** *simp*
with *assms*
show $y \in eclose(\{z\})$ **using** *tr_edrel_eclose eclose_sing* **by** *simp*
qed

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*)

lemma *Rep_simp* : $Replace(u,\lambda\ y\ z\ .\ z = f(y)) = \{\ f(y)\ .\ y \in u\}$
by(*auto*)

end

14.2 Values and check-names

context *forcing_data*
begin

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

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

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

definition

```
  rcheck :: i ⇒ i where
  rcheck(x) ≡ Memrel(eclose({x}))^+


lemma Hcheck_trancl:Hcheck(y, restrict(f,Memrel(eclose({x}))-‘‘{y}))
                = Hcheck(y, restrict(f,(Memrel(eclose({x}))^+)-‘‘{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 ∈ M ⟹ rcheck(x) ∈ M
  unfolding rcheck_def by (simp flip: setclass_iff)


lemma  aux_def_check: x ∈ y ⟹
  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=λy. Memrel(eclose({y}))
  have wfr:  ∀ w . wf(?r(w))
    using wf_Memrel ..
  then
  have check(y)= Hcheck( y, λx∈?r(y) -‘‘ {y}. wfrec(?r(y), x, Hcheck))
    using wfrec[of ?r(y) y Hcheck] checkD by simp
  also
  have  ... = Hcheck( y, λx∈y. wfrec(?r(y), x, Hcheck))
    using under_Memrel_eclose arg_into_eclose by simp
  also
  have  ... = Hcheck( y, λx∈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 ... = $\{\langle check(i),one\rangle$. $i \in n\} \cup \{\langle check(n),one\rangle\}$
by *blast*
also have ... = $check(n) \cup \{\langle check(n),one\rangle\}$
using *def_check*[*of* n,*symmetric*] **by** *simp*
finally show *?thesis* **.**
qed

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

definition
$Hv :: i \Rightarrow i \Rightarrow i \Rightarrow i$ **where**
$Hv(G,x,f) \equiv \{\ f`y$ .. $y \in domain(x), \exists p \in P.\ \langle y,p\rangle \in x \wedge p \in G\ \}$

The funcion *val* interprets a name in M according to a (generic) filter G. Note the definition in terms of the well-founded recursor.

definition
$val :: i \Rightarrow i \Rightarrow i$ **where**
$val(G,\tau) \equiv wfrec(edrel(eclose(\{\tau\})),\ \tau\ ,Hv(G))$

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

lemma *val_mono* : $x \subseteq y \Longrightarrow val(G,x) \subseteq val(G,y)$
  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* : $one \in G \Longrightarrow\ one \in P \Longrightarrow val(G,check(y))\ = y$
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 $val(G,check(y)) = val(G,\ \{\langle check(w),\ one\rangle$. $w \in y\})$
      by *simp*
    also
    have ... $= \{val(G,t)$ .. $t \in domain(?C)$, $\exists\, p \in P$. $\langle t,\ p\rangle \in ?C \wedge p \in G\ \}$
      using *def_val* **by** *blast*
    also
    have ... $= \{val(G,t)$ .. $t \in domain(?C)$, $\exists\, w \in y.\ t = check(w)\ \}$
      using *1* **by** *simp*
    also
    have ... $= \{val(G,check(w))$. $w \in y\ \}$
      by *force*
    finally
    show $val(G,check(y)) = y$
      using *1* **by** *simp*

```
  qed
qed

lemma val_of_name :
  val(G,{x∈A×P. Q(x)}) = {val(G,t) .. t∈A , ∃p∈P .  Q(⟨t,p⟩) ∧ p ∈ G }
proof -
  let
    ?n={x∈A×P. Q(x)} and
    ?r=λτ . edrel(eclose({τ}))
  let
    ?f=λz∈?r(?n)-``{?n}. val(G,z)
  have
    wfR : wf(?r(τ)) for τ
    by (simp add: wf_edrel)
  have domain(?n) ⊆ A by auto
  { fix t
    assume H:t ∈ domain({x ∈ A × P . Q(x)})
    then have ?f ` t = (if t ∈ ?r(?n)-``{?n} then val(G,t) else 0)
      by simp
    moreover have ... = val(G,t)
      using dom_under_edrel_eclose H if_P by auto
  }
  then
  have Eq1: t ∈ domain({x ∈ A × P . Q(x)}) ⟹ val(G,t) = ?f` t  for t
    by simp
  have val(G,?n) = {val(G,t) .. t∈domain(?n), ∃p ∈ P . ⟨t,p⟩ ∈ ?n ∧ p ∈ 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(G,t) else 0) .. t∈domain(?n), ∃p∈P
. ⟨t,p⟩∈?n ∧ p∈G}
    by (simp)
  also
  have Eq2:  ... = { val(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(G,t) else 0) = val(G,t)
      by auto
    then
     show { (if t∈?r(?n)-``{?n} then val(G,t) else 0) .. t∈domain(?n), ∃p∈P .
⟨t,p⟩∈?n ∧ p∈G} =
          { val(G,t) .. t∈domain(?n), ∃p∈P . ⟨t,p⟩∈?n ∧ p∈G}
      by auto
  qed
```

also
have $... = \{\ val(G,t)\ ..\ t{\in}A,\ \exists\, p{\in}P\ .\ \langle t,p\rangle{\in}?n\ \wedge\ p{\in}G\}$
by *force*
finally
show $val(G,?n)\ = \{\ val(G,t)\ ..\ t{\in}A,\ \exists\, p{\in}P\ .\ Q(\langle t,p\rangle)\ \wedge\ p{\in}G\}$
by *auto*
qed

lemma *val_of_name_alt* :
$val(G,\{x{\in}A{\times}P.\ Q(x)\}) = \{val(G,t)\ ..\ t{\in}A\ ,\ \exists\, p{\in}P{\cap}G\ .\ \ Q(\langle t,p\rangle)\ \}$
using *val_of_name* **by** *force*

lemma *val_only_names*: $val(F,\tau) = val(F,\{x{\in}\tau.\ \exists\, t{\in}domain(\tau).\ \exists\, p{\in}P.\ x{=}\langle t,p\rangle\})$
(**is** $\_ = val(F,?name)$)
proof -
have $val(F,?name) = \{val(F,\ t)..\ t{\in}domain(?name),\ \exists\, p{\in}P.\ \langle t,\ p\rangle\ \in\ ?name\ \wedge\ p\ \in\ F\}$
using *def_val* **by** *blast*
also
have $... = \{val(F,\ t).\ t{\in}\{y{\in}domain(?name).\ \exists\, p{\in}P.\ \langle y,\ p\rangle\ \in\ ?name\ \wedge\ p\ \in\ F\}\}$
using *Sep_and_Replace* **by** *simp*
also
have $... = \{val(F,\ t).\ t{\in}\{y{\in}domain(\tau).\ \exists\, p{\in}P.\ \langle y,\ p\rangle\ \in\ \tau\ \wedge\ p\ \in\ F\}\}$
by *blast*
also
have $... = \{val(F,\ t)..\ t{\in}domain(\tau),\ \exists\, p{\in}P.\ \langle t,\ p\rangle\ \in\ \tau\ \wedge\ p\ \in\ F\}$
using *Sep_and_Replace* **by** *simp*
also
have $... = val(F,\ \tau)$
using *def_val*[*symmetric*] **by** *blast*
finally
show *?thesis* **..**
qed

lemma *val_only_pairs*: $val(F,\tau) = val(F,\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\})$
proof
have $val(F,\tau) = val(F,\{x{\in}\tau.\ \exists\, t{\in}domain(\tau).\ \exists\, p{\in}P.\ x{=}\langle t,p\rangle\})$
(**is** $\_ = val(F,?name)$)
using *val_only_names* **.**
also
have $... \subseteq val(F,\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\})$
using *val_mono*[*of ?name* $\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\}$] **by** *auto*
finally
show $val(F,\tau) \subseteq val(F,\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\})$ **by** *simp*
next
show $val(F,\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\}) \subseteq val(F,\tau)$
using *val_mono*[*of* $\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\}$] **by** *auto*
qed

lemma *val_subset_domain_times_range*: $val(F,\tau) \subseteq val(F,domain(\tau)\times range(\tau))$
using *val_only_pairs*[*THEN equalityD1*]
val_mono[*of* $\{x \in \tau . \exists t\ p.\ x = \langle t,\ p\rangle\}$ $domain(\tau)\times range(\tau)$] **by** *blast*

lemma *val_subset_domain_times_P*: $val(F,\tau) \subseteq val(F,domain(\tau)\times P)$
using *val_only_names*[*of F* τ] *val_mono*[*of* $\{x\in\tau.\ \exists\, t\in domain(\tau).\ \exists\, p\in P.\ x=\langle t,p\rangle\}$ $domain(\tau)\times P$ *F*]
by *auto*

definition
GenExt :: $i\Rightarrow i$ ($\langle M[\_]\rangle$)
where $GenExt(G)\equiv \{val(G,\tau).\ \tau \in M\}$

lemma *val_of_elem*: $\langle\vartheta,p\rangle \in \pi \Longrightarrow p\in G \Longrightarrow p\in P \Longrightarrow val(G,\vartheta) \in val(G,\pi)$
proof -
assume
$\langle\vartheta,p\rangle \in \pi$
then
have $\vartheta\in domain(\pi)$ **by** *auto*
assume $p\in G$ $p\in P$
with $\langle\vartheta\in domain(\pi)\rangle$ $\langle\langle\vartheta,p\rangle \in \pi\rangle$
have $val(G,\vartheta) \in \{val(G,t)\ ..\ t\in domain(\pi)\ ,\ \exists\, p\in P\ .\ \langle t,\ p\rangle\in\pi \wedge p \in G\ \}$
by *auto*
then
show *?thesis* **by** (*subst def_val*)
qed

lemma *elem_of_val*: $x\in val(G,\pi) \Longrightarrow \exists\,\vartheta\in domain(\pi).\ val(G,\vartheta) = x$
by (*subst* (*asm*) *def_val*,*auto*)

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

lemma *elem_of_val_pair'*:
assumes $\pi\in M$ $x\in val(G,\pi)$
shows $\exists\,\vartheta\in M.\ \exists\, p\in G.\ \langle\vartheta,p\rangle\in\pi \wedge val(G,\vartheta) = x$
proof -
from *assms*
obtain ϑ p **where** $p\in G$ $\langle\vartheta,p\rangle\in\pi$ $val(G,\vartheta) = x$
using *elem_of_val_pair* **by** *blast*
moreover from *this* $\langle\pi\in M\rangle$
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*:
  $x \in M[G] \Longrightarrow \exists \tau{\in}M.\ x = val(G,\tau)$
  by (*simp add*:*GenExt_def*)

lemma *GenExtI*:
  $x \in M \Longrightarrow val(G,x) \in M[G]$
  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** $c{\in}M$ $val(G,c){\in}M[G]$ $y \in val(G,c)$
      using *GenExtD* **by** *auto*
    from ‹$y \in val(G,c)$›
    obtain ϑ **where** $\vartheta{\in}domain(c)$ $val(G,\vartheta) = y$
      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*] ⇒ *o* **where**
PHcheck(*o*,*f*,*y*,*p*) ≡ *p*∈*M* ∧ (∃*fy*[##*M*]. *fun_apply*(##*M*,*f*,*y*,*fy*) ∧ *pair*(##*M*,*fy*,*o*,*p*))

definition
is_Hcheck :: [*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_Hcheck(*o*,*z*,*f*,*hc*) ≡ *is_Replace*(##*M*,*z*,*PHcheck*(*o*,*f*),*hc*)

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

lemma *def_PHcheck*:
assumes
z∈*M f*∈*M*
shows
Hcheck(*z*,*f*) = *Replace*(*z*,*PHcheck*(*one*,*f*))
proof -
from *assms*
have ⟨*f*‘*x*,*one*⟩ ∈ *M f*‘*x*∈*M* **if** *x*∈*z* **for** *x*
using *tuples_in_M one_in_M transitivity that apply_closed* **by** *simp_all*
then
have {*y* . *x* ∈ *z*, *y* = ⟨*f* ‘ *x*, *one*⟩} = {*y* . *x* ∈ *z*, *y* = ⟨*f* ‘ *x*, *one*⟩ ∧ *y*∈*M* ∧ *f*‘*x*∈*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*))))

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*)

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 *is_wfrec_fm_def is_Hcheck_fm_def Replace_fm_def PHcheck_fm_def*
pair_fm_def upair_fm_def is_recfun_fm_def fun_apply_fm_def big_union_fm_def
pre_image_fm_def restriction_fm_def image_fm_def
by (*simp add*:*nat_simp_union*)
then

have *strong_replacement*(*##M*,$\lambda x\ z.$ *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*,$\lambda x\ z.$
$\exists\, y{\in}M.$ *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*,$\lambda x\ y.$ *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*∈*M* ; *f*∈*M* ⟧ ⟹ *univalent*(*##M*,*z*,*PHcheck*(*one*,*f*))
unfolding *univalent_def PHcheck_def* **by** *simp*

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

lemma *PHcheck_closed* :
⟦*z*∈*M* ; *f*∈*M* ; *x*∈*z*; *PHcheck*(*one*,*f*,*x*,*y*) ⟧ ⟹ (*##M*)(*y*)

unfolding *PHcheck_def* **by** *simp*

lemma *Hcheck_closed* :
$\forall\, y{\in}M.\ \forall\, g{\in}M.\ function(g) \longrightarrow Hcheck(y,g){\in}M$
proof -
have $Replace(y,PHcheck(one,f)){\in}M$ **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* : $x{\in}M \Longrightarrow wf(rcheck(x))$
unfolding *rcheck_def* **using** *wf_trancl*[*OF wf_Memrel*] **.**

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

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

lemma *check_in_M* : $x{\in}M \Longrightarrow check(x) \in M$
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**
$singleton\_fm(i,j) \equiv Exists(And(empty\_fm(0),\ cons\_fm(succ(i),0,succ(j))))$

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*:
$[\![\, nth(i,env) = x;\ nth(j,env) = y;$

$i \in nat$; $j \in nat$; $env \in list(A)\rrbracket$
$\Longrightarrow$ $is\_singleton(\#\#A,x,y) \longleftrightarrow sats(A,\ singleton\_fm(i,j),\ env)$
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* :
$\llbracket\ x{\in}M\ ;\ r{\in}M\ \rrbracket \Longrightarrow is\_rcheck(x,r) \longleftrightarrow r = rcheck(x)$
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 :: $[i,i] \Rightarrow o$ **where**
$is\_check(x,z) \equiv \exists\, rch{\in}M.\ is\_rcheck(x,rch) \wedge is\_wfrec(\#\#M,is\_Hcheck(one),rch,x,z)$

lemma *check_abs* :
assumes
$x{\in}M\ z{\in}M$
shows
$is\_check(x,z) \longleftrightarrow z = check(x)$
proof -
have
$is\_check(x,z) \longleftrightarrow is\_wfrec(\#\#M,is\_Hcheck(one),rcheck(x),x,z)$
unfolding *is_check_def* **using** *assms rcheck_abs rcheck_in_M*
unfolding *check_trancl is_check_def* **by** *simp*
then show *?thesis*
unfolding *check_trancl*
using *assms wfrec_Hcheck*[*of x*] *wf_rcheck trans_rcheck relation_rcheck rcheck_in_M*
Hcheck_closed relation2_Hcheck trans_wfrec_abs[*of rcheck*(*x*) *x z is_Hcheck*(*one*) *Hcheck*]
by (*simp flip*: *setclass_iff*)

qed

definition
check_fm :: $[i,i,i] \Rightarrow i$ **where**
check_fm$(x,o,z) \equiv$ *Exists*(*And*(*rcheck_fm*$(1\#+x,0)$,
is_wfrec_fm(*is_Hcheck_fm*$(6\#+o,2,1,0),0,1\#+x,1\#+z)))$

lemma *check_fm_type*[*TC*] :
$[\![x\in nat;o\in nat;z\in nat]\!] \Longrightarrow$ *check_fm*$(x,o,z)\in$*formula*
unfolding *check_fm_def* **by** *simp*

lemma *sats_check_fm* :
assumes
nth$(o,env) =$ *one* $x\in nat$ $z\in nat$ $o\in nat$ $env\in list(M)$ $x <$ *length*(env) $z <$ *length*(env)
shows
sats$(M,$ *check_fm*$(x,o,z),$ *env*$) \longleftrightarrow$ *is_check*(*nth*(x,env),*nth*(z,env))
proof -
have *sats_is_Hcheck_fm*:
$\bigwedge a0\ a1\ a2\ a3\ a4$. $[\![\ a0\in M;\ a1\in M;\ a2\in M;\ a3\in M;\ a4\in M\ ]\!] \Longrightarrow$
is_Hcheck(*one*,*a2*, *a1*, *a0*) $\longleftrightarrow$
sats$(M,$ *is_Hcheck_fm*$(6\#+o,2,1,0),$ $[a0,a1,a2,a3,a4,r]@env)$ **if** $r\in M$ **for**
r
using *that one_in_M assms* **by** *simp*
then
have *sats*$(M,$ *is_wfrec_fm*(*is_Hcheck_fm*$(6\#+o,2,1,0),0,1\#+x,1\#+z)$,*Cons*$(r,env))$
$\longleftrightarrow$ *is_wfrec*$(\#\#M$,*is_Hcheck*(*one*),r,*nth*(x,env),*nth*$(z,env))$ **if** $r\in M$ **for** r
using *that assms one_in_M sats_is_wfrec_fm* **by** *simp*
then
show *?thesis* **unfolding** *is_check_def check_fm_def*
using *assms rcheck_in_M one_in_M rcheck_fm_iff_sats*[*symmetric*] **by** *simp*
qed

lemma *check_replacement*:
$\{$*check*(x). $x\in P\} \in M$
proof -
have *arity*(*check_fm*$(0,2,1)) = 3$
unfolding *check_fm_def rcheck_fm_def trans_closure_fm_def is_eclose_fm_def*
mem_eclose_fm_def
is_Hcheck_fm_def Replace_fm_def PHcheck_fm_def finite_ordinal_fm_def
is_iterates_fm_def
is_wfrec_fm_def is_recfun_fm_def restriction_fm_def pre_image_fm_def
eclose_n_fm_def
is_nat_case_fm_def quasinat_fm_def Memrel_fm_def singleton_fm_def
fm_defs iterates_MH_fm_def
by (*simp add*:*nat_simp_union*)
moreover
have *check*$(x)\in M$ **if** $x\in P$ **for** x

using *that Transset_intf*[*of M x P*] *trans_M 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 *check_fm_def rcheck_fm_def trans_closure_fm_def is_eclose_fm_def*
mem_eclose_fm_def
is_Hcheck_fm_def Replace_fm_def PHcheck_fm_def finite_ordinal_fm_def
is_iterates_fm_def
is_wfrec_fm_def is_recfun_fm_def restriction_fm_def pre_image_fm_def
eclose_n_fm_def
is_nat_case_fm_def quasinat_fm_def Memrel_fm_def singleton_fm_def
fm_defs iterates_MH_fm_def
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 ⊆ P
    one ∈ G
  shows val(G,G_dot) = G
proof (intro equalityI subsetI)
  fix x
  assume x∈val(G,G_dot)
  then obtain ϑ p where p∈G ⟨ϑ,p⟩ ∈ G_dot val(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(G,check(p)) ∈ val(G,G_dot)
    using val_of_elem G_dot_in_M by blast
  with ‹p∈G› ‹G⊆P› ‹one∈G›
  show p ∈ val(G,G_dot)
    using P_sub_M valcheck by auto
qed


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


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** $p = \langle a, b \rangle$ **by** *blast*
with *True*
have $fst(p) = a$ $snd(p) = b$ **unfolding** *fst_def snd_def* **by** *simp_all*
moreover
assume $p \in M$
moreover from *this*
have $a \in M$
unfolding ‹$p = \_$› *Pair_def* **by** (*force intro*:*Transset_M*[*OF trans_M*])
moreover from ‹$p \in M$›
have $b \in M$
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* :
$0 \in M[G]$
proof -
have $0 = val(G,0)$
using *zero_in_M elem_of_val* **by** *auto*
also
have ... $\in M[G]$
using *GenExtI zero_in_M* **by** *simp*
finally show *?thesis* **.**
qed

lemma *G_nonempty*: $G \neq 0$
proof -
have $P \subseteq P$ **..**
with *P_in_M P_dense* ‹$P \subseteq P$›
show $G \neq 0$
using *generic* **unfolding** *M_generic_def* **by** *auto*
qed

end
end

15 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 :: $[i⇒o,i⇒i⇒o,i⇒i⇒o,i,i] ⇒ o$ **where**
$is\_hcomp(M,is\_f,is\_g,a,w) ≡ ∃\,z[M].\ is\_g(a,z) ∧ is\_f(z,w)$

lemma (**in** *M_trivial*) *hcomp_abs*:
assumes
$is\_f\_abs{:}⋀a\ z.\ M(a) ⟹ M(z) ⟹ is\_f(a,z) ⟷ z = f(a)$ **and**
$is\_g\_abs{:}⋀a\ z.\ M(a) ⟹ M(z) ⟹ is\_g(a,z) ⟷ z = g(a)$ **and**
$g\_closed{:}⋀a.\ M(a) ⟹ M(g(a))$
$M(a)\ M(w)$
shows
$is\_hcomp(M,is\_f,is\_g,a,w) ⟷ w = f(g(a))$
unfolding *is_hcomp_def* **using** *assms* **by** *simp*

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

lemma *sats_hcomp_fm*:
assumes
$f\_iff\_sats{:}⋀a\ b\ z.\ a∈nat ⟹ b∈nat ⟹ z∈M ⟹$
$is\_f(nth(a,Cons(z,env)),nth(b,Cons(z,env))) ⟷ sats(M,pf(a,b),Cons(z,env))$
and
$g\_iff\_sats{:}⋀a\ b\ z.\ a∈nat ⟹ b∈nat ⟹ z∈M ⟹$
$is\_g(nth(a,Cons(z,env)),nth(b,Cons(z,env))) ⟷ sats(M,pg(a,b),Cons(z,env))$
and
$a∈nat\ w∈nat\ env∈list(M)$
shows
$sats(M,hcomp\_fm(pf,pg,a,w),env) ⟷ is\_hcomp(\#\#M,is\_f,is\_g,nth(a,env),nth(w,env))$

proof -
have $sats(M,\ pf(0,\ succ(w)),\ Cons(x,\ env)) ⟷ is\_f(x,nth(w,env))$ **if** $x∈M\ w∈nat$ **for** $x\ w$
using *f_iff_sats*[*of* $0\ succ(w)\ x$] *that* **by** *simp*
moreover
have $sats(M,\ pg(succ(a),\ 0),\ Cons(x,\ env)) ⟷ is\_g(nth(a,env),x)$ **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 :: $i \Rightarrow i$ **where**
ftype $\equiv$ *fst*

definition
name1 :: $i \Rightarrow i$ **where**
name1$(x) \equiv$ *fst*(*snd*(x))

definition
name2 :: $i \Rightarrow i$ **where**
name2$(x) \equiv$ *fst*(*snd*(*snd*(x)))

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

lemma *components_simp*:
ftype$(\langle f,n1,n2,c\rangle) = f$
name1$(\langle f,n1,n2,c\rangle) = n1$
name2$(\langle f,n1,n2,c\rangle) = n2$
cond_of$(\langle f,n1,n2,c\rangle) = c$
unfolding *ftype_def name1_def name2_def cond_of_def*
by *simp_all*

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

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

lemma *components_in_eclose* :
$n1 \in$ *ecloseN*$(\langle f,n1,n2,c\rangle)$
$n2 \in$ *ecloseN*$(\langle f,n1,n2,c\rangle)$
unfolding *ecloseN_def eclose_n_def*
using *components_simp arg_into_eclose* **by** *auto*

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

lemma *ecloseNI1* :
assumes $x \in$ *eclose*$(n1) \vee x \in$ *eclose*$(n2)$
shows $x \in$ *ecloseN*$(\langle f,n1,n2,c\rangle)$
unfolding *ecloseN_def eclose_n_def*
using *assms eclose_sing names_simp*
by *auto*

lemmas *ecloseNI* = *ecloseNI1*

lemma *ecloseN_mono* :
  assumes $u \in ecloseN(x)$ $name1(x) \in ecloseN(y)$ $name2(x) \in ecloseN(y)$
  shows $u \in ecloseN(y)$
proof -
  from ‹$u \in \_$›
  consider (a) $u \in eclose(\{name1(x)\})$ | (b) $u \in eclose(\{name2(x)\})$
    unfolding *ecloseN_def eclose_n_def* **by** *auto*
  then
  show *?thesis*
  proof *cases*
    case *a*
    with ‹$name1(x) \in \_$›
    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 ‹$name2(x) \in \_$›
    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 :: $(i \Rightarrow o) \Rightarrow i \Rightarrow i \Rightarrow o$ **where**
  $is\_fst(M,x,t) \equiv (\exists\, z[M].\ pair(M,t,z,x)) \vee$
                    $(\neg(\exists\, z[M].\ \exists\, w[M].\ pair(M,w,z,x)) \wedge empty(M,t))$

definition
  fst_fm :: $[i,i] \Rightarrow i$ **where**
  $fst\_fm(x,t) \equiv Or(Exists(pair\_fm(succ(t),0,succ(x))),$
                $And(Neg(Exists(Exists(pair\_fm(0,1,2\ \#+\ x)))),empty\_fm(t)))$

lemma *sats_fst_fm* :
  $[\![\ x \in nat;\ y \in nat; env \in list(A)\ ]\!]$
    $\Longrightarrow sats(A,\ fst\_fm(x,y),\ env) \longleftrightarrow$
        $is\_fst(\#\#A,\ nth(x,env),\ nth(y,env))$
  by (*simp add*: *fst_fm_def is_fst_def*)

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

definition
  ftype_fm :: $[i,i] \Rightarrow i$ **where**

ftype_fm ≡ *fst_fm*

lemma *sats_ftype_fm* :
⟦ *x* ∈ *nat*; *y* ∈ *nat*;*env* ∈ *list*(*A*) ⟧
⟹ *sats*(*A*, *ftype_fm*(*x*,*y*), *env*) ⟷
is_ftype(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
unfolding *ftype_fm_def is_ftype_def*
by (*simp add*:*sats_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*)
using *assms*
by (*simp add*:*sats_ftype_fm*)

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*
by (*simp add:sats_name1_fm*)

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 :: $[i,i] \Rightarrow i$ **where**
$cond\_of\_fm(x,t4) \equiv hcomp\_fm(snd\_fm,snd\_snd\_fm,x,t4)$

lemma *sats_cond_of_fm* :
$[\![\ x \in nat;\ y \in nat;env \in list(A)\ ]\!]$
$\Longrightarrow sats(A,cond\_of\_fm(x,y),\ env) \longleftrightarrow$
$is\_cond\_of(\#\#A,\ nth(x,env),\ nth(y,env))$
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
$nth(a,env) = aa\ nth(b,env) = bb\ a \in nat\ b \in nat\ env \in list(A)$
shows
$is\_cond\_of(\#\#A,aa,bb) \longleftrightarrow sats(A,cond\_of\_fm(a,b),\ env)$
using *assms*
by (*simp add*:*sats_cond_of_fm*)

lemma *components_type*[*TC*] :
assumes $a \in nat\ b \in nat$
shows
$ftype\_fm(a,b) \in formula$
$name1\_fm(a,b) \in formula$
$name2\_fm(a,b) \in formula$
$cond\_of\_fm(a,b) \in formula$
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 *sats_components_fm*[*simp*] = *sats_ftype_fm sats_name1_fm sats_name2_fm sats_cond_of_fm*

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 :: $[i \Rightarrow o,[i \Rightarrow o,i,i] \Rightarrow o,i,i] \Rightarrow o$ **where**
$is\_eclose\_n(N,is\_name,en,t) \equiv$
$\exists\, n1[N].\exists\, s1[N].\ is\_name(N,t,n1) \wedge is\_singleton(N,n1,s1) \wedge is\_eclose(N,s1,en)$

definition

eclose_n1_fm :: $[i,i] \Rightarrow i$ **where**
eclose_n1_fm$(m,t) \equiv$ *Exists*(*Exists*(*And*(*And*(*name1_fm*$(t\#+2,0)$,*singleton_fm*$(0,1)$),
is_eclose_fm$(1,m\#+2)$))))

definition
eclose_n2_fm :: $[i,i] \Rightarrow i$ **where**
eclose_n2_fm$(m,t) \equiv$ *Exists*(*Exists*(*And*(*And*(*name2_fm*$(t\#+2,0)$,*singleton_fm*$(0,1)$),
is_eclose_fm$(1,m\#+2)$))))

definition
is_ecloseN :: $[i \Rightarrow o,i,i] \Rightarrow o$ **where**
is_ecloseN$(N,en,t) \equiv \exists\, en1[N].\exists\, en2[N].$
is_eclose_n$(N,is\_name1,en1,t) \wedge$ *is_eclose_n*$(N,is\_name2,en2,t) \wedge$
union$(N,en1,en2,en)$

definition
ecloseN_fm :: $[i,i] \Rightarrow i$ **where**
ecloseN_fm$(en,t) \equiv$ *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*] :
$[\![\ en \in nat\ ;\ t \in nat\ ]\!] \Longrightarrow$ *ecloseN_fm*$(en,t) \in$ *formula*
unfolding *ecloseN_fm_def eclose_n1_fm_def eclose_n2_fm_def* **by** *simp*

lemma *sats_ecloseN_fm* [*simp*]:
$[\![\ en \in nat;\ t \in nat\ ;\ env \in list(A)\ ]\!]$
$\Longrightarrow$ *sats*$(A,$ *ecloseN_fm*$(en,t),\ env) \longleftrightarrow$ *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 :: $i \Rightarrow i \Rightarrow o$ **where**
frecR$(x,y) \equiv$
$(ftype(x) = 1 \wedge ftype(y) = 0$
$\wedge\ (name1(x) \in domain(name1(y)) \cup domain(name2(y)) \wedge (name2(x) = name1(y) \vee name2(x) = name2(y))))$
$\vee\ (ftype(x) = 0 \wedge ftype(y) = 1 \wedge name1(x) = name1(y) \wedge name2(x) \in domain(name2(y)))$

lemma *frecR_ftypeD* :
assumes *frecR*(x,y)
shows $(ftype(x) = 0 \wedge ftype(y) = 1) \vee (ftype(x) = 1 \wedge ftype(y) = 0)$
using *assms* **unfolding** *frecR_def* **by** *auto*

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

unfolding *frecR_def* **by** (*simp add*:*components_simp*)

lemma *frecRI1'*: $s \in domain(n1) \cup domain(n2) \Longrightarrow frecR(\langle 1,\ s,\ n1,\ q\rangle, \langle 0,\ n1,\ n2,\ q'\rangle)$
unfolding *frecR_def* **by** (*simp add*:*components_simp*)

lemma *frecRI2*: $s \in domain(n1) \vee s \in domain(n2) \Longrightarrow frecR(\langle 1,\ s,\ n2,\ q\rangle, \langle 0,\ n1,\ n2,\ q'\rangle)$
unfolding *frecR_def* **by** (*simp add*:*components_simp*)

lemma *frecRI2'*: $s \in domain(n1) \cup domain(n2) \Longrightarrow frecR(\langle 1,\ s,\ n2,\ q\rangle, \langle 0,\ n1,\ n2,\ q'\rangle)$
unfolding *frecR_def* **by** (*simp add*:*components_simp*)

lemma *frecRI3*: $\langle s,\ r\rangle \in n2 \Longrightarrow frecR(\langle 0,\ n1,\ s,\ q\rangle, \langle 1,\ n1,\ n2,\ q'\rangle)$
unfolding *frecR_def* **by** (*auto simp add*:*components_simp*)

lemma *frecRI3'*: $s \in domain(n2) \Longrightarrow frecR(\langle 0,\ n1,\ s,\ q\rangle, \langle 1,\ n1,\ n2,\ q'\rangle)$
unfolding *frecR_def* **by** (*auto simp add*:*components_simp*)

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

lemma *frecR_D1* :
$frecR(x,y) \Longrightarrow ftype(y) = 0 \Longrightarrow ftype(x) = 1 \wedge$
$(name1(x) \in domain(name1(y)) \cup domain(name2(y)) \wedge (name2(x) = name1(y) \vee name2(x) = name2(y)))$
using *frecR_iff*
by *auto*

lemma *frecR_D2* :
$frecR(x,y) \Longrightarrow ftype(y) = 1 \Longrightarrow ftype(x) = 0 \wedge$
$ftype(x) = 0 \wedge ftype(y) = 1 \wedge name1(x) = name1(y) \wedge name2(x) \in domain(name2(y))$
using *frecR_iff*
by *auto*

lemma *frecR_DI* :
assumes $frecR(\langle a,b,c,d\rangle,\langle ftype(y),name1(y),name2(y),cond\_of(y)\rangle)$
shows $frecR(\langle a,b,c,d\rangle,y)$
using *assms* **unfolding** *frecR_def* **by** (*force simp add*:*components_simp*)

definition
is_frecR :: $[i \Rightarrow o,i,i] \Rightarrow o$ **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 is_Collect_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*$(x)\leq$*rank_names*(y)
proof -
obtain a b c d **where**
H: $a = name1(x)$ $b = name2(x)$
$c = name1(y)$ $d = name2(y)$
$(a \in domain(c) \cup domain(d) \wedge (b{=}c \vee b = d)) \vee (a = c \wedge b \in domain(d))$
using *assms* **unfolding** *frecR_def* **by** *force*
then
consider
(m) $a \in domain(c) \wedge (b = c \vee b = d)$
| (n) $a \in domain(d) \wedge (b = c \vee b = d)$
| (o) $b \in domain(d) \wedge a = c$
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 :: $i \Rightarrow i$ **where**
$\Gamma(x) = 3 ** rank\_names(x) ++ type\_form(x)$

lemma Γ_type [*TC*]:
  shows *Ord*($\Gamma(x)$)
  unfolding Γ_def **by** *simp*

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

using *max_D2*[*OF E a*]
E assms Ord_rank **by** *simp*
next
case *b*
with ‹*rank_names*(*x*) = *rank_names*(*y*) ›
show *?thesis* **unfolding** *rank_names_def mtype_form_def type_form_def*

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

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

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

lemma *frecrelD* :
assumes $\langle x,y\rangle \in frecrel(A1\times A2\times A3\times A4)$
shows $ftype(x) \in A1$ $ftype(x) \in A1$
$name1(x) \in A2$ $name1(y) \in A2$ $name2(x) \in A3$ $name2(x) \in A3$
$cond\_of(x) \in A4$ $cond\_of(y) \in A4$
$frecR(x,y)$
using *assms* **unfolding** *frecrel_def Rrel_def ftype_def* **by** (*auto simp add:components_simp*)

lemma *wf_frecrel* :
shows $wf(frecrel(A))$
proof -
have $frecrel(A) \subseteq measure(A,\Gamma)$
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
$Transset(A1)$
$\bigwedge\tau\ \vartheta\ p.\ \ p \in A2 \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![\ q\in A2\ ;\ \sigma\in domain(\vartheta)]\!] \Longrightarrow Q(0,\tau,\sigma,q)]\!] \Longrightarrow Q(1,\tau,\vartheta,p)$
$\bigwedge\tau\ \vartheta\ p.\ \ p \in A2 \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![\ q\in A2\ ;\ \sigma\in domain(\tau)\ \cup\ domain(\vartheta)]\!] \Longrightarrow Q(1,\sigma,\tau,q) \wedge Q(1,\sigma,\vartheta,q)]\!] \Longrightarrow Q(0,\tau,\vartheta,p)$
shows $a\in 2\times A1\times A1\times A2 \Longrightarrow Q(ftype(a),name1(a),name2(a),cond\_of(a))$
proof (*induct a rule:wf_induct*[*OF wf_frecrel*[*of* $2\times A1\times A1\times A2$]])
case (*1 x*)
let $?\tau = name1(x)$
let $?\vartheta = name2(x)$
let $?D = 2\times A1\times A1\times A2$
assume $x \in ?D$
then
have $cond\_of(x)\in A2$
by (*auto simp add:components_simp*)
from ‹$x\in ?D$›
consider (*eq*) $ftype(x)=0$ | (*mem*) $ftype(x)=1$
by (*auto simp add:components_simp*)
then
show *?case*
proof *cases*
case *eq*
then
have $Q(1,\ \sigma,\ ?\tau,\ q) \wedge Q(1,\ \sigma,\ ?\vartheta,\ q)$ **if** $\sigma \in domain(?\tau) \cup domain(?\vartheta)$ **and** $q\in A2$ **for** $q\ \sigma$
proof -
from *1*
have *A*: $?\tau\in A1$ $?\vartheta\in A1$ $?\tau\in eclose(A1)$ $?\vartheta\in eclose(A1)$

```
      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 ‹Trans-
set(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 ‹$q{\in}A2$› ‹$\sigma{\in}A1$› ‹$?\tau{\in}A1$› ‹$?\vartheta{\in}A1$› *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

16 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*∈*nat* ; *z*∈*nat* ⟧ ⟹ *arity*(*range_fm*(*r*,*z*)) = *succ*(*r*) ∪ *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*∈*nat* ; *y*∈*nat* ; *z*∈*nat* ⟧ ⟹ *arity*(*union_fm*(*x*,*y*,*z*)) = ⋃ {*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*∈*nat* ; *y*∈*nat* ; *z*∈*nat* ⟧ ⟹ *arity*(*image_fm*(*x*,*y*,*z*)) = ⋃ {*succ*(*x*), *succ*(*y*), *succ*(*z*)}
 unfolding *image_fm_def*
 using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
 by *auto*

lemma *arity_pre_image_fm* :
 ⟦ *x*∈*nat* ; *y*∈*nat* ; *z*∈*nat* ⟧ ⟹ *arity*(*pre_image_fm*(*x*,*y*,*z*)) = ⋃ {*succ*(*x*), *succ*(*y*), *succ*(*z*)}
 unfolding *pre_image_fm_def*
 using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
 by *auto*

lemma *arity_big_union_fm* :
 ⟦ *x*∈*nat* ; *y*∈*nat* ⟧ ⟹ *arity*(*big_union_fm*(*x*,*y*)) = *succ*(*x*) ∪ *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*∈*nat* ; *y*∈*nat* ; *f*∈*nat* ⟧ ⟹
  arity(*fun_apply_fm*(*f*,*x*,*y*)) = *succ*(*f*) ∪ *succ*(*x*) ∪ *succ*(*y*)
 unfolding *fun_apply_fm_def*
 using *arity_upair_fm arity_image_fm arity_big_union_fm nat_union_abs2 pred_Un_distrib*
 by *auto*

lemma *arity_field_fm* :
  ⟦ *r*∈*nat* ; *z*∈*nat* ⟧ ⟹ *arity*(*field_fm*(*r*,*z*)) = *succ*(*r*) ∪ *succ*(*z*)
 unfolding *field_fm_def*
 using *arity_pair_fm arity_domain_fm arity_range_fm arity_union_fm*
  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* :
$[\![x{\in}nat\ ;\ y{\in}nat]\!] \Longrightarrow arity(subset\_fm(x,y)) = succ(x) \cup succ(y)$
unfolding *subset_fm_def*
using *nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_transset_fm* :
$[\![x{\in}nat]\!] \Longrightarrow arity(transset\_fm(x)) = succ(x)$
unfolding *transset_fm_def*
using *arity_subset_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_ordinal_fm* :
$[\![x{\in}nat]\!] \Longrightarrow arity(ordinal\_fm(x)) = succ(x)$
unfolding *ordinal_fm_def*
using *arity_transset_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_limit_ordinal_fm* :
$[\![x{\in}nat]\!] \Longrightarrow arity(limit\_ordinal\_fm(x)) = succ(x)$
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* :
$[\![x{\in}nat]\!] \Longrightarrow arity(finite\_ordinal\_fm(x)) = succ(x)$
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* :
$[\![x{\in}nat]\!] \Longrightarrow arity(omega\_fm(x)) = succ(x)$
unfolding *omega_fm_def*
using *arity_limit_ordinal_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_cartprod_fm* :
$[\![\ A{\in}nat\ ;\ B{\in}nat\ ;\ z{\in}nat\ ]\!] \Longrightarrow arity(cartprod\_fm(A,B,z)) = succ(A) \cup succ(B)$
$\cup\ succ(z)$
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*∈*formula* ; *v*∈*nat* ; *n*∈*nat*; *Z*∈*nat*;*i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
arity(*is_recfun_fm*(*p*,*v*,*n*,*Z*)) = *succ*(*v*) ∪ *succ*(*n*) ∪ *succ*(*Z*) ∪ *pred*(*pred*(*pred*(*pred*(*i*))))
 unfolding *is_recfun_fm_def*
 using *arity_upair_fm arity_pair_fm arity_pre_image_fm arity_restriction_fm*
 nat_union_abs2 pred_Un_distrib
 by *auto*

lemma *arity_is_wfrec_fm* :
 ⟦*p*∈*formula* ; *v*∈*nat* ; *n*∈*nat*; *Z*∈*nat* ; *i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
 arity(*is_wfrec_fm*(*p*,*v*,*n*,*Z*)) = *succ*(*v*) ∪ *succ*(*n*) ∪ *succ*(*Z*) ∪ *pred*(*pred*(*pred*(*pred*(*pred*(*i*)))))
 unfolding *is_wfrec_fm_def*
 using *arity_succ_fm arity_is_recfun_fm*
 nat_union_abs2 pred_Un_distrib
 by *auto*

lemma *arity_is_nat_case_fm* :
 ⟦*p*∈*formula* ; *v*∈*nat* ; *n*∈*nat*; *Z*∈*nat*; *i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
 arity(*is_nat_case_fm*(*v*,*p*,*n*,*Z*)) = *succ*(*v*) ∪ *succ*(*n*) ∪ *succ*(*Z*) ∪ *pred*(*pred*(*i*))
 unfolding *is_nat_case_fm_def*
 using *arity_succ_fm arity_empty_fm arity_quasinat_fm*
 nat_union_abs2 pred_Un_distrib
 by *auto*

lemma *arity_iterates_MH_fm* :
 assumes *isF*∈*formula v*∈*nat n*∈*nat g*∈*nat z*∈*nat i*∈*nat*
 arity(*isF*) = *i*
 shows *arity*(*iterates_MH_fm*(*isF*,*v*,*n*,*g*,*z*)) =
 succ(*v*) ∪ *succ*(*n*) ∪ *succ*(*g*) ∪ *succ*(*z*) ∪ *pred*(*pred*(*pred*(*pred*(*i*))))
proof -
 let *?φ* = *Exists*(*And*(*fun_apply_fm*(*succ*(*succ*(*succ*(*g*))), *2*, *0*), *Forall*(*Implies*(*Equal*(*0*, *2*), *isF*))))
 let *?ar* = *succ*(*succ*(*succ*(*g*))) ∪ *pred*(*pred*(*i*))
 from *assms*
 have *arity*(*?φ*) =*?ar ?φ*∈*formula*

using *arity_fun_apply_fm*
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* ‹*?φ*∈_› _ _ _ _ ‹*arity*(*?φ*) = _›] *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*(*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*(*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 $x \in nat$ $y \in nat$ $p \in formula$
shows $arity(Collect\_fm(x,p,y)) = succ(x) \cup succ(y) \cup pred(arity(p))$
unfolding *Collect_fm_def*
using *assms pred_Un_distrib*
by *auto*

end

17 The definition of *forces*

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

This is the core of our development.

17.1 The relation *frecrel*

definition
$frecrelP :: [i \Rightarrow o, i] \Rightarrow o$ **where**
$frecrelP(M,xy) \equiv (\exists\, x[M].\ \exists\, y[M].\ pair(M,x,y,xy) \wedge is\_frecR(M,x,y))$

definition
$frecrelP\_fm :: i \Rightarrow i$ **where**
$frecrelP\_fm(a) \equiv Exists(Exists(And(pair\_fm(1,0,a\#+2),frecR\_fm(1,0))))$

lemma *arity_frecrelP_fm* :
$a \in nat \Longrightarrow arity(frecrelP\_fm(a)) = succ(a)$
unfolding *frecrelP_fm_def*
using *arity_frecR_fm arity_pair_fm pred_Un_distrib*
by *simp*

lemma *frecrelP_fm_type*[*TC*] :
$a \in nat \Longrightarrow frecrelP\_fm(a) \in formula$
unfolding *frecrelP_fm_def* **by** *simp*

lemma *sats_frecrelP_fm* :
assumes $a \in nat$ $env \in list(A)$
shows $sats(A,frecrelP\_fm(a),env) \longleftrightarrow frecrelP(\#\#A,nth(a,\ env))$
unfolding *frecrelP_def frecrelP_fm_def*
using *assms* **by** (*auto simp add*:*frecR_fm_iff_sats*[*symmetric*])

lemma *frecrelP_iff_sats*:
assumes
$nth(a,env) = aa$ $a \in nat$ $env \in list(A)$
shows
$frecrelP(\#\#A,aa) \longleftrightarrow sats(A,\ frecrelP\_fm(a),\ env)$

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*∈*nat*;*b*∈*nat*⟧ ⟹ *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
$P{\in}nat\ x{\in}nat\ nb{\in}nat\ env{\in}list(A)$
shows
$sats(A,is\_names\_below\_fm(P,x,nb),env)$
$\longleftrightarrow\ is\_names\_below(\#\#A,nth(P,\ env),nth(x,\ env),nth(nb,\ env))$
unfolding *is_names_below_fm_def is_names_below_def* **using** *assms* **by** *simp*

definition
$is\_tuple :: [i{\Rightarrow}o,i,i,i,i,i] \Rightarrow o$ **where**
$is\_tuple(M,z,t1,t2,p,t) \equiv \exists\, t1t2p[M].\ \exists\, t2p[M].\ pair(M,t2,p,t2p) \wedge pair(M,t1,t2p,t1t2p) \wedge$
$pair(M,z,t1t2p,t)$

definition
$is\_tuple\_fm :: [i,i,i,i,i] \Rightarrow i$ **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 $arity\_is\_tuple\_fm : [\![\ z{\in}nat\ ;\ t1{\in}nat\ ;\ t2{\in}nat\ ;\ p{\in}nat\ ;\ tup{\in}nat\ ]\!] \Longrightarrow$
$arity(is\_tuple\_fm(z,t1,t2,p,tup)) = \bigcup\ \{succ(z),succ(t1),succ(t2),succ(p),succ(tup)\}$
unfolding *is_tuple_fm_def*
using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *auto*

lemma $is\_tuple\_fm\_type[TC] :$
$z{\in}nat \Longrightarrow t1{\in}nat \Longrightarrow t2{\in}nat \Longrightarrow p{\in}nat \Longrightarrow tup{\in}nat \Longrightarrow is\_tuple\_fm(z,t1,t2,p,tup){\in}formula$
unfolding *is_tuple_fm_def* **by** *simp*

lemma *sats_is_tuple_fm* :
assumes
$z{\in}nat\ \ t1{\in}nat\ t2{\in}nat\ p{\in}nat\ tup{\in}nat\ env{\in}list(A)$
shows
$sats(A,is\_tuple\_fm(z,t1,t2,p,tup),env)$
$\longleftrightarrow\ is\_tuple(\#\#A,nth(z,\ env),nth(t1,\ env),nth(t2,\ env),nth(p,\ env),nth(tup,\ env))$
unfolding *is_tuple_def is_tuple_fm_def* **using** *assms* **by** *simp*

lemma *is_tuple_iff_sats*:
assumes
$nth(a,env) = aa\ nth(b,env) = bb\ nth(c,env) = cc\ nth(d,env) = dd\ nth(e,env) = ee$
$a{\in}nat\ b{\in}nat\ c{\in}nat\ d{\in}nat\ e{\in}nat\ \ env \in list(A)$
shows
$is\_tuple(\#\#A,aa,bb,cc,dd,ee)\ \longleftrightarrow\ sats(A,\ is\_tuple\_fm(a,b,c,d,e),\ env)$
using *assms* **by** (*simp add*: *sats_is_tuple_fm*)

17.2 Definition of *forces* for equality and membership

definition
eq_case :: $[i,i,i,i,i,i] \Rightarrow o$ **where**
$eq\_case(t1,t2,p,P,leq,f) \equiv \forall\, s.\ s{\in}domain(t1) \cup domain(t2) \longrightarrow$
$(\forall\, q.\ q{\in}P \wedge \langle q,p\rangle{\in}leq \longrightarrow (f`\langle 1,s,t1,q\rangle{=}1 \longleftrightarrow f`\langle 1,s,t2,q\rangle = 1))$

definition
is_eq_case :: $[i{\Rightarrow}o,i,i,i,i,i,i] \Rightarrow o$ **where**
$is\_eq\_case(M,t1,t2,p,P,leq,f) \equiv$
$\forall\, s[M].\ (\exists\, d[M].\ is\_domain(M,t1,d) \wedge s{\in}d) \vee (\exists\, d[M].\ is\_domain(M,t2,d) \wedge s{\in}d)$
$\longrightarrow (\forall\, q[M].\ q{\in}P \wedge (\exists\, qp[M].\ pair(M,q,p,qp) \wedge qp{\in}leq) \longrightarrow$
$(\exists\, ost1q[M].\ \exists\, ost2q[M].\ \exists\, o[M].\ \exists\, vf1[M].\ \exists\, vf2[M].$
$is\_tuple(M,o,s,t1,q,ost1q) \wedge$
$is\_tuple(M,o,s,t2,q,ost2q) \wedge number1(M,o) \wedge$
$fun\_apply(M,f,ost1q,vf1) \wedge fun\_apply(M,f,ost2q,vf2) \wedge$
$(vf1 = o \longleftrightarrow vf2 = o)))$

definition
mem_case :: $[i,i,i,i,i,i] \Rightarrow o$ **where**
$mem\_case(t1,t2,p,P,leq,f) \equiv \forall\, v{\in}P.\ \langle v,p\rangle{\in}leq \longrightarrow$
$(\exists\, q.\ \exists\, s.\ \exists\, r.\ r{\in}P \wedge q{\in}P \wedge \langle q,v\rangle{\in}leq \wedge \langle s,r\rangle \in t2 \wedge \langle q,r\rangle{\in}leq \wedge\ f`\langle 0,t1,s,q\rangle = 1)$

definition
is_mem_case :: $[i{\Rightarrow}o,i,i,i,i,i,i] \Rightarrow o$ **where**
$is\_mem\_case(M,t1,t2,p,P,leq,f) \equiv \forall\, v[M].\ \forall\, vp[M].\ v{\in}P \wedge pair(M,v,p,vp) \wedge vp{\in}leq \longrightarrow$
$(\exists\, q[M].\ \exists\, s[M].\ \exists\, r[M].\ \exists\, qv[M].\ \exists\, sr[M].\ \exists\, qr[M].\ \exists\, z[M].\ \exists\, zt1sq[M].\ \exists\, o[M].$
$r{\in}\ P \wedge q{\in}P \wedge pair(M,q,v,qv) \wedge pair(M,s,r,sr) \wedge pair(M,q,r,qr) \wedge$
$empty(M,z) \wedge is\_tuple(M,z,t1,s,q,zt1sq) \wedge$
$number1(M,o) \wedge qv{\in}leq \wedge sr{\in}t2 \wedge qr{\in}leq \wedge fun\_apply(M,f,zt1sq,o))$

schematic_goal *sats_is_mem_case_fm_auto*:
assumes
$n1{\in}nat\ n2{\in}nat\ p{\in}nat\ P{\in}nat\ leq{\in}nat\ f{\in}nat\ env{\in}list(A)$
shows
$is\_mem\_case(\#\#A, nth(n1, env), nth(n2, env), nth(p, env), nth(P, env), nth(leq, env), nth(f, env))$
$\longleftrightarrow sats(A, ?imc\_fm(n1,n2,p,P,leq,f), env)$
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 :: $[i,i,i,i] \Rightarrow i$ **where**
Hfrc_fm$(P,leq,fnnc,f) \equiv$
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*$(2,1,0,P\ \#+\ 4,leq\ \#+\ 4,f\ \#+$
$4)))))))))$

lemma *Hfrc_fm_type*$[TC]$:
$[\![P{\in}nat;leq{\in}nat;fnnc{\in}nat;f{\in}nat]\!] \Longrightarrow$ *Hfrc_fm*$(P,leq,fnnc,f){\in}formula$
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*) ≡ *Or*(*And*(*empty_fm*(*z*),*Neg*(*Hfrc_fm*(*P*,*leq*,*fnnc*,*f*))),
And(*number1_fm*(*z*),*Hfrc_fm*(*P*,*leq*,*fnnc*,*f*)))

lemma *arity_Hfrc_at_fm* :
assumes
P∈*nat leq*∈*nat fnnc*∈*nat f*∈*nat z*∈*nat*
shows
arity(*Hfrc_at_fm*(*P*,*leq*,*fnnc*,*f*,*z*)) = *succ*(*P*) ∪ *succ*(*leq*) ∪ *succ*(*fnnc*) ∪ *succ*(*f*)
∪ *succ*(*z*)
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*

17.3 The well-founded relation *forcerel*

definition
forcerel :: $i \Rightarrow i \Rightarrow i$ **where**
$forcerel(P,x) \equiv frecrel(names\_below(P,x))^+$

definition
is_forcerel :: $[i \Rightarrow o,i,i,i] \Rightarrow o$ **where**
$is\_forcerel(M,P,x,z) \equiv \exists\, r[M].\ \exists\, nb[M].\ tran\_closure(M,r,z) \wedge$
$(is\_names\_below(M,P,x,nb) \wedge is\_frecrel(M,nb,r))$

definition
forcerel_fm :: $i \Rightarrow i \Rightarrow i \Rightarrow i$ **where**
$forcerel\_fm(p,x,z) \equiv Exists(Exists(And(trans\_closure\_fm(1,\ z\#+2),$
$And(is\_names\_below\_fm(p\#+2,x\#+2,0),frecrel\_fm(0,1)))))$

lemma *arity_forcerel_fm*:
$[\![p{\in}nat;x{\in}nat;z{\in}nat]\!] \Longrightarrow arity(forcerel\_fm(p,x,z)) = succ(p) \cup succ(x) \cup succ(z)$
unfolding *forcerel_fm_def*
using *arity_frecrel_fm arity_tran_closure_fm arity_is_names_below_fm pred_Un_distrib*
by *auto*

lemma *forcerel_fm_type*[*TC*]:
$[\![p{\in}nat;x{\in}nat;z{\in}nat]\!] \Longrightarrow forcerel\_fm(p,x,z){\in}formula$
unfolding *forcerel_fm_def* **by** *simp*

lemma *sats_forcerel_fm*:
assumes
$p{\in}nat\ x{\in}nat\ \ z{\in}nat\ env{\in}list(A)$
shows
$sats(A,forcerel\_fm(p,x,z),env) \longleftrightarrow is\_forcerel(\#\#A,nth(p,env),nth(x,\ env),nth(z,\ env))$
proof -
have $sats(A,trans\_closure\_fm(1,z\ \#+\ 2),Cons(nb,Cons(r,env))) \longleftrightarrow$
$tran\_closure(\#\#A,\ r,\ nth(z,\ env))$ **if** $r{\in}A\ nb{\in}A$ **for** $r\ nb$
using *that assms trans_closure_fm_iff_sats*[*of 1* $[nb,r]@env\_\ z\#+2$,*symmetric*]
by *simp*
moreover
have $sats(A,\ is\_names\_below\_fm(succ(succ(p)),\ succ(succ(x)),\ 0),\ Cons(nb,\ Cons(r,\ env))) \longleftrightarrow$
$is\_names\_below(\#\#A,\ nth(p,env),\ nth(x,\ env),\ nb)$
if $r{\in}A\ nb{\in}A$ **for** $nb\ r$
using *assms that sats_is_names_below_fm*[*of p #+ 2 x #+ 2 0* $[nb,r]@env$]
by *simp*
moreover
have $sats(A,\ frecrel\_fm(0,\ 1),\ Cons(nb,\ Cons(r,\ env))) \longleftrightarrow$
$is\_frecrel(\#\#A,\ nb,\ r)$
if $r{\in}A\ nb{\in}A$ **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

17.4 *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*[*simp*] :
⟦*x∈M*; *y∈M* ⟧ ⟹ *is_ftype(##M,x,y) ⟷ y = ftype(x)* **unfolding** *ftype_def is_ftype_def* **by** *simp*

lemma *name1_abs*[*simp*] :
⟦*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*[*simp*]:

$[\![x{\in}M;\ y{\in}M\ ]\!] \Longrightarrow is\_name2(\#\#M,x,y) \longleftrightarrow y = name2(x)$
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*[*simp*]:
$[\![x{\in}M;\ y{\in}M\ ]\!] \Longrightarrow is\_cond\_of(\#\#M,x,y) \longleftrightarrow y = cond\_of(x)$
unfolding *cond_of_def is_cond_of_def*
by (*rule hcomp_abs*[*OF snd_abs snd_snd_abs*];*simp_all add*:*fst_snd_closed*)

lemma *tuple_abs*[*simp*]:
$[\![z{\in}M;t1{\in}M;t2{\in}M;p{\in}M;t{\in}M]\!] \Longrightarrow$
$is\_tuple(\#\#M,z,t1,t2,p,t) \longleftrightarrow t = \langle z,t1,t2,p\rangle$
unfolding *is_tuple_def* **using** *tuples_in_M* **by** *simp*

lemma *oneN_in_M*[*simp*]: $1{\in}M$
by (*simp flip*: *setclass_iff*)

lemma *twoN_in_M* : $2{\in}M$
by (*simp flip*: *setclass_iff*)

lemma *comp_in_M*:
$p \preceq q \Longrightarrow p{\in}M$
$p \preceq q \Longrightarrow q{\in}M$
using *leq_in_M transitivity*[*of _ leq*] *pair_in_M_iff* **by** *auto*

lemma *eq_case_abs* [*simp*]:
assumes
$t1{\in}M\ t2{\in}M\ p{\in}M\ f{\in}M$
shows
$is\_eq\_case(\#\#M,t1,t2,p,P,leq,f) \longleftrightarrow eq\_case(t1,t2,p,P,leq,f)$
proof -
have $q \preceq p \Longrightarrow q{\in}M$ **for** q
using *comp_in_M* **by** *simp*
moreover
have $\langle s,y\rangle{\in}t \Longrightarrow s{\in}domain(t)$ **if** $t{\in}M$ **for** $s\ y\ t$
using *that* **unfolding** *domain_def* **by** *auto*
ultimately
have
$(\forall\, s{\in}M.\ s \in domain(t1) \vee s \in domain(t2) \longrightarrow (\forall\, q{\in}M.\ q{\in}P \wedge q \preceq p \longrightarrow$
$(f\ `\ \langle 1,\ s,\ t1,\ q\rangle = 1 \longleftrightarrow f\ `\ \langle 1,\ s,\ t2,\ q\rangle{=}1))) \longleftrightarrow$
$(\forall\, s.\ s \in domain(t1) \vee s \in domain(t2) \longrightarrow (\forall\, q.\ q{\in}P \wedge q \preceq p \longrightarrow$
$(f\ `\ \langle 1,\ s,\ t1,\ q\rangle = 1 \longleftrightarrow f\ `\ \langle 1,\ s,\ t2,\ q\rangle{=}1)))$
using *assms domain_trans*[*OF trans_M*,*of t1*]
domain_trans[*OF trans_M*,*of t2*] **by** *auto*
then show *?thesis*
unfolding *eq_case_def is_eq_case_def*
using *assms pair_in_M_iff n_in_M*[*of 1*] *domain_closed tuples_in_M*

```
    apply_closed leq_in_M
  by simp
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
  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 $\exists\, q{\in}M\ .\ \exists\, s{\in}M.\ \exists\, r{\in}M.$
$r \in P \wedge q \in P \wedge \langle q,\, v\rangle \in M \wedge \langle s,\, r\rangle \in M \wedge \langle q,\, r\rangle \in M \wedge 0 \in M \wedge$
$\langle 0,\, t1,\, s,\, q\rangle \in M \wedge q \preceq v \wedge \langle s,\, r\rangle \in t2 \wedge q \preceq r \wedge f\ `\ \langle 0,\, t1,\, s,\, q\rangle = 1$
using *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*
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*

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 *is_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* :

$[\![x{\in}M;ec{\in}M]\!] \Longrightarrow is\_ecloseN(\#\#M,ec,x) \longleftrightarrow ec = ecloseN(x)$
unfolding *is_ecloseN_def ecloseN_def*
using *is_eclose_n_abs Un_closed union_abs ecloseN_closed*
by *auto*

lemma *frecR_abs* :
$x{\in}M \Longrightarrow y{\in}M \Longrightarrow frecR(x,y) \longleftrightarrow is\_frecR(\#\#M,x,y)$
unfolding *frecR_def is_frecR_def* **using** *components_closed domain_closed* **by** *simp*

lemma *frecrelP_abs* :
$z{\in}M \Longrightarrow frecrelP(\#\#M,z) \longleftrightarrow (\exists\, x\ y.\ z = \langle x,y\rangle \wedge frecR(x,y))$
using *pair_in_M_iff frecR_abs* **unfolding** *frecrelP_def* **by** *auto*

lemma *frecrel_abs*:
assumes
$A{\in}M\ r{\in}M$
shows
$is\_frecrel(\#\#M,A,r) \longleftrightarrow\ r = frecrel(A)$
proof -
from ‹$A{\in}M$›
have $z{\in}M$ **if** $z{\in}A{\times}A$ **for** z
using *cartprod_closed transitivity that* **by** *simp*
then
have $Collect(A{\times}A,frecrelP(\#\#M)) = Collect(A{\times}A,\lambda z.\ (\exists\, x\ y.\ z = \langle x,y\rangle \wedge frecR(x,y)))$
using *Collect_cong*[*of* $A{\times}A$ $A{\times}A$ $frecrelP(\#\#M)$] *assms frecrelP_abs* **by** *simp*
with *assms*
show *?thesis* **unfolding** *is_frecrel_def def_frecrel* **using** *cartprod_closed*
by *simp*
qed

lemma *frecrel_closed*:
assumes
$x{\in}M$
shows
$frecrel(x){\in}M$
proof -
have $Collect(x{\times}x,\lambda z.\ (\exists\, x\ y.\ z = \langle x,y\rangle \wedge frecR(x,y))){\in}M$
using *Collect_in_M_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 $frecR(\langle ft,n1,n2,p\rangle,\langle a,b,c,d\rangle)$ $p \in P$
shows $\langle ft,n1,n2,p\rangle \in names\_below(P,\langle a,b,c,d\rangle)$ (**is** $?x \in names\_below(\_,?y)$)
proof -
from *assms*
have $ft \in 2$ $a \in 2$
unfolding *frecR_def* **by** (*auto simp add*:*components_simp*)
from *assms*
consider (*e*) $n1 \in domain(b) \cup domain(c) \wedge (n2 = b \vee n2 = c)$
| (*m*) $n1 = b \wedge n2 \in domain(c)$
unfolding *frecR_def* **by** (*auto simp add*:*components_simp*)
then show *?thesis*
proof *cases*
case *e*
then
have $n1 \in eclose(b) \vee n1 \in eclose(c)$
using *Un_iff in_dom_in_eclose* **by** *auto*
with *e*
have $n1 \in ecloseN(?y)$ $n2 \in ecloseN(?y)$
using *ecloseNI components_in_eclose* **by** *auto*
with ‹$ft \in 2$› ‹$p \in P$›
show *?thesis* **unfolding** *names_below_def* **by** *auto*
next
case *m*
then
have $n1 \in ecloseN(?y)$ $n2 \in ecloseN(?y)$
using *mem_eclose_trans ecloseNI*
in_dom_in_eclose components_in_eclose **by** *auto*
with ‹$ft \in 2$› ‹$p \in P$›
show *?thesis* **unfolding** *names_below_def*
by *auto*
qed
qed

lemma *names_below_tr* :
  assumes $x\in$ *names_below*(P,y)
    $y\in$ *names_below*(P,z)
  shows $x\in$ *names_below*(P,z)
proof -
  let *?A*=λy . *names_below*(P,y)
  from *assms*
  obtain *fx x1 x2 px* **where**
    $x = \langle fx,x1,x2,px\rangle$ *fx*$\in$*2 x1*$\in$*ecloseN*(y) *x2*$\in$*ecloseN*(y) *px*$\in P$
    unfolding *names_below_def* **by** *auto*
  from *assms*
  obtain *fy y1 y2 py* **where**
    $y = \langle fy,y1,y2,py\rangle$ *fy*$\in$*2 y1*$\in$*ecloseN*(z) *y2*$\in$*ecloseN*(z) *py*$\in P$
    unfolding *names_below_def* **by** *auto*
  from ‹*x1*$\in$*_*› ‹*x2*$\in$*_*› ‹*y1*$\in$*_*› ‹*y2*$\in$*_*› ‹*x*=*_*› ‹*y*=*_*›
  have *x1*$\in$*ecloseN*(z) *x2*$\in$*ecloseN*(z)
    using *ecloseN_mono names_simp* **by** *auto*
  with ‹*fx*$\in$*2*› ‹*px*$\in P$› ‹*x*=*_*›
  have *x*$\in$*?A*(z)
    unfolding *names_below_def* **by** *simp*
  then show *?thesis* **using** *subsetI* **by** *simp*
qed

lemma *arg_into_names_below2* :
  assumes $\langle x,y\rangle \in$ *frecrel*(*names_below*(P,z))
  shows $x \in$ *names_below*(P,y)
proof -
  {
    from *assms*
    have *x*$\in$*names_below*(P,z) *y*$\in$*names_below*(P,z) *frecR*(x,y)
      unfolding *frecrel_def Rrel_def*
      by *auto*
    obtain *f n1 n2 p* **where**
      $x = \langle f,n1,n2,p\rangle$ *f*$\in$*2 n1*$\in$*ecloseN*(z) *n2*$\in$*ecloseN*(z) *p*$\in P$
      using ‹*x*$\in$*names_below*(P,z)›
      unfolding *names_below_def* **by** *auto*
    moreover
    obtain *fy m1 m2 q* **where**
      $q\in P$ $y = \langle fy,m1,m2,q\rangle$
      using ‹*y*$\in$*names_below*(P,z)›
      unfolding *names_below_def* **by** *auto*
    moreover
    note ‹*frecR*(x,y)›
    ultimately
    have *x*$\in$*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 $\langle x,y\rangle \in$ *frecrel*(*names_below*(P,z))
shows *forcerel*(P,x) $\subseteq$ *forcerel*(P,y)
unfolding *forcerel_def*
using *trancl_mono frecrel_mono assms* **by** *simp*

lemma *forcerel_mono_aux* :
assumes $\langle x,y\rangle \in$ *frecrel*(*names_below*(P, w))$^+$
shows *forcerel*(P,x) $\subseteq$ *forcerel*(P,y)
using *assms*
by (*rule trancl_induct*,*simp_all add*: *subset_trans forcerel_mono2*)

lemma *forcerel_mono* :
assumes $\langle x,y\rangle \in$ *forcerel*(P,z)
shows *forcerel*(P,x) $\subseteq$ *forcerel*(P,y)
using *forcerel_mono_aux assms* **unfolding** *forcerel_def* **by** *simp*

lemma *aux*: $x \in$ *names_below*(P, w) $\Longrightarrow$ $\langle x,y\rangle \in$ *forcerel*(P,z) $\Longrightarrow$
($y \in$ *names_below*(P, w) $\longrightarrow$ $\langle x,y\rangle \in$ *forcerel*(P,w))
unfolding *forcerel_def*
proof(*rule_tac* a=x **and** b=y **and** P=λ y . $y \in$ *names_below*(P, w) $\longrightarrow$ $\langle x,y\rangle \in$ *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 $u \in$*?A*(w) $\longrightarrow$ $\langle x,u\rangle \in$*?R*(w)$^+$ **if** $x \in$*?A*(w) $\langle x,y\rangle \in$*?R*(z)$^+$ $\langle x,u\rangle \in$*?R*(z)
for u
using *that frecrelD frecrelI r_into_trancl* **unfolding** *names_below_def* **by** *simp*
{
fix u v
assume $x \in$ *?A*(w)
$\langle x,\ y\rangle \in$ *?R*(z)$^+$
$\langle x,\ u\rangle \in$ *?R*(z)$^+$
$\langle u,\ v\rangle \in$ *?R*(z)
$u \in$ *?A*(w) $\Longrightarrow$ $\langle x,\ u\rangle \in$ *?R*(w)$^+$
then
have $v \in$ *?A*(w) $\Longrightarrow$ $\langle x,\ v\rangle \in$ *?R*(w)$^+$
proof -
assume $v \in$*?A*(w)
from ‹$\langle u,v\rangle \in$_›
have $u \in$*?A*(v)
using *arg_into_names_below2* **by** *simp*
with ‹$v \in$*?A*(w)›
have $u \in$*?A*(w)
using *names_below_tr* **by** *simp*
with ‹$v \in$_› ‹$\langle u,v\rangle \in$_›
have $\langle u,v\rangle \in$ *?R*(w)
using *frecrelD frecrelI r_into_trancl* **unfolding** *names_below_def* **by** *simp*
with ‹$u \in$ *?A*(w) $\Longrightarrow$ $\langle x,\ u\rangle \in$ *?R*(w)$^+$› ‹$u \in$*?A*(w)›

```
    have ⟨x, u⟩ ∈ ?R(w)^+ by simp
    with ‹⟨u,v⟩∈ ?R(w)›
    show ⟨x,v⟩∈ ?R(w)^+ using trancl_trans r_into_trancl
      by simp
  qed
 }
 then show v ∈ ?A(w) ⟶ ⟨x, v⟩ ∈ ?R(w)^+
  if x ∈ ?A(w)
    ⟨x, y⟩ ∈ ?R(z)^+
    ⟨x, u⟩ ∈ ?R(z)^+
    ⟨u, v⟩ ∈ ?R(z)
    u ∈ ?A(w) ⟶ ⟨x, u⟩ ∈ ?R(w)^+ for u v
  using that by simp
qed

lemma forcerel_eq :
 assumes ⟨z,x⟩ ∈ forcerel(P,x)
 shows forcerel(P,z) = forcerel(P,x) ∩ names_below(P,z)×names_below(P,z)
 using assms aux forcerelD forcerel_mono[of z x x] subsetI
 by auto

lemma forcerel_below_aux :
 assumes ⟨z,x⟩ ∈ forcerel(P,x) ⟨u,z⟩ ∈ forcerel(P,x)
 shows u ∈ names_below(P,z)
 using assms(2)
 unfolding forcerel_def
proof(rule trancl_induct)
 show  u ∈ names_below(P,y) if  ⟨u, y⟩ ∈ frecrel(names_below(P, x)) for y
  using that vimage_singleton_iff arg_into_names_below2 by simp
next
 show u ∈ names_below(P,z)
  if ⟨u, y⟩ ∈ frecrel(names_below(P, x))^+
    ⟨y, z⟩ ∈ frecrel(names_below(P, x))
    u ∈ names_below(P, y)
  for y z
  using that arg_into_names_below2[of y z x] names_below_tr by simp
qed

lemma forcerel_below :
 assumes ⟨z,x⟩ ∈ forcerel(P,x)
 shows forcerel(P,x) -`` {z} ⊆ names_below(P,z)
 using vimage_singleton_iff assms forcerel_below_aux by auto

lemma relation_forcerel :
 shows relation(forcerel(P,z)) trans(forcerel(P,z))
 unfolding forcerel_def using relation_trancl trans_trancl by simp_all

lemma Hfrc_restrict_trancl: bool_of_o(Hfrc(P, leq, y, restrict(f,frecrel(names_below(P,x))-``{y})))
     = bool_of_o(Hfrc(P, leq, y, restrict(f,(frecrel(names_below(P,x)))^+)-``{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*,λ*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* ∈ *domain*(*b*) ∨ *n1* ∈ *domain*(*c*) *p*∈*P d*∈*P*
shows ⟨⟨*1*, *n1*, *b*, *p*⟩, ⟨*0*,*b*,*c*,*d*⟩⟩∈ *forcerel*(*P*,⟨*0*,*b*,*c*,*d*⟩)
proof -
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 $\langle n2,\ r\rangle \in c\ p{\in}P\ d{\in}P\ r \in P$
shows $\langle\langle 0,\ b,\ n2,\ p\rangle,\langle 1,\ b,\ c,\ d\rangle\rangle \in forcerel(P,\langle 1,b,c,d\rangle)$
proof -
let $?x{=}\langle 0,\ b,\ n2,\ p\rangle$
let $?y{=}\langle 1,\ b,\ c,\ d\rangle$
from *assms*
have $frecR(?x,?y)$
using *assms frecRI3* **by** *simp*
then
have $?x{\in}names\_below(P,?y)\quad ?y \in names\_below(P,?y)$
using *names_belowI assms components_in_eclose*
unfolding *names_below_def* **by** *auto*
with ‹$frecR(?x,?y)$›
show *?thesis*
unfolding *forcerel_def frecrel_def*
using *subsetD*[*OF r_subset_trancl*[*OF relation_Rrel*]] *RrelI*
by *auto*
qed

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 $E{:}forcerel(P,z) = forcerel(P,x) \cap (?A{\times}?A)$
$forcerel(P,x)$ -‘‘ $\{z\} \subseteq ?A$
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

17.5 Recursive expression of *frc_at*

lemma *def_frc_at* :
assumes $p \in P$
shows
$frc\_at(P,leq,\langle ft,n1,n2,p\rangle) =$
$bool\_of\_o(\ p \in P \wedge$
$(\ ft = 0 \wedge\ (\forall\, s.\ s \in domain(n1) \cup domain(n2) \longrightarrow$
$(\forall\, q.\ q \in P \wedge q \preceq p \longrightarrow (frc\_at(P,leq,\langle 1,s,n1,q\rangle) = 1 \longleftrightarrow frc\_at(P,leq,\langle 1,s,n2,q\rangle) = 1)))$
$\vee\ ft = 1 \wedge (\ \forall\, v \in P.\ v \preceq p \longrightarrow$
$(\exists\, q.\ \exists\, s.\ \exists\, r.\ r \in P \wedge q \in P \wedge q \preceq v \wedge \langle s,r\rangle \in n2 \wedge q \preceq r \wedge\ frc\_at(P,leq,\langle 0,n1,s,q\rangle) = 1))))$
proof -
let $?r = \lambda y.\ forcerel(P,y)$ **and** $?Hf = \lambda x\ f.\ bool\_of\_o(Hfrc(P,leq,x,f))$
let $?t = \lambda y.\ ?r(y)$ -‘‘ $\{y\}$
let $?arg = \langle ft,n1,n2,p\rangle$
from *wf_forcerel*
have *wfr*: $\forall\, w\ .\ wf(?r(w))$ **..**
with *wfrec* [*of* $?r(?arg)$ *?arg ?Hf*]
have $frc\_at(P,leq,?arg) = ?Hf(\ ?arg,\ \lambda x \in ?r(?arg)$ -‘‘ $\{?arg\}.\ wfrec(?r(?arg),\ x,\ ?Hf))$
using *frc_at_trancl* **by** *simp*
also
have $\ldots = ?Hf(\ ?arg,\ \lambda x \in ?r(?arg)$ -‘‘ $\{?arg\}.\ frc\_at(P,leq,x))$
using *aux_def_frc_at frc_at_trancl* **by** *simp*
finally
show *?thesis*
unfolding *Hfrc_def mem_case_def eq_case_def*
using *forcerelI assms*
by *auto*
qed

17.6 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
$x \in M$
shows
$forcerel(P,x) \in M$
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*,λ*z*. ∃ *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_defs frecR_fm_def number1_fm_def components_defs nat_simp_union*)
then
have *separation*(##*M*, λ*z*. *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*) λ*z*. ∃ *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*),λ*x 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 n_in_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 $a{\in}M$ $b{\in}M$ $c{\in}M$ $d{\in}M$ $e{\in}M$ $y{\in}M$ $x{\in}M$ $z{\in}M$
for a b c d e y x z
using *that P_in_M leq_in_M* ‹$X{\in}M$› *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*)])
$\longleftrightarrow$
is_wfrec(##*M*, *is_Hfrc_at*(##*M*,*P*,*leq*),*forcerel*(*P*,*X*), *x*, *y*)
if $x{\in}M$ $y{\in}M$ $z{\in}M$ **for** x y z
using *that* ‹$X{\in}M$› *forcerel_in_M P_in_M leq_in_M*
sats_is_wfrec_fm[*OF 0*]
by *simp*
let
?f=*Exists*(*And*(*pair_fm*(*1*,*0*,*2*),*is_wfrec_fm*(*Hfrc_at_fm*(*8*,*9*,*2*,*1*,*0*),*5*,*1*,*0*)))
have *satsf*:*sats*(*M*, *?f*, [*x*,*z*,*P*,*leq*,*forcerel*(*P*,*X*)]) $\longleftrightarrow$
($\exists$ *y*$\in$*M*. *pair*(##*M*,*x*,*y*,*z*) & *is_wfrec*(##*M*, *is_Hfrc_at*(##*M*,*P*,*leq*),*forcerel*(*P*,*X*),
x, *y*))
if $x{\in}M$ $z{\in}M$ **for** x z
using *that 1* ‹$X{\in}M$› *forcerel_in_M P_in_M leq_in_M* **by** (*simp del*:*pair_abs*)
have *artyf*:*arity*(*?f*) = *5*
unfolding *is_wfrec_fm_def Hfrc_at_fm_def Hfrc_fm_def Replace_fm_def*
PHcheck_fm_def
pair_fm_def upair_fm_def is_recfun_fm_def fun_apply_fm_def big_union_fm_def
pre_image_fm_def restriction_fm_def image_fm_def fm_defs number1_fm_def
eq_case_fm_def mem_case_fm_def is_tuple_fm_def
by (*simp add*:*nat_simp_union*)
moreover
have *?f*$\in$*formula*
unfolding *fm_defs Hfrc_at_fm_def* **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* ‹$X{\in}M$› *forcerel_in_M P_in_M leq_in_M* **by**
simp
then
have *strong_replacement*(##*M*,λ*x z*.
$\exists$ *y*$\in$*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*$\in$*M*;*x*$\in$*M*;*nb*$\in$*M*⟧ $\Longrightarrow$ *is_names_below*(##*M*,*Q*,*x*,*nb*) $\longleftrightarrow$ *nb* = *names_below*(*Q*,*x*)
unfolding *is_names_below_def names_below_def*
using *succ_in_M_iff zero_in_M cartprod_closed 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*

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*

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*,*blast*)
qed

lemma *forces_nmem'_abs* :
assumes
p∈M t1∈M t2∈M
shows
is_forces_nmem'(##M,P,leq,p,t1,t2) ⟷ forces_nmem'(P,leq,p,t1,t2)
proof -
have *q∈M* **if** *q∈P* **for** *q*
using *that transitivity P_in_M* **by** *simp*
then show *?thesis*
unfolding *is_forces_nmem'_def forces_nmem'_def*
using *assms forces_mem'_abs pair_in_M_iff*
by (*auto*,*blast*)
qed

end

17.7 Forcing for general formulas

definition
ren_forces_nand :: *i⇒i* **where**
ren_forces_nand(φ) ≡ Exists(And(Equal(0,1),iterates(λp. incr_bv(p)`1 , 2, φ)))

lemma *ren_forces_nand_type*[*TC*] :
φ∈formula ⟹ ren_forces_nand(φ) ∈formula
unfolding *ren_forces_nand_def*
by *simp*

lemma *arity_ren_forces_nand* :
assumes *φ∈formula*
shows *arity(ren_forces_nand(φ)) ≤ succ(arity(φ))*
proof -

```
  consider (lt) 1<arity(φ) | (ge) ¬ 1 < arity(φ)
    by auto
  then
  show ?thesis
  proof cases
    case lt
    with ‹φ∈_›
    have 2 < succ(arity(φ)) 2<arity(φ)#+2
      using succ_ltI by auto
    with ‹φ∈_›
    have arity(iterates(λp. incr_bv(p)‘1,2,φ)) = 2#+arity(φ)
      using arity_incr_bv_lemma lt
      by auto
    with ‹φ∈_›
    show ?thesis
      unfolding ren_forces_nand_def
    using lt pred_Un_distrib 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(λp. incr_bv(p)‘5,5,φ)) = arity(φ)
      using arity_incr_bv_lemma ge
      by simp
    with ‹arity(φ) ≤ 5› ‹φ∈_› ‹pred^5(_) ≤ 5›
    show ?thesis
      unfolding ren_forces_forall_def
     using pred_Un_distrib 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 ⟹
```

$sats(M,\ ren\_forces\_forall(\varphi),[x,p,P,leq,o]\ @\ env) \longleftrightarrow sats(M,\ \varphi,[p,P,leq,o,x]\ @\ env)$
unfolding *ren_forces_forall_def*
using *sats_incr_bv_iff* [*of* _ _ *M* _ [*p*,*P*,*leq*,*o*,*x*]]
by *simp*

definition
$is\_leq :: [i\Rightarrow o,i,i,i] \Rightarrow o$ **where**
$is\_leq(A,l,q,p) \equiv \exists\, qp[A].\ (pair(A,q,p,qp) \wedge qp\in l)$

lemma (**in** *forcing_data*) *leq_abs*[*simp*]:
$[\![\ l\in M\ ;\ q\in M\ ;\ p\in M\ ]\!] \Longrightarrow is\_leq(\#\#M,l,q,p) \longleftrightarrow \langle q,p\rangle\in l$
unfolding *is_leq_def* **using** *pair_in_M_iff* **by** *simp*

definition
$leq\_fm :: [i,i,i] \Rightarrow i$ **where**
$leq\_fm(leq,q,p) \equiv Exists(And(pair\_fm(q\#+1,p\#+1,0),Member(0,leq\#+1)))$

lemma *arity_leq_fm* :
$[\![leq\in nat;q\in nat;p\in nat]\!] \Longrightarrow arity(leq\_fm(leq,q,p)) = succ(q) \cup succ(p) \cup succ(leq)$
unfolding *leq_fm_def*
using *arity_pair_fm pred_Un_distrib nat_simp_union*
by *auto*

lemma *leq_fm_type*[*TC*] :
$[\![leq\in nat;q\in nat;p\in nat]\!] \Longrightarrow leq\_fm(leq,q,p)\in formula$
unfolding *leq_fm_def* **by** *simp*

lemma *sats_leq_fm* :
$[\![\ leq\in nat;q\in nat;p\in nat;env\in list(A)\ ]\!] \Longrightarrow$
$sats(A,leq\_fm(leq,q,p),env) \longleftrightarrow is\_leq(\#\#A,nth(leq,env),nth(q,env),nth(p,env))$
unfolding *leq_fm_def is_leq_def* **by** *simp*

17.7.1 The primitive recursion

consts $forces' :: i\Rightarrow i$
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
$forces :: i\Rightarrow i$ **where**

$forces(\varphi) \equiv And(Member(0,1),forces'(\varphi))$

lemma *forces'_type* [*TC*]: $\varphi{\in}formula \Longrightarrow forces'(\varphi) \in formula$
by (*induct* φ *set:formula*; *simp*)

lemma *forces_type*[*TC*] : $\varphi{\in}formula \Longrightarrow forces(\varphi) \in formula$
unfolding *forces_def* **by** *simp*

context *forcing_data*
begin

17.8 Forcing for atomic formulas in context

definition
$forces\_eq :: [i,i,i] \Rightarrow o$ **where**
$forces\_eq \equiv forces\_eq'(P,leq)$

definition
$forces\_mem :: [i,i,i] \Rightarrow o$ **where**
$forces\_mem \equiv forces\_mem'(P,leq)$

definition
$is\_forces\_eq :: [i,i,i] \Rightarrow o$ **where**
$is\_forces\_eq \equiv is\_forces\_eq'(\#\#M,P,leq)$

definition
$is\_forces\_mem :: [i,i,i] \Rightarrow o$ **where**
$is\_forces\_mem \equiv is\_forces\_mem'(\#\#M,P,leq)$

lemma *def_forces_eq*: $p{\in}P \Longrightarrow forces\_eq(p,t1,t2) \longleftrightarrow$
$(\forall\, s{\in}domain(t1) \cup domain(t2).\ \forall\, q.\ q{\in}P \wedge q \preceq p \longrightarrow$
$(forces\_mem(q,s,t1) \longleftrightarrow forces\_mem(q,s,t2)))$
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∈M ; t1∈M ; t2∈M⟧ ⟹ is_forces_mem(p,t1,t2) ⟷ 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∈nat l∈nat q∈nat t1∈nat t2∈nat env∈list(M)*
nth(p,env)=P nth(l,env)=leq
shows *sats(M,forces_eq_fm(p,l,q,t1,t2),env) ⟷*
is_forces_eq(nth(q,env),nth(t1,env),nth(t2,env))
unfolding *forces_eq_fm_def is_forces_eq_def is_forces_eq′_def*
using *assms sats_is_tuple_fm sats_frc_at_fm*
by *simp*

lemma *sats_forces_mem_fm*:
assumes *p∈nat l∈nat q∈nat t1∈nat t2∈nat env∈list(M)*
nth(p,env)=P nth(l,env)=leq
shows *sats(M,forces_mem_fm(p,l,q,t1,t2),env) ⟷*
is_forces_mem(nth(q,env),nth(t1,env),nth(t2,env))
unfolding *forces_mem_fm_def is_forces_mem_def is_forces_mem′_def*
using *assms sats_is_tuple_fm sats_frc_at_fm*
by *simp*

definition
forces_neq :: *[i,i,i] ⇒ o* **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 $x{\in}nat$ $y{\in}nat$ $env{\in}list(M)$
$nth(x,env){=}xx$ $nth(y,env){=}yy$ $q{\in}M$
shows $sats(M,forces(Equal(x,y)),[q,P,leq,one]@env) \longleftrightarrow$
$(q{\in}P \wedge is\_forces\_eq(q,xx,yy))$
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 $\varphi{\in}formula$ $\psi{\in}formula$ $env{\in}list(M)$ $p{\in}M$
shows $sats(M,forces(Nand(\varphi,\psi)),[p,P,leq,one]@env) \longleftrightarrow$
$(p{\in}P \wedge \neg(\exists\, q{\in}M.\ q{\in}P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
$(sats(M,forces'(\varphi),[q,P,leq,one]@env) \wedge sats(M,forces'(\psi),[q,P,leq,one]@env))))$
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

17.9 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 φ∈*formula*
  shows *arity*(*forces'*(φ)) ≤ *arity*(φ) #+ *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 *?*φ' = *ren_forces_nand*(*forces'*(φ))
  let *?*ψ' = *ren_forces_nand*(*forces'*(ψ))
  have *arity*(*leq_fm*(*3*, *0*, *1*)) = *4*
    using *arity_leq_fm succ_Un_distrib 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

18 The Forcing Theorems

theory *Forcing_Theorems*
imports
Forces_Definition

begin

context *forcing_data*
begin

18.1 The forcing relation in context

abbreviation *Forces* :: [*i*, *i*, *i*] ⇒ *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*(

$\{q{\in}P.\ \exists\, s.\ \exists\, r.\ r{\in}P \wedge \langle s,r\rangle \in t2 \wedge q{\preceq}r \wedge forces\_eq(q,t1,s)\}$
,p)
using *def_forces_mem*[*of p t1 t2*] **by** *blast*

18.2 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_eq*:
assumes $p{\in}P$ $r{\in}P$ $r{\preceq}p$ $forces\_eq(p,t1,t2)$
shows $forces\_eq(r,t1,t2)$
using *assms def_forces_eq*[*of _ t1 t2*] *leq_transD* **by** *blast*

18.3 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_mem*:
assumes $p{\in}P$ $r{\in}P$ $r{\preceq}p$ $forces\_mem(p,t1,t2)$
shows $forces\_mem(r,t1,t2)$
using *assms forces_mem_iff_dense_below dense_below_under* **by** *auto*

18.4 Kunen 2013, Lemma IV.2.37(b)

lemma *density_mem*:
assumes $p{\in}P$
shows $forces\_mem(p,t1,t2) \longleftrightarrow dense\_below(\{q{\in}P.\ forces\_mem(q,t1,t2)\},p)$
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 $t1$] *leq_reflI* *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*({$q{\in}P$. *forces_eq*(q,$t1$,$t2$)},p)
using *strengthening_eq* *ideal_dense_below* **by** *auto*
next
assume *dense_below*({$q{\in}P$. *forces_eq*(q,$t1$,$t2$)},p)
{
fix s q
let *?D1*={$q'{\in}P$. $\forall\, s{\in}domain(t1) \cup domain(t2).\ \forall\, q.\ q \in P \wedge q{\preceq}q' \longrightarrow$
forces_mem(q,s,$t1$)$\longleftrightarrow$*forces_mem*(q,s,$t2$)}
let *?D2*={$q'{\in}P$. $\forall\, q.\ q{\in}P \wedge q{\preceq}q' \longrightarrow$ *forces_mem*(q,s,$t1$) $\longleftrightarrow$ *forces_mem*(q,s,$t2$)}
assume $s{\in}domain(t1) \cup domain(t2)$
then
have *?D1*$\subseteq$*?D2* **by** *blast*
with ‹*dense_below*(_,p)›
have *dense_below*({$q'{\in}P$. $\forall\, s{\in}domain(t1) \cup domain(t2).\ \forall\, q.\ q \in P \wedge q{\preceq}q'$
$\longrightarrow$
forces_mem(q,s,$t1$)$\longleftrightarrow$*forces_mem*(q,s,$t2$)},p)
using *dense_below_cong'*[*OF* ‹$p{\in}P$› *def_forces_eq*[*of* _ $t1$ $t2$]] **by** *simp*
with ‹$p{\in}P$› ‹*?D1*$\subseteq$*?D2*›
have *dense_below*({$q'{\in}P$. $\forall\, q.\ q{\in}P \wedge q{\preceq}q' \longrightarrow$
forces_mem(q,s,$t1$) $\longleftrightarrow$ *forces_mem*(q,s,$t2$)},p)
using *dense_below_mono* **by** *simp*
moreover from *this*
have *dense_below*({$q'{\in}P$. $\forall\, q.\ q{\in}P \wedge q{\preceq}q' \longrightarrow$
forces_mem(q,s,$t2$) $\longleftrightarrow$ *forces_mem*(q,s,$t1$)},p)
by *blast*
moreover

assume $q \in P$ $q \preceq p$
moreover
note ‹$p \in P$›
ultimately
have *forces_mem*(q,s,$t1$) $\Longrightarrow$ *dense_below*($\{r \in P.$ *forces_mem*(r,s,$t2$)$\}$,q)
forces_mem(q,s,$t2$) $\Longrightarrow$ *dense_below*($\{r \in P.$ *forces_mem*(r,s,$t1$)$\}$,q)
using *aux_density_eq* **by** *simp_all*
then
have *forces_mem*(q, s, $t1$) $\longleftrightarrow$ *forces_mem*(q, s, $t2$)
using *density_mem*[*OF* ‹$q \in P$›] **by** *blast*
}
with ‹$p \in P$›
show *forces_eq*(p,$t1$,$t2$) **using** *def_forces_eq* **by** *blast*
qed

18.5 Kunen 2013, Lemma IV.2.38

lemma *not_forces_neq*:
assumes $p \in P$
shows *forces_eq*(p,$t1$,$t2$) $\longleftrightarrow \neg$ ($\exists\, q \in P.\ q \preceq p \wedge$ *forces_neq*(q,$t1$,$t2$))
using *assms density_eq* **unfolding** *forces_neq_def* **by** *blast*

lemma *not_forces_nmem*:
assumes $p \in P$
shows *forces_mem*(p,$t1$,$t2$) $\longleftrightarrow \neg$ ($\exists\, q \in P.\ q \preceq p \wedge$ *forces_nmem*(q,$t1$,$t2$))
using *assms density_mem* **unfolding** *forces_nmem_def* **by** *blast*

lemma *sats_forces_Nand'*:
assumes
$p \in P$ $\varphi \in$ *formula* $\psi \in$ *formula* *env* $\in$ *list*(M)
shows
M, [p,P,*leq*,*one*] @ *env* $\models$ *forces*(*Nand*(φ,ψ)) $\longleftrightarrow$
$\neg(\exists\, q \in M.\ q \in P \wedge$ *is_leq*(\#\#M,*leq*,q,p) $\wedge$
M, [q,P,*leq*,*one*] @ *env* $\models$ *forces*(φ) $\wedge$
M, [q,P,*leq*,*one*] @ *env* $\models$ *forces*(ψ))
using *assms sats_forces_Nand*[*OF assms*(*2-4*) *transitivity*[*OF* ‹$p \in P$›]]
P_in_M leq_in_M one_in_M **unfolding** *forces_def*
by *simp*

lemma *sats_forces_Neg'*:
assumes
$p \in P$ *env* $\in$ *list*(M) $\varphi \in$ *formula*
shows
M, [p,P,*leq*,*one*] @ *env* $\models$ *forces*(*Neg*(φ)) $\longleftrightarrow$

$\neg(\exists\, q{\in}M.\ q{\in}P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
$M,\ [q,P,leq,one]@env \models forces(\varphi))$
using *assms sats_forces_Neg transitivity*
P_in_M leq_in_M one_in_M **unfolding** *forces_def*
by (*simp*, *blast*)

lemma *sats_forces_Forall'*:
assumes
$p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$
shows
$M,[p,P,leq,one]$ @ $env \models forces(Forall(\varphi)) \longleftrightarrow$
$(\forall\, x{\in}M.\quad M,\ [p,P,leq,one,x]$ @ $env \models forces(\varphi))$
using *assms sats_forces_Forall transitivity*
P_in_M leq_in_M one_in_M sats_ren_forces_forall **unfolding** *forces_def*
by *simp*

18.6 The relation of forcing and atomic formulas

lemma *Forces_Equal*:
assumes
$p{\in}P$ $t1{\in}M$ $t2{\in}M$ $env{\in}list(M)$ $nth(n,env) = t1$ $nth(m,env) = t2$ $n{\in}nat$ $m{\in}nat$
shows
$(p \Vdash Equal(n,m)\ env) \longleftrightarrow forces\_eq(p,t1,t2)$
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
$(p \Vdash Member(n,m)\ env) \longleftrightarrow forces\_mem(p,t1,t2)$
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
$(p \Vdash Neg(\varphi)\ env) \longleftrightarrow \neg(\exists\, q{\in}M.\ q{\in}P \wedge q \preceq p \wedge (q \Vdash \varphi\ env))$
using *assms sats_forces_Neg' transitivity*
P_in_M pair_in_M_iff leq_in_M leq_abs **by** *simp*

18.7 The relation of forcing and connectives

lemma *Forces_Nand*:
assumes
$p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$ $\psi{\in}formula$
shows
$(p \Vdash Nand(\varphi,\psi)\ env) \longleftrightarrow \neg(\exists\, q{\in}M.\ q{\in}P \wedge q \preceq p \wedge (q \Vdash \varphi\ env) \wedge (q \Vdash \psi\ env))$
using *assms sats_forces_Nand' transitivity*

P_in_M pair_in_M_iff leq_in_M leq_abs **by** *simp*

lemma *Forces_And_aux*:
assumes
p∈P env ∈ list(M) φ∈formula ψ∈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∈P env ∈ list(M) φ∈formula ψ∈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(G,ϑ) = x ⟹ x∈val(G,π)*
by (*subst def_val*, *auto*)

lemma *GenExtD*: *x∈M[G] ⟷ (∃τ∈M. x = val(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*

18.8 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 leq_reflI[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 $x{\in}D$
using *M_generic_compatD*[*OF* _ ‹$p{\in}G$›, *of* x]
leq_reflI compatI[*of* _ p x] **by** *force*
ultimately
show *?thesis* **by** *auto*
qed

18.9 Auxiliary results for Lemma IV.2.40(a)

lemma *IV240a_mem_Collect*:
assumes
$\pi{\in}M$ $\tau{\in}M$
shows
$\{q{\in}P.\ \exists\sigma.\ \exists r.\ r{\in}P \wedge \langle\sigma,r\rangle \in \tau \wedge q{\preceq}r \wedge forces\_eq(q,\pi,\sigma)\}{\in}M$
proof -
let *?rel_pred*= $\lambda M\ x\ a1\ a2\ a3\ a4.\ \exists\sigma[M].\ \exists r[M].\ \exists\sigma r[M].$
$r{\in}a1 \wedge pair(M,\sigma,r,\sigma r) \wedge \sigma r{\in}a4 \wedge is\_leq(M,a2,x,r) \wedge is\_forces\_eq'(M,a1,a2,x,a3,\sigma)$
let *?φ*=*Exists*(*Exists*(*Exists*(*And*(*Member*(1,4),*And*(*pair_fm*(2,1,0),
And(*Member*(0,7),*And*(*leq_fm*(5,3,1),*forces_eq_fm*(4,5,3,6,2))))))))
have $\sigma{\in}M \wedge r{\in}M$ **if** $\langle\sigma,\ r\rangle \in \tau$ **for** σ r
using *that* ‹$\tau{\in}M$› *pair_in_M_iff transitivity*[*of* $\langle\sigma,r\rangle$ τ] **by** *simp*
then
have *?rel_pred*(##*M*,*q*,*P*,*leq*,π,τ) $\longleftrightarrow$ ($\exists\sigma.\ \exists r.\ r{\in}P \wedge \langle\sigma,r\rangle \in \tau \wedge q{\preceq}r \wedge$
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*,π,τ] $\models$ *?φ*) $\longleftrightarrow$ *?rel_pred*(##*M*,*q*,*P*,*leq*,π,τ) **if** $q{\in}M$ **for** q
using *assms that sats_forces_eq'_fm sats_leq_fm P_in_M leq_in_M* **by** *simp*
moreover
have *?φ*∈*formula* **by** *simp*
moreover
have *arity*(*?φ*)=5
unfolding *leq_fm_def pair_fm_def upair_fm_def*
using *arity_forces_eq_fm* **by** (*simp add*:*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,π,τ)

$\bigwedge q\ \sigma.\ q{\in}P \Longrightarrow q{\in}G \Longrightarrow \sigma{\in}domain(\tau) \Longrightarrow forces\_eq(q,\pi,\sigma) \Longrightarrow$
$val(G,\pi) = val(G,\sigma)$
shows
$val(G,\pi){\in}val(G,\tau)$
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 $r \in G$ $val(G,\pi) = val(G,\sigma)$ **by** *blast*+
ultimately
show $\exists\ \sigma.\ \exists p{\in}P.\ p \in G \wedge \langle\sigma,\ p\rangle \in \tau \wedge val(G,\ \sigma) = val(G,\ \pi)$ **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 leq_reflI*
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:$\bigwedge q\ \sigma.\ q{\in}P \Longrightarrow q{\in}G \Longrightarrow \sigma{\in}domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(forces\_mem(q,\sigma,\tau) \longrightarrow val(G,\sigma) \in val(G,\tau)) \wedge$
$(forces\_mem(q,\sigma,\vartheta) \longrightarrow val(G,\sigma) \in val(G,\vartheta))$

shows
$val(G,\tau) \subseteq val(G,\vartheta)$
proof
fix x

assume $x{\in}val(G,\tau)$
then
obtain σ r **where** $\langle\sigma,r\rangle{\in}\tau$ $r{\in}G$ $val(G,\sigma){=}x$ **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] ‹$\langle\sigma,r\rangle{\in}\tau$› ‹$val(G,\sigma) = x$›
show $x{\in}val(G,\vartheta)$ **by** (*blast*)
qed

lemma *IV240a_eq_2nd_incl*:
assumes
$M\_generic(G)$ $p{\in}G$ $forces\_eq(p,\tau,\vartheta)$
and
$IH{:}\bigwedge q\ \sigma.\ q{\in}P \Longrightarrow q{\in}G \Longrightarrow \sigma{\in}domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(forces\_mem(q,\sigma,\tau) \longrightarrow val(G,\sigma) \in val(G,\tau)) \wedge$
$(forces\_mem(q,\sigma,\vartheta) \longrightarrow val(G,\sigma) \in val(G,\vartheta))$
shows
$val(G,\vartheta) \subseteq val(G,\tau)$
proof
fix x
assume $x{\in}val(G,\vartheta)$
then
obtain σ r **where** $\langle\sigma,r\rangle{\in}\vartheta$ $r{\in}G$ $val(G,\sigma){=}x$ **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 σ] ‹$\langle\sigma,r\rangle{\in}\vartheta$› ‹$val(G,\sigma) = x$›
show $x{\in}val(G,\tau)$ **by** (*blast*)
qed

lemma *IV240a_eq*:
assumes
M_generic(*G*) *p*∈*G forces_eq*(*p*,τ,ϑ)
and
IH:⋀*q* σ. *q*∈*P* ⟹ *q*∈*G* ⟹ σ∈*domain*(τ) ∪ *domain*(ϑ) ⟹
(*forces_mem*(*q*,σ,τ) ⟶ *val*(*G*,σ) ∈ *val*(*G*,τ)) ∧
(*forces_mem*(*q*,σ,ϑ) ⟶ *val*(*G*,σ) ∈ *val*(*G*,ϑ))
shows
val(*G*,τ) = *val*(*G*,ϑ)
using *IV240a_eq_1st_incl*[*OF assms*] *IV240a_eq_2nd_incl*[*OF assms*] *IH* **by** *blast*

18.10 Induction on names

lemma *core_induction*:
assumes
⋀τ ϑ *p*. *p* ∈ *P* ⟹ ⟦⋀*q* σ. ⟦*q*∈*P* ; σ∈*domain*(ϑ)⟧ ⟹ *Q*(*0*,τ,σ,*q*)⟧ ⟹ *Q*(*1*,τ,ϑ,*p*)
⋀τ ϑ *p*. *p* ∈ *P* ⟹ ⟦⋀*q* σ. ⟦*q*∈*P* ; σ∈*domain*(τ) ∪ *domain*(ϑ)⟧ ⟹ *Q*(*1*,σ,τ,*q*) ∧ *Q*(*1*,σ,ϑ,*q*)⟧ ⟹ *Q*(*0*,τ,ϑ,*p*)
ft ∈ *2 p* ∈ *P*
shows
Q(*ft*,τ,ϑ,*p*)
proof -
{
fix *ft p* τ ϑ
have *Transset*(*eclose*({τ,ϑ})) (**is** *Transset*(*?e*))
using *Transset_eclose* **by** *simp*
have τ ∈ *?e* ϑ ∈ *?e*
using *arg_into_eclose* **by** *simp_all*
moreover
assume *ft* ∈ *2 p* ∈ *P*
ultimately
have ⟨*ft*,τ,ϑ,*p*⟩∈ *2*×*?e*×*?e*×*P* (**is** *?a*∈*2*×*?e*×*?e*×*P*) **by** *simp*
then
have *Q*(*ftype*(*?a*), *name1*(*?a*), *name2*(*?a*), *cond_of*(*?a*))
using *core_induction_aux*[*of ?e P Q ?a*,*OF* ‹*Transset*(*?e*)› *assms*(*1*,*2*) ‹*?a*∈_›]
by (*clarify*) (*blast*)
then have *Q*(*ft*,τ,ϑ,*p*) **by** (*simp add*:*components_simp*)
}
then show *?thesis* **using** *assms* **by** *simp*
qed

lemma *forces_induction_with_conds*:
assumes
⋀τ ϑ *p*. *p* ∈ *P* ⟹ ⟦⋀*q* σ. ⟦*q*∈*P* ; σ∈*domain*(ϑ)⟧ ⟹ *Q*(*q*,τ,σ)⟧ ⟹ *R*(*p*,τ,ϑ)
⋀τ ϑ *p*. *p* ∈ *P* ⟹ ⟦⋀*q* σ. ⟦*q*∈*P* ; σ∈*domain*(τ) ∪ *domain*(ϑ)⟧ ⟹ *R*(*q*,σ,τ) ∧

```
R(q,σ,ϑ)]] ⟹ Q(p,τ,ϑ)
    p ∈ P
  shows
    Q(p,τ,ϑ) ∧ R(p,τ,ϑ)
proof -
  let ?Q=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
```

18.11 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(G,τ) = val(G,ϑ))) ∧
     (τ∈M ⟶ ϑ∈M ⟶ (∀ p∈G. forces_mem(p,τ,ϑ) ⟶ val(G,τ) ∈ val(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(G, τ) = val(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(G,τ) ∈ val(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(G, σ) ∈ val(G, τ)) ∧
          (forces_mem(q, σ, ϑ) ⟶ val(G, σ) ∈ val(G, ϑ)) for q σ
    by (auto intro:  transitivity[OF _ domain_closed[simplified]])
  ultimately
  show ∀ p∈G. forces_eq(p,τ,ϑ) ⟶ val(G,τ) = val(G,ϑ)
    using IV240a_eq[OF assms(1) _ _ IH'] by (simp)
qed
```

18.12 Lemma IV.2.40(b)

```
lemma IV240b_mem:
  assumes
    M_generic(G) val(G,π)∈val(G,τ) π∈M τ∈M
    and
    IH:⋀σ. σ∈domain(τ) ⟹ val(G,π) = val(G,σ) ⟹
      ∃ p∈G. forces_eq(p,π,σ)
  shows
    ∃ p∈G. forces_mem(p,π,τ)
proof -
  from ‹val(G,π)∈val(G,τ)›
  obtain σ r where r∈G ⟨σ,r⟩∈τ val(G,π) = val(G,σ) by auto
  moreover from this and IH
  obtain p' where p'∈G forces_eq(p',π,σ) by blast
  moreover
  note ‹M_generic(G)›
  ultimately
  obtain p where p≼r p∈G forces_eq(p,π,σ)
    using M_generic_compatD strengthening_eq[of p'] by blast
  moreover
```

note ‹*M_generic*(*G*)›
moreover from *calculation*
have *forces_eq*(*q*,π,σ) **if** *q*∈*P* *q*≼*p* **for** *q*
using *that strengthening_eq* **by** *blast*
moreover
note ‹⟨σ,*r*⟩∈τ› ‹*r*∈*G*›
ultimately
have *r*∈*P* ∧ ⟨σ,*r*⟩ ∈ τ ∧ *q*≼*r* ∧ *forces_eq*(*q*,π,σ) **if** *q*∈*P* *q*≼*p* **for** *q*
using *that leq_transD*[*of* _ *p r*] **by** *blast*
then
have *dense_below*({*q*∈*P*. ∃*s r*. *r*∈*P* ∧ ⟨*s*,*r*⟩ ∈ τ ∧ *q*≼*r* ∧ *forces_eq*(*q*,π,*s*)},*p*)
using *leq_reflI* **by** *blast*
moreover
note ‹*M_generic*(*G*)› ‹*p*∈*G*›
moreover from *calculation*
have *forces_mem*(*p*,π,τ)
using *forces_mem_iff_dense_below* **by** *blast*
ultimately
show *?thesis* **by** *blast*
qed

end

lemma *Collect_forces_eq_in_M*:
assumes τ ∈ *M* ϑ ∈ *M*
shows {*p*∈*P*. *forces_eq*(*p*,τ,ϑ)} ∈ *M*
using *assms Collect_in_M_4p*[*of forces_eq_fm*(*1*,*2*,*0*,*3*,*4*) *P leq* τ ϑ
λ*A x p l t1 t2*. *is_forces_eq*(*x*,*t1*,*t2*)
λ *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 τ ∈ *M* ϑ ∈ *M*
shows {*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*
proof -
let *?rel_pred*=λ*M x a1 a2 a3 a4*.
∃σ[*M*]. ∃*u*[*M*]. ∃*da3*[*M*]. ∃*da4*[*M*]. *is_domain*(*M*,*a3*,*da3*) ∧ *is_domain*(*M*,*a4*,*da4*)
∧
union(*M*,*da3*,*da4*,*u*) ∧ σ∈*u* ∧ *is_forces_mem′*(*M*,*a1*,*a2*,*x*,σ,*a3*) ∧
is_forces_nmem′(*M*,*a1*,*a2*,*x*,σ,*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*:σ∈*M* **if** ⟨σ,*y*⟩∈δ δ∈*M* **for** σ δ *y*
using *that pair_in_M_iff transitivity*[*of* ⟨σ,*y*⟩ δ] **by** *simp*

have *abs1*:*?rel_pred*(*##M*,*p*,*P*,*leq*,*τ*,*ϑ*) ⟷
(∃ *σ*∈*domain*(*τ*) ∪ *domain*(*ϑ*). *forces_mem′*(*P*,*leq*,*p*,*σ*,*τ*) ∧ *forces_nmem′*(*P*,*leq*,*p*,*σ*,*ϑ*))

if *p*∈*M* **for** *p*
unfolding *forces_mem_def forces_nmem_def*
using *assms that forces_mem′_abs forces_nmem′_abs P_in_M leq_in_M*
domain_closed Un_closed
by (*auto simp add*:*1*[*of _ _ τ*] *1*[*of _ _ ϑ*])
have *abs2*:*?rel_pred*(*##M*,*p*,*P*,*leq*,*ϑ*,*τ*) ⟷ (∃ *σ*∈*domain*(*τ*) ∪ *domain*(*ϑ*).
forces_nmem′(*P*,*leq*,*p*,*σ*,*τ*) ∧ *forces_mem′*(*P*,*leq*,*p*,*σ*,*ϑ*)) **if** *p*∈*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*,*τ*,*ϑ*] ⊨ *?φ*) ⟷ *?rel_pred*(*##M*,*p*,*P*,*leq*,*τ*,*ϑ*) **if** *p*∈*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*,*ϑ*,*τ*] ⊨ *?φ*) ⟷ *?rel_pred*(*##M*,*p*,*P*,*leq*,*ϑ*,*τ*) **if** *p*∈*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*:*?φ*∈*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 _ _ _ _ _*
λx 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*
λx 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*(*G*,τ) = *val*(*G*,ϑ) τ∈*M* ϑ∈*M*
and
IH:$\bigwedge\sigma$. σ∈*domain*(τ)∪*domain*(ϑ) ⟹
(*val*(*G*,σ)∈*val*(*G*,τ) ⟶ (∃ *q*∈*G*. *forces_mem*(*q*,σ,τ))) ∧
(*val*(*G*,σ)∈*val*(*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*∈*P*
have ∃ *d*∈*P*. (*forces_eq*(*d*, τ, ϑ) ∨
(∃ σ∈*domain*(τ) ∪ *domain*(ϑ). *forces_mem*(*d*, σ, τ) ∧ *forces_nmem*(*d*, σ, ϑ)) ∨
(∃ σ∈*domain*(τ) ∪ *domain*(ϑ). *forces_nmem*(*d*, σ, τ) ∧ *forces_mem*(*d*, σ, ϑ))) ∧
d ≼ *p*
proof (*cases forces_eq*(*p*, τ, ϑ))
case *True*
with ‹*p*∈*P*›
show *?thesis* **using** *leq_reflI* **by** *blast*
next
case *False*
moreover note ‹*p*∈*P*›
moreover from *calculation*
obtain σ *q* **where** σ∈*domain*(τ)∪*domain*(ϑ) *q*∈*P* *q*≼*p*
(*forces_mem*(*q*, σ, τ) ∧ ¬ *forces_mem*(*q*, σ, ϑ)) ∨
(¬ *forces_mem*(*q*, σ, τ) ∧ *forces_mem*(*q*, σ, ϑ))
using *def_forces_eq* **by** *blast*
moreover from *this*
obtain *r* **where** *r*≼*q* *r*∈*P*
(*forces_mem*(*r*, σ, τ) ∧ *forces_nmem*(*r*, σ, ϑ)) ∨

```
      (forces_nmem(r, σ, τ) ∧ forces_mem(r, σ, ϑ))
     using not_forces_nmem strengthening_mem by blast
   ultimately
   show ?thesis using leq_transD by blast
  qed
  then
  show ∃ d∈?D1 ∪ ?D2 ∪ ?D3. d ≼ p by blast
qed
moreover
have ?D ⊆ P
  by auto
moreover
note ‹M_generic(G)›
ultimately
obtain p where p∈G p∈?D
  unfolding M_generic_def by blast
then
consider
  (1) forces_eq(p,τ,ϑ) |
  (2) ∃σ∈domain(τ) ∪ domain(ϑ). forces_mem(p,σ,τ) ∧ forces_nmem(p,σ,ϑ) |
  (3) ∃σ∈domain(τ) ∪ domain(ϑ). forces_nmem(p,σ,τ) ∧ forces_mem(p,σ,ϑ)
  by blast
then
show ?thesis
proof (cases)
  case 1
  with ‹p∈G›
  show ?thesis by blast
next
  case 2
  then
 obtain σ where σ∈domain(τ) ∪ domain(ϑ) forces_mem(p,σ,τ) forces_nmem(p,σ,ϑ)

    by blast
  moreover from this and ‹p∈G› and assms
  have val(G,σ)∈val(G,τ)
   using IV240a[of G σ τ] transitivity[OF _ domain_closed[simplified]] by blast
  moreover note IH ‹val(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(G,σ)∈val(G,ϑ)
     using IV240a[of G σ ϑ] transitivity[OF _ domain_closed[simplified]] by blast
    moreover note IH ‹val(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
    (τ∈M⟶ϑ∈M⟶val(G,τ) = val(G,ϑ) ⟶ (∃ p∈G. forces_eq(p,τ,ϑ))) ∧
     (τ∈M⟶ϑ∈M⟶val(G,τ) ∈ val(G,ϑ) ⟶ (∃ p∈G. forces_mem(p,τ,ϑ)))
    (is ?Q(τ,ϑ) ∧ ?R(τ,ϑ))
proof (intro forces_induction)
  fix τ ϑ p
  assume σ∈domain(ϑ) ⟹ ?Q(τ, σ) for σ
  with assms
  show ?R(τ, ϑ)
    using IV240b_mem domain_closed transitivity by (simp)
next
  fix τ ϑ p
  assume σ ∈ domain(τ) ∪ domain(ϑ) ⟹ ?R(σ,τ) ∧ ?R(σ,ϑ) for σ
```

moreover from *this*
have *IH'*:$\tau{\in}M \Longrightarrow \vartheta{\in}M \Longrightarrow \sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(val(G,\sigma) \in val(G,\tau) \longrightarrow (\exists\, q{\in}G.\ forces\_mem(q,\sigma,\tau))) \wedge$
$(val(G,\sigma) \in val(G,\vartheta) \longrightarrow (\exists\, q{\in}G.\ forces\_mem(q,\sigma,\vartheta)))$ **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∈*list*(*M*)
shows
map(*val*(*G*),*env*)∈*list*(*M*[*G*])
unfolding *GenExt_def* **using** *assms map_type2* **by** *simp*

lemma *truth_lemma_mem*:
assumes
env∈*list*(*M*) *M_generic*(*G*)
n∈*nat m*∈*nat n*<*length*(*env*) *m*<*length*(*env*)
shows
(∃ *p*∈*G*. *p* ⊩ *Member*(*n*,*m*) *env*) ⟷ *M*[*G*], *map*(*val*(*G*),*env*) ⊨ *Member*(*n*,*m*)
using *assms IV240a*[*OF assms*(*2*), *of nth*(*n*,*env*) *nth*(*m*,*env*)]
IV240b[*OF assms*(*2*), *of nth*(*n*,*env*) *nth*(*m*,*env*)]
P_in_M leq_in_M one_in_M
Forces_Member[*of* _ *nth*(*n*,*env*) *nth*(*m*,*env*) *env n m*] *map_val_in_MG*
by (*auto*)

lemma *truth_lemma_eq*:
assumes
env∈*list*(*M*) *M_generic*(*G*)
n∈*nat m*∈*nat n*<*length*(*env*) *m*<*length*(*env*)
shows
(∃ *p*∈*G*. *p* ⊩ *Equal*(*n*,*m*) *env*) ⟷ *M*[*G*], *map*(*val*(*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*

18.13 The Strenghtening Lemma

lemma *strengthening_lemma*:
assumes
$p{\in}P$ $\varphi{\in}formula$ $r{\in}P$ $r{\preceq}p$
shows
$\bigwedge env.\ env{\in}list(M) \Longrightarrow arity(\varphi){\leq}length(env) \Longrightarrow p \Vdash \varphi\ env \Longrightarrow r \Vdash \varphi\ env$
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 $env{\in}list(M)$
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 $env{\in}list(M)$
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 $p \Vdash \varphi$ ([x] @ *env*) **if** $x{\in}M$ **for** x
using *that Forces_Forall* **by** *simp*
with *Forall*
have $r \Vdash \varphi$ ([x] @ *env*) **if** $x{\in}M$ **for** x
using *that pred_le2* **by** (*simp*)
with *assms Forall*
show *?case*

using *Forces_Forall* **by** *simp*
qed

18.14 The Density Lemma

lemma *arity_Nand_le*:
assumes $\varphi \in formula$ $\psi \in formula$ $arity(Nand(\varphi, \psi)) \leq length(env)$ $env \in list(A)$
shows $arity(\varphi) \leq length(env)$ $arity(\psi) \leq length(env)$
using *assms*
by (*rule_tac Un_leD1*, *rule_tac* [*5*] *Un_leD2*, *auto*)

lemma *dense_below_imp_forces*:
assumes
$p \in P$ $\varphi \in formula$
shows
$\bigwedge env.\ env \in list(M) \Longrightarrow arity(\varphi) \leq length(env) \Longrightarrow$
$dense\_below(\{q \in P.\ (q \Vdash \varphi\ env)\}, p) \Longrightarrow (p \Vdash \varphi\ env)$
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 $env \in list(M)$
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 $env \in list(M)$
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 $q \in M$ $q \in P$ $q \preceq p$ $q \Vdash \varphi\ env$

```
    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 leq_reflI 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
```

$p{\in}P$ $\varphi{\in}formula$ $env{\in}list(M)$ $arity(\varphi){\le}length(env)$
shows
$p \Vdash \varphi\ env \quad \longleftrightarrow \quad dense\_below(\{q{\in}P.\ (q \Vdash \varphi\ env)\},p)$
proof
assume $dense\_below(\{q{\in}P.\ (q \Vdash \varphi\ env)\},p)$
with *assms*
show $(p \Vdash \varphi\ env)$
using *dense_below_imp_forces* **by** *simp*
next
assume $p \Vdash \varphi\ env$
with *assms*
show $dense\_below(\{q{\in}P.\ q \Vdash \varphi\ env\},p)$
using *strengthening_lemma leq_reflI* **by** *auto*
qed

18.15 The Truth Lemma

lemma *Forces_And*:
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
$p \Vdash And(\varphi,\psi)\ env \quad \longleftrightarrow \quad (p \Vdash \varphi\ env) \wedge (p \Vdash \psi\ env)$
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 leq_reflI* **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: $(\exists\, p{\in}G.\ p \Vdash \varphi\ env) \longleftrightarrow M[G],\ map(val(G),env) \models \varphi$
  shows
    $(\exists\, p{\in}G.\ p \Vdash Neg(\varphi)\ env) \longleftrightarrow M[G],\ map(val(G),env) \models Neg(\varphi)$
proof (*intro iffI*, *elim bexE*, *rule ccontr*)

  fix p
  assume $p{\in}G$ $p \Vdash Neg(\varphi)$ env $\neg(M[G],map(val(G),env) \models Neg(\varphi))$
  moreover
  note *assms*
  moreover from *calculation*
  have $M[G],\ map(val(G),env) \models \varphi$
    using *map_val_in_MG* **by** *simp*
  with *IH*
  obtain r **where** $r \Vdash \varphi$ env $r{\in}G$ **by** *blast*
  moreover from *this* **and** ‹*M_generic*(*G*)› ‹$p{\in}G$›
  obtain q **where** $q{\preceq}p$ $q{\preceq}r$ $q{\in}G$
    by *blast*
  moreover from *calculation*
  have $q \Vdash \varphi$ env
    using *strengthening_lemma*[**where** $\varphi{=}\varphi$] **by** *blast*
  ultimately
  show *False*
    using *Forces_Neg*[**where** $\varphi{=}\varphi$] *transitivity*[*OF _ P_in_M*] **by** *blast*
next
  assume $M[G],\ map(val(G),env) \models Neg(\varphi)$
  with *assms*
  have $\neg\ (M[G],\ map(val(G),env) \models \varphi)$
    using *map_val_in_MG* **by** *simp*
  let *?D*=$\{p{\in}P.\ (p \Vdash \varphi\ env) \vee (p \Vdash Neg(\varphi)\ env)\}$
  have *separation*($\#\#M$,$\lambda p.\ (p \Vdash \varphi\ env)$)
      using *separation_ax arity_forces assms P_in_M leq_in_M one_in_M arity_forces_le*
    by *simp*
  moreover
  have *separation*($\#\#M$,$\lambda p.\ (p \Vdash Neg(\varphi)\ env)$)
      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 leq_reflI 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(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(G),env) ⊨ φ
        (∃p∈G. p ⊩ ψ env)  ⟷   M[G], map(val(G),env) ⊨ ψ
  shows
    (∃p∈G. (p ⊩ And(φ,ψ) env)) ⟷ M[G] , map(val(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(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(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. ∃ b∈M. ∀ q∈M. q∈a1 ∧ is_leq(##M,a2,q,x) ⟶

                  ¬(M, [q,a1,a2,a3,b] @ env ⊨ forces(φ))
  let ?ψ=Exists(Forall(Implies(And(Member(0,3),leq_fm(4,0,2)),
         Neg(ren_truth_lemma(forces(φ))))))
  have q∈M if q∈P for q using that transitivity[OF _ P_in_M] by simp
  then
  have 1:∀ q∈M. q∈P ∧ R(q) ⟶ Q(q) ⟹ (∀ q∈P. R(q) ⟶ Q(q)) for R Q
    by auto
  then
  have ⟦b ∈ M; ∀ q∈M. q ∈ P ∧ q ⪯ d ⟶ ¬(q ⊩ φ ([b]@env))⟧ ⟹
        ∃ c∈M. ∀ q∈P. q ⪯ d ⟶ ¬(q ⊩ φ ([c]@env)) for b d
    by (rule bexI,simp_all)
  then
 have ?rel_pred(M,d,P,leq,one) ⟷ (∃ b∈M. ∀ q∈P. q⪯d ⟶ ¬(q ⊩ φ ([b]@env)))
if d∈M for d
    using that leq_abs leq_in_M P_in_M one_in_M assms
    by auto
  moreover
  have ?ψ∈formula using assms by simp
  moreover
```

```
  have (M, [d,P,leq,one]@env ⊨ ?ψ) ⟷ ?rel_pred(M,d,P,leq,one) if d∈M for
d
   using assms that P_in_M leq_in_M one_in_M sats_leq_fm sats_ren_truth_lemma
    by simp
  moreover
  have arity(?ψ) ≤ 4#+length(env)
  proof -
    have eq:arity(leq_fm(4, 0, 2)) = 5
      using arity_leq_fm succ_Un_distrib nat_simp_union
      by simp
    with ‹φ∈_›
    have arity(?ψ) = 3 ∪ (pred^2(arity(ren_truth_lemma(forces(φ)))))
      using nat_union_abs1 pred_Un_distrib by simp
    moreover
    have ... ≤ 3 ∪ (pred(pred(6 ∪ succ(arity(forces(φ)))))) (is _ ≤ ?r)
      using  ‹φ∈_› Un_le_compat[OF le_refl[of 3]]
                le_imp_subset arity_ren_truth[of forces(φ)]
                pred_mono
      by auto
    finally
    have arity(?ψ) ≤ ?r by simp
    have i:?r ≤ 4 ∪ pred(arity(forces(φ)))
     using pred_Un_distrib pred_succ_eq ‹φ∈_› Un_assoc[symmetric] nat_union_abs1
by simp
    have h:4 ∪ pred(arity(forces(φ))) ≤ 4 ∪ (4#+length(env))
      using  ‹env∈_› add_commute ‹φ∈_›
           Un_le_compat[of 4 4,OF _ pred_mono[OF _ arity_forces_le[OF _ _
‹arity(φ)≤_›]] ]
           ‹env∈_› by auto
    with ‹φ∈_› ‹env∈_›
    show ?thesis
     using le_trans[OF  ‹arity(?ψ) ≤ ?r›  le_trans[OF i h]] 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(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(G),env) ⊨ Forall(φ)
      using pred_le2 map_val_in_MG
      by (auto iff:GenExtD)
  next
    assume M[G], map(val(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 leq_reflI 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
```

18.16 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(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(G),env) ⊨ φ
    using truth_lemma by blast
next
  assume 1: ∀ G.(M_generic(G)∧ p∈G)⟶ M[G] , map(val(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 $M[G]$, $map(val(G),env) \models \varphi$
by *simp*
with *assms* ‹$M\_generic(G)$›
obtain s **where** $s{\in}G$ $(s \Vdash \varphi\ env)$
using *truth_lemma* **by** *blast*
moreover from *this* **and** ‹$M\_generic(G)$› ‹$r{\in}G$›
obtain q **where** $q{\in}G$ $q{\preceq}s$ $q{\preceq}r$
by *blast*
moreover from *calculation* ‹$s{\in}G$› ‹$M\_generic(G)$›
have $s{\in}P$ $q{\in}P$
unfolding *M_generic_def filter_def* **by** *auto*
moreover
note *assms*
ultimately
have $\exists\, q{\in}P.\ q{\preceq}r \wedge (q \Vdash \varphi\ env)$
using *strengthening_lemma* **by** *blast*
}
then
have *dense_below*($\{q{\in}P.\ (q \Vdash \varphi\ env)\}$,$p$)
unfolding *dense_below_def* **by** *blast*
with *assms*
show $(p \Vdash \varphi\ env)$
using *density_lemma* **by** *blast*
qed

lemmas *definability* = *forces_type*
end

end

19 Auxiliary renamings for Separation

theory *Separation_Rename*
imports *Interface Renaming*
begin

lemmas *apply_fun* = *apply_iff*[*THEN iffD1*]

lemma *nth_concat* : $[p,t] \in list(A) \Longrightarrow env{\in}\ list(A) \Longrightarrow nth(1\ \#{+}\ length(env),[p]@\ env\ @\ [t]) = t$
by(*auto simp add*:*nth_append*)

lemma *nth_concat2* : $env{\in}\ list(A) \Longrightarrow nth(length(env),env\ @\ [p,t]) = p$
by(*auto simp add*:*nth_append*)

lemma *nth_concat3* : $env{\in}\ list(A) \Longrightarrow u = nth(succ(length(env)),\ env\ @\ [pi,\ u])$

```
  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

20 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(G),nenv)*
using *assms*
proof(*induct env*)
case *Nil*
have *map(val(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(G),l′) a = val(G,a′)
Cons(a,l) = map(val(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(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 $domain(\pi)\in M$ $domain(\pi) \times P \in M$
by (*simp_all flip:setclass_iff*)
from ‹$env \in \_$›
obtain *nenv* **where** $nenv\in list(M)$ $env = map(val(G),nenv)$ $length(nenv) = length(env)$
using *map_val* **by** *auto*
from ‹$arity(\varphi) \le \_$› ‹$env\in\_$› ‹$\varphi\in\_$›
have $arity(\varphi) \le 2\#+ length(env)$
using *le_trans*[*OF* ‹$arity(\varphi)\le\_$›] *add_le_mono*[*of 1 2*,*OF _ le_refl*]
by *auto*
with ‹$nenv\in\_$› ‹$env\in\_$› ‹$\pi\in M$› ‹$\varphi\in\_$› ‹$length(nenv) = length(env)$›
have $arity(?\chi) \le length([\vartheta] @ nenv @ [\pi])$ **for** ϑ
using *nat_union_abs2*[*OF _ _* ‹$arity(\varphi) \le 2\#+ \_$›] *nat_simp_union*
by *simp*
note *in_M* = ‹$\pi\in M$› ‹$domain(\pi) \times P \in M$› *P_in_M one_in_M leq_in_M*
{
fix *u*
assume $u \in domain(\pi) \times P$ $u \in M$
with *in_M* ‹*?new_form* ∈ *formula*› ‹$?\psi\in formula$› ‹$nenv \in \_$›
have *Eq1*: $(M, [u] @ ?Pl1 @ [\pi] @ nenv \models ?\psi) \longleftrightarrow$
$(\exists \vartheta\in M.\ \exists p\in P.\ u =\langle\vartheta,p\rangle \wedge$
$M, [\vartheta,p,u]@?Pl1@[\pi] @ nenv \models ?new\_form)$
by (*auto simp add*: *transitivity*)
have *Eq3*: $\vartheta\in M \Longrightarrow p\in P \Longrightarrow$
$(M, [\vartheta,p,u]@?Pl1@[\pi]@nenv \models ?new\_form) \longleftrightarrow$
$(\forall F.\ M\_generic(F) \wedge p \in F \longrightarrow (M[F],\ map(val(F), [\vartheta] @ nenv@[\pi]) \models ?\chi))$
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* ‹$\vartheta \in M$› ‹$p\in M$› ‹$u \in domain(\pi) \times P$› ‹$u \in M$› ‹$nenv\in\_$›
then
have $[\vartheta,u] \in list(M)$ **by** *simp*
let *?env*=$[p]@?Pl1@[\vartheta] @ nenv @ [\pi,u]$
let *?new_env*= $[\vartheta,p,u,P,leq,one,\pi] @ nenv$
let $?\psi=Exists(Exists(And(pair\_fm(0,1,2),?new\_form)))$
have $[\vartheta,\ p,\ u,\ \pi,\ leq,\ one,\ \pi] \in list(M)$
using *in_M'* **by** *simp*
have $?\chi \in formula$ $forces(?\chi)\in formula$
using *phi* **by** *simp_all*
from *in_M'*
have $?Pl1 \in list(M)$ **by** *simp*
from *in_M'* **have** $?env \in list(M)$ **by** *simp*
have *Eq1'*: $?new\_env \in list(M)$ **using** *in_M'* **by** *simp*
then

have $(M,\ [\vartheta,p,u]@?Pl1@[\pi]\ @\ nenv \models ?new\_form) \longleftrightarrow (M,\ ?new\_env \models ?new\_form)$
by *simp*
from *in_M'* ‹$env \in \_$› *Eq1'* ‹$length(nenv) = length(env)$›
‹$arity(forces(?\chi)) \leq 7\ \#+\ length(env)$› ‹$forces(?\chi) \in formula$›
‹$[\vartheta,\ p,\ u,\ \pi,\ leq,\ one,\ \pi] \in list(M)$›
have ... $\longleftrightarrow M,\ ?env \models forces(?\chi)$
using *sepren_action*[*of* $forces(?\chi)$ *nenv*,*OF* _ _ ‹$nenv \in list(M)$›]
by *simp*
also from *in_M'*
have ... $\longleftrightarrow M,\ ([p,P,\ leq,\ one,\vartheta]@nenv@\ [\pi])@[u] \models forces(?\chi)$
using *app_assoc* **by** *simp*
also
from *in_M'* ‹$env \in \_$› *phi* ‹$length(nenv) = length(env)$›
‹$arity(forces(?\chi)) \leq 6\ \#+\ length(env)$› ‹$forces(?\chi) \in formula$›
have ... $\longleftrightarrow M,\ [p,P,\ leq,\ one,\vartheta]@\ nenv\ @\ [\pi] \models forces(?\chi)$
by (*rule_tac arity_sats_iff*,*auto*)
also
from ‹$arity(forces(?\chi)) \leq 6\ \#+\ length(env)$› ‹$forces(?\chi) \in formula$› *in_M' phi*
have ... $\longleftrightarrow (\forall F.\ M\_generic(F) \wedge p \in F \longrightarrow M[F],\ map(val(F),\ [\vartheta]\ @\ nenv\ @\ [\pi]) \models\ ?\chi)$
using *definition_of_forcing*
proof (*intro iffI*)
assume *a1*: $M,\ [p,P,\ leq,\ one,\vartheta]\ @\ nenv\ @\ [\pi] \models\ forces(?\chi)$
note *definition_of_forcing* ‹$arity(\varphi) \leq\ 1\#+\_$›
with ‹$nenv \in \_$› ‹$arity(?\chi) \leq length([\vartheta]\ @\ nenv\ @\ [\pi])$› ‹$env \in \_$›
have $p \in P \Longrightarrow ?\chi \in formula \Longrightarrow [\vartheta,\pi] \in list(M) \Longrightarrow M,\ [p,P,\ leq,\ one]\ @\ [\vartheta]@\ nenv@[\pi] \models forces(?\chi) \Longrightarrow \forall G.\ M\_generic(G) \wedge p \in G \longrightarrow M[G],\ map(val(G),\ [\vartheta]\ @\ nenv\ @[\pi]) \models\ ?\chi$
by *auto*
then
show $\forall F.\ M\_generic(F) \wedge p \in F \longrightarrow M[F],\ map(val(F),\ [\vartheta]\ @\ nenv\ @\ [\pi]) \models\ ?\chi$
using ‹$?\chi \in formula$› ‹$p \in P$› *a1* ‹$\vartheta \in M$› ‹$\pi \in M$› **by** *simp*
next
assume $\forall F.\ M\_generic(F) \wedge p \in F \longrightarrow M[F],\ map(val(F),\ [\vartheta]\ @\ nenv\ @[\pi]) \models\ ?\chi$
with *definition_of_forcing* [*THEN iffD2*] ‹$arity(?\chi) \leq length([\vartheta]\ @\ nenv\ @\ [\pi])$›
show $M,\ [p,\ P,\ leq,\ one,\vartheta]\ @\ nenv\ @\ [\pi] \models\ forces(?\chi)$
using ‹$?\chi \in formula$› ‹$p \in P$› *in_M'*
by *auto*
qed
finally
show $(M,\ [\vartheta,p,u]@?Pl1@[\pi]@nenv \models ?new\_form) \longleftrightarrow (\forall F.\ M\_generic(F) \wedge p \in F \longrightarrow M[F],\ map(val(F),\ [\vartheta]\ @\ nenv\ @\ [\pi]) \models\ ?\chi)$

by *simp*
qed
with *Eq1*
have (M, $[u]$ @ *?Pl1* @ $[\pi]$ @ *nenv* $\models$ *?ψ*) $\longleftrightarrow$
($\exists\,\vartheta{\in}M.\ \exists\,p{\in}P.\ u = \langle\vartheta,p\rangle \wedge$
($\forall\,F.\ M\_generic(F) \wedge p \in F \longrightarrow M[F],\ \ map(val(F),\ [\vartheta]$ @ *nenv* @ $[\pi])$
$\models$ *?χ*))
by *auto*
}
then
have *Equivalence*: $u{\in}$ *domain*$(\pi) \times P \Longrightarrow u \in M \Longrightarrow$
(M, $[u]$ @ *?Pl1* @ $[\pi]$ @ *nenv* $\models$ *?ψ*) $\longleftrightarrow$
($\exists\,\vartheta{\in}M.\ \exists\,p{\in}P.\ u = \langle\vartheta,p\rangle \wedge$
($\forall\,F.\ M\_generic(F) \wedge p \in F \longrightarrow M[F],\ \ \ map(val(F),\ [\vartheta]$ @ *nenv* @$[\pi])$
$\models$ *?χ*))
for u
by *simp*
moreover from ‹*env* = _› ‹$\pi{\in}M$› ‹*nenv*∈*list*(M)›
have *map_nenv*:*map*(*val*(G), *nenv*@$[\pi]$) = *env* @ [*val*(G,π)]
using *map_app_distrib append1_eq_iff* **by** *auto*
ultimately
have *aux*:($\exists\,\vartheta{\in}M.\ \exists\,p{\in}P.\ u = \langle\vartheta,p\rangle \wedge (p{\in}G \longrightarrow M[G],\ [val(G,\vartheta)]$ @ *env* @
$[val(G,\pi)] \models$ *?χ*))
(**is** ($\exists\,\vartheta{\in}M.\ \exists\,p{\in}P.$ _ (_ $\longrightarrow$ _, *?vals*(ϑ) $\models$ _)))
if $u \in$ *domain*$(\pi) \times P$ $u \in M$ M, $[u]$@ *?Pl1* @$[\pi]$ @ *nenv* $\models$ *?ψ* **for** u
using *Equivalence*[*THEN iffD1*, *OF that*] *generic* **by** *force*
moreover
have $\vartheta{\in}M \Longrightarrow$ *val*(G,ϑ)∈M[G] **for** ϑ
using *GenExt_def* **by** *auto*
moreover
have $\vartheta{\in}\ M \Longrightarrow [val(G,\ \vartheta)]$ @ *env* @ $[val(G,\ \pi)] \in list(M[G])$ **for** ϑ
proof -
from ‹$\pi{\in}M$›
have *val*(G,π)∈ M[G] **using** *GenExtI* **by** *simp*
moreover
assume $\vartheta \in M$
moreover
note ‹*env* ∈ *list*(M[G])›
ultimately
show *?thesis*
using *GenExtI* **by** *simp*
qed
ultimately
have ($\exists\,\vartheta{\in}M.\ \exists\,p{\in}P.\ u{=}\langle\vartheta,p\rangle \wedge (p{\in}G \longrightarrow val(G,\vartheta){\in}nth(1\ \#{+}\ length(env),[val(G,$
$\vartheta)]$ @ *env* @ $[val(G,\ \pi)]$)
$\wedge\ M[G],\ \ ?vals(\vartheta) \models\ \varphi))$
if $u \in$ *domain*$(\pi) \times P$ $u \in M$ M, $[u]$ @ *?Pl1* @$[\pi]$ @ *nenv* $\models$ *?ψ* **for** u
using *aux*[*OF that*] **by** *simp*
moreover from ‹*env* ∈ _› ‹$\pi{\in}M$›

have *nth*:*nth*(*1 #+ length*(*env*),[*val*(*G*, *ϑ*)] @ *env* @ [*val*(*G*, *π*)]) = *val*(*G*,*π*)
if *ϑ*∈*M* **for** *ϑ*
using *nth_concat*[*of val*(*G*,*ϑ*) *val*(*G*,*π*) *M*[*G*]] **using** *that GenExtI* **by** *simp*
ultimately
have (∃ *ϑ*∈*M*. ∃ *p*∈*P*. *u*=⟨*ϑ*,*p*⟩ ∧ (*p*∈*G* ⟶ *val*(*G*,*ϑ*)∈*val*(*G*,*π*) ∧ *M*[*G*], *?vals*(*ϑ*) ⊨ *φ*))
if *u* ∈ *domain*(*π*) × *P u* ∈ *M* *M*, [*u*] @ *?Pl1* @[*π*] @ *nenv* ⊨ *?ψ* **for** *u*
using *that* ‹*π*∈*M*› ‹*env* ∈ _› **by** *simp*
with ‹*domain*(*π*)×*P*∈*M*›
have ∀ *u*∈*domain*(*π*)×*P* . (*M*, [*u*] @ *?Pl1* @[*π*] @ *nenv* ⊨ *?ψ*) ⟶ (∃ *ϑ*∈*M*. ∃ *p*∈*P*. *u* =⟨*ϑ*,*p*⟩ ∧
(*p* ∈ *G* ⟶ *val*(*G*, *ϑ*)∈*val*(*G*, *π*) ∧ *M*[*G*], *?vals*(*ϑ*) ⊨ *φ*))
by (*simp add*:*transitivity*)
then
have {*u*∈*domain*(*π*)×*P* . (*M*,[*u*] @ *?Pl1* @[*π*] @ *nenv* ⊨ *?ψ*) } ⊆
{*u*∈*domain*(*π*)×*P* . ∃ *ϑ*∈*M*. ∃ *p*∈*P*. *u* =⟨*ϑ*,*p*⟩ ∧
(*p* ∈ *G* ⟶ *val*(*G*, *ϑ*)∈*val*(*G*, *π*) ∧ (*M*[*G*], *?vals*(*ϑ*) ⊨ *φ*))}
(**is** *?n*⊆*?m*)
by *auto*
with *val_mono*
have *first_incl*: *val*(*G*,*?n*) ⊆ *val*(*G*,*?m*)
by *simp*
note ‹*val*(*G*,*π*) = *c*›
with ‹*?ψ*∈*formula*› ‹*arity*(*?ψ*) ≤ _› *in_M* ‹*nenv* ∈ _› ‹*env* ∈ _› ‹*length*(*nenv*) = _›
have *?n*∈*M*
using *separation_ax leI separation_iff* **by** *auto*
from *generic*
have *filter*(*G*) *G*⊆*P*
unfolding *M_generic_def filter_def* **by** *simp_all*
from ‹*val*(*G*,*π*) = *c*›
have *val*(*G*,*?m*) =
{*val*(*G*,*t*) .. *t*∈*domain*(*π*) , ∃ *q*∈*P* .
(∃ *ϑ*∈*M*. ∃ *p*∈*P*. ⟨*t*,*q*⟩ = ⟨*ϑ*, *p*⟩ ∧
(*p* ∈ *G* ⟶ *val*(*G*, *ϑ*) ∈ *c* ∧ (*M*[*G*], [*val*(*G*, *ϑ*)] @ *env* @ [*c*] ⊨ *φ*)) ∧ *q* ∈ *G*)}
using *val_of_name* **by** *auto*
also
have ... = {*val*(*G*,*t*) .. *t*∈*domain*(*π*) , ∃ *q*∈*P*.
val(*G*, *t*) ∈ *c* ∧ (*M*[*G*], [*val*(*G*, *t*)] @ *env* @ [*c*] ⊨ *φ*) ∧ *q* ∈ *G*}
proof -

have *t*∈*M* ⟹
(∃ *q*∈*P*. (∃ *ϑ*∈*M*. ∃ *p*∈*P*. ⟨*t*,*q*⟩ = ⟨*ϑ*, *p*⟩ ∧
(*p* ∈ *G* ⟶ *val*(*G*, *ϑ*) ∈ *c* ∧ (*M*[*G*], [*val*(*G*, *ϑ*)] @ *env* @ [*c*] ⊨ *φ*)) ∧ *q* ∈ *G*))
⟷
(∃ *q*∈*P*. *val*(*G*, *t*) ∈ *c* ∧ (*M*[*G*], [*val*(*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(G,π) = c›
     have ∃ q∈P. ∃ t∈domain(π). val(G,t) =x ∧ (M[G],  [val(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(G,?m) = {x∈c. (M[G], [x] @ env @ [c] ⊨ φ)} by simp
 have val(G,?m) ⊆ val(G,?n)
 proof
  fix x
  assume x ∈ val(G,?m)
  with val_m
  have Eq4: x ∈ {x∈c. (M[G], [x] @ env @ [c] ⊨ φ)} by simp
  with ‹val(G,π) = c›
  have x ∈ val(G,π) by simp
  then
  have ∃ ϑ. ∃ q∈G. ⟨ϑ,q⟩∈π ∧ val(G,ϑ) =x
    using elem_of_val_pair by auto
  then obtain ϑ q where
    ⟨ϑ,q⟩∈π  q∈G val(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 $[val(G,\vartheta),\ val(G,\pi)]$ @ $env \in list(M[G])$
using *GenExt_def* **by** *auto*
with *Eq4* ‹$val(G,\vartheta)=x$› ‹$val(G,\pi) = c$› ‹$x \in val(G,\pi)$› *nth* ‹$\vartheta \in M$›
have *Eq5*: $M[G],\ [val(G,\vartheta)]$ @ env @$[val(G,\pi)] \models And(Member(0,1\ \#+ length(env)),\varphi)$
by *auto*

with ‹$\vartheta \in M$› ‹$\pi \in M$› *Eq5* ‹*M_generic*(G)› ‹$\varphi \in formula$› ‹$nenv \in \_$› ‹$env = \_$› *map_nenv*
‹$arity(?\chi) \leq length([\vartheta]$ @ $nenv$ @ $[\pi])$›
have $(\exists\, r \in G.\ M,\ [r,P,leq,one,\vartheta]$ @ $nenv$ @$[\pi] \models forces(?\chi))$
using *truth_lemma*
by *auto*
then obtain r **where**
$r \in G\ M,\ [r,P,leq,one,\vartheta]$ @ $nenv$ @ $[\pi] \models forces(?\chi)$ **by** *auto*
with ‹*filter*(G)› **and** ‹$q \in G$› **obtain** p **where**
$p \in G\ p \preceq q\ p \preceq r$
unfolding *filter_def compat_in_def* **by** *force*
with ‹$r \in G$› ‹$q \in G$› ‹$G \subseteq P$›
have $p \in P\ r \in P\ q \in P\ p \in M$
using *P_in_M* **by** (*auto simp add*:*transitivity*)
with ‹$\varphi \in formula$› ‹$\vartheta \in M$› ‹$\pi \in M$› ‹$p \preceq r$› ‹$nenv \in \_$› ‹$arity(?\chi) \leq length([\vartheta]$ @ $nenv$ @ $[\pi])$›
‹$M,\ [r,P,leq,one,\vartheta]$ @ $nenv$ @ $[\pi] \models forces(?\chi)$› ‹$env \in \_$›
have $M,\ [p,P,leq,one,\vartheta]$ @ $nenv$ @ $[\pi] \models forces(?\chi)$
using *strengthening_lemma*
by *simp*
with ‹$p \in P$› ‹$\varphi \in formula$› ‹$\vartheta \in M$› ‹$\pi \in M$› ‹$nenv \in \_$› ‹$arity(?\chi) \leq length([\vartheta]$ @ $nenv$ @ $[\pi])$›
have $\forall\, F.\ M\_generic(F) \wedge p \in F \longrightarrow$
$M[F],\ \ map(val(F),\ [\vartheta]$ @ $nenv$ @$[\pi]) \models\ ?\chi$
using *definition_of_forcing*
by *simp*
with ‹$p \in P$› ‹$\vartheta \in M$›
have *Eq6*: $\exists\, \vartheta' \in M.\ \exists\, p' \in P.\ \langle\vartheta,p\rangle = \langle\vartheta',p'\rangle \wedge (\forall\, F.\ M\_generic(F) \wedge p' \in F \longrightarrow$
$M[F],\ \ map(val(F),\ [\vartheta']$ @ $nenv$ @ $[\pi]) \models\ ?\chi)$ **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 $val(G,\vartheta) \in val(G,?n)$

```
    using  val_of_elem[of ϑ p] by simp
   with ‹val(G,ϑ)=x›
   show x∈val(G,?n) by simp
  qed
  with val_m first_incl
  have val(G,?n) = {x∈c. (M[G], [x] @ env @ [c] ⊨ φ)} by auto
  also
  have  ... = {x∈c. (M[G], [x] @ env ⊨ φ)}
  proof -
   {
    fix x
    assume x∈c
    moreover from assms
    have c∈M[G]
     unfolding GenExt_def by auto
    moreover from this and ‹x∈c›
    have x∈M[G]
     using transitivity_MG
     by simp
    ultimately
    have (M[G],  ([x] @ env) @[c] ⊨  φ) ⟷ (M[G],  [x] @ env ⊨  φ)
     using phi ‹env ∈ _› by (rule_tac arity_sats_iff, simp_all)
   }
   then show ?thesis by auto
  qed
  finally
  show {x∈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(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

21 The Axiom of Pairing in $M[G]$

theory *Pairing_Axiom* **imports** *Names* **begin**

context *forcing_data*
begin

lemma *val_Upair* :
$one \in G \Longrightarrow val(G,\{\langle\tau,one\rangle,\langle\varrho,one\rangle\}) = \{val(G,\tau),val(G,\varrho)\}$
by (*insert one_in_P*, *rule trans*, *subst def_val*,*auto simp add*: *Sep_and_Replace*)

lemma *pairing_in_MG* :
assumes *M_generic*(G)
shows *upair_ax*($\#\#M[G]$)
proof -
{
fix x y
have $one{\in}G$ **using** *assms one_in_G* **by** *simp*
from *assms*
have $G{\subseteq}P$ **unfolding** *M_generic_def* **and** *filter_def* **by** *simp*
with ‹$one{\in}G$›
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 : $val(G,\tau) = x$ $val(G,\varrho) = y$ $\varrho \in M$ $\tau \in M$
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 $val(G,?\sigma) \in M[G]$ **using** *GenExtI* **by** *simp*
with *1*
have $\{val(G,\tau),val(G,\varrho)\} \in M[G]$ **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

22 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*] ⇒ *o* **where**
  Union_name_body(*P'*,*leq'*,*τ*,*ϑp*) ≡ (∃ *σ*[##*M*].
        ∃ *q*[##*M*]. (*q*∈ *P'* ∧ (⟨*σ*,*q*⟩ ∈ *τ* ∧
         (∃ *r*[##*M*].*r*∈*P'* ∧ (⟨*fst*(*ϑp*),*r*⟩ ∈ *σ* ∧ ⟨*snd*(*ϑp*),*r*⟩ ∈ *leq'* ∧ ⟨*snd*(*ϑp*),*q*⟩
∈ *leq'*)))))

definition *Union_name_fm* :: *i* **where**
  Union_name_fm ≡
    Exists(
    Exists(*And*(*pair_fm*(1,0,2),
    Exists (
    Exists (*And*(*Member*(0,7),
      Exists (*And*(*And*(*pair_fm*(2,1,0),*Member*(0,6)),
        Exists (*And*(*Member*(0,9),
         Exists (*And*(*And*(*pair_fm*(6,1,0),*Member*(0,4)),
          Exists (*And*(*And*(*pair_fm*(6,2,0),*Member*(0,10)),
          Exists (*And*(*pair_fm*(7,5,0),*Member*(0,11)))))))))))))))))

lemma *Union_name_fm_type* [*TC*]:
  Union_name_fm ∈*formula*
  unfolding *Union_name_fm_def* **by** *simp*

lemma *arity_Union_name_fm* :
  arity(*Union_name_fm*) = 4
  unfolding *Union_name_fm_def upair_fm_def pair_fm_def*
  by(*auto simp add*: *nat_simp_union*)

lemma *sats_Union_name_fm* :
  ⟦ *a* ∈ *M*; *b* ∈ *M* ; *P'* ∈ *M* ; *p* ∈ *M* ; *ϑ* ∈ *M* ; *τ* ∈ *M* ; *leq'* ∈ *M* ⟧ ⟹
     sats(*M*,*Union_name_fm*,[⟨*ϑ*,*p*⟩,*τ*,*leq'*,*P'*]@[*a*,*b*]) ⟷
     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*)

lemma *domD* :
assumes $\tau \in M$ $\sigma \in domain(\tau)$
shows $\sigma \in M$
using *assms Transset_M trans_M*
by (*simp flip*: *setclass_iff*)

definition *Union_name* :: $i \Rightarrow i$ **where**
Union_name(τ) $\equiv$
$\{u \in domain(\bigcup(domain(\tau))) \times P$. *Union_name_body*(P,*leq*,τ,u)$\}$

lemma *Union_name_M* : **assumes** $\tau \in M$
shows $\{u \in domain(\bigcup(domain(\tau))) \times P$. *Union_name_body*(P,*leq*,τ,u)$\} \in M$
unfolding *Union_name_def*
proof -
let *?P*=λ x . *sats*(M,*Union_name_fm*,[x,τ,*leq*]@[P,τ,*leq*])
let *?Q*=λ x . *Union_name_body*(P,*leq*,τ,x)
from ‹$\tau \in M$›
have *domain*($\bigcup(domain(\tau))$)$\in M$ (**is** *?d* $\in$ _) **using** *domain_closed Union_closed*
by *simp*
then
have *?d* $\times$ $P \in M$ **using** *cartprod_closed P_in_M* **by** *simp*
have *arity*(*Union_name_fm*)$\leq$6 **using** *arity_Union_name_fm* **by** *simp*
from *assms P_in_M leq_in_M arity_Union_name_fm*
have [τ,*leq*] $\in$ *list*(M) [P,τ,*leq*] $\in$ *list*(M) **by** *auto*
with *assms assms P_in_M leq_in_M* ‹*arity*(*Union_name_fm*)$\leq$6›
have *separation*(##M,*?P*)
using *separation_ax* **by** *simp*
with ‹*?d* $\times$ $P \in M$›
have A:{ $u \in$ *?d* $\times$ P . *?P*(u) } $\in M$
using *separation_iff* **by** *force*
have *?P*(x)$\longleftrightarrow$ *?Q*(x) **if** $x\in$ *?d*$\times P$ **for** x
proof -
from ‹$x\in$ *?d*$\times P$›
have $x = \langle fst(x),snd(x)\rangle$ **using** *Pair_fst_snd_eq* **by** *simp*
with ‹$x\in$*?d*$\times P$› ‹*?d*$\in M$›
have *fst*(x) $\in M$ *snd*(x) $\in M$
using *mtrans fst_type snd_type P_in_M* **unfolding** *M_trans_def* **by** *auto*
then
have *?P*($\langle fst(x),snd(x)\rangle$) $\longleftrightarrow$ *?Q*($\langle fst(x),snd(x)\rangle$)
using *P_in_M sats_Union_name_fm P_in_M* ‹$\tau \in M$› *leq_in_M* **by** *simp*
with ‹$x = \langle fst(x),snd(x)\rangle$›
show *?P*(x) $\longleftrightarrow$ *?Q*(x) **using** *that* **by** *simp*
qed
then show *?thesis* **using** *Collect_cong A* **by** *simp*
qed

lemma *Union_MG_Eq* :
assumes $a \in M[G]$ **and** $a = val(G,\tau)$ **and** *filter*(G) **and** $\tau \in M$
shows $\bigcup\ a = val(G,Union\_name(\tau))$
proof -
{
fix x
assume $x \in \bigcup\ (val(G,\tau))$
then obtain i **where** $i \in val(G,\tau)$ $x \in i$ **by** *blast*
with ‹$\tau \in M$› **obtain** σ q **where**
$q \in G$ $\langle\sigma,q\rangle \in \tau$ $val(G,\sigma) = i$ $\sigma \in M$
using *elem_of_val_pair domD* **by** *blast*
with ‹$x \in i$› **obtain** ϑ r **where**
$r \in G$ $\langle\vartheta,r\rangle \in \sigma$ $val(G,\vartheta) = x$ $\vartheta \in M$
using *elem_of_val_pair domD* **by** *blast*
with ‹$\langle\sigma,q\rangle \in \tau$› **have** $\vartheta \in domain(\bigcup(domain(\tau)))$ **by** *auto*
with ‹*filter*(G)› ‹$q \in G$› ‹$r \in G$› **obtain** p **where**
A: $p \in G$ $\langle p,r\rangle \in leq$ $\langle p,q\rangle \in leq$ $p \in P$ $r \in P$ $q \in P$
using *low_bound_filter filterD* **by** *blast*
then have $p \in M$ $q \in M$ $r \in M$
using *mtrans P_in_M* **unfolding** *M_trans_def* **by** *auto*
with *A* ‹$\langle\vartheta,r\rangle \in \sigma$› ‹$\langle\sigma,q\rangle \in \tau$› ‹$\vartheta \in M$› ‹$\vartheta \in domain(\bigcup(domain(\tau)))$› ‹$\sigma \in M$›
have
$\langle\vartheta,p\rangle \in Union\_name(\tau)$ **unfolding** *Union_name_def Union_name_body_def*
by *auto*
with ‹$p \in P$› ‹$p \in G$› **have** $val(G,\vartheta) \in val(G,Union\_name(\tau))$
using *val_of_elem* **by** *simp*
with ‹$val(G,\vartheta)=x$› **have** $x \in val(G,Union\_name(\tau))$ **by** *simp*
}
with ‹$a=val(G,\tau)$› **have** *1*: $x \in \bigcup\ a \Longrightarrow x \in val(G,Union\_name(\tau))$ **for** x **by** *simp*
{
fix x
assume $x \in (val(G,Union\_name(\tau)))$
then obtain ϑ p **where**
$p \in G$ $\langle\vartheta,p\rangle \in Union\_name(\tau)$ $val(G,\vartheta) = x$
using *elem_of_val_pair* **by** *blast*
with ‹*filter*(G)› **have** $p \in P$ **using** *filterD* **by** *simp*
from ‹$\langle\vartheta,p\rangle \in Union\_name(\tau)$› **obtain** σ q r **where**
$\sigma \in domain(\tau)$ $\langle\sigma,q\rangle \in \tau$ $\langle\vartheta,r\rangle \in \sigma$ $r \in P$ $q \in P$ $\langle p,r\rangle \in leq$ $\langle p,q\rangle \in leq$
unfolding *Union_name_def Union_name_body_def* **by** *force*
with ‹$p \in G$› ‹*filter*(G)› **have** $r \in G$ $q \in G$
using *filter_leqD* **by** *auto*
with ‹$\langle\vartheta,r\rangle \in \sigma$› ‹$\langle\sigma,q\rangle \in \tau$› ‹$q \in P$› ‹$r \in P$› **have**
$val(G,\sigma) \in val(G,\tau)$ $val(G,\vartheta) \in val(G,\sigma)$
using *val_of_elem* **by** *simp+*
then have $val(G,\vartheta) \in \bigcup\ val(G,\tau)$ **by** *blast*
with ‹$val(G,\vartheta)=x$› ‹$a=val(G,\tau)$› **have**

```
    x ∈ ⋃ a by simp
  }
  with ‹a=val(G,τ)›
  have x ∈ val(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(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(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(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 force
  }
  then
  have Union_ax(##M[G]) unfolding Union_ax_def by force
  then
  show ?thesis by simp
qed

theorem Union_MG : M_generic(G) ⟹ Union_ax(##M[G])
  by (simp add:M_generic_def union_in_MG)

end
end
```

23 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
{*sq* ∈*A*×*B* . *sats*(*M*,φ,[*snd*(*sq*),*p*,*l*,*o*,*fst*(*sq*),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 ‹$A\times B \in M$›
moreover
assume $sp \in A\times B$
moreover from *calculation*
have $fst(sp) \in A$ $snd(sp) \in B$
using *fst_type snd_type* **by** *simp_all*
ultimately
have $sp \in M$ $fst(sp) \in M$ $snd(sp) \in M$
using ‹$A\in M$› ‹$B\in M$› *transitivity*
by *simp_all*
note $inM = $ ‹$A\in M$› ‹$B\in M$› ‹$p\in M$› ‹$l\in M$› ‹$o\in M$› ‹$\chi\in M$›
‹$sp\in M$› ‹$fst(sp)\in M$› ‹$snd(sp)\in M$›
with *1* ‹$sp \in M$› ‹$?\varphi' \in formula$›
have M, $[sp,p,l,o,\chi]@[p] \models ?\psi \longleftrightarrow M,[sp,p,l,o,\chi] \models ?\psi$ (**is** M,*?env0*@ _⊨_
⟷ _)
using *arity_sats_iff*[*of ?ψ* [*p*] *M ?env0*] **by** *auto*
also from inM ‹$sp \in A\times B$›
have ... ⟷ $sats(M,?\varphi',[fst(sp),snd(sp),sp,p,l,o,\chi])$
by *auto*
also from inM ‹$\varphi \in formula$› ‹$arity(\varphi) \le 6$›
have ... ⟷ $sats(M,\varphi,[snd(sp),p,l,o,fst(sp),\chi])$
(**is** $sats(\_,\_,?env1) \longleftrightarrow sats(\_,\_,?env2)$)
using *sats_iff_sats_ren*[*of φ 6 7 ?env2 M ?env1 perm_pow_fn*] *perm_pow_thm*
unfolding *perm_pow_fn_def* **by** *simp*
finally
have $sats(M,?\psi,[sp,p,l,o,\chi,p]) \longleftrightarrow sats(M,\varphi,[snd(sp),p,l,o,fst(sp),\chi])$
by *simp*
}
then have
$?\vartheta = \{sp\in A\times B$. $sats(M,?\psi,[sp,p,l,o,\chi,p])\}$
by *auto*
also from *assms* ‹$A\times B\in M$› **have**
... ∈ M
proof -
from *1*
have $arity(?\psi) \le 6$
using *leI* **by** *simp*
moreover from ‹$?\varphi' \in formula$›
have $?\psi \in formula$
by *simp*
moreover note *assms* ‹$A\times B\in M$›
ultimately
show $\{x \in A\times B$. $sats(M, ?\psi, [x, p, l, o, \chi, p])\} \in M$
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(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(G,?π)
  from ‹?Q∈M›
  have ?π∈M
    using one_in_P P_in_M transitivity
    by (simp flip: setclass_iff)
  from ‹?π∈M›
  have ?b ∈ M[G]
    using GenExtI by simp
  have Pow(a) ∩ M[G] ⊆ ?b
```

proof
fix c
assume $c \in Pow(a) \cap M[G]$
then obtain χ **where**
$c \in M[G]$ $\chi \in M$ $val(G,\chi) = c$
using *GenExtD* **by** *auto*
let $?\vartheta = \{sp \in domain(\tau) \times P \,.\, snd(sp) \Vdash (Member(0,1))\ [fst(sp),\chi]\ \}$
have $arity(forces(Member(0,1))) = 6$
using *arity_forces_at* **by** *auto*
with ‹$domain(\tau) \in M$› ‹$\chi \in M$›
have $?\vartheta \in M$
using *P_in_M one_in_M leq_in_M sats_fst_snd_in_M*
by *simp*
then
have $?\vartheta \in ?Q$
by *auto*
then
have $val(G,?\vartheta) \in ?b$
using *one_in_G one_in_P generic val_of_elem* [*of* $?\vartheta$ *one* $?\pi$ G]
by *auto*
have $val(G,?\vartheta) = c$
proof(*intro equalityI subsetI*)
fix x
assume $x \in val(G,?\vartheta)$
then obtain σ p **where**
1: $<\sigma,p> \in ?\vartheta$ $p \in G$ $val(G,\sigma) = x$
using *elem_of_val_pair*
by *blast*
moreover from ‹$<\sigma,p> \in ?\vartheta$› ‹$?\vartheta \in M$›
have $\sigma \in M$
using *name_components_in_M*[*of* _ _ $?\vartheta$] **by** *auto*
moreover from *1*
have $(p \Vdash (Member(0,1))\ [\sigma,\chi])\ p \in P$
by *simp_all*
moreover
note ‹$val(G,\chi) = c$›
ultimately
have $sats(M[G],Member(0,1),[x,c])$
using ‹$\chi \in M$› *generic definition_of_forcing nat_simp_union*
by *auto*
moreover
have $x \in M[G]$
using ‹$val(G,\sigma) = x$› ‹$\sigma \in M$› ‹$\chi \in M$› *GenExtI* **by** *blast*
ultimately
show $x \in c$
using ‹$c \in M[G]$› **by** *simp*
next
fix x
assume $x \in c$

```
    with ‹c ∈ Pow(a) ∩ M[G]›
    have x ∈ a c∈M[G] x∈M[G]
      using transitivity_MG
      by auto
    with ‹val(G, τ) = a›
    obtain σ where
      σ∈domain(τ) val(G,σ) =  x
      using elem_of_val
      by blast
    moreover note ‹x∈c› ‹val(G,χ) = c›
    moreover from calculation
    have val(G,σ) ∈ val(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 Member(0,1)∈formula 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 unfolding M_generic_def filter_def by blast
    ultimately
    have <σ,p>∈?ϑ
      using ‹σ∈domain(τ)› by simp
    with ‹val(G,σ) =  x› ‹p∈G›
    show x∈val(G,?ϑ)
      using val_of_elem [of _ _ ?ϑ] by auto
  qed
  with ‹val(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 . sats(M[G],subset_fm(0,1),[x,a]) & x∈M[G]}
    using Transset_MG by force
  also
  have  ... = {x∈?b . sats(M[G],subset_fm(0,1),[x,a])} ∩ M[G]
    by auto
  also from ‹?b∈M[G]›
  have  ... = {x∈?b . sats(M[G],subset_fm(0,1),[x,a])}
    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

24 The Axiom of Extensionality in $M[G]$

theory *Extensionality_Axiom*
imports
  Names
begin

context *forcing_data*
begin

lemma *extensionality_in_MG* : *extensionality*(##($M[G]$))
proof -
  {
    fix x y z
    assume
      asms: $x{\in}M[G]$ $y{\in}M[G]$ $(\forall\, w{\in}M[G]\ .\ w \in x \longleftrightarrow w \in y)$
    from ‹$x{\in}M[G]$› **have**
      $z{\in}x \longleftrightarrow z{\in}M[G] \wedge z{\in}x$
      using *transitivity_MG* **by** *auto*
    also have
      $\ldots \longleftrightarrow z{\in}y$
      using *asms transitivity_MG* **by** *auto*
    finally have
      $z{\in}x \longleftrightarrow z{\in}y$ **.**
  }
  then have
    $\forall\, x{\in}M[G]\ .\ \forall\, y{\in}M[G]\ .\ (\forall\, z{\in}M[G]\ .\ z \in x \longleftrightarrow z \in y) \longrightarrow x = y$
    by *blast*
  then show *?thesis* **unfolding** *extensionality_def* **by** *simp*
qed

end
end

25 The Axiom of Foundation in $M[G]$

theory *Foundation_Axiom*
imports
  Names
begin

context *forcing_data*
begin

lemma *foundation_in_MG* : *foundation_ax*(##(*M*[*G*]))
unfolding *foundation_ax_def*
by (*rule rallI*, *cut_tac A*=*x* **in** *foundation*, *auto intro*: *transitivity_MG*)

lemma *foundation_ax*(##(*M*[*G*]))
proof -
{
fix *x*
assume *x*∈*M*[*G*] ∃ *y*∈*M*[*G*] . *y*∈*x*
then
have ∃ *y*∈*M*[*G*] . *y*∈*x*∩*M*[*G*] **by** *simp*
then
obtain *y* **where** *y*∈*x*∩*M*[*G*] ∀ *z*∈*y*. *z* ∉ *x*∩*M*[*G*]
using *foundation*[*of x*∩*M*[*G*]] **by** *blast*
then
have ∃ *y*∈*M*[*G*] . *y* ∈ *x* ∧ (∀ *z*∈*M*[*G*] . *z* ∉ *x* ∨ *z* ∉ *y*)**by** *auto*
}
then show *?thesis*
unfolding *foundation_ax_def* **by** *auto*
qed

end
end

26 The binder *Least*

theory *Least*
imports
Names

begin

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

lemma *Least_Ord*: (*μ α*. *R*(*α*)) = (*μ α*. *Ord*(*α*) ∧ *R*(*α*))
unfolding *Least_def* **by** (*simp add*:*lt_Ord*)

lemma *Ord_Least_cong*:
assumes ⋀*y*. *Ord*(*y*) ⟹ *R*(*y*) ⟷ *Q*(*y*)
shows (*μ α*. *R*(*α*)) = (*μ α*. *Q*(*α*))
proof -
from *assms*
have (*μ α*. *Ord*(*α*) ∧ *R*(*α*)) = (*μ α*. *Ord*(*α*) ∧ *Q*(*α*))
by *simp*
then
show *?thesis* **using** *Least_Ord* **by** *simp*
qed

definition

least :: $[i⇒o,i⇒o,i] ⇒ o$ **where**
least$(M,Q,i) \equiv$ *ordinal*$(M,i) \wedge$ (
$\quad$ (*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 :: $[i,i] ⇒ i$ **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*

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

26.1 Absoluteness and closure under *Least*

lemma *least_abs*:
assumes $\bigwedge x.\ Q(x) \Longrightarrow M(x)\ M(a)$
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 ‹$\bigwedge x.\ Q(x) \Longrightarrow M(x)$›
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** ‹$\bigwedge x.\ Q(x) \Longrightarrow M(x)$›
have $Ord(b) \Longrightarrow b \in a \Longrightarrow \neg\ Q(b)$ **for** b
by *blast*
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 $\bigwedge x.\ Q(x) \Longrightarrow M(x)$
shows $M(\mu\ x.\ Q(x))$
using *assms LeastI*[*of Q*] *Least_0* **by** (*cases* ($\exists\, i.\ Ord(i) \wedge Q(i)$), *auto*)

end

end

27 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*) ≡ *sum*(*renrep1_fn*,*id*(*length*(*env*)),*6*,*8*,*length*(*env*))

definition
renrep :: [*i*,*i*] ⇒ *i* **where**
renrep(*φ*,*env*) = *ren*(*φ*)‘(*6*#+*length*(*env*))‘(*8*#+*length*(*env*))‘*renrep_fn*(*env*)

lemma *renrep_type* [*TC*]:
assumes *φ*∈*formula env* ∈ *list*(*M*)
shows *renrep*(*φ*,*env*) ∈ *formula*
unfolding *renrep_def renrep_fn_def renrep1_fn_def*
using *assms renrep1_thm*(*1*) *ren_tc*
by *simp*

lemma *arity_renrep*:
assumes *φ*∈*formula arity*(*φ*)≤ *6*#+*length*(*env*) *env* ∈ *list*(*M*)
shows *arity*(*renrep*(*φ*,*env*)) ≤ *8*#+*length*(*env*)
unfolding *renrep_def renrep_fn_def renrep1_fn_def*
using *assms renrep1_thm*(*1*) *arity_ren*
by *simp*

lemma *renrep_sats* :
assumes *arity*(*φ*) ≤ *6* #+ *length*(*env*)
[*P*,*leq*,*o*,*p*,*ϱ*,*τ*] @ *env* ∈ *list*(*M*)
V ∈ *M α* ∈ *M*

φ∈*formula*
shows *sats*(M, φ, [p,P,*leq*,o,ϱ,τ] @ *env*) ⟷ *sats*(M, *renrep*(φ,*env*), [V,τ,ϱ,p,α,P,*leq*,o] @ *env*)
unfolding *renrep_def renrep_fn_def renrep1_fn_def*
by (*rule sats_iff_sats_ren*,*insert assms*, *auto simp add*:*renrep1_thm*(*1*)[*of* _ M,*simplified*]
renrep1_thm(*2*)[*simplified*,**where** p=p **and** α=α])

rename *renpbdy1* **src** [ϱ,p,α,P,*leq*,o] **tgt** [ϱ,p,x,α,P,*leq*,o]

definition *renpbdy_fn* :: $i \Rightarrow i$ **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, φ, [ϱ,p,α,P,*leq*,o] @ *nenv*) ⟷ *sats*(M, *renpbdy*(φ,*nenv*), [ϱ,p,x,α,P,*leq*,o] @ *nenv*)
unfolding *renpbdy_def renpbdy_fn_def renpbdy1_fn_def*
by (*rule sats_iff_sats_ren*,*auto simp add*: *renpbdy1_thm*(*1*)[*of* _ M,*simplified*]
renpbdy1_thm(*2*)[*simplified*,**where** α=α **and** x=x])

rename *renbody1* **src** [x,α,P,*leq*,o] **tgt** [α,x,m,P,*leq*,o]

definition *renbody_fn* :: $i \Rightarrow i$ **where**
renbody_fn(*env*) ≡ *sum*(*renbody1_fn*,*id*(*length*(*env*)),*5*,*6*,*length*(*env*))

definition
renbody :: [i,i] ⇒ i **where**
renbody(φ,*env*) = *ren*(φ)‘(*5*#+*length*(*env*))‘(*6*#+*length*(*env*))‘*renbody_fn*(*env*)

lemma
renbody_type [*TC*]: φ∈*formula* ⟹ *env*∈*list*(*M*) ⟹ *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* ⟹ *arity*(φ) ≤ *5* #+ *length*(*env*) ⟹ *env*∈*list*(*M*)
⟹
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*) ⟹ [α,*x*,*m*,*P*,*leq*,*o*] @ *nenv* ∈
list(*M*) ⟹ φ∈*formula* ⟹
sats(*M*, φ, [*x*,α,*P*,*leq*,*o*] @ *nenv*) ⟷ *sats*(*M*, *renbody*(φ,*nenv*), [α,*x*,*m*,*P*,*leq*,*o*]
@ *nenv*)
unfolding *renbody_def renbody_fn_def renbody1_fn_def*
by (*rule sats_iff_sats_ren*, *auto simp add*:*renbody1_thm*(*1*)[*of* _ *M*,*simplified*]
renbody1_thm(*2*)[**where** α=α **and** *m*=*m*,*simplified*])

context *G_generic*
begin

lemma *pow_inter_M*:
assumes
x∈*M* *y*∈*M*
shows
powerset(##*M*,*x*,*y*) ⟷ *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

lemmas *new_fm_defs* = *fm_defs is_transrec_fm_def is_eclose_fm_def mem_eclose_fm_def*
finite_ordinal_fm_def is_wfrec_fm_def Memrel_fm_def eclose_n_fm_def is_recfun_fm_def
is_iterates_fm_def
iterates_MH_fm_def is_nat_case_fm_def quasinat_fm_def pre_image_fm_def
restriction_fm_def

lemma *sats_prebody_fm*:
assumes
[*P*,*leq*,*one*,*p*,*ϱ*] @ *nenv* ∈*list*(*M*) *φ*∈*formula α*∈*M arity*(*φ*) ≤ *2* #+ *length*(*nenv*)
shows
sats(*M*,*prebody_fm*(*φ*,*nenv*),[*ϱ*,*p*,*α*,*P*,*leq*,*one*] @ *nenv*) ⟷
(∃ *τ*∈*M*. ∃ *V*∈*M*. *is_Vset*(##*M*,*α*,*V*) ∧ *τ*∈*V* ∧ *sats*(*M*,*forces*(*φ*),[*p*,*P*,*leq*,*one*,*ϱ*,*τ*]
@ *nenv*))
unfolding *prebody_fm_def* **using** *assms sats_prebody_fm_auto* **by** *force*

lemma *arity_prebody_fm*:
assumes
φ∈formula α∈M env ∈ list(M) arity(φ) ≤ 2 #+ length(env)
shows
arity(prebody_fm(φ,env))≤6 #+ length(env)
unfolding *prebody_fm_def is_HVfrom_fm_def is_powapply_fm_def*
using *assms new_fm_defs nat_simp_union*
arity_renrep[*of forces*(*φ*)] *arity_forces_le*[*simplified*] *pred_le* **by** *auto*

definition
body_fm′ :: [*i*,*i*]⇒*i* **where**
body_fm′(*φ*,*env*) ≡ *Exists*(*Exists*(*And*(*pair_fm*(*0*,*1*,*2*),*renpbdy*(*prebody_fm*(*φ*,*env*),*env*))))

lemma *body_fm′_type*[*TC*]: *φ∈formula* ⟹ *env∈list*(*M*) ⟹ *body_fm′*(*φ*,*env*)*∈formula*
unfolding *body_fm′_def* **using** *prebody_fm_type*
by *simp*

lemma *arity_body_fm′*:
assumes
φ∈formula α∈M env∈list(*M*) *arity*(*φ*) ≤ *2 #+ length*(*env*)
shows
arity(*body_fm′*(*φ*,*env*))≤*5 #+ length*(*env*)
unfolding *body_fm′_def*
using *assms new_fm_defs nat_simp_union arity_prebody_fm pred_le arity_renpbdy*[*of prebody_fm*(*φ*,*env*)]
by *auto*

lemma *sats_body_fm′*:
assumes
∃ *t p*. *x*=⟨*t*,*p*⟩ *x∈M* [*α*,*P*,*leq*,*one*,*p*,*ϱ*] @ *nenv ∈list*(*M*) *φ∈formula arity*(*φ*) ≤ *2 #+ length*(*nenv*)
shows
sats(*M*,*body_fm′*(*φ*,*nenv*),[*x*,*α*,*P*,*leq*,*one*] @ *nenv*) ⟷
sats(*M*,*renpbdy*(*prebody_fm*(*φ*,*nenv*),*nenv*),[*fst*(*x*),*snd*(*x*),*x*,*α*,*P*,*leq*,*one*] @ *nenv*)
using *assms fst_snd_closed*[*OF* ‹*x∈M*›] **unfolding** *body_fm′_def*
by (*auto*)

definition
body_fm :: [*i*,*i*]⇒*i* **where**
body_fm(*φ*,*env*) ≡ *renbody*(*body_fm′*(*φ*,*env*),*env*)

lemma *body_fm_type* [*TC*]: *env∈list*(*M*) ⟹ *φ∈formula* ⟹ *body_fm*(*φ*,*env*)*∈formula*
unfolding *body_fm_def* **by** *simp*

lemma *sats_body_fm*:
assumes
∃ *t p*. *x*=⟨*t*,*p*⟩ [*α*,*x*,*m*,*P*,*leq*,*one*] @ *nenv ∈list*(*M*)
φ∈formula arity(*φ*) ≤ *2 #+ length*(*nenv*)

shows
sats(M,*body_fm*(φ,*nenv*),[α,x,m,P,*leq*,*one*] @ *nenv*) $\longleftrightarrow$
sats(M,*renpbdy*(*prebody_fm*(φ,*nenv*),*nenv*),[*fst*(x),*snd*(x),x,α,P,*leq*,*one*] @ *nenv*)
using *assms sats_body_fm' sats_renbody*[*OF _ assms*(*2*), *symmetric*] *arity_body_fm'*
unfolding *body_fm_def*
by *auto*

lemma *sats_renpbdy_prebody_fm*:
assumes
$\exists\, t\ p.\ x{=}\langle t,p\rangle\ x{\in}M$ [α,m,P,*leq*,*one*] @ *nenv* $\in$*list*(M)
$\varphi\in$*formula arity*(φ) $\le$ *2* #+ *length*(*nenv*)
shows
sats(M,*renpbdy*(*prebody_fm*(φ,*nenv*),*nenv*),[*fst*(x),*snd*(x),x,α,P,*leq*,*one*] @ *nenv*)
$\longleftrightarrow$
sats(M,*prebody_fm*(φ,*nenv*),[*fst*(x),*snd*(x),α,P,*leq*,*one*] @ *nenv*)
using *assms fst_snd_closed*[*OF* ‹$x{\in}M$›]
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,α,m,P,*leq*,*one*] @ *nenv* $\in$*list*(M)
$\varphi\in$*formula arity*(φ) $\le$ *2* #+ *length*(*nenv*)
shows
sats(M,*body_fm*(φ,*nenv*),[α,x,m,P,*leq*,*one*] @ *nenv*) $\longleftrightarrow$
($\exists\tau{\in}M.\ \exists\ V{\in}M.$ *is_Vset*($\lambda a.$ (##M)(a),α,V) $\wedge\ \tau\in V\ \wedge$ (*snd*(x) $\Vdash$ φ ([*fst*(x),τ]@*nenv*)))
using *assms sats_body_fm*[*of x* α m *nenv*] *sats_renpbdy_prebody_fm*[*of x* α]
sats_prebody_fm[*of snd*(x) *fst*(x)] *fst_snd_closed*[*OF* ‹$x{\in}M$›]
by (*simp*, *simp flip*: *setclass_iff*,*simp*)

lemma *Replace_sats_in_MG*:
assumes
$c{\in}M[G]$ *env* $\in$ *list*($M[G]$)
φ $\in$ *formula arity*(φ) $\le$ *2* #+ *length*(*env*)
univalent(##$M[G]$, c, $\lambda x\ v.$ ($M[G]$, [x,v]@*env* $\models$ φ))
shows
{$v.\ x{\in}c,\ v{\in}M[G]\ \wedge$ ($M[G]$, [x,v]@*env* $\models$ φ)} $\in$ $M[G]$
proof -
let *?R* = $\lambda\ x\ v\ .\ v{\in}M[G]\ \wedge$ ($M[G]$, [x,v]@*env* $\models$ φ)
from ‹$c{\in}M[G]$›
obtain π' **where** *val*(G, π') = c π' $\in$ M
using *GenExt_def* **by** *auto*
then
have *domain*(π')$\times P{\in}M$ (**is** *?*$\pi{\in}M$)
using *cartprod_closed P_in_M domain_closed* **by** *simp*
from ‹*val*(G, π') = c›
have $c \subseteq$ *val*(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*(*G*),*nenv*)
using *map_val* **by** *auto*
then
have *length*(*nenv*) = *length*(*env*) **by** *simp*
define *f* **where** *f*(*ϱp*) ≡ μ α. α∈*M* ∧ (∃τ∈*M*. τ ∈ *Vset*(α) ∧
(*snd*(*ϱp*) ⊩ φ ([*fst*(*ϱp*),τ] @ *nenv*))) (**is** _ ≡ μ α. *?P*(*ϱp*,α)) **for** *ϱp*
have *f*(*ϱp*) = (μ α. α∈*M* ∧ (∃τ∈*M*. ∃ *V*∈*M*. *is_Vset*(##*M*,α,*V*) ∧ τ∈*V* ∧
(*snd*(*ϱp*) ⊩ φ ([*fst*(*ϱp*),τ] @ *nenv*)))) (**is** _ = (μ α. α∈*M* ∧ *?Q*(*ϱp*,α))) **for** *ϱp*
unfolding *f_def* **using** *Vset_abs Vset_closed Ord_Least_cong*[*of ?P*(*ϱp*) λ α. α∈*M* ∧ *?Q*(*ϱp*,α)]
by (*simp*, *simp del*:*setclass_iff*)
moreover
have *f*(*ϱp*) ∈ *M* **for** *ϱp*
unfolding *f_def* **using** *Least_closed*[*of ?P*(*ϱp*)] **by** *simp*
ultimately
have *1*:*least*(##*M*,λα. *?Q*(*ϱp*,α),*f*(*ϱp*)) **for** *ϱp*
using *least_abs*[*of* λα. α∈*M* ∧ *?Q*(*ϱp*,α) *f*(*ϱp*)] *least_conj*
by (*simp flip*: *setclass_iff*)
have *Ord*(*f*(*ϱp*)) **for** *ϱp* **unfolding** *f_def* **by** *simp*
define *QQ* **where** *QQ*≡*?Q*
from *1*
have *least*(##*M*,λα. *QQ*(*ϱp*,α),*f*(*ϱp*)) **for** *ϱp*
unfolding *QQ_def* **.**
from ‹*arity*(φ) ≤ _› ‹*length*(*nenv*) = _›
have *arity*(φ) ≤ *2* #+ *length*(*nenv*)
by *simp*
moreover
note *assms* ‹*nenv*∈*list*(*M*)› ‹*?π*∈*M*›
moreover
have *ϱp*∈*?π* ⟹ ∃ *t p*. *ϱp*=⟨*t*,*p*⟩ **for** *ϱp*
by *auto*
ultimately
have *body*:*M* , [α,*ϱp*,*m*,*P*,*leq*,*one*] @ *nenv* ⊨ *body_fm*(φ,*nenv*) ⟷ *?Q*(*ϱp*,α)
if *ϱp*∈*?π* *ϱp*∈*M* *m*∈*M* α∈*M* **for** α *ϱp m*
using *that P_in_M leq_in_M one_in_M body_lemma*[*of ϱp* α *m nenv* φ] **by** *simp*
let *?f_fm*=*least_fm*(*body_fm*(φ,*nenv*),*1*)
{
fix *ϱp m*
assume *asm*: *ϱp*∈*M* *ϱp*∈*?π* *m*∈*M*
note *inM* = *this P_in_M leq_in_M one_in_M* ‹*nenv*∈*list*(*M*)›
with *body*
have *body'*:⋀α. α ∈ *M* ⟹ (∃τ∈*M*. ∃ *V*∈*M*. *is_Vset*(λ*a*. (##*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 new_fm_defs 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 $\{x{\in}Vset(?sup).\ x \in M\} \times \{one\} \in M$ (**is** *?big_name* $\in M$)
using *cartprod_closed* **by** *simp*
then
have $val(G,?big\_name) \in M[G]$
by (*blast intro*:*GenExtI*)
{
fix $v\ x$
assume $x{\in}c$
moreover
note $\langle val(G,\pi')=c\rangle\ \langle\pi'{\in}M\rangle$
moreover
from *calculation*
obtain $\varrho\ p$ **where** $\langle\varrho,p\rangle{\in}\pi'\ \ val(G,\varrho) = x\ p{\in}G\ \varrho{\in}M$
using *elem_of_val_pair'*[*of* $\pi'\ x\ G$] **by** *blast*
moreover
assume $v{\in}M[G]$
then
obtain σ **where** $val(G,\sigma) = v\ \sigma{\in}M$
using *GenExtD* **by** *auto*
moreover
assume $sats(M[G],\ \varphi,\ [x,v]\ @\ env)$
moreover
note $\langle\varphi{\in}\_\rangle\ \langle nenv{\in}\_\rangle\ \langle env = \_\rangle\ \langle arity(\varphi){\leq}\ 2\ \#{+}\ length(env)\rangle$
ultimately
obtain q **where** $q{\in}G\ q \Vdash \varphi\ ([\varrho,\sigma]@nenv)$
using *truth_lemma*[*OF* $\langle\varphi{\in}\_\rangle$ *generic*, *symmetric*, *of* $[\varrho,\sigma]\ @\ nenv$]
by *auto*
with $\langle\langle\varrho,p\rangle{\in}\pi'\rangle\ \langle\langle\varrho,q\rangle{\in}?\pi \Longrightarrow f(\langle\varrho,q\rangle){\in}Y\rangle$
have $f(\langle\varrho,q\rangle){\in}Y$
using *generic* **unfolding** *M_generic_def filter_def* **by** *blast*
let $?\alpha{=}succ(rank(\sigma))$
note $\langle\sigma{\in}M\rangle$
moreover from *this*
have $?\alpha \in M$
using *rank_closed cons_closed* **by** (*simp flip*: *setclass_iff*)
moreover
have $\sigma \in Vset(?\alpha)$
using *Vset_Ord_rank_iff* **by** *auto*
moreover
note $\langle q \Vdash \varphi\ ([\varrho,\sigma]\ @\ nenv)\rangle$
ultimately
have $?P(\langle\varrho,q\rangle,?\alpha)$ **by** (*auto simp del*: *Vset_rank_iff*)
moreover
have $(\mu\ \alpha.\ ?P(\langle\varrho,q\rangle,\alpha)) = f(\langle\varrho,q\rangle)$
unfolding *f_def* **by** *simp*
ultimately
obtain τ **where** $\tau{\in}M\ \tau \in Vset(f(\langle\varrho,q\rangle))\ q \Vdash \varphi\ ([\varrho,\tau]\ @\ nenv)$

```
    using LeastI[of λ α. ?P(⟨ϱ,q⟩,α) ?α] by auto
  with ‹q∈G› ‹ϱ∈M› ‹nenv∈_› ‹arity(φ)≤ 2 #+ length(nenv)›
  have M[G], map(val(G),[ϱ,τ] @ nenv) ⊨ φ
    using truth_lemma[OF ‹φ∈_› generic, of [ϱ,τ] @ nenv] by auto
  moreover from ‹x∈c› ‹c∈M[G]›
  have x∈M[G] using transitivity_MG by simp
  moreover
  note ‹M[G],[x,v] @ env⊨ φ› ‹env = map(val(G),nenv)› ‹τ∈M› ‹val(G,ϱ)=x›
    ‹univalent(##M[G],_,_)› ‹x∈c› ‹v∈M[G]›
  ultimately
  have v=val(G,τ)
    using GenExtI[of τ G] unfolding univalent_def by (auto)
  from ‹τ ∈ Vset(f(⟨ϱ,q⟩))› ‹Ord(f(_))›  ‹f(⟨ϱ,q⟩)∈Y›
  have τ ∈ Vset(?sup)
    using Vset_Ord_rank_iff lt_Union_iff[of _ rank(τ)] by auto
  with ‹τ∈M›
  have val(G,τ) ∈ val(G,?big_name)
    using domain_of_prod[of one {one} {x∈Vset(?sup). x ∈ M} ] def_val[of G
?big_name]
      one_in_G[OF generic] one_in_P  by (auto simp del: Vset_rank_iff)
  with ‹v=val(G,τ)›
  have v ∈ val(G,{x∈Vset(?sup). x ∈ M} × {one})
    by simp
 }
 then
 have {v. x∈c, ?R(x,v)} ⊆ val(G,?big_name) (is ?repl⊆?big)
   by blast
 with ‹?big_name∈M›
 have ?repl = {v∈?big. ∃x∈c. sats(M[G], φ, [x,v] @ env )} (is _ = ?rhs)
 proof(intro equalityI subsetI)
   fix v
   assume v∈?repl
   with ‹?repl⊆?big›
   obtain x where x∈c M[G], [x, v] @ env ⊨ φ v∈?big
     using subsetD by auto
   with ‹univalent(##M[G],_,_)› ‹c∈M[G]›
   show v ∈ ?rhs
     unfolding univalent_def
    using transitivity_MG ReplaceI[of λ x v. ∃x∈c. M[G], [x, v] @ env ⊨ φ] by
blast
 next
   fix v
   assume v∈?rhs
   then
   obtain x where
     v∈val(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** $(\#\#M[G])(x)$ $x{\in}A$ $b{\in}M[G]$ $\wedge$ $?R(x,b)$
by *blast*
moreover from *this 1* ‹$(\#\#M[G])(b)$›
have $x{\in}M[G]$ $z{\in}M[G]$ $\wedge$ $?R(x,z) \Longrightarrow b = z$ **for** z
by *auto*
ultimately
show *?thesis*
using *ReplaceI*[*of* λ x y. $y{\in}M[G]$ $\wedge$ $?R(x,y)$] **by** *auto*
qed
qed
then
have $\forall\, b[\#\#M[G]].\ b \in ?Y \longleftrightarrow (\exists\, x[\#\#M[G]].\ x \in A \wedge ?R(x,b))$
by *simp*
with ‹$(\#\#M[G])(?Y)$›
have $(\exists\, Y[\#\#M[G]].\ \forall\, b[\#\#M[G]].\ b \in Y \longleftrightarrow (\exists\, x[\#\#M[G]].\ x \in A \wedge ?R(x,b)))$
by *auto*
}
then show *?thesis* **unfolding** *strong_replacement_def univalent_def*
by *auto*
qed

end

end

28 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: $I{\in}M$ $0 \in I$ $\forall\, y{\in}M.\ y \in I \longrightarrow succ(y) \in I$
unfolding *infinity_ax_def* **by** *auto*
then
have $check(I) \in M$
using *check_in_M* **by** *simp*
then
have $I{\in}\ M[G]$

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

29 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
begin

definition
induced_surj :: *i*⇒*i*⇒*i*⇒*i* **where**
induced_surj(*f*,*a*,*e*) ≡ *f*-‘‘(*range*(*f*)-*a*)×{*e*} ∪ *restrict*(*f*,*f*-‘‘*a*)

lemma *domain_induced_surj*: *domain*(*induced_surj*(*f*,*a*,*e*)) = *domain*(*f*)
unfolding *induced_surj_def* **using** *domain_restrict domain_of_prod* **by** *auto*

lemma *range_restrict_vimage*:
assumes *function*(*f*)
shows *range*(*restrict*(*f*,*f*-‘‘*a*)) ⊆ *a*
proof
from *assms*
have *function*(*restrict*(*f*,*f*-‘‘*a*))
using *function_restrictI* **by** *simp*
fix *y*
assume *y* ∈ *range*(*restrict*(*f*,*f*-‘‘*a*))
then
obtain *x* **where** ⟨*x*,*y*⟩ ∈ *restrict*(*f*,*f*-‘‘*a*) *x* ∈ *f*-‘‘*a* *x*∈*domain*(*f*)
using *domain_restrict domainI*[*of* _ _ *restrict*(*f*,*f*-‘‘*a*)] **by** *auto*
moreover
note ‹*function*(*restrict*(*f*,*f*-‘‘*a*))›
ultimately
have *y* = *restrict*(*f*,*f*-‘‘*a*)‘*x*

```
    using function_apply_equality by blast
  also from ‹x ∈ f-‘‘a› 
  have restrict(f,f-‘‘a)‘x = f‘x
    by simp
  finally
  have y=f‘x .
  moreover from assms ‹x∈domain(f)›
  have ⟨x,f‘x⟩ ∈ f
    using function_apply_Pair by auto
  moreover
  note assms ‹x ∈ f-‘‘a›
  ultimately
  show y∈a
    using function_image_vimage[of f a] by auto
qed

lemma induced_surj_type:
  assumes
    function(f)
  shows
    induced_surj(f,a,e): domain(f) → {e} ∪ a
    and
    x ∈ f-‘‘a ⟹ induced_surj(f,a,e)‘x = f‘x
proof -
  let ?f1=f-‘‘(range(f)-a) × {e} and ?f2=restrict(f, f-‘‘a)
  have domain(?f2) = domain(f) ∩ f-‘‘a
    using domain_restrict by simp
  moreover from assms
  have 1: domain(?f1) = f-‘‘(range(f))-f-‘‘a
    using domain_of_prod function_vimage_Diff by simp
  ultimately
  have domain(?f1) ∩ domain(?f2) = 0
    by auto
  moreover
  have function(?f1) relation(?f1) range(?f1) ⊆ {e}
    unfolding function_def relation_def range_def by auto
  moreover from this and assms
  have ?f1: domain(?f1) → range(?f1)
    using function_imp_Pi by simp
  moreover from assms
  have ?f2: domain(?f2) → range(?f2)
    using function_imp_Pi[of restrict(f, f -‘‘ a)] function_restrictI by simp
  moreover from assms
  have range(?f2) ⊆ a
    using range_restrict_vimage by simp
  ultimately
  have induced_surj(f,a,e): domain(?f1) ∪ domain(?f2) → {e} ∪ a
   unfolding induced_surj_def using fun_is_function fun_disjoint_Un fun_weaken_type
by simp
```

moreover
have *domain(?f1)* ∪ *domain(?f2)* = *domain(f)*
using *domain_restrict domain_of_prod* **by** *auto*
ultimately
show *induced_surj(f,a,e)*: *domain(f)* → {*e*} ∪ *a*
by *simp*
assume *x* ∈ *f-‘‘a*
then
have *?f2‘x* = *f‘x*
using *restrict* **by** *simp*
moreover from ‹*x* ∈ *f-‘‘a*› **and** *1*
have *x* ∉ *domain(?f1)*
by *simp*
ultimately
show *induced_surj(f,a,e)‘x* = *f‘x*
unfolding *induced_surj_def* **using** *fun_disjoint_apply2*[*of x ?f1 ?f2*] **by** *simp*
qed

lemma *induced_surj_is_surj* :
assumes
e∈*a function(f) domain(f)* = *α* ⋀*y*. *y* ∈ *a* ⟹ ∃ *x*∈*α*. *f ‘ x* = *y*
shows
induced_surj(f,a,e) ∈ *surj(α,a)*
unfolding *surj_def*
proof (*intro CollectI ballI*)
from *assms*
show *induced_surj(f,a,e)*: *α* → *a*
using *induced_surj_type*[*of f a e*] *cons_eq cons_absorb* **by** *simp*
fix *y*
assume *y* ∈ *a*
with *assms*
have ∃ *x*∈*α*. *f ‘ x* = *y*
by *simp*
then
obtain *x* **where** *x*∈*α f ‘ x* = *y* **by** *auto*
with ‹*y*∈*a*› *assms*
have *x*∈*f-‘‘a*
using *vimage_iff function_apply_Pair*[*of f x*] **by** *auto*
with ‹*f ‘ x* = *y*› *assms*
have *induced_surj*(*f*, *a*, *e*) *‘ x* = *y*
using *induced_surj_type* **by** *simp*
with ‹*x*∈*α*› **show**
∃ *x*∈*α*. *induced_surj*(*f*, *a*, *e*) *‘ x* = *y* **by** *auto*
qed

context *G_generic*
begin

definition

upair_name :: $i \Rightarrow i \Rightarrow i$ **where**
upair_name(τ,ϱ) $\equiv \{\langle\tau,one\rangle,\langle\varrho,one\rangle\}$

definition
is_upair_name :: $[i,i,i] \Rightarrow o$ **where**
is_upair_name(x,y,z) $\equiv \exists\, xo{\in}M.\ \exists\, yo{\in}M.\ pair(\#\#M,x,one,xo) \wedge pair(\#\#M,y,one,yo)$
$\wedge$
$upair(\#\#M,xo,yo,z)$

lemma *upair_name_abs* :
assumes $x{\in}M$ $y{\in}M$ $z{\in}M$
shows *is_upair_name*(x,y,z) $\longleftrightarrow$ z = *upair_name*(x,y)
unfolding *is_upair_name_def upair_name_def* **using** *assms one_in_M pair_in_M_iff*
by *simp*

lemma *upair_name_closed* :
$[\![\ x{\in}M;\ y{\in}M\ ]\!] \Longrightarrow$ *upair_name*(x,y)$\in M$
unfolding *upair_name_def* **using** *upair_in_M_iff pair_in_M_iff one_in_M*
by *simp*

definition
upair_name_fm :: $[i,i,i,i] \Rightarrow i$ **where**
upair_name_fm(x,y,o,z) $\equiv$ *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*] :
$[\![\ s{\in}nat;x{\in}nat;y{\in}nat;o{\in}nat\ ]\!] \Longrightarrow$ *upair_name_fm*(s,x,y,o)$\in$*formula*
unfolding *upair_name_fm_def* **by** *simp*

lemma *sats_upair_name_fm* :
assumes $x{\in}nat$ $y{\in}nat$ $z{\in}nat$ $o{\in}nat$ $env{\in}list(M)$ $nth(o,env){=}one$
shows
sats(M,*upair_name_fm*(x,y,o,z),env) $\longleftrightarrow$ *is_upair_name*(*nth*(x,env),*nth*(y,env),*nth*(z,env))
unfolding *upair_name_fm_def is_upair_name_def* **using** *assms* **by** *simp*

definition
opair_name :: $i \Rightarrow i \Rightarrow i$ **where**
opair_name(τ,ϱ) $\equiv$ *upair_name*(*upair_name*(τ,τ),*upair_name*(τ,ϱ))

definition
is_opair_name :: $[i,i,i] \Rightarrow o$ **where**
is_opair_name(x,y,z) $\equiv \exists\, upxx{\in}M.\ \exists\, upxy{\in}M.\ is\_upair\_name(x,x,upxx) \wedge is\_upair\_name(x,y,upxy)$
$\wedge$ *is_upair_name*($upxx$,$upxy$,z)

lemma *opair_name_abs* :
assumes $x{\in}M$ $y{\in}M$ $z{\in}M$
shows *is_opair_name*(x,y,z) $\longleftrightarrow$ 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*(*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 *simp*

lemma *val_upair_name* : *val*(*G*,*upair_name*(τ,ϱ)) = {*val*(*G*,τ),*val*(*G*,ϱ)}
 unfolding *upair_name_def* **using** *val_Upair generic one_in_G one_in_P* **by**
simp

lemma *val_opair_name* : *val*(*G*,*opair_name*(τ,ϱ)) = ⟨*val*(*G*,τ),*val*(*G*,ϱ)⟩
 unfolding *opair_name_def Pair_def* **using** *val_upair_name* **by** *simp*

lemma *val_RepFun_one*: *val*(*G*,{⟨*f*(*x*),*one*⟩ . *x*∈*a*}) = {*val*(*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*(*G*,{⟨*f*(*x*),*one*⟩ . *x* ∈ *a*}) = *val*(*G*,{*t* ∈ *?A* × *P* . *?Q*(*t*)})
  by *simp*
 also
 have ... = {*val*(*G*,*t*) .. *t* ∈ *?A* , ∃ *p*∈*P*∩*G* . *?Q*(⟨*t*,*p*⟩)}
  using *val_of_name_alt* **by** *simp*
 also
 have ... = {*val*(*G*,*t*) . *t* ∈ *?A* }
  using ‹*one*∈*P*∩*G*› **by** *force*
 also
 have ... = {*val*(*G*,*f*(*x*)) . *x* ∈ *a*}
  by *auto*

finally show *?thesis* **by** *simp*
qed

29.1 $M[G]$ is a transitive model of ZF

interpretation *mgzf*: *M_ZF_trans M[G]*
using *Transset_MG generic pairing_in_MG Union_MG*
extensionality_in_MG power_in_MG foundation_in_MG
strong_replacement_in_MG separation_in_MG infinity_in_MG
by *unfold_locales simp_all*

definition
is_opname_check :: $[i,i,i] \Rightarrow o$ **where**
$is\_opname\_check(s,x,y) \equiv \exists\, chx{\in}M.\ \exists\, sx{\in}M.\ is\_check(x,chx) \wedge fun\_apply(\#\#M,s,x,sx) \wedge$
$is\_opair\_name(chx,sx,y)$

definition
opname_check_fm :: $[i,i,i,i] \Rightarrow i$ **where**
$opname\_check\_fm(s,x,y,o) \equiv Exists(Exists(And(check\_fm(2\#{+}x,2\#{+}o,1),$
$And(fun\_apply\_fm(2\#{+}s,2\#{+}x,0),opair\_name\_fm(1,0,2\#{+}o,2\#{+}y)))))$

lemma *opname_check_fm_type*[*TC*] :
$[\![\ s{\in}nat;x{\in}nat;y{\in}nat;o{\in}nat]\!] \Longrightarrow opname\_check\_fm(s,x,y,o){\in}formula$
unfolding *opname_check_fm_def* **by** *simp*

lemma *sats_opname_check_fm*:
assumes $x{\in}nat\ y{\in}nat\ z{\in}nat\ o{\in}nat\ env{\in}list(M)\ nth(o,env){=}one$
$y{<}length(env)$
shows
$sats(M,opname\_check\_fm(x,y,z,o),env) \longleftrightarrow is\_opname\_check(nth(x,env),nth(y,env),nth(z,env))$
unfolding *opname_check_fm_def is_opname_check_def*
using *assms sats_check_fm sats_opair_name_fm one_in_M* **by** *simp*

lemma *opname_check_abs* :
assumes $s{\in}M\ x{\in}M\ y{\in}M$
shows $is\_opname\_check(s,x,y) \longleftrightarrow y = opair\_name(check(x),s`x)$
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
$A{\in}M\ f{\in}M$
shows
$\{opair\_name(check(x),f`x).\ x{\in}A\}{\in}M$
proof -

```
  have arity(opname_check_fm(3,0,1,2))= 4
    unfolding opname_check_fm_def opair_name_fm_def upair_name_fm_def
          check_fm_def rcheck_fm_def trans_closure_fm_def is_eclose_fm_def
mem_eclose_fm_def
      is_Hcheck_fm_def Replace_fm_def PHcheck_fm_def finite_ordinal_fm_def
is_iterates_fm_def
         is_wfrec_fm_def is_recfun_fm_def restriction_fm_def pre_image_fm_def
eclose_n_fm_def
        is_nat_case_fm_def quasinat_fm_def Memrel_fm_def singleton_fm_def
fm_defs iterates_MH_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)
                 λx. 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(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** $\{one\} \in M$ **using** *singletonM* **by** *simp*
define f **where**
  $f \equiv val(G,?f\_dot)$
from ‹$\{one\}\in M$› ‹$?g\in M$› ‹$?f\_dot = ?g\times\{one\}$›
have *?f_dot∈M*
  using *cartprod_closed* **by** *simp*
then
have $f \in M[G]$
  unfolding *f_def* **by** (*blast intro*:*GenExtI*)
have $f = \{val(G,opair\_name(check(\beta),s`\beta)) \ .\ \beta\in\alpha\}$
  unfolding *f_def* **using** *val_RepFun_one* **by** *simp*
also
have $... = \{\langle\beta,val(G,s`\beta)\rangle \ .\ \beta\in\alpha\}$
  using *val_opair_name valcheck generic one_in_G one_in_P* **by** *simp*
finally
have $f = \{\langle\beta,val(G,s`\beta)\rangle \ .\ \beta\in\alpha\}$ **.**
then
have *1*: $domain(f) = \alpha$ *function*(f)
  unfolding *function_def* **by** *auto*
have *2*: $y \in a \Longrightarrow \exists\, x\in\alpha.\ f\ `\ x = y$ **for** y
proof -
  fix y
  assume
    $y \in a$
  with ‹$val(G,\tau) = a$›
  obtain σ **where** $\sigma\in domain(\tau)$ $val(G,\sigma) = y$
    using *elem_of_val*[*of y _ τ*] **by** *blast*
  with ‹$s\in surj(\alpha,domain(\tau))$›
  obtain β **where** $\beta\in\alpha$ $s`\beta = \sigma$
    unfolding *surj_def* **by** *auto*
  with ‹$val(G,\sigma) = y$›
  have $val(G,s`\beta) = y$
    by *simp*
  with ‹$f = \{\langle\beta,val(G,s`\beta)\rangle \ .\ \beta\in\alpha\}$› ‹$\beta\in\alpha$›
  have $\langle\beta,y\rangle\in f$
    by *auto*
  with ‹*function*(f)›
  have $f`\beta = y$
    using *function_apply_equality* **by** *simp*
  with ‹$\beta\in\alpha$› **show**
    $\exists\,\beta\in\alpha.\ f\ `\ \beta = y$
    by *auto*
qed
then
have $\exists\,\alpha\in(M[G]).\ \exists\, f'\in(M[G]).\ Ord(\alpha) \wedge f' \in surj(\alpha,a)$
proof (*cases a=0*)
  case *True*
  then
  have $0\in surj(0,a)$

```
      unfolding surj_def by simp
    then
    show ?thesis 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
```

30 Ordinals in generic extensions

```
theory Ordinals_In_MG
 imports
   Forcing_Theorems Relative_Univ
begin

context G_generic
begin

lemma rank_val: rank(val(G,x)) ≤ rank(x) (is ?Q(x))
proof (induct rule:ed_induction[of ?Q])
 case (1 x)
 have val(G,x) = {val(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(G,x)) = (⋃u∈{t∈domain(x). ∃p∈P . ⟨t,p⟩∈x ∧ p ∈ G }.
succ(rank(val(G,u))))
   using rank[of val(G,x)] by simp
 moreover
 have succ(rank(val(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 ($\bigcup u{\in}\{t{\in}domain(x).\ \exists p{\in}P\ .\ \langle t,p\rangle{\in}x \wedge p \in G\ \}.\ succ(rank(val(G,u)))) \leq rank(x)$
by (*rule_tac UN_least_le*) (*auto*)
ultimately
show *?case* **by** *simp*
qed

lemma *Ord_MG_iff*:
assumes $Ord(\alpha)$
shows $\alpha \in M \longleftrightarrow \alpha \in M[G]$
proof
show $\alpha \in M \Longrightarrow \alpha \in M[G]$
using *generic*[*THEN one_in_G*, *THEN M_subset_MG*] **..**
next
assume $\alpha \in M[G]$
then
obtain x **where** $x{\in}M\ val(G,x) = \alpha$
using *GenExtD* **by** *auto*
then
have $rank(\alpha) \leq rank(x)$
using *rank_val* **by** *blast*
with *assms*
have $\alpha \leq rank(x)$
using *rank_of_Ord* **by** *simp*
then
have $\alpha \in succ(rank(x))$ **using** *ltD* **by** *simp*
with ‹$x{\in}M$›
show $\alpha \in M$
using *cons_closed transitivity*[*of* α $succ(rank(x))$]
rank_closed **unfolding** *succ_def* **by** *simp*
qed

end

end

31 Separative notions and proper extensions

theory *Proper_Extension*
imports
Names

begin

The key ingredient to obtain a proper extension is to have a *separative preorder*:

locale *separative_notion* = *forcing_notion* +

assumes *separative*: $p{\in}P \Longrightarrow \exists\, q{\in}P.\ \exists\, r{\in}P.\ q \preceq p \wedge r \preceq p \wedge q \perp r$
begin

For separative preorders, the complement of every filter is dense. Hence an M-generic filter can't belong to the ground model.

lemma *filter_complement_dense*:
assumes *filter*(G) **shows** *dense*(P - G)
proof
fix p
assume $p{\in}P$
show $\exists\, d{\in}P$ - $G.\ d \preceq p$
proof (*cases* $p{\in}G$)
case *True*
note ‹$p{\in}P$› *assms*
moreover
obtain $q\ r$ **where** $q \preceq p\ r \preceq p\ q \perp r\ q{\in}P\ r{\in}P$
using *separative*[*OF* ‹$p{\in}P$›]
by *force*
with ‹*filter*(G)›
obtain s **where** $s \preceq p\ s \notin G\ s \in P$
using *filter_imp_compat*[*of* $G\ q\ r$]
by *auto*
then
show *?thesis* **by** *blast*
next
case *False*
with ‹$p{\in}P$›
show *?thesis* **using** *leq_reflI* **unfolding** *Diff_def* **by** *auto*
qed
qed

end

locale *ctm_separative* = *forcing_data* + *separative_notion*
begin

lemma *generic_not_in_M*: **assumes** *M_generic*(G) **shows** $G \notin M$
proof
assume $G{\in}M$
then
have P - $G \in M$
using *P_in_M Diff_closed* **by** *simp*
moreover
have $\neg(\exists\, q{\in}G.\ q \in P$ - $G)\ (P$ - $G) \subseteq P$
unfolding *Diff_def* **by** *auto*
moreover
note *assms*
ultimately
show *False*

using *filter_complement_dense*[*of G*] *M_generic_denseD*[*of G P-G*]
M_generic_def **by** *simp* — need to put generic ==> filter in claset
qed

theorem *proper_extension*: **assumes** *M_generic*(*G*) **shows** $M \neq M[G]$
using *assms G_in_Gen_Ext*[*of G*] *one_in_G*[*of G*] *generic_not_in_M*
by *force*

end

end

32 A poset of succession

theory *Succession_Poset*
imports
Arities Proper_Extension Synthetic_Definition
Names
begin

32.1 The set of finite binary sequences

We implement the poset for adding one Cohen real, the set $2^{<\omega}$ of of finite binary sequences.

definition
seqspace :: $i \Rightarrow i$ ($\langle\_^\frown{<}\omega\rangle$ [100]100) **where**
$seqspace(B) \equiv \bigcup n{\in}nat.\ (n{\to}B)$

lemma *seqspaceI*[*intro*]: $n{\in}nat \Longrightarrow f{:}n{\to}B \Longrightarrow f{\in}seqspace(B)$
unfolding *seqspace_def* **by** *blast*

lemma *seqspaceD*[*dest*]: $f{\in}seqspace(B) \Longrightarrow \exists\, n{\in}nat.\ f{:}n{\to}B$
unfolding *seqspace_def* **by** *blast*

lemma *seqspace_type*:
$f \in B^\frown{<}\omega \Longrightarrow \exists\, n{\in}nat.\ f{:}n{\to}B$
unfolding *seqspace_def* **by** *auto*

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
$(\exists\, om{\in}A.\ omega(\#\#A,om) \wedge n \in om \wedge is\_funspace(\#\#A,\ n,\ B,\ z)) \longleftrightarrow (A,\ env \models (?sqsprp(i,j,h)))$
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: $M(B) \Longrightarrow$ *strong_replacement*(M,$\lambda n\ z.\ n{\in}nat \wedge$ *is_funspace*(M,n,B,z))
begin

lemma *seqspace_closed*:
  $M(B) \Longrightarrow M(B^{\frown}{<}\omega)$
  unfolding *seqspace_def* **using** *seqspace_replacement*[*of B*] *RepFun_closed2*
  by *simp*

end

sublocale *M_ctm* $\subseteq$ *M_seqspace* $\#\#M$
proof (*unfold_locales*, *simp*)
  fix B
  have *arity*(*seqspace_rep_fm*(0,1,2)) $\leq$ 3 *seqspace_rep_fm*(0,1,2)$\in$*formula*
    unfolding *seqspace_rep_fm_def*
   using *arity_pair_fm arity_omega_fm arity_typed_function_fm nat_simp_union*

    by *auto*
  moreover
  assume $B{\in}M$
  ultimately
  have *strong_replacement*($\#\#M$, $\lambda x\ y.\ M$, [x, y, B] $\models$ *seqspace_rep_fm*(0, 1, 2))
    using *replacement_ax*[*of seqspace_rep_fm*(0,1,2)]
    by *simp*
  moreover
  note ‹$B{\in}M$›
  moreover from *this*
  have *univalent*($\#\#M$, A, $\lambda x\ y.\ M$, [x, y, B] $\models$ *seqspace_rep_fm*(0, 1, 2))
    if $A{\in}M$ **for** A
    using *that* **unfolding** *univalent_def seqspace_rep_fm_def*
    by (*auto*, *blast dest*:*transitivity*)
  ultimately
  have *strong_replacement*($\#\#M$, $\lambda n\ z.\ \exists\, om[\#\#M].$ *omega*($\#\#M$,om) $\wedge$ $n \in om \wedge$ *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 ‹$B{\in}M$›
  show *strong_replacement*($\#\#M$, $\lambda n\ z.\ n \in nat \wedge$ *is_funspace*($\#\#M$, n, B, z))
    using *M_nat* **by** *simp*
qed

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

```
  seq_upd(f,a) ≡ λ j ∈ succ(domain(f)) . if j < domain(f) then f‘j else a

lemma seq_upd_succ_type :
  assumes n∈nat f∈n→A a∈A
  shows seq_upd(f,a)∈ succ(n) → A
proof -
  from assms
  have equ: domain(f) = n using domain_of_fun by simp
  {
    fix j
    assume j∈succ(domain(f))
    with equ ‹n∈_›
    have j≤n using ltI by auto
    with ‹n∈_›
    consider (lt) j<n | (eq) j=n using leD by auto
    then
    have (if j < n then f‘j else a) ∈ A
    proof cases
      case lt
      with ‹f∈_›
      show ?thesis using apply_type ltD[OF lt] by simp
    next
      case eq
      with ‹a∈_›
      show ?thesis by auto
    qed
  }
  with equ
  show ?thesis
    unfolding seq_upd_def
    using lam_type[of succ(domain(f))]
    by auto
qed

lemma seq_upd_type :
  assumes f∈A^<ω a∈A
  shows seq_upd(f,a) ∈ A^<ω
proof -
  from ‹f∈_›
  obtain y where y∈nat f∈y→A
    unfolding seqspace_def by blast
  with ‹a∈A›
  have seq_upd(f,a)∈succ(y)→A
    using seq_upd_succ_type by simp
  with ‹y∈_›
  show ?thesis
    unfolding seqspace_def by auto
qed
```

lemma *seq_upd_apply_domain* [*simp*]:
assumes $f{:}n{\to}A$ $n{\in}nat$
shows $seq\_upd(f,a)`n = a$
unfolding *seq_upd_def* **using** *assms domain_of_fun* **by** *auto*

lemma *zero_in_seqspace* :
shows $0 \in A^{\frown}{<}\omega$
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**
$seqlerel(A) \equiv Rrel(\lambda x\ y.\ y \subseteq x, A^{\frown}{<}\omega)$

definition
$seqle :: i$ **where**
$seqle \equiv seqlerel(2)$

lemma *seqleI*[*intro*!]:
$\langle f,g\rangle \in 2^{\frown}{<}\omega{\times}2^{\frown}{<}\omega \Longrightarrow g \subseteq f \implies \langle f,g\rangle \in seqle$
unfolding *seqspace_def seqle_def seqlerel_def Rrel_def*
by *blast*

lemma *seqleD*[*dest*!]:
$z \in seqle \Longrightarrow \exists\, x\ y.\ \langle x,y\rangle \in 2^{\frown}{<}\omega{\times}2^{\frown}{<}\omega \wedge y \subseteq x \wedge z = \langle x,y\rangle$
unfolding *seqle_def seqlerel_def Rrel_def*
by *blast*

lemma *upd_leI* :
assumes $f{\in}2^{\frown}{<}\omega$ $a{\in}2$
shows $\langle seq\_upd(f,a),f\rangle{\in}seqle$ (**is** $\langle ?f,\_\rangle{\in}\_$)
proof
show $\langle ?f,\ f\rangle \in 2^{\frown}{<}\omega \times 2^{\frown}{<}\omega$
using *assms seq_upd_type* **by** *auto*
next
show $f \subseteq seq\_upd(f,a)$
proof
fix x
assume $x \in f$
moreover from $\langle f \in 2^{\frown}{<}\omega\rangle$
obtain n **where** $n{\in}nat\ f : n \to 2$
using *seqspace_type* **by** *blast*
moreover from *calculation*
obtain y **where** $y{\in}n$ $x{=}\langle y,f`y\rangle$ **using** *Pi_memberD*[*of f n* $\lambda\_\ .\ 2$]
by *blast*

```
    moreover from ‹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 forcing_notion 2^<ω seqle 0
  using preorder_on_seqle zero_seqle_max zero_in_seqspace
  by unfold_locales simp_all

abbreviation SEQle :: [i, i] ⇒ o  (infixl ‹⪯s› 50)
  where x ⪯s y ≡ Leq(x,y)

abbreviation SEQIncompatible :: [i, i] ⇒ o  (infixl ‹⊥s› 50)
  where x ⊥s y ≡ Incompatible(x,y)

lemma seqspace_separative:
  assumes f∈2^<ω
  shows seq_upd(f,0) ⊥s seq_upd(f,1) (is ?f ⊥s ?g)
proof
  assume 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*) $\lambda\ A\ z$. *relP*(*A*,*is_rel*,*z*) $\lambda\ z.\exists\, x\ y.\ z = \langle x,\ y\rangle \wedge rel(x,\ y)$]
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*

32.2 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*: $\exists\, G.\ M\_generic(G) \wedge M \neq GenExt(G)$
using *proper_extension generic_filter_existence zero_in_seqspace*
by *force*

end

end

33 The main theorem

theory *Forcing_Main*
imports
Internal_ZFC_Axioms
Choice_Axiom
Ordinals_In_MG
Succession_Poset

begin

33.1 The generic extension is countable

definition
minimum :: $i \Rightarrow i \Rightarrow i$ **where**
$minimum(r,B) \equiv THE\ b.\ b{\in}B \wedge (\forall\, y{\in}B.\ y \neq b \longrightarrow \langle b,\, y\rangle \in r)$

lemma *well_ord_imp_min*:
assumes
$well\_ord(A,r)\ B \subseteq A\ B \neq 0$
shows
$minimum(r,B) \in B$
proof -
from ‹$well\_ord(A,r)$›
have $wf[A](r)$
using *well_ord_is_wf*[*OF* ‹$well\_ord(A,r)$›] **by** *simp*
with ‹$B{\subseteq}A$›
have $wf[B](r)$
using *Sigma_mono Int_mono wf_subset* **unfolding** *wf_on_def* **by** *simp*
then
have $\forall\ x.\ x \in B \longrightarrow (\exists\, z{\in}B.\ \forall\, y.\ \langle y,\, z\rangle \in r{\cap}B{\times}B \longrightarrow y \notin B)$
unfolding *wf_on_def* **using** *wf_eq_minimal*
by *blast*
with ‹$B{\neq}0$›
obtain z **where**
B: $z{\in}B \wedge (\forall\, y.\ \langle y,z\rangle{\in}r{\cap}B{\times}B \longrightarrow y{\notin}B)$
by *blast*
then
have $z{\in}B \wedge (\forall\, y{\in}B.\ y \neq z \longrightarrow \langle z,\, y\rangle \in r)$
proof -
{
fix y
assume $y{\in}B\ y{\neq}z$
with ‹$well\_ord(A,r)$› B ‹$B{\subseteq}A$›
have $\langle z,y\rangle{\in}r|\langle y,z\rangle{\in}r|y{=}z$
unfolding *well_ord_def tot_ord_def linear_def* **by** *auto*
with B ‹$y{\in}B$› ‹$y{\neq}z$›
have $\langle z,y\rangle{\in}r$
by (*cases*;*auto*)
}

```
    with B
    show ?thesis by blast
  qed
  have v = z if v∈B ∧ (∀ y∈B. y ≠ v ⟶ ⟨v, y⟩ ∈ r) for v
    using that B by auto
  with ‹z∈B ∧ (∀ y∈B. y ≠ z ⟶ ⟨z, y⟩ ∈ r)›
  show ?thesis
    unfolding minimum_def
    using the_equality2[OF ex1I[of λx .x∈B ∧ (∀ y∈B. y ≠ x ⟶ ⟨x, y⟩ ∈ r) z]]
    by auto
qed

lemma well_ord_surj_imp_lepoll:
  assumes well_ord(A,r) h ∈ surj(A,B)
  shows B ≲ A
proof -
  let ?f=λb∈B. minimum(r, {a∈A. h‘a=b})
  have b ∈ B ⟹ minimum(r, {a ∈ A . h ‘ a = b}) ∈ {a∈A. h‘a=b} for b
  proof -
    fix b
    assume b∈B
    with ‹h ∈ surj(A,B)›
    have ∃ a∈A. h‘a=b
      unfolding surj_def by blast
    then
    have {a∈A. h‘a=b} ≠ 0
      by auto
    with assms
    show minimum(r,{a∈A. h‘a=b}) ∈ {a∈A. h‘a=b}
      using well_ord_imp_min 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(1)[OF that(1)] calculation(1)[OF that(2)]
    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})
      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* :
$\exists f.\ f \in surj(nat,M[G])$
proof -
let *?f*=$\lambda n{\in}nat.\ val(G,enum`n)$
have $x \in nat \Longrightarrow val(G,\ enum\ `\ x){\in}\ M[G]$ **for** x
using *GenExtD*[*THEN iffD2*, *of _ G*] *bij_is_fun*[*OF M_countable*] **by** *force*
then
have *?f*: $nat \to M[G]$
using *lam_type*[*of nat* $\lambda n.\ val(G,enum`n)$ $\lambda\_.M[G]$] **by** *simp*
moreover
have $\exists\, n{\in}nat.\ ?f`n = x$ **if** $x{\in}M[G]$ **for** x
using *that GenExtD*[*of _ G*] *bij_is_surj*[*OF M_countable*]
unfolding *surj_def* **by** *auto*
ultimately
show *?thesis*
unfolding *surj_def* **by** *blast*
qed

lemma (**in** *G_generic*) *MG_eqpoll_nat*: $M[G] \approx nat$
proof -
interpret *MG*: *M_ZF_trans* $M[G]$
using *Transset_MG generic pairing_in_MG*
Union_MG extensionality_in_MG power_in_MG
foundation_in_MG strong_replacement_in_MG[*simplified*]
separation_in_MG[*simplified*] *infinity_in_MG*
by *unfold_locales simp_all*
obtain f **where** $f \in surj(nat,M[G])$
using *surj_nat_MG* **by** *blast*
then
have $M[G] \lesssim nat$
using *well_ord_surj_imp_lepoll well_ord_Memrel*[*of nat*]
by *simp*
moreover
have $nat \lesssim M[G]$
using *MG.nat_into_M subset_imp_lepoll* **by** *auto*
ultimately
show *?thesis* **using** *eqpollI*
by *simp*
qed

33.2 The main result

theorem *extensions_of_ctms*:
assumes

```
    M ≈ nat Transset(M) M ⊨ ZF
  shows
    ∃N.
      M ⊆ N ∧ N ≈ nat ∧ Transset(N) ∧ N ⊨ ZF ∧ M≠N ∧
      (∀α. Ord(α) ⟶ (α ∈ M ⟷ α ∈ N)) ∧
      (M, []⊨ AC ⟶ N ⊨ ZFC)
proof -
  from ‹M ≈ nat›
  obtain enum where enum ∈ bij(nat,M)
    using eqpoll_sym unfolding eqpoll_def by blast
  with assms
  interpret M_ctm M enum
    using M_ZF_iff_M_satT
    by intro_locales (simp_all add:M_ctm_axioms_def)
  interpret ctm_separative 2^<ω seqle 0 M enum
  proof (unfold_locales)
    fix f
    let ?q=seq_upd(f,0) and ?r=seq_upd(f,1)
    assume f ∈ 2^<ω
    then
    have ?q ≼s f ∧ ?r ≼s f ∧ ?q ⊥s ?r
      using upd_leI seqspace_separative by auto
    moreover from calculation
    have ?q ∈ 2^<ω  ?r ∈ 2^<ω
      using seq_upd_type[of f 2] by auto
    ultimately
    show ∃q∈2^<ω. ∃r∈2^<ω. q ≼s f ∧ r ≼s f ∧ q ⊥s r
      by (rule_tac bexI)+ — why the heck auto-tools don't solve this?
  next
    show 2^<ω ∈ M using nat_into_M seqspace_closed by simp
  next
    show seqle ∈ M using seqle_in_M .
  qed
  from cohen_extension_is_proper
  obtain G where M_generic(G)
    M ≠ GenExt(G) (is M≠?N)
    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
```

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).