

Formalization of Forcing in Isabelle/ZF

Emmanuel Gunther* Miguel Pagano* Pedro Sánchez Terraf*†

April 10, 2026

Abstract

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

Contents

1	Introduction	4
2	Forcing notions	4
2.1	Basic concepts	4
2.2	Towards Rasiowa-Sikorski Lemma (RSL)	10
3	A pointed version of DC	13
4	The general Rasiowa-Sikorski lemma	16
5	Auxiliary results on arithmetic	18
5.1	Some results in ordinal arithmetic	21
6	Aids to internalize formulas	23
7	Some enhanced theorems on recursion	24
8	Relativization of the cumulative hierarchy	29
8.1	Formula synthesis	30
8.2	Absoluteness results	31
9	Automatic synthesis of formulas	37

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

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

10	Interface between set models and Constructibility	40
10.1	Interface with $M_trivial$	42
10.2	Interface with M_basic	42
10.3	Interface with M_trancl	52
10.4	Interface with M_eclose	56
11	Transitive set models of ZF	69
11.1	$Collects$ in M	71
11.2	A forcing locale and generic filters	73
12	The ZFC axioms, internalized	77
12.1	The Axiom of Separation, internalized	78
12.2	The Axiom of Replacement, internalized	82
13	Renaming of variables in internalized formulas	88
13.1	Renaming of free variables	88
13.2	Renaming of formulas	96
14	Names and generic extensions	101
14.1	The well-founded relation ed	102
14.2	Values and check-names	107
15	Well-founded relation on names	123
16	Arities of internalized formulas	138
17	The definition of $forces$	146
17.1	The relation $frecrel$	146
17.2	Definition of $forces$ for equality and membership	150
17.3	The well-founded relation $forcerec$	154
17.4	frc_at , forcing for atomic formulas	155
17.5	Recursive expression of frc_at	171
17.6	Absoluteness of frc_at	171
17.7	Forcing for general formulas	175
17.7.1	The primitive recursion	178
17.8	Forcing for atomic formulas in context	179
17.9	The arity of $forces$	181
18	The Forcing Theorems	184
18.1	The forcing relation in context	184
18.2	Kunen 2013, Lemma IV.2.37(a)	185
18.3	Kunen 2013, Lemma IV.2.37(a)	185
18.4	Kunen 2013, Lemma IV.2.37(b)	185
18.5	Kunen 2013, Lemma IV.2.38	187
18.6	The relation of forcing and atomic formulas	188

18.7	The relation of forcing and connectives	188
18.8	Kunen 2013, Lemma IV.2.29	189
18.9	Auxiliary results for Lemma IV.2.40(a)	191
18.10	Induction on names	194
18.11	Lemma IV.2.40(a), in full	195
18.12	Lemma IV.2.40(b)	196
18.13	The Strenghtening Lemma	203
18.14	The Density Lemma	204
18.15	The Truth Lemma	206
18.16	The “Definition of forcing”	215
19	Auxiliary renamings for Separation	216
20	The Axiom of Separation in $M[G]$	227
21	The Axiom of Pairing in $M[G]$	236
22	The Axiom of Unions in $M[G]$	237
23	The Powerset Axiom in $M[G]$	240
24	The Axiom of Extensionality in $M[G]$	247
25	The Axiom of Foundation in $M[G]$	247
26	The binder <i>Least</i>	248
26.1	Absoluteness and closure under <i>Least</i>	250
27	The Axiom of Replacement in $M[G]$	251
28	The Axiom of Infinity in $M[G]$	263
29	The Axiom of Choice in $M[G]$	264
29.1	$M[G]$ is a transitive model of ZF	269
30	Ordinals in generic extensions	272
31	Separative notions and proper extensions	273
32	A poset of successions	275
32.1	The set of finite binary sequences	275
32.2	Cohen extension is proper	282
33	The main theorem	283
33.1	The generic extension is countable	283
33.2	The main result	285

1 Introduction

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

Release notes

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

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

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

2 Forcing notions

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

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

2.1 Basic concepts

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

definition $compat_in :: i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow o$ **where**
 $compat_in(A, r, p, q) \equiv \exists d \in A . \langle d, p \rangle \in r \wedge \langle d, q \rangle \in r$

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

lemma *compat_inI* :

$\llbracket d \in A ; \langle d, p \rangle \in r ; \langle d, g \rangle \in r \rrbracket \implies \text{compat_in}(A, r, p, g)$
by (*auto simp add: compat_in_def*)

lemma *refl_compat*:

$\llbracket \text{refl}(A, r) ; \langle p, q \rangle \in r \mid p = q \mid \langle q, p \rangle \in r ; p \in A ; q \in A \rrbracket \implies \text{compat_in}(A, r, p, q)$
by (*auto simp add: refl_def compat_inI*)

lemma *chain_compat*:

$\text{refl}(A, r) \implies \text{linear}(A, r) \implies (\forall p \in A. \forall q \in A. \text{compat_in}(A, r, p, q))$
by (*simp add: refl_compat linear_def*)

lemma *subset_fun_image*: $f: N \rightarrow P \implies f''N \subseteq P$

by (*auto simp add: image_fun_apply_funtype*)

lemma *refl_monot_domain*: $\text{refl}(B, r) \implies A \subseteq B \implies \text{refl}(A, r)$

unfolding *refl_def* **by** *blast*

definition

antichain :: $i \Rightarrow i \Rightarrow i \Rightarrow o$ **where**

$\text{antichain}(P, \text{leq}, A) \equiv A \subseteq P \wedge (\forall p \in A. \forall q \in A. (\neg \text{compat_in}(P, \text{leq}, p, q)))$

definition

ccc :: $i \Rightarrow i \Rightarrow o$ **where**

$\text{ccc}(P, \text{leq}) \equiv \forall A. \text{antichain}(P, \text{leq}, A) \longrightarrow |A| \leq \text{nat}$

locale *forcing_notion* =

fixes *P leq one*

assumes *one_in_P*: $one \in P$

and *leq_preord*: $\text{preorder_on}(P, \text{leq})$

and *one_max*: $\forall p \in P. \langle p, one \rangle \in \text{leq}$

begin

abbreviation *Leq* :: $[i, i] \Rightarrow o$ (**infixl** $\langle \preceq \rangle$ 50)

where $x \preceq y \equiv \langle x, y \rangle \in \text{leq}$

lemma *refl_leq*:

$r \in P \implies r \preceq r$

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

$\text{dense}(D) \equiv \forall p \in P. \exists d \in D. d \preceq p$

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

definition

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

lemma *P_dense*: *dense*(P)
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) $\equiv \text{compat_in}(P, \text{leq}, p, q)$

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

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

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

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

lemma *denseI [intro!]*: $\llbracket \bigwedge p. p \in P \Longrightarrow \exists d \in D. d \preceq p \rrbracket \Longrightarrow \text{dense}(D)$
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*: $p \in P \implies D = D' \implies \text{dense_below}(D, p) \longleftrightarrow \text{dense_below}(D', p)$
by *blast*

lemma *dense_below_cong'*: $p \in P \implies \llbracket \bigwedge x. x \in P \implies Q(x) \longleftrightarrow Q'(x) \rrbracket \implies$
 $\text{dense_below}(\{q \in P. Q(q)\}, p) \longleftrightarrow \text{dense_below}(\{q \in P. Q'(q)\}, p)$
by *blast*

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

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

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

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

definition

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

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

definition

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

lemma *filterD* : $\text{filter}(G) \implies x \in G \implies x \in P$
by (*auto simp add : subsetD filter_def*)

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

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

lemma *low_bound_filter*: — says the compatibility is attained inside G
assumes $\text{filter}(G)$ **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*] : $p \in P \Rightarrow a \in A \Rightarrow a \preceq p \Rightarrow p \in \text{upclosure}(A)$
by (*simp add:upclosure_def, auto*)

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

lemma *upclosureD* [*dest*] :
 $p \in \text{upclosure}(A) \Rightarrow \exists a \in A. (a \preceq p) \wedge p \in P$
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 ⊆ P›]* **by** *auto*

lemma *upclosure_in_P*: $A \subseteq P \Rightarrow \text{upclosure}(A) \subseteq P$
using *subsetI upclosure_def* **by** *simp*

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

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

lemma *closure_compat_filter*:
assumes $A \subseteq P$ ($\forall p \in A. \forall q \in A. \text{compat_in}(A, \text{leq}, p, q)$)
shows *filter*(*upclosure*(A))
unfolding *filter_def*

proof(*auto*)

show *increasing*(*upclosure*(A))
using *assms upclosure_increasing* **by** *simp*

next

let $?UA = \text{upclosure}(A)$
show *compat_in*(*upclosure*(A), *leq*, p , q) **if** $p \in ?UA$ $q \in ?UA$ **for** p q
proof -
from *that*

```

obtain  $a\ b$  where  $1:a\in A\ b\in A\ a\preceq p\ b\preceq q\ p\in P\ q\in P$ 
  using  $upclosureD[OF\ \langle p\in?UA\rangle]\ upclosureD[OF\ \langle q\in?UA\rangle]$  by auto
with  $assms(2)$ 
obtain  $d$  where  $d\in A\ d\preceq a\ d\preceq b$ 
  unfolding  $compat\_in\_def$  by auto
with  $1$ 
have  $2:d\preceq p\ d\preceq q\ d\in?UA$ 
  using  $A\_sub\_upclosure[THEN\ subsetD]\ \langle A\subseteq P\rangle$ 
   $leq\_transD'[of\ A\ d\ a]\ leq\_transD'[of\ A\ d\ b]$  by auto
then
  show  $?thesis$  unfolding  $compat\_in\_def$  by auto
qed
qed

```

```

lemma  $aux\_RS1: f\in N\ \rightarrow\ P\ \Longrightarrow\ n\in N\ \Longrightarrow\ f^n\in upclosure(f\ \text{``}N)$ 
  using  $elem\_upclosure[OF\ subset\_fun\_image]\ image\_fun$ 
  by (simp, blast)

```

```

lemma  $decr\_succ\_decr:$ 
  assumes  $f\in nat\ \rightarrow\ P\ preorder\_on(P, leq)$ 
   $\forall n\in nat. \langle f^n\ succ(n), f^n\rangle\in leq$ 
   $m\in nat$ 
  shows  $n\in nat\ \Longrightarrow\ n\leq m\ \Longrightarrow\ \langle f^n\ m, f^n\rangle\in leq$ 
  using  $\langle m\in\_ \rangle$ 
proof(induct m)
  case  $0$ 
  then show  $?case$  using  $assms\ leq\_reflI$  by simp
next
  case ( $succ\ x$ )
  then
  have  $1:f^n\ succ(x)\preceq f^n\ x\ f^n\in P\ f^n\ x\in P\ f^n\ succ(x)\in P$ 
    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  $refl(P, leq)\ f\in nat\ \rightarrow\ P$ 
   $\forall n\in nat. \langle f^n\ succ(n), f^n\rangle\in leq$ 
   $trans[P](leq)$ 

```

```

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

end

```

2.2 Towards Rasiowa-Sikorski Lemma (RSL)

```

locale countable_generic = forcing_notion +
  fixes  $\mathcal{D}$ 
  assumes countable_subs_of_P:  $\mathcal{D} \in \text{nat} \rightarrow \text{Pow}(P)$ 
    and seq_of_denses:  $\forall n \in \text{nat}. \text{dense}(\mathcal{D}‘n)$ 

```

begin

definition

```

 $D\_generic :: i \Rightarrow o$  where
 $D\_generic(G) \equiv \text{filter}(G) \wedge (\forall n \in \text{nat}. (\mathcal{D}‘n) \cap G \neq \emptyset)$ 

```

The next lemma identifies a sufficient condition for obtaining RSL.

lemma *RS_sequence_imp_rasiowa_sikorski*:

```

assumes
   $p \in P$   $f : \text{nat} \rightarrow P$   $f‘0 = p$ 
   $\bigwedge n. n \in \text{nat} \implies f‘\text{succ}(n) \preceq f‘n \wedge f‘\text{succ}(n) \in \mathcal{D}‘n$ 
shows
   $\exists G. p \in G \wedge D\_generic(G)$ 

```

```

proof -
  note assms
  moreover from this
  have  $f'^{\text{nat}} \subseteq P$ 
    by (simp add:subset_fun_image)
  moreover from calculation
  have  $\text{refl}(f'^{\text{nat}}, \text{leq}) \wedge \text{trans}[P](\text{leq})$ 
    using leq_preord unfolding preorder_on_def by (blast intro:refl_monot_domain)
  moreover from calculation
  have  $\forall n \in \text{nat}. f' \text{succ}(n) \preceq f' n$  by (simp)
  moreover from calculation
  have  $\text{linear}(f'^{\text{nat}}, \text{leq})$ 
    using leq_preord and decr_seq_linear unfolding preorder_on_def by (blast)
  moreover from calculation
  have  $(\forall p \in f'^{\text{nat}}. \forall q \in f'^{\text{nat}}. \text{compat\_in}(f'^{\text{nat}}, \text{leq}, p, q))$ 
    using chain_compat by (auto)
  ultimately
  have  $\text{filter}(\text{upclosure}(f'^{\text{nat}}))$  (is  $\text{filter}(?G)$ )
    using closure_compat_filter by simp
  moreover
  have  $\forall n \in \text{nat}. \mathcal{D}' n \cap ?G \neq 0$ 
  proof
    fix  $n$ 
    assume  $n \in \text{nat}$ 
    with assms
    have  $f' \text{succ}(n) \in ?G \wedge f' \text{succ}(n) \in \mathcal{D}' n$ 
      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
   $\text{RS\_seq}(0, P, \text{leq}, p, \text{enum}, \mathcal{D}) = p$ 
   $\text{RS\_seq}(\text{succ}(n), P, \text{leq}, p, \text{enum}, \mathcal{D}) =$ 
     $\text{enum}' m. \langle \text{enum}' m, \text{RS\_seq}(n, P, \text{leq}, p, \text{enum}, \mathcal{D}) \rangle \in \text{leq} \wedge \text{enum}' m \in \mathcal{D}' n$ 

```

```

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 $n \in nat$ $p \in P$ $P \subseteq range(enum)$ $enum: nat \rightarrow M$
 $\bigwedge x k. x \in P \implies k \in nat \implies \exists q \in P. q \preceq x \wedge q \in \mathcal{D}'k$
shows
 $f(succ(n)) \in P \wedge f(succ(n)) \preceq f(n) \wedge f(succ(n)) \in \mathcal{D}'n$
using $\langle n \in nat \rangle$
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** $\langle P \subseteq range(enum) \rangle$
obtain m **where** $m \in nat$ $enum'm = q$
using *preimage_rangeD*[*OF* $\langle enum: nat \rightarrow M \rangle$] **by** *blast*
moreover
have $\mathcal{D}'0 \subseteq P$
using *apply_funtype*[*OF* *countable_subst_of_P*] **by** *simp*
moreover note $\langle p \in P \rangle$
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** $\langle P \subseteq range(enum) \rangle$
obtain m **where** $m \in nat$ $enum'm \preceq f(succ(n))$ $enum'm \in \mathcal{D}'succ(n)$
using *preimage_rangeD*[*OF* $\langle enum: nat \rightarrow M \rangle$] **by** *blast*
moreover note *succ*
moreover from *calculation*
have $\mathcal{D}'succ(n) \subseteq P$
using *apply_funtype*[*OF* *countable_subst_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  $Q(q,k,m) \equiv \text{enum}'m \preceq q \wedge \text{enum}'m \in \mathcal{D}'k$ 
  assumes  $n \in \text{nat } p \in P \ P \subseteq \text{range}(\text{enum}) \ \text{enum}:\text{nat} \rightarrow M$ 
  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  $x \in P \ k \in \text{nat}$ 
    then
    have  $\exists q \in P. q \preceq x \wedge q \in \mathcal{D}'k$ 
      using seq_of_densens_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  $n \in \text{nat } p \in P \ P \subseteq \text{range}(\text{enum}) \ \text{enum}:\text{nat} \rightarrow M$ 
  shows  $RS\_seq(n,P,leq,p,\text{enum},\mathcal{D}) \in P$ 
  using assms countable_RS_sequence(1,3)
  by (induct;simp)

```

```

lemma RS_seq_funtype:
  assumes  $p \in P \ P \subseteq \text{range}(\text{enum}) \ \text{enum}:\text{nat} \rightarrow M$ 
  shows  $(\lambda n \in \text{nat}. RS\_seq(n,P,leq,p,\text{enum},\mathcal{D})):\text{nat} \rightarrow P$ 
  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 $a \in A$
 $(\forall X . X \neq 0 \wedge X \subseteq A \longrightarrow s'X \in X)$
 $\forall y \in A. \{x \in A. \langle y, x \rangle \in R\} \neq 0 \ n \in \text{nat}$
shows $dc_witness(n, A, a, s, R) \in A$
using $\langle n \in \text{nat} \rangle$
proof (*induct* n)
case 0
then show *?case* **using** $\langle a \in A \rangle$ **by** *simp*
next
case (*succ* x)
then
show *?case* **using** *assms* **by** *auto*
qed

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

lemma *witness_funtype*:
assumes $a \in A$
 $(\forall X . X \neq 0 \wedge X \subseteq A \longrightarrow s'X \in X)$
 $\forall y \in A. \{x \in A. \langle y, x \rangle \in R\} \neq 0$
shows $(\lambda n \in \text{nat}. dc_witness(n, A, a, s, R)) \in \text{nat} \rightarrow A$ (**is** $?f \in _ \rightarrow _$)
proof -
have $?f \in \text{nat} \rightarrow \{dc_witness(n, A, a, s, R). n \in \text{nat}\}$ (**is** $_ \in _ \rightarrow ?B$)
using *lam_funtype* *assms* **by** *simp*
then
have $?B \subseteq A$
using *witness_into_A* *assms* **by** *auto*
with $\langle ?f \in _ \rangle$
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 $\exists f \in \text{nat} \rightarrow A. \forall n \in \text{nat}. f'n = dc_witness(n, A, a, s, R)$

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

theorem *pointed_DC* :

assumes $(\forall x \in A. \exists y \in A. \langle x, y \rangle \in R)$

shows $\forall a \in A. (\exists f \in \text{nat} \rightarrow A. f'0 = a \wedge (\forall n \in \text{nat}. \langle f'n, f'succ(n) \rangle \in R))$

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 $1: g \in \text{Pow}(A) - \{0\} \rightarrow A$

$\forall X. X \neq 0 \wedge X \subseteq A \longrightarrow g'X \in X$

by *auto*

let $?f = \lambda a. \lambda n \in \text{nat}. \text{dc_witness}(n, A, a, g, R)$

{

fix *a*

assume $a \in A$

from $\langle a \in A \rangle$

have $f0: ?f(a)'0 = a$ by *simp*

with $\langle a \in A \rangle$

have $\langle ?f(a)'n, ?f(a)'succ(n) \rangle \in R$ if $n \in \text{nat}$ for *n*

using *witness_related*[*OF* $\langle a \in A \rangle$ *1*(*2*) *0*] *beta* that by *simp*

then

have $\exists f \in \text{nat} \rightarrow A. f'0 = a \wedge (\forall n \in \text{nat}. \langle f'n, f'succ(n) \rangle \in R)$ (is $\exists x \in _.$ *?P*(*x*))

using *f0* *witness_funtype* *0* *1* $\langle a \in _ \rangle$ by *blast*

}

then show *?thesis* by *auto*

qed

lemma *aux_DC_on_AxNat2* : $\forall x \in A \times \text{nat}. \exists y \in A. \langle x, \langle y, \text{succ}(\text{snd}(x)) \rangle \rangle \in R \implies$

$\forall x \in A \times \text{nat}. \exists y \in A \times \text{nat}. \langle x, y \rangle \in \{ \langle a, b \rangle \in R. \text{snd}(b) = \text{succ}(\text{snd}(a)) \}$

by (*rule* *ballI*, *erule_tac* $x=x$ in *ballE*, *simp_all*)

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

by *auto*

corollary *DC_on_A_x_nat* :

assumes $(\forall x \in A \times \text{nat}. \exists y \in A. \langle x, \langle y, \text{succ}(\text{snd}(x)) \rangle \rangle \in R)$ $a \in A$

shows $\exists f \in \text{nat} \rightarrow A. f'0 = a \wedge (\forall n \in \text{nat}. \langle \langle f'n, n \rangle, \langle f'succ(n), \text{succ}(n) \rangle \rangle \in R)$ (is $\exists x \in _.$ *?P*(*x*))

proof -

let $?R' = \{ \langle a, b \rangle \in R. \text{snd}(b) = \text{succ}(\text{snd}(a)) \}$

from *assms*(*1*)

have $\forall x \in A \times \text{nat}. \exists y \in A \times \text{nat}. \langle x, y \rangle \in ?R'$

using *aux_DC_on_AxNat2* by *simp*

with $\langle a \in _ \rangle$

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 $p \in P$ $n \in \text{nat}$

shows $\exists y \in P. \langle p, y \rangle \in (\lambda m \in \text{nat}. \{ \langle x, y \rangle \in P \times P. y \preceq x \wedge y \in \mathcal{D}'(\text{pred}(m)) \})'n$

proof -

from *seq_of_denses* $\langle n \in \text{nat} \rangle$

have $\text{dense}(\mathcal{D}' \text{ pred}(n))$ **by** *simp*

with $\langle p \in P \rangle$

have $\exists d \in \mathcal{D}' \text{ Arith.pred}(n). d \preceq p$

unfolding *dense_def* **by** *simp*

then obtain d **where** $\exists: d \in \mathcal{D}' \text{ Arith.pred}(n) \wedge d \preceq p$

by *blast*

from *countable_subs_of_P* $\langle n \in \text{nat} \rangle$

have $\mathcal{D}' \text{ Arith.pred}(n) \in \text{Pow}(P)$

by (*blast dest:apply_funtype intro:pred_type*)

then

have $\mathcal{D}' \text{ Arith.pred}(n) \subseteq P$

by (*rule PowD*)

with \exists

have $d \in P \wedge d \preceq p \wedge d \in \mathcal{D}' \text{ Arith.pred}(n)$

by *auto*

with $\langle p \in P \rangle \langle n \in \text{nat} \rangle$

show *?thesis* **by** *auto*

qed

lemma *DC_imp_RS_sequence*:

assumes $p \in P$

shows $\exists f. f: \text{nat} \rightarrow P \wedge f'0 = p \wedge$

$(\forall n \in \text{nat}. f' \text{succ}(n) \preceq f'n \wedge f' \text{succ}(n) \in \mathcal{D}' n)$

proof -

let $?S = (\lambda m \in \text{nat}. \{ \langle x, y \rangle \in P \times P. y \preceq x \wedge y \in \mathcal{D}'(\text{pred}(m)) \})$

have $\forall x \in P. \forall n \in \text{nat}. \exists y \in P. \langle x, y \rangle \in ?S'n$

using *RS_relation* **by** (*auto*)

then

have $\forall a \in P. (\exists f \in \text{nat} \rightarrow P. f'0 = a \wedge (\forall n \in \text{nat}. \langle f'n, f' \text{succ}(n) \rangle \in ?S' \text{succ}(n)))$

using *sequence_DC* **by** (*blast*)

with $\langle p \in P \rangle$

show *?thesis* **by** *auto*

qed

theorem *rasiowa_sikorski*:

$p \in P \implies \exists G. p \in G \wedge D_generic(G)$

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* : $m \in \text{nat} \implies \text{succ}(n) \in \text{succ}(m) \implies n \in m$
by (*drule_tac j=succ(m) in ltI, auto elim:ltD*)

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

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

lemma *in_succ_in_nat* : $m \in \text{nat} \implies n \in \text{succ}(m) \implies n \in \text{nat}$
by(*auto simp add:in_n_in_nat*)

lemma *ltI_neg* : $x \in \text{nat} \implies j \leq x \implies j \neq x \implies j < x$
by (*simp add: le_iff*)

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

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

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

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

lemma *succpred_leI* : $n \in \text{nat} \implies n \leq \text{succ}(\text{pred}(n))$
by (*auto elim: natE*)

lemma *succpred_n0* : $\text{succ}(n) \in p \implies p \neq 0$
by (*auto*)

lemma *funcI* : $f \in A \rightarrow B \implies a \in A \implies b = f \ ' \ a \implies \langle a, b \rangle \in f$
by(*simp_all add: apply_Pair*)

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

lemma *succ_in* : $\text{succ}(x) \leq y \implies x \in y$
by (*auto dest:ltD*)

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

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

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

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

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

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

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

lemma *succ_mono* : $m \in \text{nat} \implies n \leq m \implies \text{succ}(n) \leq \text{succ}(m)$
by *auto*

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

lemma *nat_union_abs1* :
 $\llbracket \text{Ord}(i) ; \text{Ord}(j) ; i \leq j \rrbracket \implies i \cup j = j$
by (*rule* *Un_absorb1, erule* *le_imp_subset*)

lemma *nat_union_abs2* :
 $\llbracket \text{Ord}(i) ; \text{Ord}(j) ; i \leq j \rrbracket \implies j \cup i = j$
by (*rule* *Un_absorb2, erule* *le_imp_subset*)

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

lemma *nat_max_ty* : $\text{Ord}(i) \implies \text{Ord}(j) \implies \text{Ord}(\text{max}(i, j))$
unfolding *max_def* **by** *simp*

lemma *le_not_lt_nat* : $\text{Ord}(p) \implies \text{Ord}(q) \implies \neg p \leq q \implies q \leq p$
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* : $x \in \text{nat} \implies x \leq \text{succ}(x)$ **by** *simp*

lemma *le_pred* : $x \in \text{nat} \implies \text{pred}(x) \leq x$
using *pred_le* [*OF* *le_succ*] *pred_succ_eq*

by simp

lemma *Un_le_compat* : $o \leq p \implies q \leq r \implies \text{Ord}(o) \implies \text{Ord}(p) \implies \text{Ord}(q) \implies \text{Ord}(r) \implies o \cup q \leq p \cup r$
using *le_trans*[of $q \ r \ p \cup r$, *OF _ Un_upper2_le*] *le_trans*[of $o \ p \ p \cup r$, *OF _ Un_upper1_le*]
nat_simp_union
by auto

lemma *Un_le* : $p \leq r \implies q \leq r \implies \text{Ord}(p) \implies \text{Ord}(q) \implies \text{Ord}(r) \implies p \cup q \leq r$
using *nat_simp_union* **by auto**

lemma *Un_leI3* : $o \leq r \implies p \leq r \implies q \leq r \implies \text{Ord}(o) \implies \text{Ord}(p) \implies \text{Ord}(q) \implies \text{Ord}(r) \implies o \cup p \cup q \leq r$
using *nat_simp_union* **by auto**

lemma *diff_mono* :
assumes $m \in \text{nat } n \in \text{nat } p \in \text{nat } m < n \ p \leq m$
shows $m \# - p < n \# - p$
proof -
from *assms*
have $m \# - p \in \text{nat } m \# - p \# + p = m$
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*:
 $x \in \text{nat} \implies y \in \text{nat} \implies \text{Arith.pred}(\text{succ}(x) \cup y) = x \cup \text{Arith.pred}(y)$
 $x \in \text{nat} \implies y \in \text{nat} \implies \text{Arith.pred}(x \cup \text{succ}(y)) = \text{Arith.pred}(x) \cup y$
using *pred_Un_distrib pred_succ_eq* **by** *simp_all*

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

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

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

lemma *leD* : **assumes** $n \in \text{nat } j \leq n$
shows $j < n \mid j = n$

using $leE[OF \langle j \leq n \rangle, of j < n \mid j = n]$ by *auto*

5.1 Some results in ordinal arithmetic

The following results are auxiliary to the proof of wellfoundedness of the relation $freqR$

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

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

lemma *max_cong2* :
assumes $x \leq y$ *Ord*(y) *Ord*(z) *Ord*(x)
shows $max(x, z) \leq max(y, z)$
proof -
 from *assms*
 have $x \cup z \leq y \cup z$
 using *lt_Ord Ord_Un Un_mono[OF le_imp_subset[OF \langle x \leq y \rangle]] subset_imp_le*
by *auto*
 then show *?thesis*
 using *nat_simp_union \langle Ord(x) \rangle \langle Ord(z) \rangle \langle Ord(y) \rangle* **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 \langle w < z \rangle]* *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_commutates nat_simp_union assms leI* **by**

```

simp
  then
  show ?thesis using Un_leD2 assms by simp
qed

lemma max_D2 :
  assumes  $w = y \vee w = z$   $x < y$   $\text{Ord}(x)$   $\text{Ord}(w)$   $\text{Ord}(y)$   $\text{Ord}(z)$   $\text{max}(x,w) = \text{max}(y,z)$ 
  shows  $x < w$ 
proof -
  from assms
  have  $x < z \cup y$  using Un_upper2_lt[OF  $\langle x < y \rangle$ ] 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  $\langle x < y \rangle$ ] by simp
    next
      case False
      then
      have  $w \leq y$ 
      using not_lt_iff_le[OF assms(5) assms(4)] by simp
      with  $\langle w = z \rangle$ 
      have  $\text{max}(z,y) = y$  unfolding max_def using assms by simp
      with assms
      have  $\dots = x \cup w$  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  $\text{Ord}(n)$   $\text{Ord}(\alpha)$   $\text{Ord}(\beta)$   $\alpha < \beta$   $x < n$   $y < n$   $0 < n$ 

```

```

shows  $n ** \alpha ++ x < n ** \beta ++ y$ 
proof -
  consider (0)  $\beta=0$  | (s)  $\gamma$  where  $Ord(\gamma) \beta = succ(\gamma) \mid (l) Limit(\beta)$ 
    using  $Ord\_cases[OF \langle Ord(\beta) \rangle, of ?thesis]$  by force
  then show  $?thesis$ 
  proof cases
    case 0
      then show  $?thesis$  using  $\langle \alpha < \beta \rangle$  by auto
    next
      case s
      then
        have  $\alpha \leq \gamma$  using  $\langle \alpha < \beta \rangle$  using  $leI$  by auto
        then
          have  $n ** \alpha \leq n ** \gamma$  using  $omult\_le\_mono[OF \_ \langle \alpha \leq \gamma \rangle] \langle Ord(n) \rangle$  by simp
          then
            have  $n ** \alpha ++ x < n ** \gamma ++ n$  using  $oadd\_lt\_mono[OF \_ \langle x < n \rangle]$  by
            simp
            also
              have  $\dots = n ** \beta$  using  $\langle \beta = succ(\_) \rangle omult\_succ \langle Ord(\beta) \rangle \langle Ord(n) \rangle$  by simp
            finally
              have  $n ** \alpha ++ x < n ** \beta$  by auto
            then
              show  $?thesis$  using  $oadd\_le\_self \langle Ord(\beta) \rangle lt\_trans2 \langle Ord(n) \rangle$  by auto
          next
            case l
            have  $Ord(x)$  using  $\langle x < n \rangle lt\_Ord$  by simp
            with l
              have  $succ(\alpha) < \beta$  using  $Limit\_has\_succ \langle \alpha < \beta \rangle$  by simp
              have  $n ** \alpha ++ x < n ** \alpha ++ n$ 
                using  $oadd\_lt\_mono[OF le\_refl[OF Ord\_omult[OF \_ \langle Ord(\alpha) \rangle]]] \langle x < n \rangle$ 
                 $\langle Ord(n) \rangle$  by simp
              also
                have  $\dots = n ** succ(\alpha)$  using  $omult\_succ \langle Ord(\alpha) \rangle \langle Ord(n) \rangle$  by simp
              finally
                have  $n ** \alpha ++ x < n ** succ(\alpha)$  by simp
              with  $\langle succ(\alpha) < \beta \rangle$ 
                have  $n ** \alpha ++ x < n ** \beta$  using  $lt\_trans omult\_lt\_mono \langle Ord(n) \rangle \langle 0 < n \rangle$ 
                by auto
              then show  $?thesis$  using  $oadd\_le\_self \langle Ord(\beta) \rangle lt\_trans2 \langle Ord(n) \rangle$  by auto
            qed
          qed
        end
  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  $0 \in A$  env  $\in$  list( $A$ )
  shows  $\text{nth}(n, \text{env}) \in A$ 
  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:  $\llbracket \text{nth}(n, l) = x; n \in \text{nat} \rrbracket \implies \text{nth}(\text{succ}(n), \text{Cons}(a, l)) = x$ 
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 (*termsep_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^*

```

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

```

```

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

```

```

lemma fld_restrict_dom :
  assumes  $\text{relation}(r)$   $\text{domain}(r) \subseteq A$   $\text{range}(r) \subseteq A$ 
  shows  $r \cap A \times A = r$ 
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* $tr_down(r,a) \subseteq A$, *auto simp add* : $wfrec_restr\ tr_down_def$
 $trancl_eq_r$)

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

definition

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

lemma *RrelI* : $x \in A \Longrightarrow y \in A \Longrightarrow R(x,y) \Longrightarrow \langle x,y \rangle \in Rrel(R,A)$
unfolding *Rrel_def* **by** *simp*

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

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

lemma *field_Rrel*: $field(Rrel(R,d)) \subseteq d$
unfolding *Rrel_def* **by** *auto*

lemma *Rrel_mono* : $A \subseteq B \Longrightarrow Rrel(R,A) \subseteq Rrel(R,B)$
unfolding *Rrel_def* **by** *blast*

lemma *Rrel_restr_eq* : $Rrel(R,A) \cap B \times B = Rrel(R,A \cap B)$
unfolding *Rrel_def* **by** *blast*

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

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

lemma *restrict_trancl_Rrel*:

assumes $R(w,y)$

shows $restrict(f, Rrel(R,d) - \{\{y\}\})'w$
 $= restrict(f, (Rrel(R,d)^{\hat{+}}) - \{\{y\}\})'w$

proof (*cases* $y \in d$)

let $?r = Rrel(R,d)$ **and** $?s = (Rrel(R,d))^{\hat{+}}$

case *True*

show *?thesis*

proof (*cases* $w \in d$)

case *True*

with $\langle y \in d \rangle$ *assms*

have $\langle w, y \rangle \in ?r$

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]
  fieldII[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) = wfrec(?r^+, x, λy f. H(y, restrict(f, ?r-“{y})))
    unfolding wfrec_def ..
  also
  have ... = wfrec(?r^+, x, λy f. H(y, restrict(f, (?r^+)-“{y})))
    using assms by simp
  also
  have ... = wfrec(?r^+, x, H)
    unfolding wfrec_def using trancl_eq_r[OF relation_trancl trans_trancl] by
simp
  finally
  show ?thesis .
qed

end

```

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

$\text{powerset}(M, x, y) \longleftrightarrow y = \{a \in \text{Pow}(x) . M(a)\}$

using *powerset_abs* **assms** **by** *simp*

lemma *Collect_inter_Transset*:

assumes

$\text{Transset}(M) \ b \in M$

shows

$\{x \in b . P(x)\} = \{x \in b . P(x)\} \cap M$

using *assms* **unfolding** *Transset_def*

by (*auto*)

lemma (in *M_trivial*) *family_union_closed*: $\llbracket \text{strong_replacement}(M, \lambda x y. y = f(x)); M(A); \forall x \in A. M(f(x)) \rrbracket$

$\implies M(\bigcup x \in A. f(x))$

using *RepFun_closed* ..

definition

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

definition

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

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

definition

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

definition

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

definition

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

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**
 $rrank(a) \equiv Memrel(eclose(\{a\}))^{\wedge+}$

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

$HVfrom_replacement : \llbracket M(i) ; M(A) \rrbracket \Longrightarrow$
 $transrec_replacement(M, is_HVfrom(M, A), i)$ **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 $is_powapply_abs: \llbracket M(f); M(y) \rrbracket \Longrightarrow is_powapply(M, f, y, z) \longleftrightarrow M(z) \wedge$
 $z = \{x \in Pow(f'y). M(x)\}$
unfolding $is_powapply_def$ **by** $simp$

lemma $\llbracket M(A); M(x); M(f); M(h) \rrbracket \Longrightarrow$

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

lemma *Replace_is_powapply:*

assumes
 $M(R) M(A) M(f)$
shows
 $is_Replace(M, A, is_powapply(M, f), R) \longleftrightarrow R = Replace(A, is_powapply(M, f))$
proof -
have *univalent(M, A, is_powapply(M, f))*
using $\langle M(A) \rangle \langle M(f) \rangle$ **unfolding** *univalent_def is_powapply_def by simp*
moreover
have $\bigwedge x y. \llbracket x \in A; is_powapply(M, f, x, y) \rrbracket \implies M(y)$
using $\langle M(A) \rangle \langle M(f) \rangle$ **unfolding** *is_powapply_def by simp*
ultimately
show *?thesis using* $\langle M(A) \rangle \langle M(R) \rangle$ *Replace_abs by simp*
qed

lemma *powapply_closed:*

$\llbracket M(y) ; M(f) \rrbracket \implies M(\{x \in Pow(f \text{ ' } y) . M(x)\})$
using *apply_closed power_ax unfolding power_ax_def by simp*

lemma *RepFun_is_powapply:*

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

lemma *RepFun_powapply_closed:*

assumes
 $M(f) M(A)$
shows
 $M(Replace(A, is_powapply(M, f)))$
proof -
have *univalent(M, A, is_powapply(M, f))*
using $\langle M(A) \rangle \langle M(f) \rangle$ **unfolding** *univalent_def is_powapply_def by simp*
moreover
have $\llbracket x \in A ; is_powapply(M, f, x, y) \rrbracket \implies M(y)$ **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
 $M(\bigcup y \in x. \{a \in \text{Pow}(f'y). \ M(a)\})$
proof -
have $M(\{a \in \text{Pow}(f'y). \ M(a)\})$ **if** $y \in x$ **for** y
using *that assms transM[of _ x] powapply_closed by simp*
then
have $M(\{\{a \in \text{Pow}(f'y). \ M(a)\}. \ y \in x\})$
using *assms transM[of _ x] RepFun_powapply_closed RepFun_is_powapply*
by *simp*
then show *?thesis using assms by simp*
qed

lemma *relation2_HVfrom: $M(A) \implies \text{relation2}(M, \text{is_HVfrom}(M, A), \text{HVfrom}(M, A))$*

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 :*

$M(A) \implies \forall x[M]. \ \forall g[M]. \ \text{function}(g) \longrightarrow M(\text{HVfrom}(M, A, x, g))$
unfolding *HVfrom_def using Union_powapply_closed by simp*

lemma *transrec_HVfrom:*

assumes $M(A)$
shows $\text{Ord}(i) \implies \{x \in \text{Vfrom}(A, i). \ M(x)\} = \text{transrec}(i, \text{HVfrom}(M, A))$
proof (*induct rule:trans_induct*)
case (*step i*)
have $\text{Vfrom}(A, i) = A \cup (\bigcup y \in i. \ \text{Pow}((\lambda x \in i. \ \text{Vfrom}(A, x)) \ 'y))$
using *def_transrec[OF Vfrom_def, of A i] by simp*
then
have $\text{Vfrom}(A, i) = A \cup (\bigcup y \in i. \ \text{Pow}(\text{Vfrom}(A, y)))$
by *simp*
then
have $\{x \in \text{Vfrom}(A, i). \ M(x)\} = \{x \in A. \ M(x)\} \cup (\bigcup y \in i. \ \{x \in \text{Pow}(\text{Vfrom}(A, y)). \ M(x)\})$
 $M(x)\}$
by *auto*
with $\langle M(A) \rangle$
have $\{x \in \text{Vfrom}(A, i). \ M(x)\} = A \cup (\bigcup y \in i. \ \{x \in \text{Pow}(\text{Vfrom}(A, y)). \ M(x)\})$
by (*auto intro:transM*)
also
have $\dots = A \cup (\bigcup y \in i. \ \{x \in \text{Pow}(\{z \in \text{Vfrom}(A, y). \ M(z)\}). \ M(x)\})$
proof -

```

have { $x \in \text{Pow}(V\text{from}(A, y)). M(x)$ } = { $x \in \text{Pow}(\{z \in V\text{from}(A, y). M(z)\}). M(x)$ }
  if  $y \in i$  for  $y$  by (auto intro:transM)
  then
    show ?thesis by simp
  qed
also from step
have ... =  $A \cup (\bigcup y \in i. \{x \in \text{Pow}(\text{transrec}(y, HV\text{from}(M, A))). M(x)\})$  by auto
also
have ... =  $\text{transrec}(i, HV\text{from}(M, A))$ 
  using def_transrec[of  $\lambda y. \text{transrec}(y, HV\text{from}(M, A)) HV\text{from}(M, A) i, \text{symmetric}$ ]

  unfolding HVfrom_def by simp
finally
show ?case .
qed

```

```

lemma Vfrom_abs:  $\llbracket M(A); M(i); M(V); \text{Ord}(i) \rrbracket \implies \text{is\_Vfrom}(M, A, i, V) \longleftrightarrow$ 
 $V = \{x \in V\text{from}(A, i). M(x)\}$ 
  unfolding is_Vfrom_def
  using relation2_HVfrom HVfrom_closed HVfrom_replacement
  transrec_abs[of  $\text{is\_HVfrom}(M, A) i HV\text{from}(M, A)$ ] transrec_HVfrom by simp

```

```

lemma Vfrom_closed:  $\llbracket M(A); M(i); \text{Ord}(i) \rrbracket \implies M(\{x \in V\text{from}(A, i). M(x)\})$ 
  unfolding is_Vfrom_def
  using relation2_HVfrom HVfrom_closed HVfrom_replacement
  transrec_closed[of  $\text{is\_HVfrom}(M, A) i HV\text{from}(M, A)$ ] transrec_HVfrom by
  simp

```

```

lemma Vset_abs:  $\llbracket M(i); M(V); \text{Ord}(i) \rrbracket \implies \text{is\_Vset}(M, i, V) \longleftrightarrow V = \{x \in V\text{set}(i).$ 
 $M(x)\}$ 
  using Vfrom_abs unfolding is_Vset_def by simp

```

```

lemma Vset_closed:  $\llbracket M(i); \text{Ord}(i) \rrbracket \implies M(\{x \in V\text{set}(i). M(x)\})$ 
  using Vfrom_closed unfolding is_Vset_def by simp

```

```

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

```

```

lemma rank_trancl:  $\text{rank}(x) = \text{wfrec}(\text{rrank}(x), x, \text{Hrank})$ 

```

proof -

```

have  $\text{rank}(x) = \text{wfrec}(\text{Memrel}(\text{eclose}(\{x\})), x, \text{Hrank})$ 
  (is _ = wfrec(?r, _, _))
  unfolding rank_def transrec_def Hrank_def by simp
also
have ... =  $\text{wfrec}(\text{?r} \hat{+}, x, \lambda y f. \text{Hrank}(y, \text{restrict}(f, \text{?r} - \{\{y\}\})))$ 
  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 :  $\llbracket M(z) ; M(f) \rrbracket \implies \text{univalent}(M, z, \text{PHrank}(M, f))$ 
  unfolding univalent_def PHrank_def by simp

```

```

lemma PHrank_abs :
   $\llbracket M(f) ; M(y) \rrbracket \implies \text{PHrank}(M, f, y, z) \longleftrightarrow M(z) \wedge z = \text{succ}(f'y)$ 
  unfolding PHrank_def by simp

```

```

lemma PHrank_closed :  $\text{PHrank}(M, f, y, z) \implies M(z)$ 
  unfolding PHrank_def by simp

```

```

lemma Replace_PHrank_abs:
  assumes
    M(z) M(f) M(hr)
  shows
    is_Replace(M, z, PHrank(M, f), hr)  $\longleftrightarrow$  hr = Replace(z, PHrank(M, f))
proof -
  have  $\bigwedge x y. \llbracket x \in z ; \text{PHrank}(M, f, x, y) \rrbracket \implies M(y)$ 
    using  $\langle M(z) \rangle \langle M(f) \rangle$  unfolding PHrank_def by simp
  then
  show ?thesis using  $\langle M(z) \rangle \langle M(hr) \rangle \langle M(f) \rangle$  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  $\{z . y \in A, M(z) \wedge z = \text{succ}(f'y)\} = \{z . y \in A, z = \text{succ}(f'y)\}$ 
    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  $\llbracket x \in A ; PHrank(M,f,x,y) \rrbracket \implies M(y)$  for  $x y$ 
    using assms unfolding PHrank_def by simp
  with univ_PHrank
  show ?thesis using assms PHrank_replacement by simp
qed

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

lemma Union_PHrank_closed:
  assumes
    M(x) M(f)
  shows
    M( $\bigcup y \in x. succ(f'y)$ )
proof -
  have  $M(succ(f'y))$  if  $y \in x$  for  $y$ 
    using that assms transM[of _ x] by simp
  then
  have  $M(\{succ(f'y). y \in x\})$ 
    using assms transM[of _ x] RepFun_PHrank_closed RepFun_PHrank by simp
  then show ?thesis using assms by simp
qed

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

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

lemma M_into_Vset:
  assumes M(a)
  shows  $\exists i[M]. \exists V[M]. ordinal(M,i) \wedge is_Vfrom(M,0,i,V) \wedge a \in V$ 
proof -
  let  $?i = succ(rank(a))$ 
  from assms

```

```

have  $a \in \{x \in V \text{from}(0, ?i). M(x)\}$  (is  $a \in ?V$ )
  using Vset_Ord_rank_iff by simp
moreover from assms
have  $M(?i)$ 
  using rank_closed by simp
moreover
note  $\langle M(a) \rangle$ 
moreover from calculation
have  $M(?V)$ 
  using Vfrom_closed by simp
moreover from calculation
have  $\text{ordinal}(M, ?i) \wedge \text{is\_Vfrom}(M, 0, ?i, ?V) \wedge a \in ?V$ 
  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  $\$' = \text{curry } ((\text{op } \$) \text{ o } \text{swap})$ 
  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
  end

```

```

val vars = map (fn v => (v, the (Vars.lookup inst v))) thm_vars;
val (tm,hyps) = thm_tms |> hd |> pair Thm.concl_of Thm.premis_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*] \Rightarrow *o* ($\langle _ , _ \models _ \rangle$ [36,36,36] 60)

syntax_consts

_sats \equiv *sats*

translations

(*M*, *env* \models φ) \equiv *CONST sats*(*M*, φ , *env*)

abbreviation

dec10 :: *i* ($\langle 10 \rangle$) **where** *10* \equiv *succ*(9)

abbreviation

dec11 :: *i* ($\langle 11 \rangle$) **where** *11* \equiv *succ*(10)

abbreviation

dec12 :: *i* ($\langle 12 \rangle$) **where** *12* \equiv *succ*(11)

abbreviation

dec13 :: *i* ($\langle 13 \rangle$) **where** *13* \equiv *succ*(12)

abbreviation

dec14 :: *i* ($\langle 14 \rangle$) **where** *14* \equiv *succ*(13)

definition

infinity_ax :: (*i* \Rightarrow *o*) \Rightarrow *o* **where**

infinity_ax(*M*) \equiv

($\exists I[M]. (\exists z[M]. \text{empty}(M, z) \wedge z \in I) \wedge (\forall y[M]. y \in I \longrightarrow (\exists sy[M]. \text{successor}(M, y, sy) \wedge sy \in I))$)

definition

choice_ax :: (*i* \Rightarrow *o*) \Rightarrow *o* **where**

choice_ax(*M*) \equiv $\forall x[M]. \exists a[M]. \exists f[M]. \text{ordinal}(M, a) \wedge \text{surjection}(M, a, x, f)$

context *M_basic* **begin**

lemma *choice_ax_abs* :
choice_ax(*M*) \longleftrightarrow ($\forall x[M]. \exists a[M]. \exists f[M]. \text{Ord}(a) \wedge f \in \text{surj}(a,x)$)
unfolding *choice_ax_def*
by (*simp*)

end

definition

wellfounded_trancl :: [*i* => *o*, *i*, *i*] => *o* **where**
wellfounded_trancl(*M*, *Z*, *r*, *p*) \equiv
 $\exists w[M]. \exists wx[M]. \exists rp[M].$
 $w \in Z \ \& \ \text{pair}(M, w, p, wx) \ \& \ \text{tran_closure}(M, r, rp) \ \& \ wx \in rp$

lemma *empty_intf* :
infinity_ax(*M*) \implies
($\exists z[M]. \text{empty}(M, z)$)
by (*auto simp add: empty_def infinity_ax_def*)

lemma *Transset_intf* :
Transset(*M*) $\implies y \in x \implies x \in M \implies y \in M$
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*: $\varphi \in \text{formula} \implies \text{env} \in \text{list}(M) \implies \text{arity}(\varphi) \leq 1 \ \#\+$
 $\text{length}(\text{env}) \implies$

separation($\#\#M, \lambda x. \text{sats}(M, \varphi, [x] \ @ \ \text{env})$)

and *replacement_ax*: $\varphi \in \text{formula} \implies \text{env} \in \text{list}(M) \implies \text{arity}(\varphi) \leq 2 \ \#\+$
 $\text{length}(\text{env}) \implies$

strong_replacement($\#\#M, \lambda x y. \text{sats}(M, \varphi, [x, y] \ @ \ \text{env})$)

and *trans_M*: *Transset*(*M*)

begin

lemma *TranssetI* :

($\bigwedge y x. y \in x \implies x \in M \implies y \in M$) $\implies \text{Transset}(M)$

by (*auto simp add: Transset_def*)

lemma *zero_in_M*: $0 \in M$

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)

```

```

 $\longleftrightarrow$  sats( $M, ifm(0,1), env$ )
and
 $ifm(0,1) \in formula$ 
and
 $arity(ifm(0,1)) = 2$ 
using  $\langle A \in M \rangle inter\_fm\_auto$ 
by (simp del:FOL_sats_iff add: nat_simp_union)
then
have  $\forall a \in M. separation(\#\#M, \lambda x. sats(M, ifm(0,1), [x, a]))$ 
using separation_ax by simp
moreover
have  $(\forall y \in M. y \in a \longrightarrow x \in y) \longleftrightarrow sats(M, ifm(0,1), [x, a])$ 
if  $a \in M$   $x \in M$  for  $a$   $x$ 
using that fmsats[of [x,a]] by simp
ultimately
have  $\forall a \in M. separation(\#\#M, \lambda x. \forall y \in M. y \in a \longrightarrow x \in y)$ 
unfolding separation_def by simp
with  $\langle A \in M \rangle$  show ?thesis by simp
qed

```

```

schematic_goal diff_fm_auto:
assumes
 $nth(i, env) = x$   $nth(j, env) = B$ 
 $i \in nat$   $j \in nat$   $env \in list(A)$ 
shows
 $x \notin B \longleftrightarrow sats(A, ?dfm(i,j), env)$ 
by (insert assms ; (rule sep_rules | simp)+)

```

```

lemma diff_sep_intf :
assumes
 $B \in M$ 
shows
 $separation(\#\#M, \lambda x. x \notin B)$ 
proof -
obtain dfm where
 $fmsats: \bigwedge env. env \in list(M) \implies nth(0, env) \notin nth(1, env)$ 
 $\longleftrightarrow sats(M, dfm(0,1), env)$ 
and
 $dfm(0,1) \in formula$ 
and
 $arity(dfm(0,1)) = 2$ 
using  $\langle B \in M \rangle$  diff_fm_auto
by (simp del:FOL_sats_iff add: nat_simp_union)
then
have  $\forall b \in M. separation(\#\#M, \lambda x. sats(M, dfm(0,1), [x, b]))$ 
using separation_ax by simp
moreover

```

```

have  $x \notin b \iff \text{sats}(M, \text{dfm}(0,1), [x,b])$ 
if  $b \in M$   $x \in M$  for  $b$   $x$ 
using that fmsats[of [x,b]] by simp
ultimately
have  $\forall b \in M. \text{separation}(\#\#M, \lambda x. x \notin b)$ 
unfolding separation_def by simp
with  $\langle B \in M \rangle$  show ?thesis by simp
qed

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

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

```

schematic_goal *im_fm_auto*:

assumes

$nth(i,env) = y \ nth(j,env) = r \ nth(h,env) = B$
 $i \in nat \ j \in nat \ h \in nat \ env \in list(A)$

shows

$(\exists p \in A. p \in r \ \& \ (\exists x \in A. x \in B \ \& \ pair(\#\#A,x,y,p))) \longleftrightarrow sats(A,?imfm(i,j,h),env)$
by (*insert assms ; (rule sep_rules | simp)+*)

lemma *image_sep_intf* :

assumes

$A \in M$
and
 $r \in M$

shows

$separation(\#\#M, \lambda y. \exists p \in M. p \in r \ \& \ (\exists x \in M. x \in A \ \& \ pair(\#\#M,x,y,p)))$

proof -

obtain *imfm* **where**

$fmsats: \bigwedge env. env \in list(M) \implies$

$(\exists p \in M. p \in nth(1,env) \ \& \ (\exists x \in M. x \in nth(2,env) \ \& \ pair(\#\#M,x,nth(0,env),p)))$
 $\longleftrightarrow sats(M,imfm(0,1,2),env)$

and

$imfm(0,1,2) \in formula$

and

$arity(imfm(0,1,2)) = 3$

using *im_fm_auto* **by** (*simp del:FOL_sats_iff pair_abs add: fm_defs nat_simp_union*)

then

have $\forall r \in M. \forall a \in M. separation(\#\#M, \lambda y. sats(M,imfm(0,1,2), [y,r,a]))$

using *separation_ax* **by** *simp*

moreover

have $(\exists p \in M. p \in k \ \& \ (\exists x \in M. x \in a \ \& \ pair(\#\#M,x,y,p))) \longleftrightarrow sats(M,imfm(0,1,2),[y,k,a])$

if $k \in M \ a \in M \ y \in M$ **for** $k \ a \ y$

using *that fmsats[of [y,k,a]]* **by** *simp*

ultimately

have $\forall k \in M. \forall a \in M. separation(\#\#M, \lambda y. \exists p \in M. p \in k \ \& \ (\exists x \in M. x \in a \ \& \ pair(\#\#M,x,y,p)))$

unfolding *separation_def* **by** *simp*

with $\langle r \in M \rangle \langle A \in M \rangle$ **show** *?thesis* **by** *simp*

qed

schematic_goal *con_fm_auto*:

assumes

$nth(i,env) = z \ nth(j,env) = R$
 $i \in nat \ j \in nat \ env \in list(A)$

shows

$(\exists p \in A. p \in R \ \& \ (\exists x \in A. \exists y \in A. pair(\#\#A,x,y,p) \ \& \ pair(\#\#A,y,x,z)))$
 $\longleftrightarrow sats(A,?cfm(i,j),env)$

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

```

lemma converse_sep_intf :
  assumes
     $R \in M$ 
  shows
    separation( $\#\#M, \lambda z. \exists p \in M. p \in R \ \& \ (\exists x \in M. \exists y \in M. \text{pair}(\#\#M, x, y, p) \ \& \ \text{pair}(\#\#M, y, x, z))$ )
  proof -
    obtain cfm where
      fmsats:  $\bigwedge env. env \in list(M) \implies$ 
      ( $\exists p \in M. p \in nth(1, env) \ \& \ (\exists x \in M. \exists y \in M. \text{pair}(\#\#M, x, y, p) \ \& \ \text{pair}(\#\#M, y, x, nth(0, env))))$ )
       $\longleftrightarrow \text{sats}(M, cfm(0, 1), env)$ 
    and
      cfm(0, 1)  $\in$  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  $\forall r \in M. \text{separation}(\#\#M, \lambda z. \text{sats}(M, cfm(0, 1), [z, r]))$ 
      using separation_ax by simp
    moreover
      have ( $\exists p \in M. p \in r \ \& \ (\exists x \in M. \exists y \in M. \text{pair}(\#\#M, x, y, p) \ \& \ \text{pair}(\#\#M, y, x, z))$ )
       $\longleftrightarrow$ 
        sats(M, cfm(0, 1), [z, r])
      if  $z \in M \ r \in M$  for  $z \ r$ 
      using that fmsats[of [z, r]] by simp
    ultimately
      have  $\forall r \in M. \text{separation}(\#\#M, \lambda z. \exists p \in M. p \in r \ \& \ (\exists x \in M. \exists y \in M. \text{pair}(\#\#M, x, y, p) \ \& \ \text{pair}(\#\#M, y, x, z)))$ 
      unfolding separation_def by simp
    with  $\langle R \in M \rangle$  show ?thesis by simp
  qed

```

```

schematic_goal rest_fm_auto:
  assumes
    nth(i, env) = z nth(j, env) = C
    i  $\in$  nat j  $\in$  nat env  $\in$  list(A)
  shows
    ( $\exists x \in A. x \in C \ \& \ (\exists y \in A. \text{pair}(\#\#A, x, y, z))$ )
     $\longleftrightarrow \text{sats}(A, ?rfm(i, j), env)$ 
  by (insert assms ; (rule sep_rules | simp)+)

```

```

lemma restrict_sep_intf :
  assumes
     $A \in M$ 
  shows
    separation( $\#\#M, \lambda z. \exists x \in M. x \in A \ \& \ (\exists y \in M. \text{pair}(\#\#M, x, y, z))$ )
  proof -

```

obtain rfm where
 $fmsats: \bigwedge env. env \in list(M) \implies$
 $(\exists x \in M. x \in nth(1, env) \ \& \ (\exists y \in M. pair(\#\#M, x, y, nth(0, env))))$
 $\longleftrightarrow sats(M, rfm(0, 1), env)$
and
 $rfm(0, 1) \in formula$
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 $\forall a \in M. separation(\#\#M, \lambda z. sats(M, rfm(0, 1), [z, a]))$
using *separation_ax* **by** *simp*
moreover
have $(\exists x \in M. x \in a \ \& \ (\exists y \in M. pair(\#\#M, x, y, z))) \longleftrightarrow$
 $sats(M, rfm(0, 1), [z, a])$
if $z \in M \ a \in M$ **for** $z \ a$
using *that fmsats[of [z, a]]* **by** *simp*
ultimately
have $\forall a \in M. separation(\#\#M, \lambda z. \exists x \in M. x \in a \ \& \ (\exists y \in M. pair(\#\#M, x, y, z)))$
unfolding *separation_def* **by** *simp*
with $\langle A \in M \rangle$ **show** *?thesis* **by** *simp*
qed

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

lemma comp_sep_intf :
assumes
 $R \in M$
and
 $S \in M$
shows
 $separation(\#\#M, \lambda xz. \exists x \in M. \exists y \in M. \exists z \in M. \exists xy \in M. \exists yz \in M.$
 $pair(\#\#M, x, z, xz) \ \& \ pair(\#\#M, x, y, xy) \ \& \ pair(\#\#M, y, z, yz) \ \& \ xy \in S$
 $\ \& \ yz \in R)$

proof -

obtain cfm where
 $fmsats: \bigwedge env. env \in list(M) \implies$
 $(\exists x \in M. \exists y \in M. \exists z \in M. \exists xy \in M. \exists yz \in M. pair(\#\#M, x, z, nth(0, env)) \ \&$
 $pair(\#\#M, x, y, xy) \ \& \ pair(\#\#M, y, z, yz) \ \& \ xy \in nth(1, env) \ \& \ yz \in nth(2, env))$

```

     $\longleftrightarrow$   $sats(M, cfm(0,1,2), env)$ 
  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  $\forall r \in M. \forall s \in M. separation(\#\#M, \lambda y. sats(M, cfm(0,1,2), [y,s,r]))$ 
  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  $\forall s \in M. \forall r \in M. separation(\#\#M, \lambda xz. \exists x \in M. \exists y \in M. \exists z \in M. \exists xy \in M.$ 
 $\exists yz \in M.$ 
 $pair(\#\#M, x, z, xz) \ \& \ pair(\#\#M, x, y, xy) \ \& \ pair(\#\#M, y, z, yz) \ \& \ xy \in s$ 
 $\ \& \ yz \in r$ )
  unfolding  $separation\_def$  by  $simp$ 
  with  $\langle S \in M \rangle \langle R \in M \rangle$  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

$separation(\#\#M, \lambda y. \exists p \in M. p \in R \ \& \ pair(\#\#M, y, X, p))$

proof -

obtain pfm **where**

$fmsats: \bigwedge env. env \in list(M) \implies$

$(\exists p \in M. p \in nth(1, env) \ \& \ pair(\#\#M, nth(0, env), nth(2, env), p)) \longleftrightarrow sats(M, pfm(0,1,2), env)$

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  $\forall x \in M. \forall r \in M. \text{separation}(\#\#M, \lambda y. \text{sats}(M, \text{pfm}(0,1,2)) , [y,r,x])$ 
  using separation_ax by simp
moreover
have  $(\exists p \in M. p \in r \ \& \ \text{pair}(\#\#M, y, x, p))$ 
   $\longleftrightarrow \text{sats}(M, \text{pfm}(0,1,2)) , [y,r,x]$ 
  if  $y \in M \ r \in M \ x \in M$  for  $y \ x \ r$ 
  using that_fmsats[of [y,r,x]] by simp
ultimately
have  $\forall x \in M. \forall r \in M. \text{separation}(\#\#M, \lambda y. \exists p \in M. p \in r \ \& \ \text{pair}(\#\#M, y, x, p))$ 
  unfolding separation_def by simp
with  $\langle X \in M \rangle \langle R \in M \rangle$  show ?thesis by simp
qed

```

```

schematic_goal mem_fm_auto:
assumes
  nth(i,env) = z i  $\in$  nat env  $\in$  list(A)
shows
   $(\exists x \in A. \exists y \in A. \text{pair}(\#\#A, x, y, z) \ \& \ x \in y) \longleftrightarrow \text{sats}(A, ?mfm(i), env)$ 
by (insert assms ; (rule sep_rules | simp)+)

```

```

lemma memrel_sep_intf:
  separation(\#\#M,  $\lambda z. \exists x \in M. \exists y \in M. \text{pair}(\#\#M, x, y, z) \ \& \ x \in y$ )
proof -
obtain mfm where
  fmsats: $\bigwedge env. env \in \text{list}(M) \implies$ 
   $(\exists x \in M. \exists y \in M. \text{pair}(\#\#M, x, y, \text{nth}(0, env)) \ \& \ x \in y) \longleftrightarrow \text{sats}(M, mfm(0), env)$ 
  and
  mfm(0)  $\in$  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,  $\lambda z. \text{sats}(M, mfm(0)) , [z]$ )
  using separation_ax by simp
moreover
have  $(\exists x \in M. \exists y \in M. \text{pair}(\#\#M, x, y, z) \ \& \ x \in y) \longleftrightarrow \text{sats}(M, mfm(0), [z])$ 
  if  $z \in M$  for  $z$ 
  using that_fmsats[of [z]] by simp
ultimately
have separation(\#\#M,  $\lambda z. \exists x \in M. \exists y \in M. \text{pair}(\#\#M, x, y, z) \ \& \ x \in y$ )
  unfolding separation_def by simp
then show ?thesis by simp
qed

```

schematic_goal *recfun_fm_auto*:

assumes
 $nth(i1,env) = x \ nth(i2,env) = r \ nth(i3,env) = f \ nth(i4,env) = g \ nth(i5,env)$
 $= a$
 $nth(i6,env) = b \ i1 \in nat \ i2 \in nat \ i3 \in nat \ i4 \in nat \ i5 \in nat \ i6 \in nat \ env \in list(A)$
shows
 $(\exists xa \in A. \exists xb \in A. pair(\#\#A,x,a,xa) \ \& \ xa \in r \ \& \ pair(\#\#A,x,b,xb) \ \& \ xb \in r \ \&$
 $(\exists fx \in A. \exists gx \in A. fun_apply(\#\#A,f,x,fx) \ \& \ fun_apply(\#\#A,g,x,gx)$
 $\ \& \ fx \neq gx))$
 $\longleftrightarrow sats(A,?rffm(i1,i2,i3,i4,i5,i6),env)$
by (*insert assms ; (rule sep_rules | simp)+*)

lemma *is_recfun_sep_intf* :

assumes
 $r \in M \ f \in M \ g \in M \ a \in M \ b \in M$
shows
 $separation(\#\#M,\lambda x. \exists xa \in M. \exists xb \in M.$
 $pair(\#\#M,x,a,xa) \ \& \ xa \in r \ \& \ pair(\#\#M,x,b,xb) \ \& \ xb \in r \ \&$
 $(\exists fx \in M. \exists gx \in M. fun_apply(\#\#M,f,x,fx) \ \& \ fun_apply(\#\#M,g,x,gx)$
 $\ \&$
 $fx \neq gx))$

proof -

obtain *rffm* **where**
 $fmsats:\bigwedge env. env \in list(M) \implies$
 $(\exists xa \in M. \exists xb \in M. pair(\#\#M,nth(0,env),nth(4,env),xa) \ \& \ xa \in nth(1,env) \ \&$
 $pair(\#\#M,nth(0,env),nth(5,env),xb) \ \& \ xb \in nth(1,env) \ \& \ (\exists fx \in M. \exists gx \in M.$
 $fun_apply(\#\#M,nth(2,env),nth(0,env),fx) \ \& \ fun_apply(\#\#M,nth(3,env),nth(0,env),gx)$
 $\ \& \ fx \neq gx))$
 $\longleftrightarrow sats(M,rffm(0,1,2,3,4,5),env)$
and
 $rffm(0,1,2,3,4,5) \in formula$
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 $\forall a1 \in M. \forall a2 \in M. \forall a3 \in M. \forall a4 \in M. \forall a5 \in M.$
 $separation(\#\#M, \lambda x. sats(M,rffm(0,1,2,3,4,5), [x,a1,a2,a3,a4,a5]))$
using *separation_ax* **by** *simp*
moreover
have $(\exists xa \in M. \exists xb \in M. pair(\#\#M,x,a4,xa) \ \& \ xa \in a1 \ \& \ pair(\#\#M,x,a5,xb)$
 $\ \& \ xb \in a1 \ \&$
 $(\exists fx \in M. \exists gx \in M. fun_apply(\#\#M,a2,x,fx) \ \& \ fun_apply(\#\#M,a3,x,gx)$
 $\ \& \ fx \neq gx))$
 $\longleftrightarrow sats(M,rffm(0,1,2,3,4,5), [x,a1,a2,a3,a4,a5])$
if $x \in M \ a1 \in M \ a2 \in M \ a3 \in M \ a4 \in M \ a5 \in M$ **for** $x \ a1 \ a2 \ a3 \ a4 \ a5$
using *that fmsats[of [x,a1,a2,a3,a4,a5]]* **by** *simp*
ultimately

have $\forall a1 \in M. \forall a2 \in M. \forall a3 \in M. \forall a4 \in M. \forall a5 \in M. \text{separation}(\#\#M, \lambda x .$
 $\exists xa \in M. \exists xb \in M. \text{pair}(\#\#M, x, a4, xa) \ \& \ xa \in a1 \ \& \ \text{pair}(\#\#M, x, a5, xb)$
 $\ \& \ xb \in a1 \ \&$
 $(\exists fx \in M. \exists gx \in M. \text{fun_apply}(\#\#M, a2, x, fx) \ \& \ \text{fun_apply}(\#\#M, a3, x, gx)$
 $\ \& \ fx \neq gx))$
unfolding *separation_def* **by** *simp*
with $\langle r \in M \rangle \langle f \in M \rangle \langle g \in M \rangle \langle a \in M \rangle \langle b \in M \rangle$ **show** *?thesis* **by** *simp*
qed

schematic_goal *funsp_fm_auto*:

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

lemma *funspace_succ_rep_intf* :

assumes
 $n \in M$
shows
 $\text{strong_replacement}(\#\#M,$
 $\lambda p \ z. \exists f \in M. \exists b \in M. \exists nb \in M. \exists \text{cnbf} \in M.$
 $\text{pair}(\#\#M, f, b, p) \ \& \ \text{pair}(\#\#M, n, b, nb) \ \& \ \text{is_cons}(\#\#M, nb, f, \text{cnbf})$
 $\ \&$
 $\text{upair}(\#\#M, \text{cnbf}, \text{cnbf}, z))$

proof -

obtain *fsfm* **where**
 $\text{fmsats} : \text{env} \in \text{list}(M) \implies$
 $(\exists f \in M. \exists b \in M. \exists nb \in M. \exists \text{cnbf} \in M. \text{pair}(\#\#M, f, b, \text{nth}(0, \text{env})) \ \& \ \text{pair}(\#\#M, \text{nth}(2, \text{env}), b, nb)$
 $\ \& \ \text{is_cons}(\#\#M, nb, f, \text{cnbf}) \ \& \ \text{upair}(\#\#M, \text{cnbf}, \text{cnbf}, \text{nth}(1, \text{env})))$
 $\longleftrightarrow \text{sats}(M, \text{fsfm}(0, 1, 2), \text{env})$
and $\text{fsfm}(0, 1, 2) \in \text{formula}$ **and** $\text{arity}(\text{fsfm}(0, 1, 2)) = 3$ **for** env
using *funsp_fm_auto*[*of concl:M*] **by** (*simp del:FOL_sats_iff pair_abs add:*
 $\text{fm_defs nat_simp_union}$)
then
have $\forall n0 \in M. \text{strong_replacement}(\#\#M, \lambda p \ z. \text{sats}(M, \text{fsfm}(0, 1, 2), [p, z, n0]))$
using *replacement_ax* **by** *simp*
moreover
have $(\exists f \in M. \exists b \in M. \exists nb \in M. \exists \text{cnbf} \in M. \text{pair}(\#\#M, f, b, p) \ \& \ \text{pair}(\#\#M, n0, b, nb)$
 $\ \&$
 $\text{is_cons}(\#\#M, nb, f, \text{cnbf}) \ \& \ \text{upair}(\#\#M, \text{cnbf}, \text{cnbf}, z))$
 $\longleftrightarrow \text{sats}(M, \text{fsfm}(0, 1, 2), [p, z, n0])$

```

    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  $\forall n0 \in M. strong\_replacement(\#\#M, \lambda p z.$ 
     $\exists f \in M. \exists b \in M. \exists nb \in M. \exists cnbf \in M. pair(\#\#M, f, b, p) \ \& \ pair(\#\#M, n0, b, nb)$ 
  &
     $is\_cons(\#\#M, nb, f, cnbf) \ \& \ upair(\#\#M, cnbf, cnbf, z))$ 
    unfolding strong_replacement_def univalent_def by simp
  with  $\langle n \in M \rangle$  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:  $\bigwedge env. env \in list(M) \implies$ 
  (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)  $\in$  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  $\forall x \in M. \forall a \in M. separation(\#\#M, \lambda y. sats(M, rcfm(0, 1, 2), [y, x, a]))$ 
  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  $y \in M \ x \in M \ a \in M$  for  $y \ x \ a$ 
  using that fmsats[of [y, x, a]] by simp
  ultimately
  have  $\forall x \in M. \forall a \in M. separation(\#\#M, rtran\_closure\_mem(\#\#M, a, x))$ 
  unfolding separation_def by simp
  with  $\langle r \in M \rangle \langle A \in M \rangle$  show ?thesis by simp
qed

```

```

schematic_goal rtran_closure_fm_auto:
  assumes
     $nth(i, env) = r \ nth(j, env) = rp$ 
     $i \in nat \ j \in nat \ env \in list(A)$ 
  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
     $nth(i, env) = r \ nth(j, env) = rp$ 
     $i \in nat \ j \in nat \ env \in list(A)$ 
  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
     $nth(i, env) = p \ nth(j, env) = r \ nth(k, env) = B$ 
     $i \in nat \ j \in nat \ k \in nat \ env \in list(A)$ 
  shows
     $wellfounded\_trancl(\#\#A, B, r, p) \longleftrightarrow sats(A, ?wtf(i, j, k), env)$ 

```

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: $\bigwedge env. env \in list(M) \implies$
 (*wellfounded_trancl*(##*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 *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 $\forall x \in M. \forall z \in M. \text{separation}(\## M, \lambda y. \text{sats}(M, \text{rcfm}(0,1,2), [y,x,z]))$
using *separation_ax* **by** *simp*
moreover
have (*wellfounded_trancl*(##*M*,*z*,*x*,*y*)
 \longleftrightarrow *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 $\forall x \in M. \forall z \in M. \text{separation}(\## M, \text{wellfounded_trancl}(\## M, z, x))$
unfolding *separation_def* **by** *simp*
with $\langle r \in M \rangle \langle Z \in M \rangle$ **show** *?thesis* **by** *simp*
qed

lemma (**in** *M_ZF_trans*) *finite_sep_intf*:
separation(##*M*, $\lambda x. x \in nat$)
proof -
have *arity*(*finite_ordinal_fm*(0)) = 1
unfolding *finite_ordinal_fm_def limit_ordinal_fm_def empty_fm_def succ_fm_def*
cons_fm_def
 union_fm_def upair_fm_def
by (*simp add: nat_union_abs1 Un_commute*)
with *separation_ax*
have ($\forall v \in M. \text{separation}(\## M, \lambda x. \text{sats}(M, \text{finite_ordinal_fm}(0), [x,v]))$)
by *simp*
then have ($\forall v \in M. \text{separation}(\## M, \text{finite_ordinal}(\## M))$)

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'* :
 $\llbracket I \in M ; 0 \in I ; \bigwedge x. x \in I \implies \text{succ}(x) \in I \rrbracket \implies \text{nat} \subseteq I$
by (*rule subsetI,induct_tac x,simp+*)

lemma (in *M_ZF_trans*) *nat_subset_I* :
 $\exists I \in M. \text{nat} \subseteq I$

proof -
have $\exists I \in M. 0 \in I \wedge (\forall x \in M. x \in I \longrightarrow \text{succ}(x) \in I)$
using *infinity_ax unfolding infinity_ax_def* **by** *auto*
then obtain *I* **where**
 $I \in M \ 0 \in I \ (\forall x \in M. x \in I \longrightarrow \text{succ}(x) \in I)$
by *auto*
then have $\bigwedge x. x \in I \implies \text{succ}(x) \in I$
using *Transset_intf[OF trans_M]* **by** *simp*
then have $\text{nat} \subseteq I$
using $\langle I \in M \rangle \langle 0 \in I \rangle$ *nat_subset_I'* **by** *simp*
then show *?thesis* **using** $\langle I \in M \rangle$ **by** *auto*
qed

lemma (in *M_ZF_trans*) *nat_in_M* :
 $\text{nat} \in M$

proof -
have $1 : \{x \in B . x \in A\} = A$ **if** $A \subseteq B$ **for** *A B*
using *that* **by** *auto*
obtain *I* **where**
 $I \in M \ \text{nat} \subseteq I$
using *nat_subset_I* **by** *auto*
then have $\{x \in I . x \in \text{nat}\} \in M$
using *finite_sep_intf separation_closed[of $\lambda x . x \in \text{nat}$]* **by** *simp*
then show *?thesis*
using $\langle \text{nat} \subseteq I \rangle$ *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* \subseteq *M_trancl ##M*

by (rule mtranc1)

10.4 Interface with M_eclose

lemma *repl_sats*:

assumes

$sat: \bigwedge x z. x \in M \implies z \in M \implies sats(M, \varphi, Cons(x, Cons(z, env))) \longleftrightarrow P(x, z)$

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 $\langle n \in nat \rangle$ *nat_trans_M* **by** *simp*

then have $1: Memrel(succ(n)) \in M$

using $\langle n \in nat \rangle$ *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 $1 \langle A \in M \rangle$ *list_functor_iff_sats* **by** *simp*

then have $sats(M, iterates_MH_fm(list_functor_fm(13, 1, 0), 10, 2, 1, 0), [a0, a1, a2, a3, a4, y, x, z, Memrel(succ(n)), A, 0])$

$\longleftrightarrow iterates_MH(\#\#M, is_list_functor(\#\#M, A), 0, a2, a1, a0)$

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 $sats_iterates_MH_fm[of M is_list_functor(\#\#M, A) _] 1 \langle 0 \in M \rangle$

$\langle A \in M \rangle$ **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

$is_wfrec(\#\#M, iterates_MH(\#\#M, is_list_functor(\#\#M, A), 0), Memrel(succ(n)), x, y)$

if $y \in M$ $x \in M$ $z \in M$ **for** y x z

using that $sats_is_wfrec_fm 1 \langle 0 \in M \rangle \langle A \in M \rangle$ **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])))))
     $\longleftrightarrow$  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])
     $\longleftrightarrow$  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])
   $\longleftrightarrow$ 
  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])
   $\longleftrightarrow$ 
  ( $\exists y$ ∈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.
   $\exists y$ ∈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 $0 \in M$
 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 $2:formula_functor_fm(1,0) \in formula$ **by** *simp*
 have *is_formula_functor*($\#\#M, a, b$) \longleftrightarrow
 sats($M, formula_functor_fm(1,0), [b, a]$)
 if $a \in M$ $b \in M$ **for** a b
 using *that* **by** *simp*
 then show *?thesis* **using** $\langle 0 \in M \rangle$ 1 2 *iterates_repl_intf* **by** *simp*
qed

lemma (in M_ZF_trans) *nth_repl_intf*:
assumes
 $l \in M$
shows
 iterates_replacement($\#\#M, \lambda l' t. is_tl(\#\#M, l', t), l$)
proof -
 have $1:arity(tl_fm(1,0)) = 2$
 unfolding *tl_fm_def fm_defs quaselist_fm_def Cons_fm_def Nil_fm_def*
Inr_fm_def number1_fm_def
 Inl_fm_def **by** (*simp add:nat_simp_union*)
 have $2:tl_fm(1,0) \in formula$ **by** *simp*
 have *is_tl*($\#\#M, a, b$) \longleftrightarrow *sats*($M, tl_fm(1,0), [b, a]$)
 if $a \in M$ $b \in M$ **for** a b
 using *that* **by** *simp*
 then show *?thesis* **using** $\langle l \in M \rangle$ 1 2 *iterates_repl_intf* **by** *simp*
qed

lemma (in M_ZF_trans) *eclose_repl1_intf*:
assumes
 $A \in M$
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 $2:big_union_fm(1,0) \in formula$ **by** *simp*
 have *big_union*($\#\#M, a, b$) \longleftrightarrow *sats*($M, big_union_fm(1,0), [b, a]$)
 if $a \in M$ $b \in M$ **for** a b
 using *that* **by** *simp*
 then show *?thesis* **using** $\langle A \in M \rangle$ 1 2 *iterates_repl_intf* **by** *simp*
qed

```

lemma (in M_ZF_trans) list_repl2_intf:
  assumes
     $A \in M$ 
  shows
    strong_replacement( $\#\#M, \lambda n y. n \in \text{nat} \ \& \ \text{is\_iterates}(\#\#M, \text{is\_list\_functor}(\#\#M, A), 0, n, y)$ )
  proof -
    have  $0 \in M$ 
    using nat_0I nat_trans M by simp
    have is_list_functor( $\#\#M, A, a, b$ )  $\longleftrightarrow$ 
      sats( $M, \text{list\_functor\_fm}(13, 1, 0), [b, a, c, d, e, f, g, h, i, j, k, n, y, A, 0, \text{nat}]$ )
    if  $a \in M \ b \in M \ c \in M \ d \in M \ e \in M \ f \in M \ g \in M \ h \in M \ i \in M \ j \in M \ k \in M \ n \in M \ y \in M$ 
    for  $a \ b \ c \ d \ e \ f \ g \ h \ i \ j \ k \ n \ y$ 
    using that  $\langle 0 \in M \rangle \text{ nat\_in\_M } \langle A \in M \rangle$  by simp
    then
      have  $1: \text{sats}(M, \text{is\_iterates\_fm}(\text{list\_functor\_fm}(13, 1, 0), 3, 0, 1), [n, y, A, 0, \text{nat}])$ 
       $\longleftrightarrow$ 
        is_iterates( $\#\#M, \text{is\_list\_functor}(\#\#M, A), 0, n, y$ )
      if  $n \in M \ y \in M$  for  $n \ y$ 
      using that  $\langle 0 \in M \rangle \langle A \in M \rangle \text{ nat\_in\_M}$ 
      sats_is_iterates_fm[of  $M \ \text{is\_list\_functor}(\#\#M, A)$ ] by simp
    let  $?f = \text{And}(\text{Member}(0, 4), \text{is\_iterates\_fm}(\text{list\_functor\_fm}(13, 1, 0), 3, 0, 1))$ 
    have satsf: sats( $M, ?f, [n, y, A, 0, \text{nat}]$ )  $\longleftrightarrow$ 
       $n \in \text{nat} \ \& \ \text{is\_iterates}(\#\#M, \text{is\_list\_functor}(\#\#M, A), 0, n, y)$ 
    if  $n \in M \ y \in M$  for  $n \ y$ 
    using that  $\langle 0 \in M \rangle \langle A \in M \rangle \text{ nat\_in\_M } 1$  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( $\#\#M, \lambda n y. \text{sats}(M, ?f, [n, y, A, 0, \text{nat}])$ )
      using replacement_ax 1 nat_in_M  $\langle A \in M \rangle \langle 0 \in M \rangle$  by simp
    then
      show thesis using repl_sats[of  $M \ ?f \ [A, 0, \text{nat}]$ ] satsf by simp
  qed

```

```

lemma (in M_ZF_trans) formula_repl2_intf:
  strong_replacement( $\#\#M, \lambda n y. n \in \text{nat} \ \& \ \text{is\_iterates}(\#\#M, \text{is\_formula\_functor}(\#\#M), 0, n, y)$ )
  proof -
    have  $0 \in M$ 
    using nat_0I nat_trans M by simp
    have is_formula_functor( $\#\#M, a, b$ )  $\longleftrightarrow$ 

```

```

      sats(M, formula_function_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_function_fm(1,0),2,0,1), [n,y,0,nat] )
  ←→
    is_iterates(##M, is_formula_function(##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_function(##M)] by simp
  let ?f = And(Member(0,3), is_iterates_fm(formula_function_fm(1,0),2,0,1))
  have satsf:sats(M, ?f, [n,y,0,nat] ) ←→
    n∈nat & is_iterates(##M, is_formula_function(##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_function_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

$is_powapply_fm :: [i,i,i] \Rightarrow i$ **where**
 $is_powapply_fm(f,y,z) \equiv$
 $Exists(And(fun_apply_fm(succ(f), succ(y), 0),$
 $Forall(Iff(Member(0, succ(succ(z))),$
 $Forall(Implies(Member(0, 1), Member(0, 2))))))$

lemma $is_powapply_type [TC]$:

$[f \in nat ; y \in nat ; z \in nat] \Rightarrow is_powapply_fm(f,y,z) \in formula$
unfolding $is_powapply_fm_def$ **by** *simp*

lemma $sats_is_powapply_fm$:

assumes

$f \in nat \ y \in nat \ z \in nat \ env \in list(A) \ 0 \in A$

shows

$is_powapply(\#\#A, nth(f, env), nth(y, env), nth(z, env))$

$\longleftrightarrow sats(A, is_powapply_fm(f,y,z), env)$

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

$f \in M$

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 $\forall f0 \in M. strong_replacement(\#\#M, \lambda p z. sats(M, is_powapply_fm(2,0,1)$
 $, [p,z,f0]))$

using $replacement_ax$ **by** *simp*

moreover

have $is_powapply(\#\#M, f0, p, z) \longleftrightarrow sats(M, is_powapply_fm(2,0,1) , [p,z,f0])$

if $p \in M \ z \in M \ f0 \in M$ **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 $\forall f0 \in M. strong_replacement(\#\#M, is_powapply(\#\#M, f0))$

unfolding $strong_replacement_def$ $univalent_def$ **by** *simp*

with $\langle f \in M \rangle$ **show** $?thesis$ **by** *simp*

qed

definition

$PHrank_fm :: [i,i,i] \Rightarrow i$ **where**
 $PHrank_fm(f,y,z) \equiv \text{Exists}(\text{And}(\text{fun_apply_fm}(\text{succ}(f),\text{succ}(y),0)$
 $\quad ,\text{succ_fm}(0,\text{succ}(z))))$

lemma $PHrank_type$ [TC]:

$\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \Longrightarrow PHrank_fm(x,y,z) \in \text{formula}$
by (*simp add:PHrank_fm_def*)

lemma (in M_ZF_trans) $sats_PHrank_fm$ [*simp*]:

$\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; \text{env} \in \text{list}(M) \rrbracket$
 $\Longrightarrow \text{sats}(M,PHrank_fm(x,y,z),\text{env}) \longleftrightarrow$
 $PHrank(\#\#M,\text{nth}(x,\text{env}),\text{nth}(y,\text{env}),\text{nth}(z,\text{env}))$
using *zero_in_M Internalizations.nth_closed* **by** (*simp add: PHrank_def PHrank_fm_def*)

lemma (in M_ZF_trans) $phrank_repl$:**assumes** $f \in M$ **shows** $\text{strong_replacement}(\#\#M,PHrank(\#\#M,f))$ **proof** -**have** $\text{arity}(PHrank_fm(2,0,1)) = 3$ **unfolding** $PHrank_fm_def$ **by** (*simp add: fm_defs nat_simp_union*)**then****have** $\forall f0 \in M. \text{strong_replacement}(\#\#M, \lambda p z. \text{sats}(M,PHrank_fm(2,0,1) ,$
 $[p,z,f0]))$ **using** replacement_ax **by** *simp***then****have** $\forall f0 \in M. \text{strong_replacement}(\#\#M, PHrank(\#\#M,f0))$ **unfolding** $\text{strong_replacement_def univalent_def}$ **by** *simp***with** $\langle f \in M \rangle$ **show** *?thesis* **by** *simp***qed****definition** $is_Hrank_fm :: [i,i,i] \Rightarrow i$ **where**

$is_Hrank_fm(x,f,hc) \equiv \text{Exists}(\text{And}(\text{big_union_fm}(0,\text{succ}(hc)),$
 $\quad \text{Replace_fm}(\text{succ}(x),PHrank_fm(\text{succ}(\text{succ}(\text{succ}(f))),0,1),0)))$

lemma is_Hrank_type [TC]:

$\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \Longrightarrow is_Hrank_fm(x,y,z) \in \text{formula}$
by (*simp add:is_Hrank_fm_def*)

lemma (in M_ZF_trans) $sats_is_Hrank_fm$ [*simp*]:

$\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; \text{env} \in \text{list}(M) \rrbracket$

```

    ==> sats(M, is_Hrank_fm(x,y,z), env) <=>
      is_Hrank(##M, nth(x, env), nth(y, env), nth(z, env))
using zero_in_M is_Hrank_def is_Hrank_fm_def sats_Replace_fm
by simp

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

```

definition

$is_HVfrom_fm :: [i,i,i,i] \Rightarrow i$ **where**
 $is_HVfrom_fm(A,x,f,h) \equiv \text{Exists}(\text{Exists}(\text{And}(\text{union_fm}(A \# + 2,1,h \# + 2),$
 $\text{And}(\text{big_union_fm}(0,1),$
 $\text{Replace_fm}(x \# + 2,is_powapply_fm(f \# + 4,0,1),0))))))$

lemma is_HVfrom_type [TC]:

$\llbracket A \in nat; x \in nat; f \in nat; h \in nat \rrbracket \Longrightarrow is_HVfrom_fm(A,x,f,h) \in formula$
by (*simp add:is_HVfrom_fm_def*)

lemma $sats_is_HVfrom_fm$:

$\llbracket a \in nat; x \in nat; f \in nat; h \in nat; env \in list(A); 0 \in A \rrbracket$
 $\Longrightarrow sats(A, is_HVfrom_fm(a,x,f,h), env) \longleftrightarrow$
 $is_HVfrom(\#\#A, nth(a, env), nth(x, env), nth(f, env), nth(h, env))$
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 $\langle nth(i, env) = ii \rangle [symmetric] \langle nth(v, env) = vv \rangle [symmetric]$
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:  $\bigwedge x y. \llbracket x \in M; y \in M \rrbracket \implies is\_F(##M, x, y) \longleftrightarrow y = f(x)$ 
  and
    f_sats:  $\bigwedge x y. \llbracket x \in M; y \in M \rrbracket \implies$ 
      sats(M, f_fm, Cons(x, Cons(y, env)))  $\longleftrightarrow is\_F(##M, x, y)$ 
  and
    f_form: f_fm ∈ formula
  and
    f_arty: arity(f_fm) = 2
  and
    env ∈ list(M)
  shows
    strong_replacement(##M,  $\lambda x y. y = f(x)$ )
proof -
  have sats(M, f_fm, [x, y]@env)  $\longleftrightarrow is\_F(##M, x, y)$  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,  $\lambda x y. sats(M, f_fm, [x, y]@env)$ )
    using ⟨env ∈ list(M)⟩ replacement_ax by simp
  ultimately
  have strong_replacement(##M, is_F(##M))
    using strong_replacement_cong[of ##M is_F(##M)  $\lambda x y. sats(M, f_fm, [x, y]@env)$  is_F(##M)]
  by simp
  with f_abs show ?thesis
  using strong_replacement_cong[of ##M is_F(##M)  $\lambda x y. y = f(x)$ ] 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:  $\bigwedge x. x \in M \implies sats(M, \varphi, [x]@env) \longleftrightarrow Q(x)$ 
  shows
    {y ∈ A . Q(y)} ∈ M

```

```

proof -
  have separation(##M,λx. sats(M,φ,[x] @ env))
    using assms separation_ax by simp
  then show ?thesis using
    ⟨A∈M⟩ satsQ trans_M
    separation_cong[of ##M λy. sats(M,φ,[y]@env) Q]
    separation_closed by simp
qed

end

```

11 Transitive set models of ZF

This theory defines the locale M_ZF_trans for transitive models of ZF, and the associated *forcing_data* that adds a forcing notion

```

theory Forcing_Data
  imports
    Forcing_Notions
    Interface

```

```

begin

```

```

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

```

```

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

```

```

locale M_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 $M_ctm \subseteq M_trivial \#\#M$
by (rule mtriv)

sublocale $M_ctm \subseteq M_trans \#\#M$
by (rule mtrans)

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

sublocale $M_ctm \subseteq M_trancl \#\#M$
by (rule mtrancl)

sublocale $M_ctm \subseteq M_datatypes \#\#M$
by (rule mdatatypes)

sublocale $M_ctm \subseteq M_eclose \#\#M$
by (rule meclose)

sublocale $M_ctm \subseteq M_eclose_pow \#\#M$
by (rule meclose_pow)

context M_ctm
begin

11.1 Collects in M

lemma $Collect_in_M_Op$:

assumes

$Qfm : Q_fm \in formula$ **and**

$Qarty : arity(Q_fm) = 1$ **and**

$Qsats : \bigwedge x. x \in M \implies sats(M, Q_fm, [x]) \longleftrightarrow is_Q(\#\#M, x)$ **and**

$Qabs : \bigwedge x. x \in M \implies is_Q(\#\#M, x) \longleftrightarrow Q(x)$ **and**

$A \in M$

shows

$Collect(A, Q) \in M$

proof -

have $z \in A \implies z \in M$ **for** z

using $\langle A \in M \rangle$ $transitivity[of\ z\ A]$ **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

$separation_cong[of\ \#\#M\ \lambda y. sats(M, Q_fm, [y])\ is_Q(\#\#M)]$

by $simp$

then

have $Collect(A, is_Q(\#\#M)) \in M$

using *separation_closed* $\langle A \in M \rangle$ **by** *simp*
then
show *?thesis using 1 by simp*
qed

lemma *Collect_in_M_2p* :

assumes

$Q_{fm} : Q_{fm} \in \text{formula}$ **and**

$Q_{arty} : \text{arity}(Q_{fm}) = 3$ **and**

$\text{params_M} : y \in M \ z \in M$ **and**

$Q_{sats} : \bigwedge x. x \in M \implies \text{sats}(M, Q_{fm}, [x, y, z]) \longleftrightarrow \text{is_Q}(\#\#M, x, y, z)$ **and**

$Q_{abs} : \bigwedge x. x \in M \implies \text{is_Q}(\#\#M, x, y, z) \longleftrightarrow Q(x, y, z)$ **and**

$A \in M$

shows

$\text{Collect}(A, \lambda x. Q(x, y, z)) \in M$

proof -

have $z \in A \implies z \in M$ **for** z

using $\langle A \in M \rangle$ *transitivity[of z A]* **by** *simp*

then

have $1 : \text{Collect}(A, \lambda x. \text{is_Q}(\#\#M, x, y, z)) = \text{Collect}(A, \lambda x. Q(x, y, z))$

using Q_{abs} *Collect_cong[of A A $\lambda x. \text{is_Q}(\#\#M, x, y, z)$ $\lambda x. Q(x, y, z)$]* **by** *simp*

have *separation* $(\#\#M, \lambda x. \text{is_Q}(\#\#M, x, y, z))$

using *separation_ax* Q_{sats} Q_{arty} Q_{fm} params_M

separation_cong[of $\#\#M$ $\lambda x. \text{sats}(M, Q_{fm}, [x, y, z])$ $\lambda x. \text{is_Q}(\#\#M, x, y, z)$]

by *simp*

then

have $\text{Collect}(A, \lambda x. \text{is_Q}(\#\#M, x, y, z)) \in M$

using *separation_closed* $\langle A \in M \rangle$ **by** *simp*

then

show *?thesis using 1 by simp*

qed

lemma *Collect_in_M_4p* :

assumes

$Q_{fm} : Q_{fm} \in \text{formula}$ **and**

$Q_{arty} : \text{arity}(Q_{fm}) = 5$ **and**

$\text{params_M} : a1 \in M \ a2 \in M \ a3 \in M \ a4 \in M$ **and**

$Q_{sats} : \bigwedge x. x \in M \implies \text{sats}(M, Q_{fm}, [x, a1, a2, a3, a4]) \longleftrightarrow \text{is_Q}(\#\#M, x, a1, a2, a3, a4)$

and

$Q_{abs} : \bigwedge x. x \in M \implies \text{is_Q}(\#\#M, x, a1, a2, a3, a4) \longleftrightarrow Q(x, a1, a2, a3, a4)$ **and**

$A \in M$

shows

$\text{Collect}(A, \lambda x. Q(x, a1, a2, a3, a4)) \in M$

proof -

have $z \in A \implies z \in M$ **for** z

using $\langle A \in M \rangle$ *transitivity[of z A]* **by** *simp*

then

have $1 : \text{Collect}(A, \lambda x. \text{is_Q}(\#\#M, x, a1, a2, a3, a4)) = \text{Collect}(A, \lambda x. Q(x, a1, a2, a3, a4))$

```

using Qabs Collect_cong[of  $A$   $\lambda x. is\_Q(\#\#M, x, a1, a2, a3, a4) \lambda x. Q(x, a1, a2, a3, a4)$ ]

by simp
have separation( $\#\#M, \lambda x. is\_Q(\#\#M, x, a1, a2, a3, a4)$ )
using separation_ax Qsats Qarty Qfm params_M
separation_cong[of  $\#\#M \lambda x. sats(M, Q\_fm, [x, a1, a2, a3, a4])$ 
 $\lambda x. is\_Q(\#\#M, x, a1, a2, a3, a4)$ ]
by simp
then
have Collect( $A, \lambda x. is\_Q(\#\#M, x, a1, a2, a3, a4)$ )  $\in M$ 
using separation_closed  $\langle A \in M \rangle$  by simp
then
show ?thesis using 1 by simp
qed

```

lemma *Repl_in_M* :

```

assumes
  f_fm:  $f\_fm \in formula$  and
  f_ar:  $arity(f\_fm) \leq 2 \ \#\ + \ length(env)$  and
  fsats:  $\bigwedge x y. x \in M \implies y \in M \implies sats(M, f\_fm, [x, y]@env) \longleftrightarrow is\_f(x, y)$  and
  fabs:  $\bigwedge x y. x \in M \implies y \in M \implies is\_f(x, y) \longleftrightarrow y = f(x)$  and
  fclosed:  $\bigwedge x. x \in A \implies f(x) \in M$  and
   $A \in M \ env \in list(M)$ 
shows  $\{f(x). x \in A\} \in M$ 
proof -
have strong_replacement( $\#\#M, \lambda x y. sats(M, f\_fm, [x, y]@env)$ )
using replacement_ax f_fm f_ar  $\langle env \in list(M) \rangle$  by simp
then
have strong_replacement( $\#\#M, \lambda x y. y = f(x)$ )
using fsats fabs
strong_replacement_cong[of  $\#\#M \lambda x y. sats(M, f\_fm, [x, y]@env) \lambda x y. y =$ 
 $f(x)$ ]
by simp
then
have  $\{y . x \in A, y = f(x)\} \in M$ 
using  $\langle A \in M \rangle$  fclosed strong_replacement_closed by simp
moreover
have  $\{f(x). x \in A\} = \{y . x \in A, y = f(x)\}$ 
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:  $P \in M$ 
and leq_in_M:  $leq \in M$ 

```

begin

lemma *transD* : $\text{Transset}(M) \implies y \in M \implies y \subseteq M$
by (*unfold Transset_def*, *blast*)

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

definition

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

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

lemma *M_generic_leqD* [*dest*]: $M_generic(G) \implies p \in G \implies q \in P \implies p \preceq q \implies q \in G$
unfolding *M_generic_def* **by** (*blast dest:filter_leqD*)

lemma *M_generic_compatD* [*dest*]: $M_generic(G) \implies p \in G \implies r \in G \implies \exists q \in G. q \preceq p \wedge q \preceq r$
unfolding *M_generic_def* **by** (*blast dest:low_bound_filter*)

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

lemma *G_nonempty*: $M_generic(G) \implies G \neq 0$

proof -

have $P \subseteq P$..

assume

$M_generic(G)$

with *P_in_M P_dense* $\langle P \subseteq P \rangle$ **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 $\langle M_generic(G) \rangle$ **have** *increasing*(G)

unfolding *M_generic_def* **and** *filter_def* **by** *simp*

with $\langle G \subseteq P \rangle$ **and** $\langle M_generic(G) \rangle$

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)  $\implies$  G  $\subseteq$  M
  using transitivity[OF _ P_in_M] by auto

declare iff_trans [trans]

lemma generic_filter_existence:
  p  $\in$  P  $\implies$   $\exists$  G. p  $\in$  G  $\wedge$  M_generic(G)
proof -
  assume p  $\in$  P
  let ?D =  $\lambda$ n  $\in$  nat. (if (enum' n  $\subseteq$  P  $\wedge$  dense(enum' n)) then enum' n else P)
  have  $\forall$  n  $\in$  nat. ?D' n  $\in$  Pow(P)
    by auto
  then
  have ?D: nat  $\rightarrow$  Pow(P)
    using lam_type by auto
  have Eq4:  $\forall$  n  $\in$  nat. dense(?D' n)
  proof (intro ballI)
    fix n
    assume n  $\in$  nat
    then
    have dense(?D' n)  $\longleftrightarrow$  dense(if enum' n  $\subseteq$  P  $\wedge$  dense(enum' n) then enum' n
else P)
      by simp
    also
    have ...  $\longleftrightarrow$  ( $\neg$ (enum' n  $\subseteq$  P  $\wedge$  dense(enum' n))  $\longrightarrow$  dense(P))
      using split_if by simp
    finally
    show dense(?D' n)
      using P_dense <n  $\in$  nat> by auto
  qed
  from <?D  $\in$  _> and Eq4
  interpret cg: countable_generic P leq one ?D
    by (unfold_locales, auto)
  from <p  $\in$  P>
  obtain G where Eq6: p  $\in$  G  $\wedge$  filter(G)  $\wedge$  ( $\forall$  n  $\in$  nat. (?D' n)  $\cap$  G  $\neq$   $\emptyset$ )
    using cg.countable_rasiowa_sikorski[where M =  $\lambda$ _. M] P_sub_M
      M_countable[THEN bij_is_fun] M_countable[THEN bij_is_surj, THEN
surj_range]
    unfolding cg.D_generic_def by blast
  then
  have Eq7: ( $\forall$  D  $\in$  M. D  $\subseteq$  P  $\wedge$  dense(D)  $\longrightarrow$  D  $\cap$  G  $\neq$   $\emptyset$ )
  proof (intro ballI impI)
    fix D
    assume D  $\in$  M and Eq9: D  $\subseteq$  P  $\wedge$  dense(D)
    have  $\forall$  y  $\in$  M.  $\exists$  x  $\in$  nat. enum' x = y
      using M_countable and bij_is_surj unfolding surj_def by (simp)

```

```

with ⟨D∈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*:

$infinity_ax(\#\#A) \longleftrightarrow (A, [] \models (?φ(i,j,h)))$

```

unfolding infinity_ax_def
by ((rule sep_rules | simp)+)

synthesize ZF_infinity_fm from_schematic ZF_infinity_auto

schematic_goal ZF_choice_auto:
  choice_ax(##A)  $\longleftrightarrow$  (A, []  $\models$  (? $\varphi$ (i,j,h)))
unfolding choice_ax_def
by ((rule sep_rules | simp)+)

synthesize ZF_choice_fm from_schematic ZF_choice_auto

syntax
  _choice :: i ( $\langle AC \rangle$ )
syntax_consts
  _choice  $\equiv$  ZF_choice_fm
translations
  AC  $\rightarrow$  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  $\equiv$  { 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  $\equiv$  ZF_fin  $\cup$  {ZF_choice_fm}

lemma ZFC_fin_type : ZFC_fin  $\subseteq$  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]:
   $\llbracket n \in \text{nat}; p \in \text{formula} \rrbracket \implies \text{Forall}^n(p) \in \text{formula}$ 
by (induct set:nat, auto)

lemma last_init_eq :
assumes l  $\in$  list(A) length(l) = succ(n)
shows  $\exists a \in A. \exists l' \in \text{list}(A). l = l'@[a]$ 
proof-
from  $\langle l \in \_ \rangle \langle \text{length}(\_) = \_ \rangle$ 
have rev(l)  $\in$  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  $\bigwedge n . n < \text{succ}(\text{length}(l)) \implies l = \text{take}(n,l) @ \text{drop}(n,l)$ 
  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)  $\exists k \in \text{nat} . i = \text{succ}(k) \wedge k < \text{succ}(\text{length}(l))$ 
      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  $\exists re \in \text{list}(M) . \exists st \in \text{list}(M) . \text{rest} = re @ st \wedge \text{length}(re) = \text{pred}(n)$ 
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  $length(?re) = pred(n) \ ?re \in list(M) \ ?st \in list(M)$ 
  using  $length\_take[rule\_format, OF \_ \langle pred(n) \in \_ \rangle \langle pred(n) \leq \_ \rangle \langle rest \in \_ \rangle]$ 
  unfolding  $min\_def$ 
  by  $auto$ 
then
show  $?thesis$ 
  using  $rev\_bexI[of \_ \_ \lambda re. \exists st \in list(M). rest = re @ st \wedge length(re) = pred(n)]$ 
   $\langle length(?re) = \_ \rangle \langle rest = \_ \rangle$ 
  by  $auto$ 
qed

```

lemma $sats_nForall$:

```

assumes
   $\varphi \in formula$ 
shows
   $n \in nat \implies ms \in list(M) \implies$ 
   $M, ms \models (Forall^n(\varphi)) \longleftrightarrow$ 
   $(\forall rest \in list(M). length(rest) = n \longrightarrow M, rest @ ms \models \varphi)$ 
proof ( $induct\ n\ arbitrary:ms\ set:nat$ )
  case  $0$ 
  with  $assms$ 
  show  $?case$  by  $simp$ 
next
  case ( $succ\ n$ )
  have  $(\forall rest \in list(M). length(rest) = succ(n) \longrightarrow P(rest, n)) \longleftrightarrow$ 
   $(\forall t \in M. \forall res \in list(M). length(res) = n \longrightarrow P(res @ [t], n))$ 
  if  $n \in nat$  for  $n\ P$ 
  using  $that\ last\_init\_eq$  by  $force$ 
  from  $this[of \_ \lambda rest \_ . (M, rest @ ms \models \varphi)] \langle n \in nat \rangle$ 
  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  $arity\_sats\_iff[of\ \varphi \_ M\ Cons(\_,ms\ @ \_)]\ app\_assoc$ 
  by ( $simp$ )
qed

```

definition

```

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

```

lemma $sep_body_fm_type$ [TC]: $p \in formula \implies sep_body_fm(p) \in formula$
by ($simp\ add: sep_body_fm_def$)

lemma $sats_sep_body_fm$:

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

definition

$ZF_separation_fm :: i \Rightarrow i$ **where**
 $ZF_separation_fm(p) \equiv \text{Forall}^\wedge(\text{pred}(\text{arity}(p)))(\text{sep_body_fm}(p))$

lemma *ZF_separation_fm_type* [TC]: $p \in \text{formula} \implies ZF_separation_fm(p) \in \text{formula}$

by (*simp add: ZF_separation_fm_def*)

lemma *sats_ZF_separation_fm_iff*:

assumes
 $\varphi \in \text{formula}$
shows
 $(M, [] \models (ZF_separation_fm(\varphi)))$
 \longleftrightarrow
 $(\forall env \in \text{list}(M). \text{arity}(\varphi) \leq 1 \#+ \text{length}(env) \longrightarrow$
 $\text{separation}(\#\#M, \lambda x. M, [x] @ env \models \varphi))$

proof (*intro iffI ballI impI*)

let $?n = \text{Arith.pred}(\text{arity}(\varphi))$

fix *env*

assume $M, [] \models ZF_separation_fm(\varphi)$

assume $\text{arity}(\varphi) \leq 1 \#+ \text{length}(env) \text{ env} \in \text{list}(M)$

moreover from *this*

have $\text{arity}(\varphi) \leq \text{succ}(\text{length}(env))$ **by** *simp*

then

obtain *some rest* **where** $\text{some} \in \text{list}(M) \text{ rest} \in \text{list}(M)$

$env = \text{some} @ \text{rest} \text{ length}(\text{some}) = \text{Arith.pred}(\text{arity}(\varphi))$

using *list_split[OF <arity> <succ> <env>]* **by** *force*

moreover from $\langle \varphi \in _ \rangle$

have $\text{arity}(\varphi) \leq \text{succ}(\text{Arith.pred}(\text{arity}(\varphi)))$

using *succpred_leI* **by** *simp*

moreover

note *assms*

moreover

assume $M, [] \models ZF_separation_fm(\varphi)$

moreover from *calculation*

have $M, \text{some} \models \text{sep_body_fm}(\varphi)$

using *sats_nForall[of sep_body_fm(\varphi) ?n]*

unfolding *ZF_separation_fm_def* **by** *simp*

ultimately

show $\text{separation}(\#\#M, \lambda x. M, [x] @ env \models \varphi)$

unfolding *ZF_separation_fm_def*

```

using sats_sep_body_fm[of  $\varphi$  []  $M$  some]
      arity_sats_iff[of  $\varphi$  rest  $M$  [] @ some]
      separation_cong[of ## $M$   $\lambda x. M$ , Cons( $x$ , some @ rest)  $\models \varphi$  _]
by simp
next — almost equal to the previous implication
let ? $n$ =Arith.pred(arity( $\varphi$ ))
assume asm: $\forall env \in list(M). \text{arity}(\varphi) \leq 1 \# + \text{length}(env) \longrightarrow$ 
      separation(## $M$ ,  $\lambda x. M$ , [ $x$ ] @ env  $\models \varphi$ )
{
  fix some
  assume some $\in list(M)$  length(some) = Arith.pred(arity( $\varphi$ ))
  moreover
  note  $\langle \varphi \in \_ \rangle$ 
  moreover from calculation
  have arity( $\varphi$ )  $\leq 1 \# + \text{length}(some)$ 
    using le_trans[OF succpred_leI] succpred_leI by simp
  moreover from calculation and asm
  have separation(## $M$ ,  $\lambda x. M$ , [ $x$ ] @ some  $\models \varphi$ ) by blast
  ultimately
  have  $M, some \models \text{sep\_body\_fm}(\varphi)$ 
  using sats_sep_body_fm[of  $\varphi$  []  $M$  some]
      arity_sats_iff[of  $\varphi$  _  $M$  [],_] @ some]
      strong_replacement_cong[of ## $M$   $\lambda x y. M$ , Cons( $x$ , Cons( $y$ , some @ _))  $\models$ 
 $\varphi$  _]
  by simp
}
with  $\langle \varphi \in \_ \rangle$ 
show  $M, [] \models ZF\_separation\_fm(\varphi)$ 
  using sats_nForall[of sep_body_fm( $\varphi$ ) ? $n$ ]
  unfolding ZF_separation_fm_def
  by simp
qed

```

12.2 The Axiom of Replacement, internalized

schematic_goal sats_univalent_fm_auto:

assumes

$$\begin{aligned}
 Q_iff_sats: & \bigwedge x y z. x \in A \implies y \in A \implies z \in A \implies \\
 & Q(x, z) \longleftrightarrow (A, \text{Cons}(z, \text{Cons}(y, \text{Cons}(x, env)))) \models Q1_fm) \\
 \bigwedge x y z. & x \in A \implies y \in A \implies z \in A \implies \\
 & Q(x, y) \longleftrightarrow (A, \text{Cons}(z, \text{Cons}(y, \text{Cons}(x, env)))) \models Q2_fm)
 \end{aligned}$$

and

$$asms: nth(i, env) = B \ i \in nat \ env \in list(A)$$

shows

$$\text{univalent}(\#\#A, B, Q) \longleftrightarrow A, env \models ?ufm(i)$$

unfolding univalent_def

by (insert asms; (rule sep_rules Q_iff_sats | simp)+)

synthesize_notc *univalent_fm from_schematic sats_univalent_fm_auto*

lemma *univalent_fm_type* [TC]: $q1 \in \text{formula} \implies q2 \in \text{formula} \implies i \in \text{nat} \implies$
 $\text{univalent_fm}(q2, q1, i) \in \text{formula}$
by (*simp add:univalent_fm_def*)

lemma *sats_univalent_fm* :

assumes

$Q_iff_sats: \bigwedge x y z. x \in A \implies y \in A \implies z \in A \implies$
 $Q(x, z) \longleftrightarrow (A, \text{Cons}(z, \text{Cons}(y, \text{Cons}(x, \text{env})))) \models Q1_fm)$
 $\bigwedge x y z. x \in A \implies y \in A \implies z \in A \implies$
 $Q(x, y) \longleftrightarrow (A, \text{Cons}(z, \text{Cons}(y, \text{Cons}(x, \text{env})))) \models Q2_fm)$

and

asms: nth(i, env) = B i ∈ nat env ∈ list(A)

shows

$A, \text{env} \models \text{univalent_fm}(Q1_fm, Q2_fm, i) \longleftrightarrow \text{univalent}(\#\#A, B, Q)$

unfolding *univalent_fm_def using asms sats_univalent_fm_auto[OF Q_iff_sats]*
by *simp*

definition

swap_vars :: $i \Rightarrow i$ **where**

swap_vars(φ) \equiv

$\text{Exists}(\text{Exists}(\text{And}(\text{Equal}(0, 3), \text{And}(\text{Equal}(1, 2), \text{iterates}(\lambda p. \text{incr_bv}(p)'2, 2, \varphi))))))$

lemma *swap_vars_type*[TC] :

$\varphi \in \text{formula} \implies \text{swap_vars}(\varphi) \in \text{formula}$

unfolding *swap_vars_def* **by** *simp*

lemma *sats_swap_vars* :

$[x, y] @ \text{env} \in \text{list}(M) \implies \varphi \in \text{formula} \implies$

$M, [x, y] @ \text{env} \models \text{swap_vars}(\varphi) \longleftrightarrow M, [y, x] @ \text{env} \models \varphi$

unfolding *swap_vars_def*

using *sats_incr_bv_iff [of _ _ M _ [y, x]]* **by** *simp*

definition

univalent_Q1 :: $i \Rightarrow i$ **where**

univalent_Q1(φ) $\equiv \text{incr_bv1}(\text{swap_vars}(\varphi))$

definition

univalent_Q2 :: $i \Rightarrow i$ **where**

univalent_Q2(φ) $\equiv \text{incr_bv}(\text{swap_vars}(\varphi))'0$

lemma *univalent_Qs_type* [TC]:

assumes $\varphi \in \text{formula}$

shows $\text{univalent_Q1}(\varphi) \in \text{formula}$ $\text{univalent_Q2}(\varphi) \in \text{formula}$

unfolding *univalent_Q1_def univalent_Q2_def* **using** *asms* **by** *simp_all*

lemma *sats_univalent_fm_assm*:

assumes

$x \in A \ y \in A \ z \in A \ env \in list(A) \ \varphi \in formula$

shows

$(A, ([x,z] @ env) \models \varphi) \longleftrightarrow (A, Cons(z, Cons(y, Cons(x, env))) \models (univalent_Q1(\varphi)))$

$(A, ([x,y] @ env) \models \varphi) \longleftrightarrow (A, Cons(z, Cons(y, Cons(x, env))) \models (univalent_Q2(\varphi)))$

unfolding *univalent_Q1_def univalent_Q2_def*

using

sats_incr_bv_iff[of _ _ A _ []] — simplifies iterates of $\lambda x. incr_bv(x) \text{ ' } 0$

sats_incr_bv1_iff[of _ Cons(x,env) A z y]

sats_swap_vars assms

by *simp_all*

definition

rep_body_fm :: $i \Rightarrow i$ **where**

rep_body_fm(*p*) \equiv *Forall*(*Implies*(

univalent_fm(*univalent_Q1*(*incr_bv*(*p*)^{'2}), *univalent_Q2*(*incr_bv*(*p*)^{'2}), 0),

Exists(*Forall*(

Iff(*Member*(0,1), *Exists*(*And*(*Member*(0,3), *incr_bv*(*incr_bv*(*p*)^{'2})^{'2}))))))

lemma *rep_body_fm_type* [*TC*]: $p \in formula \Longrightarrow rep_body_fm(p) \in formula$

by (*simp add: rep_body_fm_def*)

lemmas *ZF_replacement_simps* = *formula_add_params1*[*of* φ 2 _ *M* [*_*, *_*]]

sats_incr_bv_iff[*of* _ _ *M* _ []] — simplifies iterates of $\lambda x. incr_bv(x) \text{ ' } 0$

sats_incr_bv_iff[*of* _ _ *M* _ [*_*, *_*]] — simplifies $\lambda x. incr_bv(x) \text{ ' } 2$

sats_incr_bv1_iff[*of* _ _ *M*] *sats_swap_vars* **for** φ *M*

lemma *sats_rep_body_fm*:

assumes

$\varphi \in formula \ ms \in list(M) \ rest \in list(M)$

shows

$M, rest @ ms \models rep_body_fm(\varphi) \longleftrightarrow$

strong_replacement($\#\#M, \lambda x y. M, [x,y] @ rest @ ms \models \varphi$)

using *assms ZF_replacement_simps*

unfolding *rep_body_fm_def strong_replacement_def univalent_def*

unfolding *univalent_fm_def univalent_Q1_def univalent_Q2_def*

by *simp*

definition

ZF_replacement_fm :: $i \Rightarrow i$ **where**

ZF_replacement_fm(*p*) \equiv *Forall*[^](*pred*(*pred*(*arity*(*p*))))(*rep_body_fm*(*p*))

lemma *ZF_replacement_fm_type* [*TC*]: $p \in formula \Longrightarrow ZF_replacement_fm(p) \in formula$

by (*simp add: ZF_replacement_fm_def*)

lemma *sats_ZF_replacement_fm_iff*:

assumes

$\varphi \in formula$

```

shows
(M, [] ⊨ (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( $\varphi$ )
using sats_rep_body_fm[of  $\varphi$  [] M some]
  arity_sats_iff[of  $\varphi$  _ M [_,_] @ some]
  strong_replacement_cong[of ##M  $\lambda x y. M, Cons(x, Cons(y, some @ \_))$ ]  $\models$ 
 $\varphi$  _ ]
  by simp
}
with  $\langle \varphi \in \_ \rangle$ 
show M, []  $\models$  ZF_replacement_fm( $\varphi$ )
  using sats_nForall[of rep_body_fm( $\varphi$ ) ?n]
  unfolding ZF_replacement_fm_def
  by simp
qed

```

definition

```

ZF_inf :: i where
ZF_inf  $\equiv$  {ZF_separation_fm( $p$ ) .  $p \in$  formula }  $\cup$  {ZF_replacement_fm( $p$ ) .
 $p \in$  formula }

```

lemma Un_subset_formula: $A \subseteq$ formula $\wedge B \subseteq$ formula $\implies A \cup B \subseteq$ formula
 by auto

lemma ZF_inf_subset_formula : ZF_inf \subseteq formula
 unfolding ZF_inf_def by auto

definition

```

ZFC :: i where
ZFC  $\equiv$  ZF_inf  $\cup$  ZFC_fin

```

definition

```

ZF :: i where
ZF  $\equiv$  ZF_inf  $\cup$  ZF_fin

```

definition

```

ZF_minus_P :: i where
ZF_minus_P  $\equiv$  ZF - { ZF_power_fm }

```

lemma ZFC_subset_formula: ZFC \subseteq 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]  $\Rightarrow$  o ( $\langle \_ \models \_ \rangle$  [36,36] 60) where
A  $\models \Phi \equiv \forall \varphi \in \Phi. (A, [] \models \varphi)$ 

```

lemma satTI [intro]:
 assumes $\bigwedge \varphi. \varphi \in \Phi \implies A, [] \models \varphi$
 shows $A \models \Phi$

```

using assms unfolding satT_def by simp

lemma satTD [dest] :  $A \models \Phi \implies \varphi \in \Phi \implies A, [] \models \varphi$ 
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  $\varphi$  env
    assume  $\varphi \in \text{formula } env \in \text{list}(M)$ 
    moreover from  $\langle M \models ZF \rangle$ 
    have  $\forall p \in \text{formula}. (M, [] \models (ZF\_separation\_fm(p)))$ 
       $\forall p \in \text{formula}. (M, [] \models (ZF\_replacement\_fm(p)))$ 
    unfolding ZF_def ZF_inf_def by auto
    moreover from calculation
    have  $\text{arity}(\varphi) \leq \text{succ}(\text{length}(\text{env})) \implies \text{separation}(\## M, \lambda x. (M, \text{Cons}(x, \text{env}) \models \varphi))$ 
       $\text{arity}(\varphi) \leq \text{succ}(\text{succ}(\text{length}(\text{env}))) \implies \text{strong\_replacement}(\## M, \lambda x y. \text{sats}(M, \varphi, \text{Cons}(x, \text{Cons}(y, \text{env}))))$ 
    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  $\langle M\_ZF(M) \rangle$ 
  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  $\langle M\_ZF(M) \rangle$ 
  have  $\forall p \in \text{formula}. (M, [] \models (ZF\_separation\_fm(p)))$ 
     $\forall p \in \text{formula}. (M, [] \models (ZF\_replacement\_fm(p)))$ 
  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 $n \in \text{nat}$ $m \in \text{nat}$ $f \in n \rightarrow m$ $x \in \text{nat}$

shows $f'x \in \text{nat}$

proof(*cases* $x \in n$)

case *True*

then show *?thesis* **using** *assms in_n_in_nat apply_type* **by** *simp*

next

case *False*

then show *?thesis* **using** *assms apply_0 domain_of_fun* **by** *simp*

qed

13.1 Renaming of free variables

definition

union_fun :: $[i, i, i, i] \Rightarrow i$ **where**

union_fun(f, g, m, p) $\equiv \lambda j \in m \cup p$. *if* $j \in m$ *then* $f'j$ *else* $g'j$

lemma *union_fun_type*:

assumes $f \in m \rightarrow n$

$g \in p \rightarrow q$

shows $\text{union_fun}(f, g, m, p) \in m \cup p \rightarrow n \cup q$

proof -

let $?h = \text{union_fun}(f, g, m, p)$

have

D: $?h'x \in n \cup q$ **if** $x \in m \cup p$ **for** x

proof (*cases* $x \in m$)

case *True*

then have

$x \in m \cup p$ **by** *simp*

with $\langle x \in m \rangle$

have $?h'x = f'x$

unfolding *union_fun_def* *beta* **by** *simp*

with $\langle f \in m \rightarrow n \rangle \langle x \in m \rangle$

have $?h'x \in n$ **by** *simp*

then show *?thesis* ..

next

case *False*

```

with  $\langle x \in m \cup p \rangle$ 
have  $x \in p$ 
  by auto
with  $\langle x \notin m \rangle$ 
have  $?h'x = g'x$ 
  unfolding union_fun_def using beta by simp
with  $\langle g \in p \rightarrow q \rangle \langle x \in p \rangle$ 
have  $?h'x \in q$  by simp
then show ?thesis ..
qed
have  $A: \text{function}(?h)$  unfolding union_fun_def using function_lam by simp
have  $x \in (m \cup p) \times (n \cup q)$  if  $x \in ?h$  for  $x$ 
  using that lamE[of x m U p _ x \in (m U p) \times (n U q)] D unfolding
union_fun_def
  by auto
then have  $B: ?h \subseteq (m \cup p) \times (n \cup q)$  ..
have  $m \cup p \subseteq \text{domain}(?h)$ 
  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
   $env \in \text{list}(M)$ 
   $env' \in \text{list}(M)$ 
   $\text{length}(env) = m \cup p$ 
   $\forall i . i \in m \rightarrow \text{nth}(f'i, env') = \text{nth}(i, env)$ 
   $\forall j . j \in p \rightarrow \text{nth}(g'j, env') = \text{nth}(j, env)$ 
shows  $\forall i . i \in m \cup p \rightarrow$ 
   $\text{nth}(i, env) = \text{nth}(\text{union\_fun}(f, g, m, p)'i, env')$ 
proof -
let  $?h = \text{union\_fun}(f, g, m, p)$ 
have  $\text{nth}(x, env) = \text{nth}(?h'x, env')$  if  $x \in m \cup p$  for  $x$ 
  using that
proof (cases x \in m)
  case True
with  $\langle x \in m \rangle$ 
have  $?h'x = f'x$ 
  unfolding union_fun_def beta by simp
with assms  $\langle x \in m \rangle$ 
have  $\text{nth}(x, env) = \text{nth}(?h'x, env')$  by simp
then show ?thesis .
next
case False
with  $\langle x \in m \cup p \rangle$ 
have
   $x \in p \wedge x \notin m$  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  $\bigwedge j . j < n \implies \text{nth}(j, \text{env}) = \text{nth}(\text{id}(n)'j, \text{env})$ 
proof -
  show  $\text{nth}(j, \text{env}) = \text{nth}(\text{id}(n)'j, \text{env})$  if  $j < n$  for  $j$  using that ⟨n∈nat⟩ ltD by
  simp
qed

```

definition

```

sum :: [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  $sum(f,g,m,n,p) \text{ `}x = g \text{ `}(x\#-m)\#+n$ 
proof -
  from assms
  have  $x \in nat$ 
    using in_n_in_nat[of  $m\#+p$ ] ltD
    by simp
  with assms
  have  $\neg x < m$ 
    using not_lt_iff_le[THEN iffD2] by simp
  from assms
  have  $x \in m\#+p$ 
    using ltD by simp
  with  $\langle \neg x < m \rangle$ 
  show ?thesis unfolding sum_def by simp
qed

```

lemma *sum_action* :

```

assumes  $m \in nat \ n \in nat \ p \in nat \ q \in nat$ 
   $f \in m \rightarrow n \ g \in p \rightarrow q$ 
   $env \in list(M)$ 
   $env' \in list(M)$ 
   $env1 \in list(M)$ 
   $env2 \in list(M)$ 
   $length(env) = m$ 
   $length(env1) = p$ 
   $length(env') = n$ 
   $\wedge i . i < m \implies nth(i,env) = nth(f \text{ `}i,env')$ 
   $\wedge j . j < p \implies nth(j,env1) = nth(g \text{ `}j,env2)$ 
shows  $\forall i . i < m\#+p \longrightarrow$ 
   $nth(i,env@env1) = nth(sum(f,g,m,n,p) \text{ `}i,env'@env2)$ 

```

proof -

```

let ?h = sum(f,g,m,n,p)
from  $\langle m \in nat \rangle \langle n \in nat \rangle \langle q \in nat \rangle$ 
have  $m \leq m\#+p \ n \leq n\#+q \ q \leq n\#+q$ 
  using add_le_self[of m] add_le_self2[of n q] by simp_all
from  $\langle p \in nat \rangle$ 
have  $p = (m\#+p)\#-m$  using diff_add_inverse2 by simp
have  $nth(x, env @ env1) = nth(\text{?h} \text{ `}x,env'@env2)$  if  $x < m\#+p$  for x
proof (cases  $x < m$ )
  case True
  then
  have ?:  $\text{?h} \text{ `}x = f \text{ `}x \ x \in m \ f \text{ `}x \in n \ x \in nat$ 
    using assms sum_inl ltD apply_type[of f m _ x] in_n_in_nat by simp_all
  with  $\langle x < m \rangle$  assms
  have  $f \text{ `}x < n \ f \text{ `}x < length(env') \ f \text{ `}x \in nat$ 
    using ltI in_n_in_nat by simp_all
  with ?  $\langle x < m \rangle$  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 ‹q∈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

$f \in \text{length}(env) \rightarrow \text{length}(env')$

$env \in \text{list}(M)$

$env' \in \text{list}(M)$

$env1 \in \text{list}(M)$

shows

$\text{sum}(f, \text{id}(\text{length}(env1)), \text{length}(env), \text{length}(env'), \text{length}(env1)) \in$
 $(\text{length}(env) \#+ \text{length}(env1)) \rightarrow (\text{length}(env') \#+ \text{length}(env1))$

using *assms length_type_id_fn_type sum_type*

by *simp*

lemma *sum_type_id_aux2* :

assumes

$f \in m \rightarrow n$

$m \in \text{nat } n \in \text{nat}$

$env1 \in \text{list}(M)$

by auto

definition

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

lemma $sum_id0 : m \in nat \Longrightarrow sum_id(m,f) \cdot 0 = 0$
by (unfold sum_id_def, subst sum_inl, auto)

lemma $sum_idS : p \in nat \Longrightarrow q \in nat \Longrightarrow f \in p \rightarrow q \Longrightarrow x \in p \Longrightarrow sum_id(p,f) \cdot (succ(x)) = succ(f \cdot x)$
by (subgoal_tac $x \in nat$, unfold sum_id_def, subst sum_inr, simp_all add: ltI, simp_all add: app_nm in_n in_nat)

lemma $sum_id_tc_aux :$
 $p \in nat \Longrightarrow q \in nat \Longrightarrow f \in p \rightarrow q \Longrightarrow sum_id(p,f) \in 1 \# + p \rightarrow 1 \# + q$
by (unfold sum_id_def, rule sum_type, simp_all)

lemma $sum_id_tc :$
 $n \in nat \Longrightarrow m \in nat \Longrightarrow f \in n \rightarrow m \Longrightarrow sum_id(n,f) \in succ(n) \rightarrow succ(m)$
by (rule ssubst[of $succ(n) \rightarrow succ(m)$ $1 \# + n \rightarrow 1 \# + m$], simp, rule sum_id_tc_aux, simp_all)

13.2 Renaming of formulas

consts $ren :: i \Rightarrow i$

primrec

$ren(Member(x,y)) =$
 $(\lambda n \in nat . \lambda m \in nat . \lambda f \in n \rightarrow m . Member(f \cdot x, f \cdot y))$

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

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

$ren(Forall(p)) =$
 $(\lambda n \in nat . \lambda m \in nat . \lambda f \in n \rightarrow m . Forall(ren(p) \cdot succ(n) \cdot succ(m) \cdot sum_id(n,f)))$

lemma $arity_meml : l \in nat \Longrightarrow Member(x,y) \in formula \Longrightarrow arity(Member(x,y)) \leq l \Longrightarrow x \in l$

by (simp, rule subsetD, rule le_imp_subset, assumption, simp)

lemma $arity_memr : l \in nat \Longrightarrow Member(x,y) \in formula \Longrightarrow arity(Member(x,y)) \leq l \Longrightarrow y \in l$

by (simp, rule subsetD, rule le_imp_subset, assumption, simp)

lemma $arity_eq1 : l \in nat \Longrightarrow Equal(x,y) \in formula \Longrightarrow arity(Equal(x,y)) \leq l \Longrightarrow x \in l$

by (simp, rule subsetD, rule le_imp_subset, assumption, simp)

lemma *arity_eqr* : $l \in \text{nat} \implies \text{Equal}(x,y) \in \text{formula} \implies \text{arity}(\text{Equal}(x,y)) \leq l \implies y \in l$

by (*simp*, *rule subsetD*, *rule le_imp_subset*, *assumption*, *simp*)

lemma *nand_ar1* : $p \in \text{formula} \implies q \in \text{formula} \implies \text{arity}(p) \leq \text{arity}(\text{Nand}(p,q))$

by (*simp*, *rule Un_upper1_le*, *simp+*)

lemma *nand_ar2* : $p \in \text{formula} \implies q \in \text{formula} \implies \text{arity}(q) \leq \text{arity}(\text{Nand}(p,q))$

by (*simp*, *rule Un_upper2_le*, *simp+*)

lemma *nand_ar1D* : $p \in \text{formula} \implies q \in \text{formula} \implies \text{arity}(\text{Nand}(p,q)) \leq n \implies \text{arity}(p) \leq n$

by (*auto simp add: le_trans[OF Un_upper1_le[of arity(p) arity(q)]]*)

lemma *nand_ar2D* : $p \in \text{formula} \implies q \in \text{formula} \implies \text{arity}(\text{Nand}(p,q)) \leq n \implies \text{arity}(q) \leq n$

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

lemma *ren_tc* : $p \in \text{formula} \implies$

$(\bigwedge n m f . n \in \text{nat} \implies m \in \text{nat} \implies f \in n \rightarrow m \implies \text{ren}(p) 'n 'm 'f \in \text{formula})$

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

lemma *arity_ren* :

fixes *p*

assumes $p \in \text{formula}$

shows $\bigwedge n m f . n \in \text{nat} \implies m \in \text{nat} \implies f \in n \rightarrow m \implies \text{arity}(p) \leq n \implies \text{arity}(\text{ren}(p) 'n 'm 'f) \leq m$

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 $\text{arity}(p) \leq \text{arity}(\text{Nand}(p,q))$

$\text{arity}(q) \leq \text{arity}(\text{Nand}(p,q))$

by (*subst nand_ar1*, *simp*, *simp*, *simp*, *subst nand_ar2*, *simp+*)

then have $\text{arity}(p) \leq n$

and $\text{arity}(q) \leq n$ **using** *Nand*

by (*rule_tac j=arity(Nand(p,q)) in le_trans*, *simp*, *simp+*)

then have $\text{arity}(\text{ren}(p) 'n 'm 'f) \leq m$ **and** $\text{arity}(\text{ren}(q) 'n 'm 'f) \leq m$

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  $\varphi$ 
assumes  $\varphi \in \text{formula}$ 
shows  $\llbracket n \in \text{nat} ; m \in \text{nat} ; \varrho \in \text{list}(M) ; \varrho' \in \text{list}(M) ; f \in n \rightarrow m ;$ 
 $\text{arity}(\varphi) \leq n ;$ 
 $\bigwedge i . i < n \implies \text{nth}(i, \varrho) = \text{nth}(f^i, \varrho') \rrbracket \implies$ 
 $\text{sats}(M, \varphi, \varrho) \longleftrightarrow \text{sats}(M, \text{ren}(\varphi) 'n 'm 'f, \varrho')$ 
using  $\langle \varphi \in \text{formula} \rangle$ 
proof(induct  $\varphi$  arbitrary: $n$   $m$   $\varrho$   $\varrho'$   $f$ )
  case (Member  $x$   $y$ )
    have  $\text{ren}(\text{Member}(x, y)) 'n 'm 'f = \text{Member}(f^x, f^y)$  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  $\text{ren}(\text{Equal}(x, y)) 'n 'm 'f = \text{Equal}(f^x, f^y)$  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  $\text{ren}(\text{Nand}(p, q)) 'n 'm 'f = \text{Nand}(\text{ren}(p) 'n 'm 'f, \text{ren}(q) 'n 'm 'f)$  using Nand by
simp
    moreover
    have  $\text{arity}(p) \leq n$  using Nand nand_ar1D by simp
    moreover from this
    have  $i \in \text{arity}(p) \implies i \in n$  for  $i$  using subsetD[OF le_imp_subset[OF  $\langle \text{arity}(p) \leq n \rangle$ ]] by simp
    moreover from this
    have  $i \in \text{arity}(p) \implies \text{nth}(i, \varrho) = \text{nth}(f^i, \varrho')$  for  $i$  using Nand ltI by simp
    moreover from this
    have  $\text{sats}(M, p, \varrho) \longleftrightarrow \text{sats}(M, \text{ren}(p) 'n 'm 'f, \varrho')$  using  $\langle \text{arity}(p) \leq n \rangle$  Nand by
simp
    have  $\text{arity}(q) \leq n$  using Nand nand_ar2D by simp
    moreover from this
    have  $i \in \text{arity}(q) \implies i \in n$  for  $i$  using subsetD[OF le_imp_subset[OF  $\langle \text{arity}(q) \leq n \rangle$ ]] by simp
    moreover from this
    have  $i \in \text{arity}(q) \implies \text{nth}(i, \varrho) = \text{nth}(f^i, \varrho')$  for  $i$  using Nand ltI by simp
    moreover from this
    have  $\text{sats}(M, q, \varrho) \longleftrightarrow \text{sats}(M, \text{ren}(q) 'n 'm 'f, \varrho')$  using assms  $\langle \text{arity}(q) \leq n \rangle$  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
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

$_SepReplace :: [i, pttm, i, o] \Rightarrow i$ ($\langle (1\{ _ .. / _ \in _, _ \} \rangle)$)

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*) $y \in A \mid$ (*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 ∈ eclose(B) B ∈ A
    by auto
  then
    show ?thesis using 2 ecloseD[of y B z] by auto
qed

```

```

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

```

```

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

```

```

lemma in_dom_in_eclose :
  assumes x ∈ domain(z)
  shows x ∈ eclose(z)
proof -
  from assms
  obtain y where ⟨x, y⟩ ∈ 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

```

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

```

definition
  ed :: [i, i] ⇒ o where
  ed(x, y) ≡ x ∈ domain(y)

```

```

definition
  edrel :: i ⇒ i where

```

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

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

lemma $edD[dest!]: ed(t, x) \implies t \in domain(x)$
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 $edrel_dest [dest]: x \in edrel(A) \implies \exists a \in A. \exists b \in A. x = \langle a, b \rangle$
by $(auto simp add: ed_def edrel_def Rrel_def)$

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

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

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

lemma $domain_trans: Transset(A) \implies y \in A \implies x \in domain(y) \implies x \in A$
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  $\bigwedge x. [\bigwedge y. ed(y, x) \implies Q(y)] \implies Q(x)$ 
  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 asms[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 $z \in \text{domain}(x)$
shows $\text{tr_down}(\text{edrel}(\text{eclose}(\{x\})), z) \subseteq \text{eclose}(\{z\})$
proof(*intro subsetI*)
let $?r = \lambda x . \text{edrel}(\text{eclose}(\{x\}))$
fix y
assume $y \in \text{tr_down}(?r(x), z)$
then
have $\langle y, z \rangle \in ?r(x)^{\wedge+}$ **using** *tr_downD* **by** *simp*
with *assms*
show $y \in \text{eclose}(\{z\})$ **using** *tr_edrel_eclose_eclose_sing* **by** *simp*
qed

context *M_ctm*
begin

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

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

lemma *Rep_simp* : $\text{Replace}(u, \lambda y z . z = f(y)) = \{ f(y) . y \in u \}$
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**
 $check(x) \equiv \text{transrec}(x, Hcheck)$

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

definition

rcheck :: $i \Rightarrow i$ **where**
rcheck(x) \equiv $\text{Memrel}(\text{eclose}(\{x\}))^{\wedge+}$

lemma *Hcheck_trancl*: $H\text{check}(y, \text{restrict}(f, \text{Memrel}(\text{eclose}(\{x\})) - \{\{y\}\}))$
 $= H\text{check}(y, \text{restrict}(f, (\text{Memrel}(\text{eclose}(\{x\}))^{\wedge+}) - \{\{y\}\}))$
unfolding *Hcheck_def*
using *restrict_trans_eq* **by** *simp*

lemma *check_trancl*: $\text{check}(x) = \text{wfrec}(\text{rcheck}(x), x, H\text{check})$
using *checkD wf_eq_trancl Hcheck_trancl* **unfolding** *rcheck_def* **by** *simp*

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

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

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

proof -

let

$?r = \lambda y. \text{Memrel}(\text{eclose}(\{y\}))$

have *wfr*: $\forall w . \text{wf}(\text{?r}(w))$

using *wf_Memrel ..*

then

have $\text{check}(y) = H\text{check}(y, \lambda x \in \text{?r}(y) . \{\{y\}\}. \text{wfrec}(\text{?r}(y), x, H\text{check}))$

using *wfrec[of ?r(y) y Hcheck] checkD* **by** *simp*

also

have $\dots = H\text{check}(y, \lambda x \in y. \text{wfrec}(\text{?r}(y), x, H\text{check}))$

using *under_Memrel_eclose arg_into_eclose* **by** *simp*

also

have $\dots = H\text{check}(y, \lambda x \in y. \text{check}(x))$

using *aux_def_check checkD* **by** *simp*

finally show *?thesis* **using** *Hcheck_def* **by** *simp*

qed

lemma *def_checkS* :

fixes n

assumes $n \in \text{nat}$

shows $\text{check}(\text{succ}(n)) = \text{check}(n) \cup \{ \langle \text{check}(n), \text{one} \rangle \}$

proof -

have $\text{check}(\text{succ}(n)) = \{ \langle \text{check}(i), \text{one} \rangle . i \in \text{succ}(n) \}$

using *def_check* **by** *blast*

also have ... = $\{\langle \text{check}(i), \text{one} \rangle . i \in n\} \cup \{\langle \text{check}(n), \text{one} \rangle\}$
by *blast*
also have ... = $\text{check}(n) \cup \{\langle \text{check}(n), \text{one} \rangle\}$
using *def_check[of n,symmetric]* **by** *simp*
finally show *?thesis* .
qed

lemma *field_Memrel2* :
assumes $x \in M$
shows $\text{field}(\text{Memrel}(\text{eclose}(\{x\}))) \subseteq M$
proof -
have $\text{field}(\text{Memrel}(\text{eclose}(\{x\}))) \subseteq \text{eclose}(\{x\})$ $\text{eclose}(\{x\}) \subseteq M$
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
 $Hv :: i \Rightarrow i \Rightarrow i$ **where**
 $Hv(G, x, f) \equiv \{ f'y .. y \in \text{domain}(x), \exists p \in P. \langle y, p \rangle \in x \wedge p \in G \}$

The function *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**
 $val(G, \tau) \equiv \text{wfrec}(\text{edrel}(\text{eclose}(\{\tau\})), \tau, Hv(G))$

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

The next lemma provides the usual recursive expression for the definition of *termval*.

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

$?f = \lambda z \in ?r(x) . \text{wfrec} (?r(x), z, Hv(G))$

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* : $x \subseteq y \implies val(G,x) \subseteq val(G,y)$

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* : $one \in G \implies one \in P \implies val(G,check(y)) = y$

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* :

$val(G, \{x \in A \times P. Q(x)\}) = \{val(G, t) .. t \in A, \exists p \in P. Q(\langle t, p \rangle) \wedge p \in G\}$

proof -

let

$?n = \{x \in A \times P. Q(x)\}$ **and**
 $?r = \lambda \tau. edrel(eclose(\{\tau\}))$

let

$?f = \lambda z \in ?r(?n). val(G, z)$

have

$wfR : wf(?r(\tau))$ **for** τ
by (*simp add: wf_edrel*)

have $domain(?n) \subseteq A$ **by** *auto*

{ fix t

assume $H : t \in domain(\{x \in A \times P. Q(x)\})$

then have $?f' t = (if t \in ?r(?n) then val(G, t) else 0)$

by *simp*

moreover have $... = val(G, t)$

using *dom_under_edrel_eclose H if_P by auto*

}

then

have $Eq1 : t \in domain(\{x \in A \times P. Q(x)\}) \implies val(G, t) = ?f' t$ **for** t

by *simp*

have $val(G, ?n) = \{val(G, t) .. t \in domain(?n), \exists p \in P. \langle t, p \rangle \in ?n \wedge p \in G\}$

by (*subst def_val, simp*)

also

have $... = \{?f' t .. t \in domain(?n), \exists p \in P. \langle t, p \rangle \in ?n \wedge p \in G\}$

unfolding *Hv_def*

by (*subst SepReplace_dom_implies, auto simp add: Eq1*)

also

have $... = \{(if t \in ?r(?n) then val(G, t) else 0) .. t \in domain(?n), \exists p \in P. \langle t, p \rangle \in ?n \wedge p \in G\}$

by (*simp*)

also

have $Eq2 : ... = \{val(G, t) .. t \in domain(?n), \exists p \in P. \langle t, p \rangle \in ?n \wedge p \in G\}$

proof -

have $domain(?n) \subseteq ?r(?n)$

using *dom_under_edrel_eclose by simp*

then

have $\forall t \in domain(?n). (if t \in ?r(?n) then val(G, t) else 0) = val(G, t)$

by *auto*

then

show $\{(if t \in ?r(?n) then val(G, t) else 0) .. t \in domain(?n), \exists p \in P. \langle t, p \rangle \in ?n \wedge p \in G\} =$

$\{val(G, t) .. t \in domain(?n), \exists p \in P. \langle t, p \rangle \in ?n \wedge p \in G\}$

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, τ) = val($F, \{x \in \tau . \exists t \in \text{domain}(\tau) . \exists p \in P . x = \langle t, p \rangle\}$)
(is _ = val($F, ?name$))

proof -

have val($F, ?name$) = { val(F, t) .. $t \in \text{domain}(?name), \exists p \in P . \langle t, p \rangle \in ?name \wedge p \in F$ }

using *def_val* **by blast**

also

have ... = { val(F, t) .. $t \in \{y \in \text{domain}(?name) . \exists p \in P . \langle y, p \rangle \in ?name \wedge p \in F\}$ }

using *Sep_and_Replace* **by simp**

also

have ... = { val(F, t) .. $t \in \{y \in \text{domain}(\tau) . \exists p \in P . \langle y, p \rangle \in \tau \wedge p \in F\}$ }

by blast

also

have ... = { val(F, t) .. $t \in \text{domain}(\tau), \exists p \in P . \langle t, p \rangle \in \tau \wedge p \in F$ }

using *Sep_and_Replace* **by simp**

also

have ... = val(F, τ)

using *def_val[symmetric]* **by blast**

finally

show *?thesis* ..

qed

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

proof

have val(F, τ) = val($F, \{x \in \tau . \exists t \in \text{domain}(\tau) . \exists p \in P . x = \langle t, p \rangle\}$)

(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, τ) \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, τ)

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 $F \ \tau$] *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 :: $i \Rightarrow i \quad (\langle M[_] \rangle)$
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 $\vartheta \ p$ **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:
   $x \in M[G] \implies \exists \tau \in M. x = \text{val}(G, \tau)$ 
  by (simp add: GenExt_def)

lemma GenExtI:
   $x \in M \implies \text{val}(G, x) \in M[G]$ 
  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  $c \in M$   $\text{val}(G, c) \in M[G]$   $y \in \text{val}(G, c)$ 
      using GenExtD by auto
    from  $\langle y \in \text{val}(G, c) \rangle$ 
    obtain  $\vartheta$  where  $\vartheta \in \text{domain}(c)$   $\text{val}(G, \vartheta) = y$ 
      using elem_of_val by blast
    with trans_M  $\langle c \in M \rangle$ 
    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  $n \in \text{nat}$ 
  shows  $\text{check}(n) \in M$ 
  using  $\langle n \in \text{nat} \rangle$ 
proof (induct n)
  case 0
  then show ?case using zero_in_M by (subst def_check, simp)
next
  case (succ x)
  have  $\text{one} \in M$  using one_in_P P_sub_M subsetD by simp
  with  $\langle \text{check}(x) \in M \rangle$ 
  have  $\langle \text{check}(x), \text{one} \rangle \in M$ 
    using tuples_in_M by simp
  then
  have  $\{ \langle \text{check}(x), \text{one} \rangle \} \in M$ 
    using singletonM by simp
  with  $\langle \text{check}(x) \in M \rangle$ 
  have  $\text{check}(x) \cup \{ \langle \text{check}(x), \text{one} \rangle \} \in M$ 
    using Un_closed by simp
  then show ?case using  $\langle x \in \text{nat} \rangle$  def_checkS by simp

```

qed

definition

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

definition

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

lemma one_in_M : $one \in M$

by ($insert_one_in_P$ P_in_M , $simp$ add : $transitivity$)

lemma $def_PHcheck$:

assumes

$z \in M$ $f \in M$

shows

$Hcheck(z,f) = Replace(z,PHcheck(one,f))$

proof -

from $assms$

have $\langle f'x, one \rangle \in M$ $f'x \in M$ **if** $x \in z$ **for** x

using $tuples_in_M$ one_in_M $transitivity$ **that** $apply_closed$ **by** $simp_all$

then

have $\{y . x \in z, y = \langle f'x, one \rangle\} = \{y . x \in z, y = \langle f'x, one \rangle \wedge y \in M \wedge f'x \in M\}$

by $simp$

then

show $?thesis$

using $\langle z \in M \rangle$ $\langle f \in M \rangle$ $transitivity$

unfolding $Hcheck_def$ $PHcheck_def$ $RepFun_def$

by $auto$

qed

definition

$PHcheck_fm :: [i,i,i,i] \Rightarrow i$ **where**
 $PHcheck_fm(o,f,y,p) \equiv Exists(And(fun_apply_fm(succ(f),succ(y),0), pair_fm(0,succ(o),succ(p))))$

lemma $PHcheck_type$ [TC]:

$\llbracket x \in nat; y \in nat; z \in nat; u \in nat \rrbracket \Longrightarrow PHcheck_fm(x,y,z,u) \in formula$
by ($simp$ add : $PHcheck_fm_def$)

lemma $sats_PHcheck_fm$ [$simp$]:

$\llbracket x \in nat; y \in nat; z \in nat; u \in nat; env \in list(M) \rrbracket$

$\Longrightarrow sats(M,PHcheck_fm(x,y,z,u),env) \longleftrightarrow$

$PHcheck(nth(x,env),nth(y,env),nth(z,env),nth(u,env))$

using $zero_in_M$ $Internalizations.nth_closed$ **by** ($simp$ add : $PHcheck_def$ $PHcheck_fm_def$)

definition

$is_Hcheck_fm :: [i,i,i,i] \Rightarrow i$ **where**
 $is_Hcheck_fm(o,z,f,hc) \equiv Replace_fm(z,PHcheck_fm(succ(succ(o)),succ(succ(f))),0,1),hc)$

lemma is_Hcheck_type [TC]:

$\llbracket x \in nat; y \in nat; z \in nat; u \in nat \rrbracket \Longrightarrow is_Hcheck_fm(x,y,z,u) \in formula$
by ($simp$ $add:is_Hcheck_fm_def$)

lemma $sats_is_Hcheck_fm$ [$simp$]:

$\llbracket x \in nat; y \in nat; z \in nat; u \in nat ; env \in list(M) \rrbracket$
 $\Longrightarrow sats(M,is_Hcheck_fm(x,y,z,u),env) \longleftrightarrow$
 $is_Hcheck(nth(x,env),nth(y,env),nth(z,env),nth(u,env))$
using $sats_Replace_fm$ **unfolding** is_Hcheck_def $is_Hcheck_fm_def$
by $simp$

lemma $wfrec_Hcheck$:

assumes

$X \in M$

shows

$wfrec_replacement(\#\#M,is_Hcheck(one),rcheck(X))$

proof -

have $is_Hcheck(one,a,b,c) \longleftrightarrow$

$sats(M,is_Hcheck_fm(8,2,1,0),[c,b,a,d,e,y,x,z,one,rcheck(x)])$

if $a \in M$ $b \in M$ $c \in M$ $d \in M$ $e \in M$ $y \in M$ $x \in M$ $z \in M$

for a b c d e y x z

using $that$ one_in_M $\langle X \in M \rangle$ $rcheck_in_M$ **by** $simp$

then have $1:sats(M,is_wfrec_fm(is_Hcheck_fm(8,2,1,0),4,1,0),$

$[y,x,z,one,rcheck(X)]) \longleftrightarrow$

$is_wfrec(\#\#M, is_Hcheck(one),rcheck(X), x, y)$

if $x \in M$ $y \in M$ $z \in M$ **for** x y z

using $that$ $sats_is_wfrec_fm$ $\langle X \in M \rangle$ $rcheck_in_M$ 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 $satsf:sats(M, ?f, [x,z,one,rcheck(X)]) \longleftrightarrow$

$(\exists y \in M. pair(\#\#M,x,y,z) \ \& \ is_wfrec(\#\#M, is_Hcheck(one),rcheck(X),$

$x, y))$

if $x \in M$ $z \in M$ **for** x z

using $that$ 1 $\langle X \in M \rangle$ $rcheck_in_M$ 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,λx z. sats(M,?f,[x,z,one,rcheck(X)]))
  using replacement_ax 1 artyf ⟨X∈M⟩ rcheck_in_M one_in_M by simp
then
have strong_replacement(##M,λx z.
  ∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, is_Hcheck(one),rcheck(X),
x, y))
  using repl_sats[of M ?f [one,rcheck(X)]] satsf by (simp del:pair_abs)
then
show ?thesis unfolding wfrec_replacement_def by simp
qed

```

```

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

```

```

lemma univ_PHcheck : [ z∈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* :

$\forall y \in M. \forall g \in M. \text{function}(g) \longrightarrow Hcheck(y,g) \in M$

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* : $x \in M \implies wf(rcheck(x))$

unfolding *rcheck_def* **using** *wf_trancl*[*OF wf_Memrel*] .

lemma *trans_rcheck* : $x \in M \implies trans(rcheck(x))$

unfolding *rcheck_def* **using** *trans_trancl* .

lemma *relation_rcheck* : $x \in M \implies relation(rcheck(x))$

unfolding *rcheck_def* **using** *relation_trancl* .

lemma *check_in_M* : $x \in M \implies check(x) \in M$

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**

$is_singleton(A, x, z) \equiv \exists c[A]. \text{empty}(A, c) \wedge is_cons(A, x, c, z)$

lemma (**in** *M_trivial*) *singleton_abs*[*simp*] : $\llbracket M(x) ; M(s) \rrbracket \implies is_singleton(M, x, s)$
 $\longleftrightarrow s = \{x\}$

unfolding *is_singleton_def* **using** *nonempty* **by** *simp*

definition

singleton_fm :: $[i, i] \Rightarrow i$ **where**

$singleton_fm(i, j) \equiv \text{Exists}(\text{And}(\text{empty_fm}(0), \text{cons_fm}(\text{succ}(i), 0, \text{succ}(j))))$

lemma *singleton_type*[*TC*] : $\llbracket x \in nat; y \in nat \rrbracket \implies singleton_fm(x, y) \in \text{formula}$

unfolding *singleton_fm_def* **by** *simp*

lemma *is_singleton_iff_sats*:

$\llbracket nth(i, env) = x; nth(j, env) = y; \rrbracket$

$i \in \text{nat}; j \in \text{nat}; \text{env} \in \text{list}(A)$
 $\implies \text{is_singleton}(\#\#A, x, y) \longleftrightarrow \text{sats}(A, \text{singleton_fm}(i, j), \text{env})$
unfolding is_singleton_def singleton_fm_def **by** simp

context forcing_data **begin**

definition

$\text{is_rcheck} :: [i, i] \Rightarrow o$ **where**
 $\text{is_rcheck}(x, z) \equiv \exists r \in M. \text{tran_closure}(\#\#M, r, z) \wedge (\exists ec \in M. \text{membership}(\#\#M, ec, r)$
 \wedge
 $(\exists s \in M. \text{is_singleton}(\#\#M, x, s) \wedge \text{is_eclose}(\#\#M, s, ec)))$

lemma rcheck_abs :

$\llbracket x \in M; r \in M \rrbracket \implies \text{is_rcheck}(x, r) \longleftrightarrow r = \text{rcheck}(x)$
unfolding rcheck_def is_rcheck_def
using singletonM trancl_closed Memrel_closed eclose_closed **by** simp

schematic_goal rcheck_fm_auto :

assumes
 $i \in \text{nat}$ $j \in \text{nat}$ $\text{env} \in \text{list}(M)$
shows
 $\text{is_rcheck}(\text{nth}(i, \text{env}), \text{nth}(j, \text{env})) \longleftrightarrow \text{sats}(M, ?\text{rch}(i, j), \text{env})$
unfolding is_rcheck_def
by (insert assms ; (rule sep_rules $\text{is_singleton_iff_sats}$ $\text{is_eclose_iff_sats}$ $\text{trans_closure_fm_iff_sats}$ | simp)+)

synthesize rcheck_fm **from_schematic** rcheck_fm_auto

definition

$\text{is_check} :: [i, i] \Rightarrow o$ **where**
 $\text{is_check}(x, z) \equiv \exists \text{rch} \in M. \text{is_rcheck}(x, \text{rch}) \wedge \text{is_wfrec}(\#\#M, \text{is_Hcheck}(\text{one}), \text{rch}, x, z)$

lemma check_abs :

assumes
 $x \in M$ $z \in M$
shows
 $\text{is_check}(x, z) \longleftrightarrow z = \text{check}(x)$
proof -
have
 $\text{is_check}(x, z) \longleftrightarrow \text{is_wfrec}(\#\#M, \text{is_Hcheck}(\text{one}), \text{rcheck}(x), x, z)$
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 $\text{rcheck}(x)$ x z $\text{is_Hcheck}(\text{one})$ Hcheck]
by (simp flip : setclass_iff)

qed

definition

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

lemma $check_fm_type[TC]$:

$\llbracket x \in nat; o \in nat; z \in nat \rrbracket \Longrightarrow check_fm(x,o,z) \in formula$
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 $sats_is_Hcheck_fm$:

$\bigwedge a0\ a1\ a2\ a3\ a4. \llbracket a0 \in M; a1 \in M; a2 \in M; a3 \in M; a4 \in M \rrbracket \Longrightarrow$

$is_Hcheck(one,a2, a1, a0) \longleftrightarrow$

$sats(M, is_Hcheck_fm(6\#+o,2,1,0), [a0,a1,a2,a3,a4,r]@env)$ **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 $M \times P$] *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* : $\llbracket p \in M ; y \in M \rrbracket \implies (\exists c \in M. \text{is_check}(p,c) \wedge \text{pair}(\#\#M,c,p,y))$
 $\longleftrightarrow y = \langle \text{check}(p), p \rangle$
using *check_abs check_in_M tuples_in_M* **by simp**

lemma *M_subset_MG* : $one \in G \implies M \subseteq M[G]$
using *check_in_M one_in_P GenExtI*
by (*intro subsetI, subst valcheck* [of *G,symmetric*], *auto*)

The name for the generic filter

definition
G_dot :: *i* **where**
G_dot $\equiv \{ \langle \text{check}(p), p \rangle . p \in P \}$

lemma *G_dot_in_M* :
G_dot $\in M$

proof -

let *?is_pcheck* = $\lambda x y. \exists ch \in M. \text{is_check}(x,ch) \wedge \text{pair}(\#\#M,ch,x,y)$
let *?pcheck_fm* = *Exists*(*And*(*check_fm*(1,3,0),*pair_fm*(0,1,2)))
have *sats*(*M*,*?pcheck_fm*,[*x,y,one*]) \longleftrightarrow *?is_pcheck*(*x,y*) **if** $x \in M \ y \in M$ **for** *x y*
using *sats_check_fm that one_in_M* **by simp**
moreover
have *?is_pcheck*(*x,y*) \longleftrightarrow $y = \langle \text{check}(x), x \rangle$ **if** $x \in M \ y \in M$ **for** *x y*
using that *check_abs check_in_M* **by simp**
moreover
have *?pcheck_fm* \in *formula* **by simp**
moreover
have *arity*(*?pcheck_fm*) = 3
unfolding *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 $\langle \text{check}(p), p \rangle \in M$ **if** $p \in P$ **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  $\text{fst}(p) = a\ \text{snd}(p) = b$  unfolding fst_def snd_def by simp_all
moreover
assume  $p \in M$ 
moreover from this
have  $a \in M$ 
  unfolding  $\langle p = \_ \rangle$  Pair_def by (force intro: Transset_M[OF trans_M])
moreover from  $\langle p \in M \rangle$ 
have  $b \in M$ 
  using Transset_M[OF trans_M, of {a,b} p] Transset_M[OF trans_M, of b
{a,b}]
  unfolding  $\langle p = \_ \rangle$  Pair_def by (simp)
  ultimately
  show ?thesis by simp
qed

end

locale  $G\_generic = \text{forcing\_data} +$ 
  fixes  $G :: i$ 
  assumes generic :  $M\_generic(G)$ 
begin

lemma zero_in_MG :
   $0 \in M[G]$ 
proof -
  have  $0 = \text{val}(G, 0)$ 
    using zero_in_M elem_of_val by auto
  also
  have  $\dots \in M[G]$ 
    using GenExtI zero_in_M by simp
  finally show ?thesis .
qed

lemma  $G\_nonempty: G \neq 0$ 
proof -
  have  $P \subseteq P$  ..
  with  $P\_in\_M\ P\_dense\ \langle P \subseteq P \rangle$ 
  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

freqR 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

$f_{type} :: i \Rightarrow i$ **where**
 $f_{type} \equiv fst$

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*:

$f_{type}(\langle f, n1, n2, c \rangle) = f$
 $name1(\langle f, n1, n2, c \rangle) = n1$
 $name2(\langle f, n1, n2, c \rangle) = n2$
 $cond_of(\langle f, n1, n2, c \rangle) = c$
unfolding f_{type_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  $u \in \text{ecloseN}(x)$   $\text{name1}(x) \in \text{ecloseN}(y)$   $\text{name2}(x) \in \text{ecloseN}(y)$ 
  shows  $u \in \text{ecloseN}(y)$ 
proof -
  from  $\langle u \in \_ \rangle$ 
  consider  $(a) u \in \text{eclose}(\{\text{name1}(x)\}) \mid (b) u \in \text{eclose}(\{\text{name2}(x)\})$ 
    unfolding ecloseN_def eclose_n_def by auto
  then
  show ?thesis
  proof cases
    case a
    with  $\langle \text{name1}(x) \in \_ \rangle$ 
    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  $\langle \text{name2}(x) \in \_ \rangle$ 
    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( $M, x, t$ )  $\equiv (\exists z[M]. \text{pair}(M, t, z, x)) \vee$ 
   $(\neg(\exists z[M]. \exists w[M]. \text{pair}(M, w, z, x)) \wedge \text{empty}(M, t))$ 

```

definition

```

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

```

lemma *sats_fst_fm* :

```

 $\llbracket x \in \text{nat}; y \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$ 
 $\implies \text{sats}(A, \text{fst\_fm}(x, y), \text{env}) \longleftrightarrow$ 
   $\text{is\_fst}(\#\#A, \text{nth}(x, \text{env}), \text{nth}(y, \text{env}))$ 
by (simp add: fst_fm_def is_fst_def)

```

definition

```

is_ftype ::  $(i \Rightarrow o) \Rightarrow i \Rightarrow i \Rightarrow o$  where
is_ftype  $\equiv \text{is\_fst}$ 

```

definition

```

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

```

$f_{type_fm} \equiv f_{st_fm}$

lemma $sats_f_{type_fm}$:
 $\llbracket x \in nat; y \in nat; env \in list(A) \rrbracket$
 $\implies sats(A, f_{type_fm}(x,y), env) \longleftrightarrow$
 $is_f_{type}(\#\#A, nth(x,env), nth(y,env))$
unfolding $f_{type_fm_def}$ $is_f_{type_def}$
by (*simp add:sats_fst_fm*)

lemma $is_f_{type_iff_sats}$:
assumes
 $nth(a,env) = aa \ nth(b,env) = bb \ a \in nat \ b \in nat \ env \in list(A)$
shows
 $is_f_{type}(\#\#A, aa, bb) \longleftrightarrow sats(A, f_{type_fm}(a,b), env)$
using *assms*
by (*simp add:sats_ftype_fm*)

definition
 $is_snd :: (i \Rightarrow o) \Rightarrow i \Rightarrow i \Rightarrow o$ **where**
 $is_snd(M,x,t) \equiv (\exists z[M]. pair(M,z,t,x)) \vee$
 $(\neg(\exists z[M]. \exists w[M]. pair(M,z,w,x)) \wedge empty(M,t))$

definition
 $snd_fm :: [i,i] \Rightarrow i$ **where**
 $snd_fm(x,t) \equiv Or(Exists(pair_fm(0,succ(t),succ(x))),$
 $And(Neg(Exists(Exists(pair_fm(1,0,2 \#+ x)))),empty_fm(t)))$

lemma $sats_snd_fm$:
 $\llbracket x \in nat; y \in nat; env \in list(A) \rrbracket$
 $\implies sats(A, snd_fm(x,y), env) \longleftrightarrow$
 $is_snd(\#\#A, nth(x,env), nth(y,env))$
by (*simp add: snd_fm_def is_snd_def*)

definition
 $is_name1 :: (i \Rightarrow o) \Rightarrow i \Rightarrow i \Rightarrow o$ **where**
 $is_name1(M,x,t2) \equiv is_hcomp(M, is_fst(M), is_snd(M), x, t2)$

definition
 $name1_fm :: [i,i] \Rightarrow i$ **where**
 $name1_fm(x,t) \equiv hcomp_fm(fst_fm, snd_fm, x, t)$

lemma $sats_name1_fm$:
 $\llbracket x \in nat; y \in nat; env \in list(A) \rrbracket$
 $\implies sats(A, name1_fm(x,y), env) \longleftrightarrow$
 $is_name1(\#\#A, nth(x,env), nth(y,env))$
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
 $nth(a, env) = aa \quad nth(b, env) = bb \quad a \in nat \quad b \in nat \quad env \in list(A)$
shows
 $is_name1(\#\#A, aa, bb) \longleftrightarrow sats(A, name1_fm(a, b), env)$
using *assms*
by (*simp add:sats_name1_fm*)

definition
 $is_snd_snd :: (i \Rightarrow o) \Rightarrow i \Rightarrow i \Rightarrow o$ **where**
 $is_snd_snd(M, x, t) \equiv is_hcomp(M, is_snd(M), is_snd(M), x, t)$

definition
 $snd_snd_fm :: [i, i] \Rightarrow i$ **where**
 $snd_snd_fm(x, t) \equiv hcomp_fm(snd_fm, snd_fm, x, t)$

lemma *sats_snd2_fm* :
 $\llbracket x \in nat; y \in nat; env \in list(A) \rrbracket$
 $\implies sats(A, snd_snd_fm(x, y), env) \longleftrightarrow$
 $is_snd_snd(\#\#A, nth(x, env), nth(y, env))$
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
 $is_name2 :: (i \Rightarrow o) \Rightarrow i \Rightarrow i \Rightarrow o$ **where**
 $is_name2(M, x, t3) \equiv is_hcomp(M, is_fst(M), is_snd_snd(M), x, t3)$

definition
 $name2_fm :: [i, i] \Rightarrow i$ **where**
 $name2_fm(x, t3) \equiv hcomp_fm(fst_fm, snd_snd_fm, x, t3)$

lemma *sats_name2_fm* :
 $\llbracket x \in nat; y \in nat; env \in list(A) \rrbracket$
 $\implies sats(A, name2_fm(x, y), env) \longleftrightarrow$
 $is_name2(\#\#A, nth(x, env), nth(y, env))$
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
 $nth(a, env) = aa \quad nth(b, env) = bb \quad a \in nat \quad b \in nat \quad env \in list(A)$
shows
 $is_name2(\#\#A, aa, bb) \longleftrightarrow sats(A, name2_fm(a, b), env)$
using *assms*
by (*simp add:sats_name2_fm*)

definition
 $is_cond_of :: (i \Rightarrow o) \Rightarrow i \Rightarrow i \Rightarrow o$ **where**
 $is_cond_of(M, x, t4) \equiv is_hcomp(M, is_snd(M), is_snd_snd(M), x, t4)$

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 :

$\llbracket x \in nat; y \in nat; env \in list(A) \rrbracket$
 $\implies sats(A,cond_of_fm(x,y), env) \longleftrightarrow$
 $is_cond_of(\#\#A, nth(x,env), nth(y,env))$
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
 $f_type_fm(a,b) \in formula$
 $name1_fm(a,b) \in formula$
 $name2_fm(a,b) \in formula$
 $cond_of_fm(a,b) \in formula$
using $assms$
unfolding $f_type_fm_def$ 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 $sats_components_fm[simp] = sats_f_type_fm\ sats_name1_fm\ sats_name2_fm$
 $sats_cond_of_fm$

lemmas $components_iff_sats = is_f_type_iff_sats\ is_name1_iff_sats\ is_name2_iff_sats$
 $is_cond_of_iff_sats$

lemmas $components_defs = fst_fm_def\ f_type_fm_def\ snd_fm_def\ snd_snd_fm_def$
 $hcomp_fm_def$
 $name1_fm_def\ name2_fm_def\ cond_of_fm_def$

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] :

$\llbracket en \in nat ; t \in nat \rrbracket \Longrightarrow ecloseN_fm(en,t) \in formula$

unfolding $ecloseN_fm_def$ $eclose_n1_fm_def$ $eclose_n2_fm_def$ **by** $simp$

lemma $sats_ecloseN_fm$ [$simp$]:

$\llbracket en \in nat; t \in nat ; env \in list(A) \rrbracket$

$\Longrightarrow sats(A, ecloseN_fm(en,t), env) \longleftrightarrow is_ecloseN(\#\#A, nth(en,env), nth(t,env))$

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'*: $s \in \text{domain}(n1) \cup \text{domain}(n2) \implies \text{frecR}(\langle 1, s, n1, q \rangle, \langle 0, n1, n2, q \rangle)$
unfolding *frecR_def* **by** (*simp add:components_simp*)

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

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

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

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

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

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

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

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

definition

$is_frecR :: [i \Rightarrow o, i, i] \Rightarrow o$ **where**
 $is_frecR(M, x, y) \equiv \exists ftx[M]. \exists n1x[M]. \exists n2x[M]. \exists fty[M]. \exists n1y[M]. \exists n2y[M].$
 $\exists dn1[M]. \exists dn2[M].$
 $is_ftype(M, x, ftx) \wedge is_name1(M, x, n1x) \wedge is_name2(M, x, n2x) \wedge$
 $is_ftype(M, y, fty) \wedge is_name1(M, y, n1y) \wedge is_name2(M, y, n2y)$
 $\wedge is_domain(M, n1y, dn1) \wedge is_domain(M, n2y, dn2) \wedge$
 $(number1(M, ftx) \wedge empty(M, fty) \wedge (n1x \in dn1 \vee n1x \in dn2) \wedge (n2x$
 $= n1y \vee n2x = n2y))$
 $\vee (empty(M, ftx) \wedge number1(M, fty) \wedge n1x = n1y \wedge n2x \in dn2))$

schematic_goal $sats_frecR_fm_auto$:

assumes

$i \in nat \ j \in nat \ env \in list(A) \ nth(i, env) = a \ nth(j, env) = b$

shows

$is_frecR(\#\#A, a, b) \longleftrightarrow sats(A, ?fr_fm(i, j), env)$

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 $eq_ftypep_not_frecR$:

assumes $ftype(x) = ftype(y)$

shows $\neg frecR(x, y)$

using $assms\ frecR_ftypeD$ **by** $force$

definition

$rank_names :: i \Rightarrow i$ **where**

$rank_names(x) \equiv max(rank(name1(x)), rank(name2(x)))$

lemma $rank_names_types$ [TC]:

shows $Ord(rank_names(x))$

unfolding $rank_names_def\ max_def$ **using** $Ord_rank\ Ord_Un$ **by** $auto$

definition

$mtype_form :: i \Rightarrow i$ **where**

$mtype_form(x) \equiv if\ rank(name1(x)) < rank(name2(x))\ then\ 0\ else\ 2$

definition

$type_form :: i \Rightarrow i$ **where**

$type_form(x) \equiv if\ ftype(x) = 0\ then\ 1\ else\ mtype_form(x)$

lemma $type_form_tc$ [TC]:

shows $type_form(x) \in 3$

unfolding $type_form_def\ mtype_form_def$ **by** $auto$

```

lemma freqR_le_rnk_names :
  assumes freqR(x,y)
  shows rank_names(x) ≤ rank_names(y)
proof -
  obtain a b c d where
    H: a = name1(x) b = name2(x)
    c = name1(y) d = name2(y)
    (a ∈ domain(c) ∪ domain(d) ∧ (b=c ∨ b = d)) ∨ (a = c ∧ b ∈ domain(d))
  using assms unfolding freqR_def by force
  then
  consider
    (m) a ∈ domain(c) ∧ (b = c ∨ b = d)
  | (n) a ∈ domain(d) ∧ (b = c ∨ b = d)
  | (o) b ∈ domain(d) ∧ a = c
  by auto
  then show ?thesis proof(cases)
    case m
    then
    have rank(a) < rank(c)
      using eclose_rank_lt_in_dom_in_eclose by simp
    with ⟨rank(a) < rank(c)⟩ H m
    show ?thesis unfolding rank_names_def using Ord_rank_max_cong_max_cong2
  leI by auto
    next
    case n
    then
    have rank(a) < rank(d)
      using eclose_rank_lt_in_dom_in_eclose by simp
    with ⟨rank(a) < rank(d)⟩ H n
    show ?thesis unfolding rank_names_def
      using Ord_rank_max_cong2_max_cong_max_commutes[of rank(c) rank(d)]
  leI by auto
    next
    case o
    then
    have rank(b) < rank(d) (is ?b < ?d) rank(a) = rank(c) (is ?a = _)
      using eclose_rank_lt_in_dom_in_eclose by simp_all
    with H
    show ?thesis unfolding rank_names_def
      using Ord_rank_max_commutes_max_cong2[OF leI[OF ⟨?b < ?d⟩], of ?a]
  by simp
  qed
qed

```

definition

```

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

```

```

lemma  $\Gamma\_type$  [TC]:
  shows  $Ord(\Gamma(x))$ 
  unfolding  $\Gamma\_def$  by simp

lemma  $\Gamma\_mono$  :
  assumes  $freqR(x,y)$ 
  shows  $\Gamma(x) < \Gamma(y)$ 
proof -
  have  $F: type\_form(x) < 3 \ type\_form(y) < 3$ 
    using  $ltI$  by simp_all
  from  $assms$ 
  have  $A: rank\_names(x) \leq rank\_names(y)$  (is  $?x \leq ?y$ )
    using  $freqR\_le\_rnk\_names$  by simp
  then
  have  $Ord(?y)$  unfolding  $rank\_names\_def$  using  $Ord\_rank$   $max\_def$  by simp
  note  $leE[OF \langle ?x \leq ?y \rangle]$ 
  then
  show  $?thesis$ 
proof (cases)
  case 1
  then
  show  $?thesis$  unfolding  $\Gamma\_def$  using  $oadd\_lt\_mono2$   $\langle ?x < ?y \rangle$   $F$  by auto
next
  case 2
  consider (a)  $ftype(x) = 0 \wedge ftype(y) = 1$  | (b)  $ftype(x) = 1 \wedge ftype(y) = 0$ 
    using  $freqR\_ftypeD[OF \langle freqR(x,y) \rangle]$  by auto
  then show  $?thesis$  proof (cases)
    case b
    then
    have  $type\_form(y) = 1$ 
      using  $type\_form\_def$  by simp
    from b
    have  $H: name2(x) = name1(y) \vee name2(x) = name2(y)$  (is  $? \tau = ? \sigma' \vee ? \tau = ? \tau'$ )
       $name1(x) \in domain(name1(y)) \cup domain(name2(y))$ 
      (is  $? \sigma \in domain(? \sigma') \cup domain(? \tau')$ )
      using  $assms$  unfolding  $type\_form\_def$   $freqR\_def$  by auto
    then
    have  $E: rank(? \tau) = rank(? \sigma') \vee rank(? \tau) = rank(? \tau')$  by auto
    from H
    consider (a)  $rank(? \sigma) < rank(? \sigma')$  | (b)  $rank(? \sigma) < rank(? \tau')$ 
      using  $eclose\_rank\_lt$   $in\_dom$   $in\_eclose$  by force
    then
    have  $rank(? \sigma) < rank(? \tau)$  proof (cases)
      case a
      with  $\langle rank\_names(x) = rank\_names(y) \rangle$ 
      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  $\langle \text{rank\_names}(x) = \text{rank\_names}(y) \rangle$ 
  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  $\text{type\_form}(x) = 0$  unfolding type_form_def mtype_form_def by simp
  with  $\langle \text{rank\_names}(x) = \text{rank\_names}(y) \rangle \langle \text{type\_form}(y) = 1 \rangle \langle \text{type\_form}(x) = 0 \rangle$ 
  show ?thesis
    unfolding  $\Gamma\_def$  by auto
next
  case a
  then
  have  $\text{name1}(x) = \text{name1}(y)$  (is  $? \sigma = ? \sigma'$ )
     $\text{name2}(x) \in \text{domain}(\text{name2}(y))$  (is  $? \tau \in \text{domain}(? \tau')$ )
     $\text{type\_form}(x) = 1$ 
  using assms unfolding type_form_def frecR_def by auto
  then
  have  $\text{rank}(? \sigma) = \text{rank}(? \sigma')$   $\text{rank}(? \tau) < \text{rank}(? \tau')$ 
    using eclose_rank_lt_in_dom_in_eclose by simp_all
  with  $\langle \text{rank\_names}(x) = \text{rank\_names}(y) \rangle$ 
  have  $\text{rank}(? \tau') \leq \text{rank}(? \sigma')$ 
    unfolding rank_names_def using Ord_rank max_D1 by simp
  with a
  have  $\text{type\_form}(y) = 2$ 
    unfolding type_form_def mtype_form_def using not_lt_iff_le assms by simp
  with  $\langle \text{rank\_names}(x) = \text{rank\_names}(y) \rangle \langle \text{type\_form}(y) = 2 \rangle \langle \text{type\_form}(x) = 1 \rangle$ 
  show ?thesis
    unfolding  $\Gamma\_def$  by auto
  qed
qed
qed

```

definition

```

frecrel ::  $i \Rightarrow i$  where
frecrel(A)  $\equiv$  Rrel(frecR,A)

```

lemma *frecrelI* :

```

assumes  $x \in A$   $y \in A$  frecR(x,y)
shows  $\langle x, y \rangle \in \text{frecrel}(A)$ 
using assms unfolding frecrel_def Rrel_def by auto

```

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

lemma *wf_frecrel* :
shows $\text{wf}(\text{frecrel}(A))$
proof -
have $\text{frecrel}(A) \subseteq \text{measure}(A,\Gamma)$
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
 $\text{Transset}(A1)$
 $\bigwedge \tau \vartheta p. p \in A2 \implies \llbracket \bigwedge q \sigma. \llbracket q \in A2 ; \sigma \in \text{domain}(\vartheta) \rrbracket \implies Q(0,\tau,\sigma,q) \rrbracket \implies$
 $Q(1,\tau,\vartheta,p)$
 $\bigwedge \tau \vartheta p. p \in A2 \implies \llbracket \bigwedge q \sigma. \llbracket q \in A2 ; \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta) \rrbracket \implies$
 $Q(1,\sigma,\tau,q) \wedge Q(1,\sigma,\vartheta,q) \rrbracket \implies Q(0,\tau,\vartheta,p)$
shows $a \in 2 \times A1 \times A1 \times A2 \implies Q(\text{ftype}(a), \text{name1}(a), \text{name2}(a), \text{cond_of}(a))$
proof (*induct a rule:wf_induct[OF wf_frecrel[of 2 × A1 × A1 × A2]]*)
case (1 x)
let $? \tau = \text{name1}(x)$
let $? \vartheta = \text{name2}(x)$
let $?D = 2 \times A1 \times A1 \times A2$
assume $x \in ?D$
then
have $\text{cond_of}(x) \in A2$
by (*auto simp add:components_simp*)
from $\langle x \in ?D \rangle$
consider (*eq*) $\text{ftype}(x)=0$ | (*mem*) $\text{ftype}(x)=1$
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** $\sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta)$ **and**
 $q \in A2$ **for** $q \sigma$
proof -
from 1
have $A: ?\tau \in A1 ?\vartheta \in A1 ?\tau \in \text{eclose}(A1) ?\vartheta \in \text{eclose}(A1)$

```

    using arg_into_eclose by (auto simp add:components_simp)
  with ‹Transset(A1)› that(1)
  have  $\sigma \in \text{eclose}(\tau) \cup \text{eclose}(\vartheta)$ 
    using in_dom_in_eclose by auto
  then
  have  $\sigma \in A1$ 
    using mem_eclose_subset[OF ‹ $\tau \in A1$ ›] mem_eclose_subset[OF ‹ $\vartheta \in A1$ ›]

    Transset_eclose_eq_arg[OF ‹Transset(A1)›]
  by auto
  with ‹ $q \in A2$ › ‹ $\vartheta \in A1$ › ‹cond_of(x) ∈ A2› ‹ $\tau \in A1$ ›
  have  $\text{frecR}(\langle 1, \sigma, \tau, q \rangle, x)$  (is  $\text{frecR}(\tau, \_)$ )
     $\text{frecR}(\langle 1, \sigma, \vartheta, q \rangle, x)$  (is  $\text{frecR}(\vartheta, \_)$ )
    using  $\text{frecRI1}'$ [OF that(1)]  $\text{frecR\_DI}$  ‹ftype(x) = 0›
     $\text{frecRI2}'$ [OF that(1)]
  by (auto simp add:components_simp)
  with ‹ $x \in ?D$ › ‹ $\sigma \in A1$ › ‹ $q \in A2$ ›
  have ‹ $\tau, x \in \text{frecrel}(?D)$ › ‹ $\vartheta, x \in \text{frecrel}(?D)$ ›
  using  $\text{frecrelI}$ [of ?T ?D x]  $\text{frecrelI}$ [of ?U ?D x] by (auto simp add:components_simp)
  with ‹ $q \in A2$ › ‹ $\sigma \in A1$ › ‹ $\tau \in A1$ › ‹ $\vartheta \in A1$ ›
  have  $Q(1, \sigma, \tau, q)$  using 1 by (force simp add:components_simp)
  moreover from ‹ $q \in A2$ › ‹ $\sigma \in A1$ › ‹ $\tau \in A1$ › ‹ $\vartheta \in A1$ › ‹ $\vartheta, x \in \text{frecrel}(?D)$ ›
  have  $Q(1, \sigma, \vartheta, q)$  using 1 by (force simp add:components_simp)
  ultimately
  show ?thesis using A by simp
qed
then show ?thesis using  $\text{assms}(3)$  ‹ftype(x) = 0› ‹cond_of(x) ∈ A2› by auto
next
case mem
have  $Q(0, \tau, \sigma, q)$  if  $\sigma \in \text{domain}(\vartheta)$  and  $q \in A2$  for  $q \sigma$ 
proof -
  from 1  $\text{assms}$ 
  have ‹ $\tau \in A1$ › ‹ $\vartheta \in A1$ › ‹cond_of(x) ∈ A2› ‹ $\tau \in \text{eclose}(A1)$ › ‹ $\vartheta \in \text{eclose}(A1)$ ›
    using arg_into_eclose by (auto simp add:components_simp)
  with ‹Transset(A1)› that(1)
  have  $\sigma \in \text{eclose}(\vartheta)$ 
    using in_dom_in_eclose by auto
  then
  have  $\sigma \in A1$ 
    using mem_eclose_subset[OF ‹ $\vartheta \in A1$ ›] Transset_eclose_eq_arg[OF ‹Trans-
set(A1)›]
  by auto
  with ‹ $q \in A2$ › ‹ $\vartheta \in A1$ › ‹cond_of(x) ∈ A2› ‹ $\tau \in A1$ ›
  have  $\text{frecR}(\langle 0, \tau, \sigma, q \rangle, x)$  (is  $\text{frecR}(\tau, \_)$ )
    using  $\text{frecRI3}'$ [OF that(1)]  $\text{frecR\_DI}$  ‹ftype(x) = 1›
  by (auto simp add:components_simp)
  with ‹ $x \in ?D$ › ‹ $\sigma \in A1$ › ‹ $q \in A2$ › ‹ $\tau \in A1$ ›
  have ‹ $\tau, x \in \text{frecrel}(?D)$ › ‹ $\tau \in ?D$ ›
    using  $\text{frecrelI}$ [of ?T ?D x] by (auto simp add:components_simp)

```

```

with ⟨q∈A2⟩ ⟨σ∈A1⟩ ⟨τ∈A1⟩ ⟨∅∈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))) ∪ do-
main(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* :
 $\llbracket r \in \text{nat} ; z \in \text{nat} \rrbracket \implies \text{arity}(\text{domain_fm}(r,z)) = \text{succ}(r) \cup \text{succ}(z)$
unfolding *domain_fm_def*
using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_range_fm* :
 $\llbracket r \in \text{nat} ; z \in \text{nat} \rrbracket \implies \text{arity}(\text{range_fm}(r,z)) = \text{succ}(r) \cup \text{succ}(z)$
unfolding *range_fm_def*
using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_union_fm* :
 $\llbracket x \in \text{nat} ; y \in \text{nat} ; z \in \text{nat} \rrbracket \implies \text{arity}(\text{union_fm}(x,y,z)) = \bigcup \{ \text{succ}(x), \text{succ}(y), \text{succ}(z) \}$
unfolding *union_fm_def*
using *nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_image_fm* :
 $\llbracket x \in \text{nat} ; y \in \text{nat} ; z \in \text{nat} \rrbracket \implies \text{arity}(\text{image_fm}(x,y,z)) = \bigcup \{ \text{succ}(x), \text{succ}(y), \text{succ}(z) \}$
unfolding *image_fm_def*
using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_pre_image_fm* :
 $\llbracket x \in \text{nat} ; y \in \text{nat} ; z \in \text{nat} \rrbracket \implies \text{arity}(\text{pre_image_fm}(x,y,z)) = \bigcup \{ \text{succ}(x), \text{succ}(y), \text{succ}(z) \}$
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* :
 $\llbracket x \in \text{nat} ; y \in \text{nat} \rrbracket \implies \text{arity}(\text{big_union_fm}(x,y)) = \text{succ}(x) \cup \text{succ}(y)$
unfolding *big_union_fm_def*
using *nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_fun_apply_fm* :
 $\llbracket x \in \text{nat} ; y \in \text{nat} ; f \in \text{nat} \rrbracket \implies$
 $\text{arity}(\text{fun_apply_fm}(f,x,y)) = \text{succ}(f) \cup \text{succ}(x) \cup \text{succ}(y)$
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* :
 $\llbracket r \in \text{nat} ; z \in \text{nat} \rrbracket \implies \text{arity}(\text{field_fm}(r,z)) = \text{succ}(r) \cup \text{succ}(z)$
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* :
 $\llbracket r \in \text{nat} \rrbracket \implies \text{arity}(\text{empty_fm}(r)) = \text{succ}(r)$
unfolding *empty_fm_def*
using *nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *simp*

lemma *arity_succ_fm* :
 $\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \implies \text{arity}(\text{succ_fm}(x,y)) = \text{succ}(x) \cup \text{succ}(y)$
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* :
 $\llbracket r \in \text{nat} \rrbracket \implies \text{arity}(\text{number1_fm}(r)) = \text{succ}(r)$
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* :
 $\llbracket r \in \text{nat} \rrbracket \implies \text{arity}(\text{function_fm}(r)) = \text{succ}(r)$
unfolding *function_fm_def*
using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *simp*

lemma *arity_relation_fm* :
 $\llbracket r \in \text{nat} \rrbracket \implies \text{arity}(\text{relation_fm}(r)) = \text{succ}(r)$
unfolding *relation_fm_def*
using *arity_pair_fm nat_union_abs1 nat_union_abs2 pred_Un_distrib*
by *simp*

lemma *arity_restriction_fm* :
 $\llbracket r \in \text{nat} ; z \in \text{nat} ; A \in \text{nat} \rrbracket \implies \text{arity}(\text{restriction_fm}(A,z,r)) = \text{succ}(A) \cup \text{succ}(r)$
 $\cup \text{succ}(z)$
unfolding *restriction_fm_def*
using *arity_pair_fm nat_union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_typed_function_fm* :
 $\llbracket x \in \text{nat} ; y \in \text{nat} ; f \in \text{nat} \rrbracket \implies$
 $\text{arity}(\text{typed_function_fm}(f,x,y)) = \bigcup \{ \text{succ}(f), \text{succ}(x), \text{succ}(y) \}$

```

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 :
   $[[x \in \text{nat} ; y \in \text{nat}]] \implies \text{arity}(\text{subset\_fm}(x,y)) = \text{succ}(x) \cup \text{succ}(y)$ 
unfolding subset_fm_def
using nat_union_abs2 pred_Un_distrib
by auto

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

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

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

lemma arity_cartprod_fm :
   $[[A \in \text{nat} ; B \in \text{nat} ; z \in \text{nat}]] \implies \text{arity}(\text{cartprod\_fm}(A,B,z)) = \text{succ}(A) \cup \text{succ}(B)$ 
 $\cup \text{succ}(z)$ 
unfolding cartprod_fm_def
using arity_pair_fm nat_union_abs2 pred_Un_distrib

```

by *auto*

lemma *arity_fst_fm* :

$\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{fst_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$

unfolding *fst_fm_def*

using *arity_pair_fm arity_empty_fm nat_union_abs2 pred_Un_distrib*

by *auto*

lemma *arity_snd_fm* :

$\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{snd_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$

unfolding *snd_fm_def*

using *arity_pair_fm arity_empty_fm nat_union_abs2 pred_Un_distrib*

by *auto*

lemma *arity_snd_snd_fm* :

$\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{snd_snd_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$

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_fstype_fm* :

$\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{fstype_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$

unfolding *fstype_fm_def*

using *arity_fst_fm*

by *auto*

lemma *name1arity_fm* :

$\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{name1_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$

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* :

$\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{name2_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$

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* :

$\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{cond_of_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$

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* :

$\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{singleton_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$

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* :

$\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{Memrel_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$

unfolding *Memrel_fm_def*

using *arity_pair_fm nat_union_abs2 pred_Un_distrib*

by *auto*

lemma *arity_quasinat_fm* :

$\llbracket x \in \text{nat} \rrbracket \implies \text{arity}(\text{quasinat_fm}(x)) = \text{succ}(x)$

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* :

$\llbracket p \in \text{formula} ; v \in \text{nat} ; n \in \text{nat} ; Z \in \text{nat} ; i \in \text{nat} \rrbracket \implies \text{arity}(p) = i \implies$

$\text{arity}(\text{is_recfun_fm}(p,v,n,Z)) = \text{succ}(v) \cup \text{succ}(n) \cup \text{succ}(Z) \cup \text{pred}(\text{pred}(\text{pred}(\text{pred}(i))))$

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* :

$\llbracket p \in \text{formula} ; v \in \text{nat} ; n \in \text{nat} ; Z \in \text{nat} ; i \in \text{nat} \rrbracket \implies \text{arity}(p) = i \implies$

$\text{arity}(\text{is_wfrec_fm}(p,v,n,Z)) = \text{succ}(v) \cup \text{succ}(n) \cup \text{succ}(Z) \cup \text{pred}(\text{pred}(\text{pred}(\text{pred}(\text{pred}(i))))))$

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* :

$\llbracket p \in \text{formula} ; v \in \text{nat} ; n \in \text{nat} ; Z \in \text{nat} ; i \in \text{nat} \rrbracket \implies \text{arity}(p) = i \implies$

$\text{arity}(\text{is_nat_case_fm}(v,p,n,Z)) = \text{succ}(v) \cup \text{succ}(n) \cup \text{succ}(Z) \cup \text{pred}(\text{pred}(i))$

unfolding *is_nat_case_fm_def*

using *arity_succ_fm arity_empty_fm arity_quasinat_fm*

nat_union_abs2 pred_Un_distrib

by *auto*

lemma *arity_iterates_MH_fm* :

assumes $isF \in \text{formula} \ v \in \text{nat} \ n \in \text{nat} \ g \in \text{nat} \ z \in \text{nat} \ i \in \text{nat}$

$\text{arity}(isF) = i$

shows $\text{arity}(\text{iterates_MH_fm}(isF,v,n,g,z)) =$

$\text{succ}(v) \cup \text{succ}(n) \cup \text{succ}(g) \cup \text{succ}(z) \cup \text{pred}(\text{pred}(\text{pred}(\text{pred}(i))))$

proof -

let $? \varphi = \text{Exists}(\text{And}(\text{fun_apply_fm}(\text{succ}(\text{succ}(\text{succ}(g))), 2, 0), \text{Forall}(\text{Implies}(\text{Equal}(0, 2), isF))))$

let $?ar = \text{succ}(\text{succ}(\text{succ}(g))) \cup \text{pred}(\text{pred}(i))$

from *assms*

have $\text{arity}(? \varphi) = ?ar \ ? \varphi \in \text{formula}$

```

    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(pred(i)))))))))))
  proof -
    let ?φ = iterates_MH_fm(p, 7#+v, 2, 1, 0)
    let ?ψ = is_wfrec_fm(?φ, 0, succ(succ(n)),succ(succ(Z)))
    from ‹v∈›
    have arity(?φ) = (8#+v) ∪ pred(pred(pred(pred(i)))) ?φ∈formula
      using assms arity_iterates_MH_fm nat_union_abs2
      by simp_all
    then
    have arity(?ψ) = succ(succ(succ(n))) ∪ succ(succ(succ(Z))) ∪ (3#+v) ∪
      pred(pred(pred(pred(pred(pred(pred(pred(pred(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

```

qed

lemma *arity_mem_eclose_fm* :

assumes $x \in \text{nat}$ $t \in \text{nat}$

shows $\text{arity}(\text{mem_eclose_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$

proof -

let $?\varphi = \text{eclose_n_fm}(x \# + 2, 1, 0)$

from $\langle x \in \text{nat} \rangle$

have $\text{arity}(?\varphi) = x \# + 3$

using *arity_eclose_n_fm_nat_union_abs2*

by *simp*

with *assms*

show $?\text{thesis}$

unfolding *mem_eclose_fm_def*

using *arity_finite_ordinal_fm_nat_union_abs2 pred_Un_distrib*

by *simp*

qed

lemma *arity_is_eclose_fm* :

$\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{is_eclose_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$

unfolding *is_eclose_fm_def*

using *arity_mem_eclose_fm_nat_union_abs2 pred_Un_distrib*

by *auto*

lemma *eclose_n1arity_fm* :

$\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{eclose_n1_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$

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* :

$\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{eclose_n2_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$

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* :

$\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{ecloseN_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$

unfolding *ecloseN_fm_def*

using *eclose_n1arity_fm eclose_n2arity_fm arity_union_fm nat_union_abs2 pred_Un_distrib*

by *auto*

lemma *arity_freR_fm* :

$\llbracket a \in \text{nat}; b \in \text{nat} \rrbracket \implies \text{arity}(\text{freR_fm}(a,b)) = \text{succ}(a) \cup \text{succ}(b)$

unfolding *freR_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 $x \in \text{nat } y \in \text{nat } p \in \text{formula}$
shows $\text{arity}(\text{Collect_fm}(x,p,y)) = \text{succ}(x) \cup \text{succ}(y) \cup \text{pred}(\text{arity}(p))$
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) $\equiv (\exists x[M]. \exists y[M]. \text{pair}(M, x, y, xy) \wedge \text{is_frecR}(M, x, y))$

definition

frecrelP_fm :: $i \Rightarrow i$ **where**
frecrelP_fm(a) $\equiv \text{Exists}(\text{Exists}(\text{And}(\text{pair_fm}(1, 0, a\#\ +2), \text{frecR_fm}(1, 0))))$

lemma *arity_frecrelP_fm* :

$a \in \text{nat} \implies \text{arity}(\text{frecrelP_fm}(a)) = \text{succ}(a)$
unfolding *frecrelP_fm_def*
using *arity_frecR_fm arity_pair_fm pred_Un_distrib*
by simp

lemma *frecrelP_fm_type[TC]* :

$a \in \text{nat} \implies \text{frecrelP_fm}(a) \in \text{formula}$
unfolding *frecrelP_fm_def* **by simp**

lemma *sats_frecrelP_fm* :

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

lemma *frecrelP_iff_sats*:

assumes
 $\text{nth}(a, \text{env}) = aa \ a \in \text{nat } \text{env} \in \text{list}(A)$
shows
 $\text{frecrelP}(\#\#A, aa) \longleftrightarrow \text{sats}(A, \text{frecrelP_fm}(a), \text{env})$

using *assms*
by (*simp add:sats_frecrelP_fm*)

definition

is_frecrel :: $[i \Rightarrow o, i, i] \Rightarrow o$ **where**
is_frecrel(M, A, r) $\equiv \exists A2[M]. \text{cartprod}(M, A, A, A2) \wedge \text{is_Collect}(M, A2, \text{frecrelP}(M), r)$

definition

frecrel_fm :: $[i, i] \Rightarrow i$ **where**
frecrel_fm(a, r) $\equiv \text{Exists}(\text{And}(\text{cartprod_fm}(a\#+1, a\#+1, 0), \text{Collect_fm}(0, \text{frecrelP_fm}(0), r\#+1)))$

lemma *frecrel_fm_type*[*TC*] :

$\llbracket a \in \text{nat}; b \in \text{nat} \rrbracket \Longrightarrow \text{frecrel_fm}(a, b) \in \text{formula}$
unfolding *frecrel_fm_def* **by** *simp*

lemma *arity_frecrel_fm* :

assumes $a \in \text{nat}$ $b \in \text{nat}$
shows $\text{arity}(\text{frecrel_fm}(a, b)) = \text{succ}(a) \cup \text{succ}(b)$
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
 $a \in \text{nat}$ $r \in \text{nat}$ $\text{env} \in \text{list}(A)$
shows
 $\text{sats}(A, \text{frecrel_fm}(a, r), \text{env})$
 $\longleftrightarrow \text{is_frecrel}(\#\#A, \text{nth}(a, \text{env}), \text{nth}(r, \text{env}))$
unfolding *is_frecrel_def frecrel_fm_def*
using *assms*
by (*simp add:sats_Collect_fm sats_frecrelP_fm*)

lemma *is_frecrel_iff_sats*:

assumes
 $\text{nth}(a, \text{env}) = aa$ $\text{nth}(r, \text{env}) = rr$ $a \in \text{nat}$ $r \in \text{nat}$ $\text{env} \in \text{list}(A)$
shows
 $\text{is_frecrel}(\#\#A, aa, rr) \longleftrightarrow \text{sats}(A, \text{frecrel_fm}(a, r), \text{env})$
using *assms*
by (*simp add:sats_frecrel_fm*)

definition

names_below :: $i \Rightarrow i \Rightarrow i$ **where**
names_below(P, x) $\equiv 2 \times \text{ecloseN}(x) \times \text{ecloseN}(x) \times P$

lemma *names_belowsD*:

assumes $x \in \text{names_below}(P, z)$
obtains f $n1$ $n2$ p **where**
 $x = \langle f, n1, n2, p \rangle$ $f \in 2$ $n1 \in \text{ecloseN}(z)$ $n2 \in \text{ecloseN}(z)$ $p \in P$

using *assms* **unfolding** *names_below_def* **by** *auto*

definition

is_names_below :: $[i \Rightarrow o, i, i, i] \Rightarrow o$ **where**
is_names_below(*M*,*P*,*x*,*nb*) $\equiv \exists p1[M]. \exists p0[M]. \exists t[M]. \exists ec[M].$
 $is_ecloseN(M, ec, x) \wedge number2(M, t) \wedge cartprod(M, ec, P, p0) \wedge cart-$
 $prod(M, ec, p0, p1)$
 $\wedge cartprod(M, t, p1, nb)$

definition

number2_fm :: $i \Rightarrow i$ **where**
number2_fm(*a*) $\equiv Exists(And(number1_fm(0), succ_fm(0, succ(a))))$

lemma *number2_fm_type*[*TC*] :

$a \in nat \implies number2_fm(a) \in formula$
unfolding *number2_fm_def* **by** *simp*

lemma *number2arity_fm* :

$a \in nat \implies arity(number2_fm(a)) = succ(a)$
unfolding *number2_fm_def*
using *number1arity_fm* *arity_succ_fm* *nat_union_abs2* *pred_Un_distrib*
by *simp*

lemma *sats_number2_fm* [*simp*]:

$\llbracket x \in nat; env \in list(A) \rrbracket$
 $\implies sats(A, number2_fm(x), env) \longleftrightarrow number2(\#\#A, nth(x, env))$
by (*simp* *add: number2_fm_def number2_def*)

definition

is_names_below_fm :: $[i, i, i] \Rightarrow i$ **where**
is_names_below_fm(*P*,*x*,*nb*) $\equiv Exists(Exists(Exists(Exists($
 $And(ecloseN_fm(0, x \# + 4), And(number2_fm(1),$
 $And(cartprod_fm(0, P \# + 4, 2), And(cartprod_fm(0, 2, 3), cartprod_fm(1, 3, nb \# + 4))))))))))$

lemma *arity_is_names_below_fm* :

$\llbracket P \in nat; x \in nat; nb \in nat \rrbracket \implies arity(is_names_below_fm(P, x, nb)) = succ(P) \cup$
 $succ(x) \cup succ(nb)$
unfolding *is_names_below_fm_def*
using *arity_cartprod_fm* *number2arity_fm* *arity_ecloseN_fm* *nat_union_abs2*
pred_Un_distrib
by *auto*

lemma *is_names_below_fm_type*[*TC*]:

$\llbracket P \in nat; x \in nat; nb \in nat \rrbracket \implies is_names_below_fm(P, x, nb) \in formula$
unfolding *is_names_below_fm_def* **by** *simp*

lemma *sats_is_names_below_fm* :

assumes

$P \in \text{nat } x \in \text{nat } nb \in \text{nat } env \in \text{list}(A)$

shows

$\text{sats}(A, \text{is_names_below_fm}(P, x, nb), env)$

$\longleftrightarrow \text{is_names_below}(\#\#A, \text{nth}(P, env), \text{nth}(x, env), \text{nth}(nb, env))$

unfolding is_names_below_fm_def is_names_below_def using assms by simp

definition

$\text{is_tuple} :: [i \Rightarrow o, i, i, i, i, i] \Rightarrow o$ **where**

$\text{is_tuple}(M, z, t1, t2, p, t) \equiv \exists t1t2p[M]. \exists t2p[M]. \text{pair}(M, t2, p, t2p) \wedge \text{pair}(M, t1, t2p, t1t2p)$
 \wedge
 $\text{pair}(M, z, t1t2p, t)$

definition

$\text{is_tuple_fm} :: [i, i, i, i, i] \Rightarrow i$ **where**

$\text{is_tuple_fm}(z, t1, t2, p, tup) = \text{Exists}(\text{Exists}(\text{And}(\text{pair_fm}(t2 \#+ 2, p \#+ 2, 0),$
 $\text{And}(\text{pair_fm}(t1 \#+ 2, 0, 1), \text{pair_fm}(z \#+ 2, 1, tup \#+ 2))))))$

lemma arity_is_tuple_fm : $\llbracket z \in \text{nat} ; t1 \in \text{nat} ; t2 \in \text{nat} ; p \in \text{nat} ; tup \in \text{nat} \rrbracket \Longrightarrow$

$\text{arity}(\text{is_tuple_fm}(z, t1, t2, p, tup)) = \bigcup \{ \text{succ}(z), \text{succ}(t1), \text{succ}(t2), \text{succ}(p), \text{succ}(tup) \}$

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] :

$z \in \text{nat} \Longrightarrow t1 \in \text{nat} \Longrightarrow t2 \in \text{nat} \Longrightarrow p \in \text{nat} \Longrightarrow tup \in \text{nat} \Longrightarrow \text{is_tuple_fm}(z, t1, t2, p, tup) \in \text{formula}$

unfolding is_tuple_fm_def by simp

lemma sats_is_tuple_fm :

assumes

$z \in \text{nat } t1 \in \text{nat } t2 \in \text{nat } p \in \text{nat } tup \in \text{nat } env \in \text{list}(A)$

shows

$\text{sats}(A, \text{is_tuple_fm}(z, t1, t2, p, tup), env)$

$\longleftrightarrow \text{is_tuple}(\#\#A, \text{nth}(z, env), \text{nth}(t1, env), \text{nth}(t2, env), \text{nth}(p, env), \text{nth}(tup, env))$

unfolding is_tuple_def is_tuple_fm_def using assms by simp

lemma is_tuple_iff_sats:

assumes

$\text{nth}(a, env) = aa \text{ nth}(b, env) = bb \text{ nth}(c, env) = cc \text{ nth}(d, env) = dd \text{ nth}(e, env)$
 $= ee$

$a \in \text{nat } b \in \text{nat } c \in \text{nat } d \in \text{nat } e \in \text{nat } env \in \text{list}(A)$

shows

$\text{is_tuple}(\#\#A, aa, bb, cc, dd, ee) \longleftrightarrow \text{sats}(A, \text{is_tuple_fm}(a, b, c, d, e), env)$

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

$mem_case :: [i,i,i,i,i,i,i] \Rightarrow o$ **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
 $n1 \in \text{nat } n2 \in \text{nat } p \in \text{nat } P \in \text{nat } \text{leq} \in \text{nat } f \in \text{nat}$
shows
 $\text{arity}(\text{mem_case_fm}(n1, n2, p, P, \text{leq}, f)) =$
 $\text{succ}(n1) \cup \text{succ}(n2) \cup \text{succ}(p) \cup \text{succ}(P) \cup \text{succ}(\text{leq}) \cup \text{succ}(f)$
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
 $n1 \in \text{nat } n2 \in \text{nat } p \in \text{nat } P \in \text{nat } \text{leq} \in \text{nat } f \in \text{nat } \text{env} \in \text{list}(A)$
shows
 $\text{is_eq_case}(\#\#A, \text{nth}(n1, \text{env}), \text{nth}(n2, \text{env}), \text{nth}(p, \text{env}), \text{nth}(P, \text{env}), \text{nth}(\text{leq},$
 $\text{env}), \text{nth}(f, \text{env}))$
 $\longleftrightarrow \text{sats}(A, ?\text{iec_fm}(n1, n2, p, P, \text{leq}, f), \text{env})$
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
 $n1 \in \text{nat } n2 \in \text{nat } p \in \text{nat } P \in \text{nat } \text{leq} \in \text{nat } f \in \text{nat}$
shows
 $\text{arity}(\text{eq_case_fm}(n1, n2, p, P, \text{leq}, f)) =$
 $\text{succ}(n1) \cup \text{succ}(n2) \cup \text{succ}(p) \cup \text{succ}(P) \cup \text{succ}(\text{leq}) \cup \text{succ}(f)$
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
 $H\text{frc} :: [i, i, i, i] \Rightarrow o$ **where**
 $H\text{frc}(P, \text{leq}, \text{fnnc}, f) \equiv \exists \text{ft}. \exists n1. \exists n2. \exists c. c \in P \wedge \text{fnnc} = \langle \text{ft}, n1, n2, c \rangle \wedge$
 $(\text{ft} = 0 \wedge \text{eq_case}(n1, n2, c, P, \text{leq}, f)$
 $\vee \text{ft} = 1 \wedge \text{mem_case}(n1, n2, c, P, \text{leq}, f))$

definition
 $\text{is_Hfrc} :: [i \Rightarrow o, i, i, i] \Rightarrow o$ **where**
 $\text{is_Hfrc}(M, P, \text{leq}, \text{fnnc}, f) \equiv$
 $\exists \text{ft}[M]. \exists n1[M]. \exists n2[M]. \exists \text{co}[M].$
 $\text{co} \in P \wedge \text{is_tuple}(M, \text{ft}, n1, n2, \text{co}, \text{fnnc}) \wedge$
 $(\text{empty}(M, \text{ft}) \wedge \text{is_eq_case}(M, n1, n2, \text{co}, P, \text{leq}, f)$
 $\vee (\text{number1}(M, \text{ft}) \wedge \text{is_mem_case}(M, n1, n2, \text{co}, P, \text{leq}, f)))$

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 $Hfrc_fm_type[TC]$:

$\llbracket P \in nat; leq \in nat; fnnc \in nat; f \in nat \rrbracket \implies Hfrc_fm(P,leq,fnnc,f) \in formula$
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

$is_Hfrc_at :: [i \Rightarrow o, i, i, i, i] \Rightarrow o$ **where**

$is_Hfrc_at(M,P,leq,fnnc,f,z) \equiv$

$(empty(M,z) \wedge \neg is_Hfrc(M,P,leq,fnnc,f))$

$\vee (number1(M,z) \wedge is_Hfrc(M,P,leq,fnnc,f))$

definition

$Hfrc_at_fm :: [i,i,i,i,i] \Rightarrow i$ **where**
 $Hfrc_at_fm(P,leq,fnnc,f,z) \equiv Or(And(empty_fm(z),Neg(Hfrc_fm(P,leq,fnnc,f))),$
 $And(number1_fm(z),Hfrc_fm(P,leq,fnnc,f)))$

lemma *arity_Hfrc_at_fm* :**assumes** $P \in nat$ $leq \in nat$ $fnnc \in nat$ $f \in nat$ $z \in nat$ **shows**

$$arity(Hfrc_at_fm(P,leq,fnnc,f,z)) = succ(P) \cup succ(leq) \cup succ(fnnc) \cup succ(f)$$

$$\cup succ(z)$$
unfolding *Hfrc_at_fm_def***using** *assms arity_Hfrc_fm arity_empty_fm number1arity_fm pred_Un_distrib*
by *auto***lemma** *Hfrc_at_fm_type[TC]* : $\llbracket P \in nat; leq \in nat; fnnc \in nat; f \in nat; z \in nat \rrbracket \Longrightarrow Hfrc_at_fm(P,leq,fnnc,f,z) \in formula$ **unfolding** *Hfrc_at_fm_def* **by** *simp***lemma** *sats_Hfrc_at_fm*:**assumes** $P \in nat$ $leq \in nat$ $fnnc \in nat$ $f \in nat$ $z \in nat$ $env \in list(A)$ **shows**

$$sats(A,Hfrc_at_fm(P,leq,fnnc,f,z),env)$$

$$\longleftrightarrow is_Hfrc_at(\#\#A,nth(P,env),nth(leq,env),nth(fnnc,env),nth(f,env),nth(z,env)))$$
unfolding *is_Hfrc_at_def Hfrc_at_fm_def* **using** *assms sats_Hfrc_fm***by** *simp***lemma** *is_Hfrc_at_iff_sats*:**assumes** $P \in nat$ $leq \in nat$ $fnnc \in nat$ $f \in nat$ $z \in nat$ $env \in list(A)$ $nth(P,env) = PP$ $nth(leq,env) = lleq$ $nth(fnnc,env) = ffnc$ $nth(f,env) = ff$ $nth(z,env) = zz$ **shows** $is_Hfrc_at(\#\#A,PP,lleq,ffnc,ff,zz)$ $\longleftrightarrow sats(A,Hfrc_at_fm(P,leq,fnnc,f,z),env)$ **using** *assms* **by** (*simp add:sats_Hfrc_at_fm*)**lemma** *arity_tran_closure_fm* : $\llbracket x \in nat; f \in nat \rrbracket \Longrightarrow arity(trans_closure_fm(x,f)) = succ(x) \cup succ(f)$ **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 *forcere*

definition

forcere :: $i \Rightarrow i \Rightarrow i$ **where**
forcere(P, x) \equiv *frecre*(*names_below*(P, x)) $^{\wedge+}$

definition

is_forcere :: $[i \Rightarrow o, i, i, i] \Rightarrow o$ **where**
is_forcere(M, P, x, z) \equiv $\exists r[M]. \exists nb[M]. \text{tran_closure}(M, r, z) \wedge$
 $(\text{is_names_below}(M, P, x, nb) \wedge \text{is_frecre}(M, nb, r))$

definition

forcere_fm :: $i \Rightarrow i \Rightarrow i \Rightarrow i$ **where**
forcere_fm(p, x, z) \equiv *Exists*(*Exists*(*And*(*trans_closure_fm*($1, z\#\#2$),
 $\text{And}(\text{is_names_below_fm}(p\#\#2, x\#\#2, 0), \text{frecre_fm}(0, 1))))$))

lemma *arity_forcere_fm*:

$\llbracket p \in \text{nat}; x \in \text{nat}; z \in \text{nat} \rrbracket \Longrightarrow \text{arity}(\text{forcere_fm}(p, x, z)) = \text{succ}(p) \cup \text{succ}(x) \cup \text{succ}(z)$

unfolding *forcere_fm_def*

using *arity_frecre_fm* *arity_tran_closure_fm* *arity_is_names_below_fm* *pred_Un_distrib*
by *auto*

lemma *forcere_fm_type*[*TC*]:

$\llbracket p \in \text{nat}; x \in \text{nat}; z \in \text{nat} \rrbracket \Longrightarrow \text{forcere_fm}(p, x, z) \in \text{formula}$

unfolding *forcere_fm_def* **by** *simp*

lemma *sats_forcere_fm*:

assumes

$p \in \text{nat} \ x \in \text{nat} \ z \in \text{nat} \ \text{env} \in \text{list}(A)$

shows

$\text{sats}(A, \text{forcere_fm}(p, x, z), \text{env}) \longleftrightarrow \text{is_forcere}(\#\#A, \text{nth}(p, \text{env}), \text{nth}(x, \text{env}), \text{nth}(z, \text{env}))$

proof -

have $\text{sats}(A, \text{trans_closure_fm}(1, z\#\#2), \text{Cons}(nb, \text{Cons}(r, \text{env}))) \longleftrightarrow$

$\text{tran_closure}(\#\#A, r, \text{nth}(z, \text{env}))$ **if** $r \in A \ \text{nb} \in A$ **for** $r \ \text{nb}$

using *that* *assms* *trans_closure_fm_iff_sats*[*of* $1 \ [nb, r]@env \ z\#\#2, \text{symmetric}$]

by *simp*

moreover

have $\text{sats}(A, \text{is_names_below_fm}(\text{succ}(\text{succ}(p)), \text{succ}(\text{succ}(x)), 0), \text{Cons}(nb, \text{Cons}(r, \text{env}))) \longleftrightarrow$

$\text{is_names_below}(\#\#A, \text{nth}(p, \text{env}), \text{nth}(x, \text{env}), nb)$

if $r \in A \ \text{nb} \in A$ **for** $nb \ r$

using *assms* *that* *sats_is_names_below_fm*[*of* $p \ \#\#2 \ x \ \#\#2 \ 0 \ [nb, r]@env$]

by *simp*

moreover

have $\text{sats}(A, \text{frecre_fm}(0, 1), \text{Cons}(nb, \text{Cons}(r, \text{env}))) \longleftrightarrow$

$\text{is_frecre}(\#\#A, nb, r)$

if $r \in A \ \text{nb} \in A$ **for** $r \ \text{nb}$

using *assms* *that* *sats_frecre_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

$p \in \text{nat } l \in \text{nat } i \in \text{nat } j \in \text{nat } \text{env} \in \text{list}(A) \ i < \text{length}(\text{env}) \ j < \text{length}(\text{env})$

shows

$\text{sats}(A, \text{frc_at_fm}(p, l, i, j), \text{env}) \longleftrightarrow$

$\text{is_frc_at}(\#\#A, \text{nth}(p, \text{env}), \text{nth}(l, \text{env}), \text{nth}(i, \text{env}), \text{nth}(j, \text{env}))$

proof -

{

fix $r \ pp \ ll$

assume $r \in A$

have $0 : \text{is_Hfrc_at}(\#\#A, \text{nth}(p, \text{env}), \text{nth}(l, \text{env}), a2, a1, a0) \longleftrightarrow$

$\text{sats}(A, \text{Hfrc_at_fm}(6\#+p, 6\#+l, 2, 1, 0), [a0, a1, a2, a3, a4, r]@env)$

if $a0 \in A \ a1 \in A \ a2 \in A \ a3 \in A \ a4 \in A$ for $a0 \ a1 \ a2 \ a3 \ a4$

using *that assms* $\langle r \in A \rangle$

$\text{is_Hfrc_at_iff_sats}[of \ 6\#+p \ 6\#+l \ 2 \ 1 \ 0 \ [a0, a1, a2, a3, a4, r]@env \ A]$ by

simp

have $\text{sats}(A, \text{is_wfrec_fm}(\text{Hfrc_at_fm}(6\#+p, 6\#+l, 2, 1, 0), 0, \text{succ}(i), \text{succ}(j)), [r]@env) \longleftrightarrow$

$\text{is_wfrec}(\#\#A, \text{is_Hfrc_at}(\#\#A, \text{nth}(p, \text{env}), \text{nth}(l, \text{env})), r, \text{nth}(i, \text{env}), \text{nth}(j, \text{env}))$

using *assms* $\langle r \in A \rangle$

$\text{sats_is_wfrec_fm}[OF \ 0[\text{simplified}]]$

by *simp*

}

moreover

have $\text{sats}(A, \text{forcere_l_fm}(\text{succ}(p), \text{succ}(i), 0), \text{Cons}(r, \text{env})) \longleftrightarrow$

$\text{is_forcere_l}(\#\#A, \text{nth}(p, \text{env}), \text{nth}(i, \text{env}), r)$ if $r \in A$ for r

using *assms* *sats_forcere_l_fm* *that* by *simp*

ultimately

show *?thesis* unfolding *is_frc_at_def* *frc_at_fm_def*

using *assms* by *simp*

qed

definition

$\text{forces_eq}' :: [i, i, i, i] \Rightarrow o$ where

$\text{forces_eq}'(P, l, p, t1, t2) \equiv \text{frc_at}(P, l, \langle 0, t1, t2, p \rangle) = 1$

definition

$\text{forces_mem}' :: [i, i, i, i] \Rightarrow o$ where

$\text{forces_mem}'(P, l, p, t1, t2) \equiv \text{frc_at}(P, l, \langle 1, t1, t2, p \rangle) = 1$

definition

$\text{forces_neg}' :: [i, i, i, i] \Rightarrow o$ where

$\text{forces_neg}'(P, l, p, t1, t2) \equiv \neg (\exists q \in P. \langle q, p \rangle \in l \wedge \text{forces_eq}'(P, l, q, t1, t2))$

definition

$forces_nmem' :: [i,i,i,i,i] \Rightarrow o$ **where**
 $forces_nmem'(P,l,p,t1,t2) \equiv \neg (\exists q \in P. \langle q,p \rangle \in l \wedge forces_mem'(P,l,q,t1,t2))$

definition

$is_forces_eq' :: [i \Rightarrow o, i, i, i, i, i] \Rightarrow o$ **where**
 $is_forces_eq'(M,P,l,p,t1,t2) \equiv \exists o[M]. \exists z[M]. \exists t[M]. number1(M,o) \wedge empty(M,z)$
 \wedge
 $is_tuple(M,z,t1,t2,p,t) \wedge is_frc_at(M,P,l,t,o)$

definition

$is_forces_mem' :: [i \Rightarrow o, i, i, i, i, i] \Rightarrow o$ **where**
 $is_forces_mem'(M,P,l,p,t1,t2) \equiv \exists o[M]. \exists t[M]. number1(M,o) \wedge$
 $is_tuple(M,o,t1,t2,p,t) \wedge is_frc_at(M,P,l,t,o)$

definition

$is_forces_neg' :: [i \Rightarrow o, i, i, i, i, i] \Rightarrow o$ **where**
 $is_forces_neg'(M,P,l,p,t1,t2) \equiv$
 $\neg (\exists q[M]. q \in P \wedge (\exists qp[M]. pair(M,q,p,qp) \wedge qp \in l \wedge is_forces_eq'(M,P,l,q,t1,t2)))$

definition

$is_forces_nmem' :: [i \Rightarrow o, i, i, i, i, i] \Rightarrow o$ **where**
 $is_forces_nmem'(M,P,l,p,t1,t2) \equiv$
 $\neg (\exists q[M]. \exists qp[M]. q \in P \wedge pair(M,q,p,qp) \wedge qp \in l \wedge is_forces_mem'(M,P,l,q,t1,t2))$

definition

$forces_eq_fm :: [i, i, i, i, i] \Rightarrow i$ **where**
 $forces_eq_fm(p,l,q,t1,t2) \equiv$
 $Exists(Exists(Exists(And(number1_fm(2), And(empty_fm(1),$
 $And(is_tuple_fm(1, t1 \# + 3, t2 \# + 3, q \# + 3, 0), frc_at_fm(p \# + 3, l \# + 3, 0, 2)$
 $))))))$

definition

$forces_mem_fm :: [i, i, i, i, i] \Rightarrow i$ **where**
 $forces_mem_fm(p,l,q,t1,t2) \equiv Exists(Exists(And(number1_fm(1),$
 $And(is_tuple_fm(1, t1 \# + 2, t2 \# + 2, q \# + 2, 0), frc_at_fm(p \# + 2, l \# + 2, 0, 1))))))$

definition

$forces_neg_fm :: [i, i, i, i, i] \Rightarrow i$ **where**
 $forces_neg_fm(p,l,q,t1,t2) \equiv Neg(Exists(Exists(And(Member(1, p \# + 2),$
 $And(pair_fm(1, q \# + 2, 0), And(Member(0, l \# + 2), forces_eq_fm(p \# + 2, l \# + 2, 1, t1 \# + 2, t2 \# + 2))))))))$

definition

$forces_nmem_fm :: [i, i, i, i, i] \Rightarrow i$ **where**
 $forces_nmem_fm(p,l,q,t1,t2) \equiv Neg(Exists(Exists(And(Member(1, p \# + 2),$
 $And(pair_fm(1, q \# + 2, 0), And(Member(0, l \# + 2), forces_mem_fm(p \# + 2, l \# + 2, 1, t1 \# + 2, t2 \# + 2))))))))$

lemma $forces_eq_fm_type$ [TC]:

$\llbracket p \in nat; l \in nat; q \in nat; t1 \in nat; t2 \in nat \rrbracket \implies forces_eq_fm(p,l,q,t1,t2) \in formula$

unfolding *forces_eq_fm_def*
by *simp*

lemma *forces_mem_fm_type [TC]*:
 $\llbracket p \in \text{nat}; l \in \text{nat}; q \in \text{nat}; t1 \in \text{nat}; t2 \in \text{nat} \rrbracket \implies \text{forces_mem_fm}(p, l, q, t1, t2) \in \text{formula}$
unfolding *forces_mem_fm_def*
by *simp*

lemma *forces_neq_fm_type [TC]*:
 $\llbracket p \in \text{nat}; l \in \text{nat}; q \in \text{nat}; t1 \in \text{nat}; t2 \in \text{nat} \rrbracket \implies \text{forces_neq_fm}(p, l, q, t1, t2) \in \text{formula}$
unfolding *forces_neq_fm_def*
by *simp*

lemma *forces_nmem_fm_type [TC]*:
 $\llbracket p \in \text{nat}; l \in \text{nat}; q \in \text{nat}; t1 \in \text{nat}; t2 \in \text{nat} \rrbracket \implies \text{forces_nmem_fm}(p, l, q, t1, t2) \in \text{formula}$
unfolding *forces_nmem_fm_def*
by *simp*

lemma *arity_forces_eq_fm* :
 $p \in \text{nat} \implies l \in \text{nat} \implies q \in \text{nat} \implies t1 \in \text{nat} \implies t2 \in \text{nat} \implies$
 $\text{arity}(\text{forces_eq_fm}(p, l, q, t1, t2)) = \text{succ}(t1) \cup \text{succ}(t2) \cup \text{succ}(q) \cup \text{succ}(p) \cup$
 $\text{succ}(l)$
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* :
 $p \in \text{nat} \implies l \in \text{nat} \implies q \in \text{nat} \implies t1 \in \text{nat} \implies t2 \in \text{nat} \implies$
 $\text{arity}(\text{forces_mem_fm}(p, l, q, t1, t2)) = \text{succ}(t1) \cup \text{succ}(t2) \cup \text{succ}(q) \cup \text{succ}(p)$
 $\cup \text{succ}(l)$
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 $p \in \text{nat} \ l \in \text{nat} \ q \in \text{nat} \ t1 \in \text{nat} \ t2 \in \text{nat} \ \text{env} \in \text{list}(M)$
shows $\text{sats}(M, \text{forces_eq_fm}(p, l, q, t1, t2), \text{env}) \longleftrightarrow$
 $\text{is_forces_eq}'(\#\#M, \text{nth}(p, \text{env}), \text{nth}(l, \text{env}), \text{nth}(q, \text{env}), \text{nth}(t1, \text{env}), \text{nth}(t2, \text{env}))$
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 $p \in \text{nat} \ l \in \text{nat} \ q \in \text{nat} \ t1 \in \text{nat} \ t2 \in \text{nat} \ \text{env} \in \text{list}(M)$
shows $\text{sats}(M, \text{forces_mem_fm}(p, l, q, t1, t2), \text{env}) \longleftrightarrow$
 $\text{is_forces_mem}'(\#\#M, \text{nth}(p, \text{env}), \text{nth}(l, \text{env}), \text{nth}(q, \text{env}), \text{nth}(t1, \text{env}), \text{nth}(t2, \text{env}))$

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 $p \in \text{nat } l \in \text{nat } q \in \text{nat } t1 \in \text{nat } t2 \in \text{nat } \text{env} \in \text{list}(M)$
shows $\text{sats}(M, \text{forces_neq_fm}(p, l, q, t1, t2), \text{env}) \longleftrightarrow$
 $\text{is_forces_neq}'(\#\#M, \text{nth}(p, \text{env}), \text{nth}(l, \text{env}), \text{nth}(q, \text{env}), \text{nth}(t1, \text{env}), \text{nth}(t2, \text{env}))$
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 $p \in \text{nat } l \in \text{nat } q \in \text{nat } t1 \in \text{nat } t2 \in \text{nat } \text{env} \in \text{list}(M)$
shows $\text{sats}(M, \text{forces_nmem_fm}(p, l, q, t1, t2), \text{env}) \longleftrightarrow$
 $\text{is_forces_nmem}'(\#\#M, \text{nth}(p, \text{env}), \text{nth}(l, \text{env}), \text{nth}(q, \text{env}), \text{nth}(t1, \text{env}), \text{nth}(t2, \text{env}))$
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]*:
 $\llbracket x \in M; y \in M \rrbracket \implies \text{is_fst}(\#\#M, x, y) \longleftrightarrow y = \text{fst}(x)$
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]*:
 $\llbracket x \in M; y \in M \rrbracket \implies \text{is_snd}(\#\#M, x, y) \longleftrightarrow y = \text{snd}(x)$
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]* :
 $\llbracket x \in M; y \in M \rrbracket \implies \text{is_ftype}(\#\#M, x, y) \longleftrightarrow y = \text{ftype}(x)$ **unfolding** *ftype_def*
is_fctype_def **by** *simp*

lemma *name1_abs [simp]* :
 $\llbracket x \in M; y \in M \rrbracket \implies \text{is_name1}(\#\#M, x, y) \longleftrightarrow y = \text{name1}(x)$
unfolding *name1_def is_name1_def*
by *(rule hcomp_abs[OF fst_abs]; simp_all add:fst_snd_closed)*

lemma *snd_snd_abs*:
 $\llbracket x \in M; y \in M \rrbracket \implies \text{is_snd_snd}(\#\#M, x, y) \longleftrightarrow y = \text{snd}(\text{snd}(x))$
unfolding *is_snd_snd_def*
by *(rule hcomp_abs[OF snd_abs]; simp_all add:fst_snd_closed)*

lemma *name2_abs [simp]*:

$\llbracket x \in M; y \in M \rrbracket \implies \text{is_name2}(\#\#M, x, y) \longleftrightarrow y = \text{name2}(x)$
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]*:
 $\llbracket x \in M; y \in M \rrbracket \implies \text{is_cond_of}(\#\#M, x, y) \longleftrightarrow y = \text{cond_of}(x)$
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]*:
 $\llbracket z \in M; t1 \in M; t2 \in M; p \in M; t \in M \rrbracket \implies$
 $\text{is_tuple}(\#\#M, z, t1, t2, p, t) \longleftrightarrow t = \langle z, t1, t2, p \rangle$
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*:
 $p \preceq q \implies p \in M$
 $p \preceq q \implies q \in M$
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
 $\text{is_eq_case}(\#\#M, t1, t2, p, P, \text{leq}, f) \longleftrightarrow \text{eq_case}(t1, t2, p, P, \text{leq}, f)$
proof -
have $q \preceq p \implies q \in M$ **for** q
using *comp_in_M by simp*
moreover
have $\langle s, y \rangle \in t \implies s \in \text{domain}(t)$ **if** $t \in M$ **for** $s \ y \ t$
using *that unfolding domain_def by auto*
ultimately
have
 $(\forall s \in M. s \in \text{domain}(t1) \vee s \in \text{domain}(t2) \longrightarrow (\forall q \in M. q \in P \wedge q \preceq p \longrightarrow$
 $(f \text{ ` } \langle 1, s, t1, q \rangle = 1 \longleftrightarrow f \text{ ` } \langle 1, s, t2, q \rangle = 1))) \longleftrightarrow$
 $(\forall s. s \in \text{domain}(t1) \vee s \in \text{domain}(t2) \longrightarrow (\forall q. q \in P \wedge q \preceq p \longrightarrow$
 $(f \text{ ` } \langle 1, s, t1, q \rangle = 1 \longleftrightarrow f \text{ ` } \langle 1, s, t2, q \rangle = 1)))$
using *assms domain_trans[OF trans_M, of t1]*
 $\text{domain_trans}[OF trans_M, of t2]$ **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:
   $\llbracket f \text{fnnc} \in M; f \in M \rrbracket \implies$ 
   $is\_Hfrc(\#\#M, P, leq, f \text{fnnc}, f) \longleftrightarrow Hfrc(P, leq, f \text{fnnc}, f)$ 
  unfolding is_Hfrc_def Hfrc_def using pair_in_M_iff
  by auto

```

```

lemma Hfrc_at_abs:
   $\llbracket f \text{fnnc} \in M; f \in M; z \in M \rrbracket \implies$ 
   $is\_Hfrc\_at(\#\#M, P, leq, f \text{fnnc}, f, z) \longleftrightarrow z = bool\_of\_o(Hfrc(P, leq, f \text{fnnc}, f))$ 
  unfolding is_Hfrc_at_def using Hfrc_abs
  by auto

```

```

lemma components_closed :
   $x \in M \implies ftype(x) \in M$ 
   $x \in M \implies name1(x) \in M$ 
   $x \in M \implies name2(x) \in M$ 
   $x \in M \implies cond\_of(x) \in M$ 
  unfolding ftype_def name1_def name2_def cond_of_def using fst_snd_closed
  by simp_all

```

```

lemma ecloseN_closed:
   $(\#\#M)(A) \implies (\#\#M)(ecloseN(A))$ 
   $(\#\#M)(A) \implies (\#\#M)(eclose\_n(name1, A))$ 
   $(\#\#M)(A) \implies (\#\#M)(eclose\_n(name2, A))$ 
  unfolding ecloseN_def eclose_n_def
  using components_closed eclose_closed singletonM Un_closed by auto

```

```

lemma is_eclose_n_abs :
  assumes  $x \in M ec \in M$ 
  shows  $is\_eclose\_n(\#\#M, is\_name1, ec, x) \longleftrightarrow ec = eclose\_n(name1, x)$ 
   $is\_eclose\_n(\#\#M, is\_name2, ec, x) \longleftrightarrow ec = eclose\_n(name2, x)$ 
  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 :

```

$\llbracket x \in M; ec \in M \rrbracket \implies is_ecloseN(\#\#M, ec, x) \longleftrightarrow ec = ecloseN(x)$
unfolding *is_ecloseN_def ecloseN_def*
using *is_eclose_n_abs Un_closed union_abs ecloseN_closed*
by *auto*

lemma *freqR_abs* :

$x \in M \implies y \in M \implies freqR(x, y) \longleftrightarrow is_freqR(\#\#M, x, y)$
unfolding *freqR_def is_freqR_def* **using** *components_closed domain_closed* **by**
simp

lemma *frecrelP_abs* :

$z \in M \implies frecrelP(\#\#M, z) \longleftrightarrow (\exists x y. z = \langle x, y \rangle \wedge freqR(x, y))$
using *pair_in_M_iff freqR_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 $\langle A \in M \rangle$
have $z \in M$ **if** $z \in A \times A$ **for** z
using *cartprod_closed transitivity that* **by** *simp*
then
have $Collect(A \times A, frecrelP(\#\#M)) = Collect(A \times A, \lambda z. (\exists x y. z = \langle x, y \rangle \wedge freqR(x, y)))$
using *Collect_cong[of A × A A × 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 $Collect(x \times x, \lambda z. (\exists x y. z = \langle x, y \rangle \wedge freqR(x, y))) \in M$
using *Collect_in_M_0p[of frecrelP_fm(0)]* *arity_frecrelP_fm sats_frecrelP_fm*
 $frecrelP_abs \langle x \in M \rangle$ *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* : $uv \in \text{forcerel}(P,x) \implies uv \in \text{names_below}(P,x) \times \text{names_below}(P,x)$
unfolding *forcerel_def*
using *trancl_type field_frecrel* **by** *blast*

lemma *wf_forcerel* :
 $wf(\text{forcerel}(P,x))$
unfolding *forcerel_def* **using** *wf_trancl wf_frecrel* .

lemma *restrict_trancl_forcerel*:
assumes $\text{frecR}(w,y)$
shows $\text{restrict}(f,\text{frecrel}(\text{names_below}(P,x))-\{\!-\}\{y\})'w$
 $= \text{restrict}(f,\text{forcerel}(P,x)-\{\!-\}\{y\})'w$
unfolding *forcerel_def frecrel_def* **using** *assms restrict_trancl_Rrel[of frecR]*
by *simp*

lemma *names_belowI* :
assumes $\text{frecR}(\langle ft,n1,n2,p \rangle, \langle a,b,c,d \rangle)$ $p \in P$
shows $\langle ft,n1,n2,p \rangle \in \text{names_below}(P, \langle a,b,c,d \rangle)$ (**is** $?x \in \text{names_below}(_,?y)$)
proof -

from *assms*
have $ft \in 2$ $a \in 2$
unfolding *frecR_def* **by** (*auto simp add:components_simp*)
from *assms*
consider (*e*) $n1 \in \text{domain}(b) \cup \text{domain}(c) \wedge (n2 = b \vee n2 = c)$
 $|$ (*m*) $n1 = b \wedge n2 \in \text{domain}(c)$
unfolding *frecR_def* **by** (*auto simp add:components_simp*)
then show *?thesis*
proof cases

case e
then
have $n1 \in \text{eclose}(b) \vee n1 \in \text{eclose}(c)$
using *Un_iff in_dom_in_eclose* **by** *auto*
with e
have $n1 \in \text{ecloseN}(?y)$ $n2 \in \text{ecloseN}(?y)$
using *ecloseNI components_in_eclose* **by** *auto*
with $\langle ft \in 2 \rangle$ $\langle p \in P \rangle$
show *?thesis* **unfolding** *names_below_def* **by** *auto*

next
case m
then
have $n1 \in \text{ecloseN}(?y)$ $n2 \in \text{ecloseN}(?y)$
using *mem_eclose_trans* *ecloseNI*
 in_dom_in_eclose *components_in_eclose* **by** *auto*
with $\langle ft \in 2 \rangle$ $\langle p \in P \rangle$
show *?thesis* **unfolding** *names_below_def*
by *auto*

qed
qed

```

lemma names_below_tr :
  assumes  $x \in \text{names\_below}(P,y)$ 
     $y \in \text{names\_below}(P,z)$ 
  shows  $x \in \text{names\_below}(P,z)$ 
proof -
  let  $?A = \lambda y . \text{names\_below}(P,y)$ 
  from assms
  obtain  $fx\ x1\ x2\ px$  where
     $x = \langle fx,x1,x2,px \rangle$   $fx \in 2$   $x1 \in \text{ecloseN}(y)$   $x2 \in \text{ecloseN}(y)$   $px \in P$ 
  unfolding names_below_def by auto
  from assms
  obtain  $fy\ y1\ y2\ py$  where
     $y = \langle fy,y1,y2,py \rangle$   $fy \in 2$   $y1 \in \text{ecloseN}(z)$   $y2 \in \text{ecloseN}(z)$   $py \in P$ 
  unfolding names_below_def by auto
  from  $\langle x1 \in \_ \rangle$   $\langle x2 \in \_ \rangle$   $\langle y1 \in \_ \rangle$   $\langle y2 \in \_ \rangle$   $\langle x = \_ \rangle$   $\langle y = \_ \rangle$ 
  have  $x1 \in \text{ecloseN}(z)$   $x2 \in \text{ecloseN}(z)$ 
    using ecloseN_mono names_simp by auto
  with  $\langle fx \in 2 \rangle$   $\langle px \in P \rangle$   $\langle x = \_ \rangle$ 
  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  $\langle x,y \rangle \in \text{frecrel}(\text{names\_below}(P,z))$ 
  shows  $x \in \text{names\_below}(P,y)$ 
proof -
  {
    from assms
    have  $x \in \text{names\_below}(P,z)$   $y \in \text{names\_below}(P,z)$  frecR( $x,y$ )
      unfolding frecrel_def Rrel_def
      by auto
    obtain  $f\ n1\ n2\ p$  where
       $x = \langle f,n1,n2,p \rangle$   $f \in 2$   $n1 \in \text{ecloseN}(z)$   $n2 \in \text{ecloseN}(z)$   $p \in P$ 
      using  $\langle x \in \text{names\_below}(P,z) \rangle$ 
      unfolding names_below_def by auto
    moreover
    obtain  $fy\ m1\ m2\ q$  where
       $q \in P$   $y = \langle fy,m1,m2,q \rangle$ 
      using  $\langle y \in \text{names\_below}(P,z) \rangle$ 
      unfolding names_below_def by auto
    moreover
    note  $\langle \text{frecR}(x,y) \rangle$ 
    ultimately
    have  $x \in \text{names\_below}(P,y)$  using names_belowI by simp
  }
  then show ?thesis .
qed

```

```

lemma arg_into_names_below :
  assumes  $\langle x,y \rangle \in \text{frecrel}(\text{names\_below}(P,z))$ 
  shows  $x \in \text{names\_below}(P,x)$ 
proof -
  {
    from assms
    have  $x \in \text{names\_below}(P,z)$ 
      unfolding frecrel_def Rrel_def
      by auto
    from  $\langle x \in \text{names\_below}(P,z) \rangle$ 
    obtain  $f\ n1\ n2\ p$  where
       $x = \langle f,n1,n2,p \rangle$   $f \in 2$   $n1 \in \text{eclose}N(z)$   $n2 \in \text{eclose}N(z)$   $p \in P$ 
      unfolding names_below_def by auto
    then
      have  $n1 \in \text{eclose}N(x)$   $n2 \in \text{eclose}N(x)$ 
        using components_in_eclose by simp_all
      with  $\langle f \in 2 \rangle$   $\langle p \in P \rangle$   $\langle x = \langle f,n1,n2,p \rangle \rangle$ 
      have  $x \in \text{names\_below}(P,x)$ 
        unfolding names_below_def by simp
    }
  then show ?thesis .
qed

lemma forcerec_arg_into_names_below :
  assumes  $\langle x,y \rangle \in \text{forcerec}(P,z)$ 
  shows  $x \in \text{names\_below}(P,x)$ 
  using assms
  unfolding forcerec_def
  by(rule trancl_induct;auto simp add: arg_into_names_below)

lemma names_below_mono :
  assumes  $\langle x,y \rangle \in \text{frecrel}(\text{names\_below}(P,z))$ 
  shows  $\text{names\_below}(P,x) \subseteq \text{names\_below}(P,y)$ 
proof -
  from assms
  have  $x \in \text{names\_below}(P,y)$ 
    using arg_into_names_below2 by simp
  then
  show ?thesis
    using names_below_tr subsetI by simp
qed

lemma frecrel_mono :
  assumes  $\langle x,y \rangle \in \text{frecrel}(\text{names\_below}(P,z))$ 
  shows  $\text{frecrel}(\text{names\_below}(P,x)) \subseteq \text{frecrel}(\text{names\_below}(P,y))$ 
  unfolding frecrel_def
  using Rrel_mono names_below_mono assms by simp

lemma forcerec_mono2 :

```

```

assumes  $\langle x,y \rangle \in \text{frecrel}(\text{names\_below}(P,z))$ 
shows  $\text{forcere}l(P,x) \subseteq \text{forcere}l(P,y)$ 
unfolding forcere}_def
using trancl_mono frecrel_mono assms by simp

lemma forcere}_mono_aux :
assumes  $\langle x,y \rangle \in \text{frecrel}(\text{names\_below}(P,w))^+$ 
shows  $\text{forcere}l(P,x) \subseteq \text{forcere}l(P,y)$ 
using assms
by (rule trancl_induct,simp_all add: subset_trans forcere}_mono2)

lemma forcere}_mono :
assumes  $\langle x,y \rangle \in \text{forcere}l(P,z)$ 
shows  $\text{forcere}l(P,x) \subseteq \text{forcere}l(P,y)$ 
using forcere}_mono_aux assms unfolding forcere}_def by simp

lemma aux:  $x \in \text{names\_below}(P,w) \implies \langle x,y \rangle \in \text{forcere}l(P,z) \implies$ 
 $(y \in \text{names\_below}(P,w) \longrightarrow \langle x,y \rangle \in \text{forcere}l(P,w))$ 
unfolding forcere}_def
proof(rule_tac a=x and b=y and P=\lambda y . y \in \text{names\_below}(P,w) \longrightarrow \langle x,y \rangle \in
frecrel(\text{names\_below}(P,w))^+ in trancl_induct,simp)
  let  $?A = \lambda a . \text{names\_below}(P,a)$ 
  let  $?R = \lambda a . \text{frecrel}(?A(a))$ 
  let  $?fR = \lambda a . \text{forcere}l(a)$ 
  show  $u \in ?A(w) \longrightarrow \langle x,u \rangle \in ?R(w)^+ \text{ if } x \in ?A(w) \langle x,y \rangle \in ?R(z)^+ \langle x,u \rangle \in ?R(z)$ 
for  $u$ 
  using that frecrelD frecrelI r_into_trancl unfolding names\_below_def by
simp
  {
    fix  $u v$ 
    assume  $x \in ?A(w)$ 
     $\langle x,y \rangle \in ?R(z)^+$ 
     $\langle x,u \rangle \in ?R(z)^+$ 
     $\langle u,v \rangle \in ?R(z)$ 
     $u \in ?A(w) \implies \langle x,u \rangle \in ?R(w)^+$ 
    then
    have  $v \in ?A(w) \implies \langle x,v \rangle \in ?R(w)^+$ 
    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  $\langle v \in ?A(w) \rangle$ 
      have  $u \in ?A(w)$ 
      using names\_below_tr by simp
      with  $\langle v \in \_ \rangle \langle u,v \rangle \in \_$ 
      have  $\langle u,v \rangle \in ?R(w)$ 
      using frecrelD frecrelI r_into_trancl unfolding names\_below_def by simp
      with  $\langle u \in ?A(w) \implies \langle x,u \rangle \in ?R(w)^+ \rangle \langle u \in ?A(w) \rangle$ 

```

```

    have  $\langle x, u \rangle \in ?R(w)^{\wedge+}$  by simp
    with  $\langle u, v \rangle \in ?R(w)$ 
    show  $\langle x, v \rangle \in ?R(w)^{\wedge+}$  using trancl_trans r_into_trancl
    by simp
  qed
}
then show  $v \in ?A(w) \longrightarrow \langle x, v \rangle \in ?R(w)^{\wedge+}$ 
  if  $x \in ?A(w)$ 
   $\langle x, y \rangle \in ?R(z)^{\wedge+}$ 
   $\langle x, u \rangle \in ?R(z)^{\wedge+}$ 
   $\langle u, v \rangle \in ?R(z)$ 
   $u \in ?A(w) \longrightarrow \langle x, u \rangle \in ?R(w)^{\wedge+}$  for  $u v$ 
  using that by simp
qed

```

```

lemma forcereq_eq :
  assumes  $\langle z, x \rangle \in forcereq(P, x)$ 
  shows  $forcereq(P, z) = forcereq(P, x) \cap \text{names\_below}(P, z) \times \text{names\_below}(P, z)$ 
  using assms aux forcereqD forcereq_mono[of z x x] subsetI
  by auto

```

```

lemma forcereq_below_aux :
  assumes  $\langle z, x \rangle \in forcereq(P, x)$   $\langle u, z \rangle \in forcereq(P, x)$ 
  shows  $u \in \text{names\_below}(P, z)$ 
  using assms(2)
  unfolding forcereq_def
proof(rule trancl_induct)
  show  $u \in \text{names\_below}(P, y)$  if  $\langle u, y \rangle \in \text{frecrel}(\text{names\_below}(P, x))$  for  $y$ 
    using that vimage_singleton_iff arg_into_names_below2 by simp
next
  show  $u \in \text{names\_below}(P, z)$ 
    if  $\langle u, y \rangle \in \text{frecrel}(\text{names\_below}(P, x))^{\wedge+}$ 
     $\langle y, z \rangle \in \text{frecrel}(\text{names\_below}(P, x))$ 
     $u \in \text{names\_below}(P, y)$ 
    for  $y z$ 
    using that arg_into_names_below2[of y z x] names_below_tr by simp
qed

```

```

lemma forcereq_below :
  assumes  $\langle z, x \rangle \in forcereq(P, x)$ 
  shows  $forcereq(P, x) - \{\{z\}\} \subseteq \text{names\_below}(P, z)$ 
  using vimage_singleton_iff assms forcereq_below_aux by auto

```

```

lemma relation_forcereq :
  shows  $\text{relation}(forcereq(P, z)) \text{trans}(forcereq(P, z))$ 
  unfolding forcereq_def using relation_trancl trans_trancl by simp_all

```

```

lemma Hfrc_restrict_trancl:  $\text{bool\_of\_o}(Hfrc(P, \text{leq}, y, \text{restrict}(f, \text{frecrel}(\text{names\_below}(P, x)) - \{\{y\}\})))$ 
  =  $\text{bool\_of\_o}(Hfrc(P, \text{leq}, y, \text{restrict}(f, (\text{frecrel}(\text{names\_below}(P, x))^{\wedge+} - \{\{y\}\})))$ 

```

unfolding *Hfrc_def bool_of_o_def eq_case_def mem_case_def*
using *restrict_trancl_forcerel frecRI1 frecRI2 frecRI3*
unfolding *forcerel_def*
by *simp*

lemma *frc_at_trancl*: $frc_at(P, leq, z) = wfrec(forcerel(P, z), z, \lambda x f. bool_of_o(Hfrc(P, leq, x, f)))$
unfolding *frc_at_def forcerel_def* **using** *wf_eq_trancl Hfrc_restrict_trancl* **by**
simp

lemma *forcerelI1* :
assumes $n1 \in domain(b) \vee n1 \in domain(c) \ p \in P \ d \in P$
shows $\langle \langle 1, n1, b, p \rangle, \langle 0, b, c, d \rangle \rangle \in forcerel(P, \langle 0, b, c, d \rangle)$
proof -
let $?x = \langle 1, n1, b, p \rangle$
let $?y = \langle 0, b, c, d \rangle$
from *assms*
have *frecR(?x, ?y)*
using *frecRI1* **by** *simp*
then
have $?x \in names_below(P, ?y) \ ?y \in names_below(P, ?y)$
using *names_belowI assms components_in_eclose*
unfolding *names_below_def* **by** *auto*
with $\langle frecR(?x, ?y) \rangle$
show *?thesis*
unfolding *forcerel_def frecrel_def*
using *subsetD[OF r_subset_trancl[OF relation_Rrel]] RrelI*
by *auto*
qed

lemma *forcerelI2* :
assumes $n1 \in domain(b) \vee n1 \in domain(c) \ p \in P \ d \in P$
shows $\langle \langle 1, n1, c, p \rangle, \langle 0, b, c, d \rangle \rangle \in forcerel(P, \langle 0, b, c, d \rangle)$
proof -
let $?x = \langle 1, n1, c, p \rangle$
let $?y = \langle 0, b, c, d \rangle$
from *assms*
have *frecR(?x, ?y)*
using *frecRI2* **by** *simp*
then
have $?x \in names_below(P, ?y) \ ?y \in names_below(P, ?y)$
using *names_belowI assms components_in_eclose*
unfolding *names_below_def* **by** *auto*
with $\langle frecR(?x, ?y) \rangle$
show *?thesis*
unfolding *forcerel_def frecrel_def*
using *subsetD[OF r_subset_trancl[OF relation_Rrel]] RrelI*
by *auto*

qed

lemma *forcereI3* :

assumes $\langle n2, r \rangle \in c$ $p \in P$ $d \in P$ $r \in P$
shows $\langle \langle 0, b, n2, p \rangle, \langle 1, b, c, d \rangle \rangle \in \text{forcereI}(P, \langle 1, b, c, d \rangle)$

proof -

let $?x = \langle 0, b, n2, p \rangle$
let $?y = \langle 1, b, c, d \rangle$
from *assms*
have $\text{frecR}(\?x, \?y)$
using *assms* *frecRI3* **by** *simp*
then
have $?x \in \text{names_below}(P, \?y)$ $?y \in \text{names_below}(P, \?y)$
using *names_belowI* *assms* *components_in_eclose*
unfolding *names_below_def* **by** *auto*
with $\langle \text{frecR}(\?x, \?y) \rangle$
show *?thesis*
unfolding *forcereI_def* *frecrI_def*
using *subsetD* [*OF* *r_subset_trancI* [*OF* *relation_Rrel*]] *RrelI*
by *auto*

qed

lemmas *forcereI1* = *forcereI1* [*THEN* *vimage_singleton_iff* [*THEN* *iffD2*]]
forcereI2 [*THEN* *vimage_singleton_iff* [*THEN* *iffD2*]]
forcereI3 [*THEN* *vimage_singleton_iff* [*THEN* *iffD2*]]

lemma *aux_def_frc_at*:

assumes $z \in \text{forcereI}(P, x)$ -“ $\{x\}$
shows $\text{wfrec}(\text{forcereI}(P, x), z, H) = \text{wfrec}(\text{forcereI}(P, z), z, H)$

proof -

let $?A = \text{names_below}(P, z)$
from *assms*
have $\langle z, x \rangle \in \text{forcereI}(P, x)$
using *vimage_singleton_iff* **by** *simp*
then
have $z \in ?A$
using *forcereI_arg_into_names_below* **by** *simp*
from $\langle \langle z, x \rangle \in \text{forcereI}(P, x) \rangle$
have $E: \text{forcereI}(P, z) = \text{forcereI}(P, x) \cap (?A \times ?A)$
 $\text{forcereI}(P, x)$ -“ $\{z\} \subseteq ?A$
using *forcereI_eq_forcereI_below*
by *auto*
with $\langle z \in ?A \rangle$
have $\text{wfrec}(\text{forcereI}(P, x), z, H) = \text{wfrec}[?A](\text{forcereI}(P, x), z, H)$
using *wfrec_trans_restr* [*OF* *relation_forcereI(1)* *wf_forcereI_relation_forcereI(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

$$\begin{aligned} & frc_at(P, leq, \langle ft, n1, n2, p \rangle) = \\ & bool_of_o(p \in P \wedge \\ & (ft = 0 \wedge (\forall s. s \in domain(n1) \cup domain(n2) \longrightarrow \\ & (\forall q. q \in P \wedge q \preceq p \longrightarrow (frc_at(P, leq, \langle 1, s, n1, q \rangle) = 1 \longleftrightarrow frc_at(P, leq, \langle 1, s, n2, q \rangle) \\ & = 1))) \\ & \vee ft = 1 \wedge (\forall v \in P. v \preceq p \longrightarrow \\ & (\exists q. \exists s. \exists r. r \in P \wedge q \in P \wedge q \preceq v \wedge \langle s, r \rangle \in n2 \wedge q \preceq r \wedge frc_at(P, leq, \langle 0, n1, s, q \rangle) \\ & = 1)))) \end{aligned}$$

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 $\dots = ?Hf(?arg, \lambda x \in ?r(?arg) - \{?arg\}. frc_at(P, leq, x))$

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, λz. ∃ x y. z = ⟨x, y⟩ ∧ frecR(x, y))
  proof -
    have arity(frecrelP_fm(0)) = 1
      unfolding number1_fm_def frecrelP_fm_def
      by (simp del:FOL_sats_iff pair_abs empty_abs
          add: fm_defs frecR_fm_def number1_fm_def components_defs nat_simp_union)
    then
    have separation(##M, λz. sats(M, frecrelP_fm(0), [z]))
      using separation_ax by simp
    moreover
    have frecrelP(##M, z) ↔ sats(M, frecrelP_fm(0), [z])
      if z∈M for z
      using that sats_frecrelP_fm[of 0 [z]] by simp
    ultimately
    have separation(##M, frecrelP(##M))
      unfolding separation_def by simp
    then
    show ?thesis using frecrelP_abs
      separation_cong[of ##M frecrelP(##M) λz. ∃ x y. z = ⟨x, y⟩ ∧ frecR(x,
y)]
      by simp
    qed
  ultimately
  show {z ∈ ?Q × ?Q . ∃ x y. z = ⟨x, y⟩ ∧ frecR(x, y)}+ ∈ M
    using separation_closed frecrelP_abs trancl_closed by simp
  qed

lemma relation2_Hfrc_at_abs:
  relation2(##M, is_Hfrc_at(##M, P, leq), λx f. bool_of_o(Hfrc(P, leq, x, f)))
  unfolding relation2_def using Hfrc_at_abs
  by simp

lemma Hfrc_at_closed :
  ∀ x∈M. ∀ g∈M. function(g) → bool_of_o(Hfrc(P, leq, x, g))∈M
  unfolding bool_of_o_def using zero_in_M 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  $M$   $\langle X \in M \rangle$  forcerel in  $M$ 
  is Hfrc at iff sats[of concl: $M$   $P$  leq  $a$   $b$   $c$   $8$   $9$   $2$   $1$   $0$ 
    [ $c, b, a, d, e, y, x, z, P, \text{leq}, \text{forcerel}(P, X)$ ]] by simp
have  $1$ :sats( $M, \text{is\_wfrec\_fm}(\text{Hfrc\_at\_fm}(8, 9, 2, 1, 0), 5, 1, 0), [y, x, z, P, \text{leq}, \text{forcerel}(P, X)])$ )
 $\longleftrightarrow$ 
  is wfrec( $\#\#M, \text{is\_Hfrc\_at}(\#\#M, P, \text{leq}), \text{forcerel}(P, X), x, y)$ )
if  $x \in M$   $y \in M$   $z \in M$  for  $x$   $y$   $z$ 
using that  $\langle X \in M \rangle$  forcerel in  $M$   $P$  in  $M$  leq in  $M$ 
  sats is wfrec fm[ $OF$   $0$ ]
by simp
let
   $?f = \text{Exists}(\text{And}(\text{pair\_fm}(1, 0, 2), \text{is\_wfrec\_fm}(\text{Hfrc\_at\_fm}(8, 9, 2, 1, 0), 5, 1, 0)))$ 
have satsf:sats( $M, ?f, [x, z, P, \text{leq}, \text{forcerel}(P, X)]$ )  $\longleftrightarrow$ 
  ( $\exists y \in M. \text{pair}(\#\#M, x, y, z) \ \& \ \text{is\_wfrec}(\#\#M, \text{is\_Hfrc\_at}(\#\#M, P, \text{leq}), \text{forcerel}(P, X),$ 
 $x, y)$ )
if  $x \in M$   $z \in M$  for  $x$   $z$ 
using that  $1$   $\langle X \in M \rangle$  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  $?f \in \text{formula}$ 
unfolding fm defs Hfrc at fm def by simp
ultimately
have strong replacement( $\#\#M, \lambda x z. \text{sats}(M, ?f, [x, z, P, \text{leq}, \text{forcerel}(P, X)])$ )
using replacement ax  $1$  artyf  $\langle X \in M \rangle$  forcerel in  $M$   $P$  in  $M$  leq in  $M$  by
simp
then
have strong replacement( $\#\#M, \lambda x z. \exists y \in M. \text{pair}(\#\#M, x, y, z) \ \& \ \text{is\_wfrec}(\#\#M, \text{is\_Hfrc\_at}(\#\#M, P, \text{leq}), \text{forcerel}(P, X),$ 
 $x, y)$ )
using repl sats[of  $M$   $?f$  [ $P, \text{leq}, \text{forcerel}(P, X)$ ]] satsf by (simp del:pair abs)
then
show thesis unfolding wfrec replacement def by simp
qed

lemma names below abs :
[[ $Q \in M; x \in M; nb \in M$ ]]  $\implies \text{is\_names\_below}(\#\#M, Q, x, nb) \longleftrightarrow nb = \text{names\_below}(Q, x)$ 
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:

```

$\llbracket Q \in M; x \in M \rrbracket \implies \text{names_below}(Q, x) \in M$
unfolding *names_below_def*
using *zero_in_M cartprod_closed ecloseN_closed succ_in_M_iff*
by *simp*

lemma *names_below_productE* :
assumes $Q \in M \ x \in M$
 $\bigwedge A1 \ A2 \ A3 \ A4. A1 \in M \implies A2 \in M \implies A3 \in M \implies A4 \in M \implies R(A1$
 $\times A2 \times A3 \times A4)$
shows $R(\text{names_below}(Q, x))$
unfolding *names_below_def* **using** *assms zero_in_M ecloseN_closed[of x] twoN_in_M*
by *auto*

lemma *forcerel_abs* :
 $\llbracket x \in M; z \in M \rrbracket \implies \text{is_forcerel}(\#\#M, P, x, z) \longleftrightarrow z = \text{forcerel}(P, x)$
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
 $\text{names_below_productE}[of\ concl:\lambda p. \text{is_frecrel}(\#\#M, p, _) \longleftrightarrow _ = \text{frecrel}(p)]$
frecrel_closed
by *simp*

lemma *frc_at_abs*:
assumes $f\text{nn}c \in M \ z \in M$
shows $\text{is_frc_at}(\#\#M, P, \text{leq}, f\text{nn}c, z) \longleftrightarrow z = \text{frc_at}(P, \text{leq}, f\text{nn}c)$
proof -
from *assms*
have $(\exists r \in M. \text{is_forcerel}(\#\#M, P, f\text{nn}c, r) \wedge \text{is_wfrec}(\#\#M, \text{is_Hfrc_at}(\#\#M,$
 $P, \text{leq}), r, f\text{nn}c, z))$
 $\longleftrightarrow \text{is_wfrec}(\#\#M, \text{is_Hfrc_at}(\#\#M, P, \text{leq}), \text{forcerel}(P, f\text{nn}c), f\text{nn}c, z)$
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 fnn] wf_forcerel trans_forcerel_t relation_forcerel_t*
forcerel_in_M
 $\text{Hfrc_at_closed_relation2_Hfrc_at_abs}$
 $\text{trans_wfrec_abs}[of\ forcerel}(P, f\text{nn}c) \ f\text{nn}c \ z \ \text{is_Hfrc_at}(\#\#M, P, \text{leq}) \ \lambda x \ f.$
 $\text{bool_of_o}(\text{Hfrc}(P, \text{leq}, x, f))]$
by (*simp flip:setclass_iff*)
qed

lemma *forces_eq'_abs* :
 $\llbracket p \in M ; t1 \in M ; t2 \in M \rrbracket \implies \text{is_forces_eq}'(\#\#M, P, \text{leq}, p, t1, t2) \longleftrightarrow \text{forces_eq}'(P, \text{leq}, p, t1, t2)$
unfolding *is_forces_eq'_def forces_eq'_def*
using *frc_at_abs zero_in_M tuples_in_M* **by** *auto*

lemma *forces_mem'_abs* :
 $\llbracket p \in M ; t1 \in M ; t2 \in M \rrbracket \implies \text{is_forces_mem}'(\#\#M, P, \text{leq}, p, t1, t2) \longleftrightarrow \text{forces_mem}'(P, \text{leq}, p, t1, t2)$

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($\lambda p. incr_bv(p)$ '1 , 2, φ)))*

lemma *ren_forces_nand_type*[*TC*] :

φ ∈ *formula* ⇒ *ren_forces_nand*(φ) ∈ *formula*

unfolding *ren_forces_nand_def*

by *simp*

lemma *arity_ren_forces_nand* :

assumes φ ∈ *formula*

shows *arity*(*ren_forces_nand*(φ)) ≤ *succ*(*arity*(φ))

proof -

```

consider (lt) 1 < arity( $\varphi$ ) | (ge)  $\neg$  1 < arity( $\varphi$ )
  by auto
then
show ?thesis
proof cases
  case lt
    with  $\langle \varphi \in \_ \rangle$ 
    have 2 < succ(arity( $\varphi$ )) 2 < arity( $\varphi$ )# + 2
      using succ_ltI by auto
    with  $\langle \varphi \in \_ \rangle$ 
    have arity(iterates( $\lambda p$ . incr_bv(p)'1,2, $\varphi$ )) = 2# + arity( $\varphi$ )
      using arity_incr_bv_lemma lt
      by auto
    with  $\langle \varphi \in \_ \rangle$ 
    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  $\langle \varphi \in \_ \rangle$ 
      have arity( $\varphi$ )  $\leq$  1 pred(arity( $\varphi$ ))  $\leq$  1
        using not_lt_iff_le le_trans[OF le_pred]
        by simp_all
      with  $\langle \varphi \in \_ \rangle$ 
      have arity(iterates( $\lambda p$ . incr_bv(p)'1,2, $\varphi$ )) = (arity( $\varphi$ ))
        using arity_incr_bv_lemma ge
        by simp
      with  $\langle$ arity( $\varphi$ )  $\leq$  1 $\rangle$   $\langle \varphi \in \_ \rangle$   $\langle$ pred( $\_$ )  $\leq$  1 $\rangle$ 
      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  $\in$  list(M)  $\implies$   $\varphi \in$  formula  $\implies$ 
  sats(M, ren_forces_nand( $\varphi$ ), [q,p,P,leq,o] @ env)  $\longleftrightarrow$  sats(M,  $\varphi$ , [q,P,leq,o] @
env)
  unfolding ren_forces_nand_def
  using sats_incr_bv_iff [of _ _ M _ [q]]
  by simp

```

definition

```

ren_forces_forall ::  $i \Rightarrow i$  where
ren_forces_forall( $\varphi$ )  $\equiv$ 
  Exists(Exists(Exists(Exists(Exists(

```

$And(Equal(0,6),And(Equal(1,7),And(Equal(2,8),And(Equal(3,9),$
 $And(Equal(4,5),iterates(\lambda p. incr_bv(p)'5 , 5, \varphi))))))))))$

lemma *arity_ren_forces_all* :
assumes $\varphi \in \text{formula}$
shows $\text{arity}(\text{ren_forces_forall}(\varphi)) = 5 \cup \text{arity}(\varphi)$
proof -
consider $(lt) \ 5 < \text{arity}(\varphi) \mid (ge) \ \neg \ 5 < \text{arity}(\varphi)$
by *auto*
then
show *?thesis*
proof *cases*
case *lt*
with $\langle \varphi \in _ \rangle$
have $5 < \text{succ}(\text{arity}(\varphi)) \ 5 < \text{arity}(\varphi) \# + 2 \ 5 < \text{arity}(\varphi) \# + 3 \ 5 < \text{arity}(\varphi) \# + 4$
using *succ_ltI* **by** *auto*
with $\langle \varphi \in _ \rangle$
have $\text{arity}(\text{iterates}(\lambda p. incr_bv(p)'5, 5, \varphi)) = 5 \# + \text{arity}(\varphi)$
using *arity_incr_bv_lemma* *lt*
by *simp*
with $\langle \varphi \in _ \rangle$
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 $\langle \varphi \in _ \rangle$
have $\text{arity}(\varphi) \leq 5 \ \text{pred}^5(\text{arity}(\varphi)) \leq 5$
using *not_lt_iff_le* *le_trans[OF le_pred]*
by *simp_all*
with $\langle \varphi \in _ \rangle$
have $\text{arity}(\text{iterates}(\lambda p. incr_bv(p)'5, 5, \varphi)) = \text{arity}(\varphi)$
using *arity_incr_bv_lemma* *ge*
by *simp*
with $\langle \text{arity}(\varphi) \leq 5 \rangle \ \langle \varphi \in _ \rangle \ \langle \text{pred}^5(_) \leq 5 \rangle$
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]* :
 $\varphi \in \text{formula} \implies \text{ren_forces_forall}(\varphi) \in \text{formula}$
unfolding *ren_forces_forall_def* **by** *simp*

lemma *sats_ren_forces_forall* :
 $[x, P, leq, o, p] \ @ \ env \in \text{list}(M) \implies \varphi \in \text{formula} \implies$

$sats(M, ren_forces_forall(\varphi), [x, p, P, leq, o] @ env) \longleftrightarrow sats(M, \varphi, [p, P, leq, o, x]$
 $@ env)$
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]: $\varphi \in formula \implies forces'(\varphi) \in formula$
by (*induct* φ *set:formula*; *simp*)

lemma *forces_type*[TC] : $\varphi \in formula \implies forces(\varphi) \in formula$
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*: $p \in P \implies forces_eq(p,t1,t2) \longleftrightarrow$

$(\forall s \in domain(t1) \cup domain(t2). \forall q. q \in P \wedge q \preceq p \longrightarrow$
 $(forces_mem(q,s,t1) \longleftrightarrow forces_mem(q,s,t2)))$

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*: $p \in P \implies forces_mem(p,t1,t2) \longleftrightarrow$

$(\forall v \in P. v \preceq p \longrightarrow$

$(\exists q. \exists s. \exists r. r \in P \wedge q \in P \wedge q \preceq v \wedge \langle s,r \rangle \in t2 \wedge q \preceq r \wedge forces_eq(q,t1,s)))$

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* :

$[p \in M ; t1 \in M ; t2 \in M] \implies is_forces_eq(p,t1,t2) \longleftrightarrow forces_eq(p,t1,t2)$
unfolding *is_forces_eq_def* *forces_eq_def*

using forces_eq'_abs by simp

lemma forces_mem_abs :

$\llbracket p \in M ; t1 \in M ; t2 \in M \rrbracket \implies is_forces_mem(p, t1, t2) \longleftrightarrow forces_mem(p, t1, t2)$

unfolding is_forces_mem_def forces_mem_def

using forces_mem'_abs by simp

lemma sats_forces_eq_fm:

assumes $p \in nat$ $l \in nat$ $q \in nat$ $t1 \in nat$ $t2 \in nat$ $env \in list(M)$

$nth(p, env) = P$ $nth(l, env) = leq$

shows $sats(M, forces_eq_fm(p, l, q, t1, t2), env) \longleftrightarrow$

$is_forces_eq(nth(q, env), nth(t1, env), nth(t2, env))$

unfolding forces_eq_fm_def is_forces_eq_def is_forces_eq'_def

using assms sats_is_tuple_fm sats_frc_at_fm

by simp

lemma sats_forces_mem_fm:

assumes $p \in nat$ $l \in nat$ $q \in nat$ $t1 \in nat$ $t2 \in nat$ $env \in list(M)$

$nth(p, env) = P$ $nth(l, env) = leq$

shows $sats(M, forces_mem_fm(p, l, q, t1, t2), env) \longleftrightarrow$

$is_forces_mem(nth(q, env), nth(t1, env), nth(t2, env))$

unfolding forces_mem_fm_def is_forces_mem_def is_forces_mem'_def

using assms sats_is_tuple_fm sats_frc_at_fm

by simp

definition

$forces_neq :: [i, i, i] \Rightarrow o$ **where**

$forces_neq(p, t1, t2) \equiv \neg (\exists q \in P. q \preceq p \wedge forces_eq(q, t1, t2))$

definition

$forces_nmem :: [i, i, i] \Rightarrow o$ **where**

$forces_nmem(p, t1, t2) \equiv \neg (\exists q \in P. q \preceq p \wedge forces_mem(q, t1, t2))$

lemma forces_neq :

$forces_neq(p, t1, t2) \longleftrightarrow forces_neq'(P, leq, p, t1, t2)$

unfolding forces_neq_def forces_neq'_def forces_eq_def by simp

lemma forces_nmem :

$forces_nmem(p, t1, t2) \longleftrightarrow forces_nmem'(P, leq, p, t1, t2)$

unfolding forces_nmem_def forces_nmem'_def forces_mem_def by simp

lemma sats_forces_Member :

assumes $x \in nat$ $y \in nat$ $env \in list(M)$

$nth(x, env) = xx$ $nth(y, env) = yy$ $q \in M$

shows $sats(M, forces(Member(x, y)), [q, P, leq, one]@env) \longleftrightarrow$

$(q \in P \wedge is_forces_mem(q, xx, yy))$

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 $x \in \text{nat } y \in \text{nat } \text{env} \in \text{list}(M)$
 $\text{nth}(x, \text{env}) = xx \text{ nth}(y, \text{env}) = yy \ q \in M$
shows $\text{sats}(M, \text{forces}(\text{Equal}(x, y)), [q, P, \text{leq}, \text{one}]@ \text{env}) \longleftrightarrow$
 $(q \in P \wedge \text{is_forces_eq}(q, xx, yy))$
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 $\varphi \in \text{formula } \psi \in \text{formula } \text{env} \in \text{list}(M) \ p \in M$
shows $\text{sats}(M, \text{forces}(\text{Nand}(\varphi, \psi)), [p, P, \text{leq}, \text{one}]@ \text{env}) \longleftrightarrow$
 $(p \in P \wedge \neg(\exists q \in M. q \in P \wedge \text{is_leq}(\#\#M, \text{leq}, q, p) \wedge$
 $(\text{sats}(M, \text{forces}'(\varphi), [q, P, \text{leq}, \text{one}]@ \text{env}) \wedge \text{sats}(M, \text{forces}'(\psi), [q, P, \text{leq}, \text{one}]@ \text{env}))))$
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 $\varphi \in \text{formula } \text{env} \in \text{list}(M) \ p \in M$
shows $\text{sats}(M, \text{forces}(\text{Neg}(\varphi)), [p, P, \text{leq}, \text{one}]@ \text{env}) \longleftrightarrow$
 $(p \in P \wedge \neg(\exists q \in M. q \in P \wedge \text{is_leq}(\#\#M, \text{leq}, q, p) \wedge$
 $(\text{sats}(M, \text{forces}'(\varphi), [q, P, \text{leq}, \text{one}]@ \text{env}))))$
unfolding *Neg_def* **using** *assms sats_forces_Nand*
by *simp*

lemma *sats_forces_Forall* :
assumes $\varphi \in \text{formula } \text{env} \in \text{list}(M) \ p \in M$
shows $\text{sats}(M, \text{forces}(\text{Forall}(\varphi)), [p, P, \text{leq}, \text{one}]@ \text{env}) \longleftrightarrow$
 $p \in P \wedge (\forall x \in M. \text{sats}(M, \text{forces}'(\varphi), [p, P, \text{leq}, \text{one}, x]@ \text{env}))$
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 $x \in \text{nat } y \in \text{nat}$
shows $\text{arity}(\text{forces}(\text{Member}(x, y))) = (\text{succ}(x) \cup \text{succ}(y)) \# + 4$
 $\text{arity}(\text{forces}(\text{Equal}(x, y))) = (\text{succ}(x) \cup \text{succ}(y)) \# + 4$
unfolding *forces_def*
using *assms arity_forces_mem_fm arity_forces_eq_fm succ_Un_distrib nat_simp_union*
by *auto*

```

lemma arity_forces':
  assumes  $\varphi \in \text{formula}$ 
  shows  $\text{arity}(\text{forces}'(\varphi)) \leq \text{arity}(\varphi) \# + 4$ 
  using assms
proof (induct set:formula)
  case (Member x y)
  then
  show ?case
    using arity_forces_mem_fm succ_Un_distrib nat_simp_union
    by simp
next
  case (Equal x y)
  then
  show ?case
    using arity_forces_eq_fm succ_Un_distrib nat_simp_union
    by simp
next
  case (Nand  $\varphi \psi$ )
  let ? $\varphi'$  = ren_forces_nand(forces'( $\varphi$ ))
  let ? $\psi'$  = ren_forces_nand(forces'( $\psi$ ))
  have  $\text{arity}(\text{leq\_fm}(3, 0, 1)) = 4$ 
    using arity_leq_fm succ_Un_distrib nat_simp_union
    by simp
  have  $3 \leq (4 \# + \text{arity}(\varphi)) \cup (4 \# + \text{arity}(\psi))$  (is  $\_ \leq ?\text{rhs}$ )
    using nat_simp_union by simp
  from  $\langle \varphi \in \_ \rangle$  Nand
  have  $\text{pred}(\text{arity}(?\varphi')) \leq ?\text{rhs}$   $\text{pred}(\text{arity}(?\psi')) \leq ?\text{rhs}$ 
  proof -
    from  $\langle \varphi \in \_ \rangle$   $\langle \psi \in \_ \rangle$ 
    have  $A: \text{pred}(\text{arity}(?\varphi')) \leq \text{arity}(\text{forces}'(\varphi))$ 
       $\text{pred}(\text{arity}(?\psi')) \leq \text{arity}(\text{forces}'(\psi))$ 
      using pred_mono[OF _ arity_ren_forces_nand] pred_succ_eq
      by simp_all
    from Nand
    have  $3 \cup \text{arity}(\text{forces}'(\varphi)) \leq \text{arity}(\varphi) \# + 4$ 
       $3 \cup \text{arity}(\text{forces}'(\psi)) \leq \text{arity}(\psi) \# + 4$ 
      using Un_le by simp_all
    with Nand
    show  $\text{pred}(\text{arity}(?\varphi')) \leq ?\text{rhs}$ 
       $\text{pred}(\text{arity}(?\psi')) \leq ?\text{rhs}$ 
      using le_trans[OF A(1)] le_trans[OF A(2)] le_Un_iff
      by simp_all
  qed
  with Nand  $\langle \_ = 4 \rangle$ 
  show ?case
    using pred_Un_distrib Un_assoc[symmetric] succ_Un_distrib nat_union_abs1
    Un_leI3[OF  $\langle 3 \leq ?\text{rhs} \rangle$ ]
    by simp

```

```

next
  case (Forall  $\varphi$ )
  let  $?\varphi' = \text{ren\_forces\_forall}(\text{forces}'(\varphi))$ 
  show ?case
  proof (cases  $\text{arity}(\varphi) = 0$ )
    case True
    with Forall
    show ?thesis
    proof -
      from Forall True
      have  $\text{arity}(\text{forces}'(\varphi)) \leq 5$ 
        using  $\text{le\_trans}[\text{of } 4\ 5]$  by auto
      with  $\langle \varphi \in \_ \rangle$ 
      have  $\text{arity}(\varphi) \leq 5$ 
        using  $\text{arity\_ren\_forces\_all}[\text{OF } \text{forces}'\_type[\text{OF } \langle \varphi \in \_ \rangle]] \text{ nat\_union\_abs2}$ 
        by auto
      with Forall True
      show ?thesis
        using  $\text{pred\_mono}[\text{OF } \_ \langle \text{arity}(\varphi) \leq 5 \rangle]$ 
        by simp
    qed
  next
  case False
  with Forall
  show ?thesis
  proof -
    from Forall False
    have  $\text{arity}(\varphi) = 5 \cup \text{arity}(\text{forces}'(\varphi))$ 
       $\text{arity}(\text{forces}'(\varphi)) \leq 5 \# + \text{arity}(\varphi)$ 
       $4 \leq \text{succ}(\text{succ}(\text{succ}(\text{arity}(\varphi))))$ 
      using  $\text{Ord\_0\_lt\_arity\_ren\_forces\_all}$ 
       $\text{le\_trans}[\text{OF } \_ \text{add\_le\_mono}[\text{of } 4\ 5, \text{OF } \_ \text{le\_refl}]]$ 
      by auto
    with  $\langle \varphi \in \_ \rangle$ 
    have  $5 \cup \text{arity}(\text{forces}'(\varphi)) \leq 5 \# + \text{arity}(\varphi)$ 
      using  $\text{nat\_simp\_union}$  by auto
    with  $\langle \varphi \in \_ \rangle \langle \text{arity}(\varphi) = 5 \cup \_ \rangle$ 
    show ?thesis
      using  $\text{pred\_Un\_distrib\_succ\_pred\_eq}[\text{OF } \_ \langle \text{arity}(\varphi) \neq 0 \rangle]$ 
       $\text{pred\_mono}[\text{OF } \_ \text{Forall}(2)] \text{ Un\_le}[\text{OF } \_ \langle 4 \leq \text{succ}(\_) \rangle]$ 
      by simp
    qed
  qed
qed

lemma  $\text{arity\_forces}$  :
  assumes  $\varphi \in \text{formula}$ 
  shows  $\text{arity}(\text{forces}(\varphi)) \leq 4 \# + \text{arity}(\varphi)$ 
  unfolding  $\text{forces\_def}$ 

```

using *assms* *arity_forces'* *le_trans* *nat_simp_union* **by** *auto*

lemma *arity_forces_le* :

assumes $\varphi \in \text{formula}$ $n \in \text{nat}$ $\text{arity}(\varphi) \leq n$

shows $\text{arity}(\text{forces}(\varphi)) \leq 4 \# + n$

using *assms* *le_trans*[*OF* *_add_le_mono*[*OF* *le_refl*[*of* 5] $\langle \text{arity}(\varphi) \leq _ \rangle$]] *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* :: $[i, i, i] \Rightarrow o$ ($\langle _ \Vdash _ \rangle$ [*36,36,36*] 60) **where**

$p \Vdash \varphi \text{ env} \equiv M, ([p, P, \text{leq}, \text{one}] @ \text{env}) \models \text{forces}(\varphi)$

lemma *Collect_forces* :

assumes

fty: $\varphi \in \text{formula}$ **and**

far: $\text{arity}(\varphi) \leq \text{length}(\text{env})$ **and**

envty: $\text{env} \in \text{list}(M)$

shows

$\{p \in P . p \Vdash \varphi \text{ env}\} \in M$

proof -

have $z \in P \implies z \in M$ **for** z

using *P_in_M* *transitivity*[*of* z *P*] **by** *simp*

moreover

have *separation*($\#\#M, \lambda p. (p \Vdash \varphi \text{ env})$)

using *separation_ax* *arity_forces* *far* *fty* *P_in_M* *leq_in_M* *one_in_M*

envty *arity_forces_le*

by *simp*

then

have *Collect*($P, \lambda p. (p \Vdash \varphi \text{ env})$) $\in M$

using *separation_closed* *P_in_M* **by** *simp*

then show *?thesis* **by** *simp*

qed

lemma *forces_mem_iff_dense_below*: $p \in P \implies \text{forces_mem}(p, t1, t2) \longleftrightarrow \text{dense_below}(\text{forces_mem}(p, t1, t2))$

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

18.2 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_eq*:
assumes $p \in P \ r \in P \ r \preceq p \ \text{forces_eq}(p, t1, t2)$
shows $\text{forces_eq}(r, t1, t2)$
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 $p \in P \ r \in P \ r \preceq p \ \text{forces_mem}(p, t1, t2)$
shows $\text{forces_mem}(r, t1, t2)$
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 $\text{forces_mem}(p, t1, t2) \longleftrightarrow \text{dense_below}(\{q \in P. \text{forces_mem}(q, t1, t2)\}, p)$
proof
assume $\text{forces_mem}(p, t1, t2)$
with *assms*
show $\text{dense_below}(\{q \in P. \text{forces_mem}(q, t1, t2)\}, p)$
using *forces_mem_iff_dense_below strengthening_mem*[of *p*] *ideal_dense_below*
by *auto*
next
assume $\text{dense_below}(\{q \in P. \text{forces_mem}(q, t1, t2)\}, p)$
with *assms*
have $\text{dense_below}(\{q \in P. \text{dense_below}(\{q' \in P. \exists s r. r \in P \wedge \langle s, r \rangle \in t2 \wedge q' \preceq r \wedge \text{forces_eq}(q', t1, s)\}, q)\}, p)$
using *forces_mem_iff_dense_below* **by** *simp*
with *assms*
show $\text{forces_mem}(p, t1, t2)$
using *dense_below_dense_below forces_mem_iff_dense_below*[of *p t1 t2*] **by** *blast*
qed

lemma *aux_density_eq*:
assumes
 $\text{dense_below}(\{q' \in P. \forall q. q \in P \wedge q \preceq q' \longrightarrow \text{forces_mem}(q, s, t1) \longleftrightarrow \text{forces_mem}(q, s, t2)\}, p)$
 $\text{forces_mem}(q, s, t1) \ q \in P \ p \in P \ q \preceq p$
shows
 $\text{dense_below}(\{r \in P. \text{forces_mem}(r, s, t2)\}, q)$

```

proof
  fix  $r$ 
  assume  $r \in P$   $r \preceq q$ 
  moreover from this and  $\langle p \in P \rangle \langle q \preceq p \rangle \langle q \in P \rangle$ 
  have  $r \preceq p$ 
    using  $leq\_transD$  by  $simp$ 
  moreover
  note  $\langle forces\_mem(q, s, t1) \rangle \langle dense\_below(\_, p) \rangle \langle q \in P \rangle$ 
  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  $forces\_eq(p, t1, t2) \iff dense\_below(\{q \in P . forces\_eq(q, t1, t2)\}, p)$ 
proof
  assume  $forces\_eq(p, t1, t2)$ 
  with  $\langle p \in P \rangle$ 
  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  $?D1 = \{q' \in P . \forall s \in domain(t1) \cup domain(t2) . \forall q . q \in P \wedge q \preceq q' \implies forces\_mem(q, s, t1) \iff forces\_mem(q, s, t2)\}$ 
    let  $?D2 = \{q' \in P . \forall q . q \in P \wedge q \preceq q' \implies forces\_mem(q, s, t1) \iff forces\_mem(q, s, t2)\}$ 
    assume  $s \in domain(t1) \cup domain(t2)$ 
    then
    have  $?D1 \subseteq ?D2$  by  $blast$ 
    with  $\langle dense\_below(\_, p) \rangle$ 
    have  $dense\_below(\{q' \in P . \forall s \in domain(t1) \cup domain(t2) . \forall q . q \in P \wedge q \preceq q' \implies forces\_mem(q, s, t1) \iff forces\_mem(q, s, t2)\}, p)$ 
    using  $dense\_below\_cong'[OF\ \langle p \in P \rangle\ def\_forces\_eq[of\ \_ t1\ t2]]$  by  $simp$ 
    with  $\langle p \in P \rangle \langle ?D1 \subseteq ?D2 \rangle$ 
    have  $dense\_below(\{q' \in P . \forall q . q \in P \wedge q \preceq q' \implies forces\_mem(q, s, t1) \iff forces\_mem(q, s, t2)\}, p)$ 
      using  $dense\_below\_mono$  by  $simp$ 
    moreover from this
    have  $dense\_below(\{q' \in P . \forall q . q \in P \wedge q \preceq q' \implies forces\_mem(q, s, t2) \iff forces\_mem(q, s, t1)\}, p)$ 
      by  $blast$ 
    moreover

```

```

assume  $q \in P \ q \preceq p$ 
moreover
note  $\langle p \in P \rangle$ 
ultimately
have  $\text{forces\_mem}(q, s, t1) \implies \text{dense\_below}(\{r \in P. \text{forces\_mem}(r, s, t2)\}, q)$ 
       $\text{forces\_mem}(q, s, t2) \implies \text{dense\_below}(\{r \in P. \text{forces\_mem}(r, s, t1)\}, q)$ 
      using  $\text{aux\_density\_eq}$  by  $\text{simp\_all}$ 
then
have  $\text{forces\_mem}(q, s, t1) \longleftrightarrow \text{forces\_mem}(q, s, t2)$ 
      using  $\text{density\_mem}[OF \langle q \in P \rangle]$  by  $\text{blast}$ 
}
with  $\langle p \in P \rangle$ 
show  $\text{forces\_eq}(p, t1, t2)$  using  $\text{def\_forces\_eq}$  by  $\text{blast}$ 
qed

```

18.5 Kunen 2013, Lemma IV.2.38

```

lemma  $\text{not\_forces\_neq}$ :
assumes  $p \in P$ 
shows  $\text{forces\_eq}(p, t1, t2) \longleftrightarrow \neg (\exists q \in P. q \preceq p \wedge \text{forces\_neq}(q, t1, t2))$ 
using  $\text{assms density\_eq unfolding forces\_neq\_def}$  by  $\text{blast}$ 

```

```

lemma  $\text{not\_forces\_nmem}$ :
assumes  $p \in P$ 
shows  $\text{forces\_mem}(p, t1, t2) \longleftrightarrow \neg (\exists q \in P. q \preceq p \wedge \text{forces\_nmem}(q, t1, t2))$ 
using  $\text{assms density\_mem unfolding forces\_nmem\_def}$  by  $\text{blast}$ 

```

```

lemma  $\text{sats\_forces\_Nand}'$ :
assumes
 $p \in P \ \varphi \in \text{formula} \ \psi \in \text{formula} \ \text{env} \in \text{list}(M)$ 
shows
 $M, [p, P, \text{leq}, \text{one}] @ \text{env} \models \text{forces}(\text{Nand}(\varphi, \psi)) \longleftrightarrow$ 
 $\neg (\exists q \in M. q \in P \wedge \text{is\_leq}(\#\#M, \text{leq}, q, p) \wedge$ 
 $M, [q, P, \text{leq}, \text{one}] @ \text{env} \models \text{forces}(\varphi) \wedge$ 
 $M, [q, P, \text{leq}, \text{one}] @ \text{env} \models \text{forces}(\psi))$ 
using  $\text{assms sats\_forces\_Nand}[OF \text{assms}(2-4) \text{transitivity}[OF \langle p \in P \rangle]]$ 
 $P\_in\_M \ \text{leq\_in\_M} \ \text{one\_in\_M}$  unfolding  $\text{forces\_def}$ 
by  $\text{simp}$ 

```

```

lemma  $\text{sats\_forces\_Neg}'$ :
assumes
 $p \in P \ \text{env} \in \text{list}(M) \ \varphi \in \text{formula}$ 
shows
 $M, [p, P, \text{leq}, \text{one}] @ \text{env} \models \text{forces}(\text{Neg}(\varphi)) \longleftrightarrow$ 

```

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

lemma *sats_forces_Forall'*:

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

18.6 The relation of forcing and atomic formulas

lemma *Forces_Equal*:

assumes
 $p \in P \ t1 \in M \ t2 \in M \ \text{env} \in \text{list}(M) \ \text{nth}(n, \text{env}) = t1 \ \text{nth}(m, \text{env}) = t2 \ n \in \text{nat} \ m \in \text{nat}$
shows
 $(p \Vdash \text{Equal}(n, m) \ \text{env}) \longleftrightarrow \text{forces_eq}(p, t1, t2)$
using *assms sats_forces_Equal forces_eq_abs transitivity P_in_M*
by *simp*

lemma *Forces_Member*:

assumes
 $p \in P \ t1 \in M \ t2 \in M \ \text{env} \in \text{list}(M) \ \text{nth}(n, \text{env}) = t1 \ \text{nth}(m, \text{env}) = t2 \ n \in \text{nat} \ m \in \text{nat}$
shows
 $(p \Vdash \text{Member}(n, m) \ \text{env}) \longleftrightarrow \text{forces_mem}(p, t1, t2)$
using *assms sats_forces_Member forces_mem_abs transitivity P_in_M*
by *simp*

lemma *Forces_Neg*:

assumes
 $p \in P \ \text{env} \in \text{list}(M) \ \varphi \in \text{formula}$
shows
 $(p \Vdash \text{Neg}(\varphi) \ \text{env}) \longleftrightarrow \neg(\exists q \in M. q \in P \wedge q \preceq p \wedge (q \Vdash \varphi \ \text{env}))$
using *assms sats_forces_Neg' transitivity*
 $P_in_M \text{pair_in_M iff_leq_in_M leq_abs}$ **by** *simp*

18.7 The relation of forcing and connectives

lemma *Forces_Nand*:

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

P_in_M pair_in_M iff leq_in_M leq_abs by simp

lemma *Forces_And_aux*:

assumes

$p \in P$ env \in list(M) $\varphi \in$ formula $\psi \in$ formula

shows

$p \Vdash And(\varphi, \psi) \text{ env} \iff$

$(\forall q \in M. q \in P \wedge q \preceq p \implies (\exists r \in M. r \in P \wedge r \preceq q \wedge (r \Vdash \varphi \text{ env}) \wedge (r \Vdash \psi \text{ env})))$

unfolding *And_def* **using** *assms Forces_Neg Forces_Nand* **by** (*auto simp only*.)

lemma *Forces_And_iff_dense_below*:

assumes

$p \in P$ env \in list(M) $\varphi \in$ formula $\psi \in$ formula

shows

$(p \Vdash And(\varphi, \psi) \text{ env}) \iff dense_below(\{r \in P. (r \Vdash \varphi \text{ env}) \wedge (r \Vdash \psi \text{ env})\}, p)$

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

$p \in P$ env \in list(M) $\varphi \in$ formula

shows

$(p \Vdash Forall(\varphi) \text{ env}) \iff (\forall x \in M. (p \Vdash \varphi ([x] @ \text{env})))$

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*: $\exists \vartheta. \exists p \in P. p \in G \wedge \langle \vartheta, p \rangle \in \pi \wedge val(G, \vartheta) = x \implies x \in val(G, \pi)$

by (*subst def_val, auto*)

lemma *GenExtD*: $x \in M[G] \iff (\exists \tau \in M. x = val(G, \tau))$

unfolding *GenExt_def* **by** *simp*

lemma *left_in_M* : $\tau \in M \implies \langle a, b \rangle \in \tau \implies a \in M$

using *fst_snd_closed*[*of* $\langle a, b \rangle$] *transitivity* **by** *auto*

18.8 Kunen 2013, Lemma IV.2.29

lemma *generic_inter_dense_below*:

assumes $D \in M$ M _generic(G) *dense_below*(D, p) $p \in G$

shows $D \cap G \neq 0$

proof -

```

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_refl[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)⟩

```


$\bigwedge q \sigma. q \in P \implies q \in G \implies \sigma \in \text{domain}(\tau) \implies \text{forces_eq}(q, \pi, \sigma) \implies \text{val}(G, \pi) = \text{val}(G, \sigma)$
shows
 $\text{val}(G, \pi) \in \text{val}(G, \tau)$
proof (*intro elem_of_valI*)
let $?D = \{q \in P. \exists \sigma. \exists r. r \in P \wedge \langle \sigma, r \rangle \in \tau \wedge q \preceq r \wedge \text{forces_eq}(q, \pi, \sigma)\}$
from $\langle M_generic(G) \rangle \langle p \in G \rangle$
have $p \in P$ **by** *blast*
moreover
note $\langle \pi \in M \rangle \langle \tau \in M \rangle$
ultimately
have $?D \in M$ **using** *IV240a_mem_Collect* **by** *simp*
moreover from *assms* $\langle p \in P \rangle$
have *dense_below*($?D, p$)
using *forces_mem_iff_dense_below* **by** *simp*
moreover
note $\langle M_generic(G) \rangle \langle p \in G \rangle$
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, π, σ) **by** *blast*
moreover from *this* **and** $\langle q \in G \rangle$ *assms*
have $r \in G$ $\text{val}(G, \pi) = \text{val}(G, \sigma)$ **by** *blast+*
ultimately
show $\exists \sigma. \exists p \in P. p \in G \wedge \langle \sigma, p \rangle \in \tau \wedge \text{val}(G, \sigma) = \text{val}(G, \pi)$ **by** *auto*
qed

lemma *refl_forces_eq*: $p \in P \implies \text{forces_eq}(p, x, x)$
using *def_forces_eq* **by** *simp*

lemma *forces_memI*: $\langle \sigma, r \rangle \in \tau \implies p \in P \implies r \in P \implies p \preceq r \implies \text{forces_mem}(p, \sigma, \tau)$
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, τ, ϑ)

and

IH: $\bigwedge q \sigma. q \in P \implies q \in G \implies \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta) \implies$

$(\text{forces_mem}(q, \sigma, \tau) \longrightarrow \text{val}(G, \sigma) \in \text{val}(G, \tau)) \wedge$

$(\text{forces_mem}(q, \sigma, \vartheta) \longrightarrow \text{val}(G, \sigma) \in \text{val}(G, \vartheta))$

shows

$\text{val}(G, \tau) \subseteq \text{val}(G, \vartheta)$

proof

fix x

assume $x \in \text{val}(G, \tau)$
then
obtain $\sigma \ r$ **where** $\langle \sigma, r \rangle \in \tau \ r \in G \ \text{val}(G, \sigma) = x$ **by** *blast*
moreover from this and $\langle p \in G \rangle \langle M_generic(G) \rangle$
obtain q **where** $q \in G \ q \preceq p \ q \preceq r$ **by** *force*
moreover from this and $\langle p \in G \rangle \langle M_generic(G) \rangle$
have $q \in P \ p \in P$ **by** *blast+*
moreover from calculation and $\langle M_generic(G) \rangle$
have $\text{forces_mem}(q, \sigma, \tau)$
using forces_memI **by** *blast*
moreover
note $\langle \text{forces_eq}(p, \tau, \vartheta) \rangle$
ultimately
have $\text{forces_mem}(q, \sigma, \vartheta)$
using def_forces_eq **by** *blast*
with $\langle q \in P \rangle \langle q \in G \rangle \text{IH}[\text{of } q \ \sigma] \langle \langle \sigma, r \rangle \in \tau \rangle \langle \text{val}(G, \sigma) = x \rangle$
show $x \in \text{val}(G, \vartheta)$ **by** (*blast*)
qed

lemma *IV240a_eq_2nd_incl*:

assumes
 $M_generic(G) \ p \in G \ \text{forces_eq}(p, \tau, \vartheta)$
and
 $\text{IH}: \bigwedge q \ \sigma. \ q \in P \implies q \in G \implies \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta) \implies$
 $(\text{forces_mem}(q, \sigma, \tau) \longrightarrow \text{val}(G, \sigma) \in \text{val}(G, \tau)) \wedge$
 $(\text{forces_mem}(q, \sigma, \vartheta) \longrightarrow \text{val}(G, \sigma) \in \text{val}(G, \vartheta))$
shows
 $\text{val}(G, \vartheta) \subseteq \text{val}(G, \tau)$
proof
fix x
assume $x \in \text{val}(G, \vartheta)$
then
obtain $\sigma \ r$ **where** $\langle \sigma, r \rangle \in \vartheta \ r \in G \ \text{val}(G, \sigma) = x$ **by** *blast*
moreover from this and $\langle p \in G \rangle \langle M_generic(G) \rangle$
obtain q **where** $q \in G \ q \preceq p \ q \preceq r$ **by** *force*
moreover from this and $\langle p \in G \rangle \langle M_generic(G) \rangle$
have $q \in P \ p \in P$ **by** *blast+*
moreover from calculation and $\langle M_generic(G) \rangle$
have $\text{forces_mem}(q, \sigma, \vartheta)$
using forces_memI **by** *blast*
moreover
note $\langle \text{forces_eq}(p, \tau, \vartheta) \rangle$
ultimately
have $\text{forces_mem}(q, \sigma, \tau)$
using def_forces_eq **by** *blast*
with $\langle q \in P \rangle \langle q \in G \rangle \text{IH}[\text{of } q \ \sigma] \langle \langle \sigma, r \rangle \in \vartheta \rangle \langle \text{val}(G, \sigma) = x \rangle$
show $x \in \text{val}(G, \tau)$ **by** (*blast*)
qed

lemma *IV240a_eq*:

assumes

$M_generic(G) \ p \in G \ forces_eq(p, \tau, \vartheta)$

and

$IH: \bigwedge q \ \sigma. \ q \in P \implies q \in G \implies \sigma \in domain(\tau) \cup domain(\vartheta) \implies$

$(forces_mem(q, \sigma, \tau) \longrightarrow val(G, \sigma) \in val(G, \tau)) \wedge$

$(forces_mem(q, \sigma, \vartheta) \longrightarrow val(G, \sigma) \in val(G, \vartheta))$

shows

$val(G, \tau) = val(G, \vartheta)$

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

$\bigwedge \tau \ \vartheta \ p. \ p \in P \implies \llbracket \bigwedge q \ \sigma. \ [q \in P ; \sigma \in domain(\vartheta)] \implies Q(0, \tau, \sigma, q) \rrbracket \implies Q(1, \tau, \vartheta, p)$

$\bigwedge \tau \ \vartheta \ p. \ p \in P \implies \llbracket \bigwedge q \ \sigma. \ [q \in P ; \sigma \in domain(\tau) \cup domain(\vartheta)] \implies Q(1, \sigma, \tau, q) \rrbracket$

$\wedge Q(1, \sigma, \vartheta, q) \implies Q(0, \tau, \vartheta, p)$

$ft \in 2 \ p \in P$

shows

$Q(ft, \tau, \vartheta, p)$

proof -

{

fix *ft p τ ϑ*

have *Transset(eclose({τ, ϑ})) (is Transset(?e))*

using *Transset_eclose* **by** *simp*

have $\tau \in ?e \ \vartheta \in ?e$

using *arg_into_eclose* **by** *simp_all*

moreover

assume $ft \in 2 \ p \in P$

ultimately

have $\langle ft, \tau, \vartheta, p \rangle \in 2 \times ?e \times ?e \times P$ (**is** $?a \in 2 \times ?e \times ?e \times P$) **by** *simp*

then

have $Q(fttype(?a), name1(?a), name2(?a), cond_of(?a))$

using *core_induction_aux[of ?e P Q ?a, OF ‹Transset(?e)›* *assms(1, 2)*

‹?a ∈ ‹›

by (*clarify*) (*blast*)

then have $Q(ft, \tau, \vartheta, p)$ **by** (*simp add: components_simp*)

}

then show *?thesis* **using** *assms* **by** *simp*

qed

lemma *forces_induction_with_conds*:

assumes

$\bigwedge \tau \ \vartheta \ p. \ p \in P \implies \llbracket \bigwedge q \ \sigma. \ [q \in P ; \sigma \in domain(\vartheta)] \implies Q(q, \tau, \sigma) \rrbracket \implies R(p, \tau, \vartheta)$

$\bigwedge \tau \ \vartheta \ p. \ p \in P \implies \llbracket \bigwedge q \ \sigma. \ [q \in P ; \sigma \in domain(\tau) \cup domain(\vartheta)] \implies R(q, \sigma, \tau) \wedge$

```

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

```

$(\tau \in M \longrightarrow \vartheta \in M \longrightarrow (\forall p \in G. \text{forces_eq}(p, \tau, \vartheta) \longrightarrow \text{val}(G, \tau) = \text{val}(G, \vartheta))) \wedge$
 $(\tau \in M \longrightarrow \vartheta \in M \longrightarrow (\forall p \in G. \text{forces_mem}(p, \tau, \vartheta) \longrightarrow \text{val}(G, \tau) \in \text{val}(G, \vartheta)))$
 $(\text{is } ?Q(\tau, \vartheta) \wedge ?R(\tau, \vartheta))$

proof (intro forces_induction[of ?Q ?R] impI)
fix $\tau \vartheta$
assume $\tau \in M \vartheta \in M \sigma \in \text{domain}(\vartheta) \Longrightarrow ?Q(\tau, \sigma)$ **for** σ
moreover from this
have $\sigma \in \text{domain}(\vartheta) \Longrightarrow \text{forces_eq}(q, \tau, \sigma) \Longrightarrow \text{val}(G, \tau) = \text{val}(G, \sigma)$
if $q \in P \ q \in G$ **for** $q \ \sigma$
using that domain_closed[of ϑ] transitivity **by** auto
moreover
note *assms*
ultimately
show $\forall p \in G. \text{forces_mem}(p, \tau, \vartheta) \longrightarrow \text{val}(G, \tau) \in \text{val}(G, \vartheta)$
using IV240a_mem domain_closed transitivity **by** (simp)

next
fix $\tau \vartheta$
assume $\tau \in M \vartheta \in M \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta) \Longrightarrow ?R(\sigma, \tau) \wedge ?R(\sigma, \vartheta)$ **for** σ
moreover from this
have $IH^1: \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta) \Longrightarrow q \in G \Longrightarrow$
 $(\text{forces_mem}(q, \sigma, \tau) \longrightarrow \text{val}(G, \sigma) \in \text{val}(G, \tau)) \wedge$
 $(\text{forces_mem}(q, \sigma, \vartheta) \longrightarrow \text{val}(G, \sigma) \in \text{val}(G, \vartheta))$ **for** $q \ \sigma$
by (auto intro: transitivity[OF _ domain_closed[simplified]])
ultimately
show $\forall p \in G. \text{forces_eq}(p, \tau, \vartheta) \longrightarrow \text{val}(G, \tau) = \text{val}(G, \vartheta)$
using IV240a_eq[OF *assms*(1) _ _ IH^1] **by** (simp)

qed

18.12 Lemma IV.2.40(b)

lemma IV240b_mem:

assumes
 $M_generic(G) \ \text{val}(G, \pi) \in \text{val}(G, \tau) \ \pi \in M \ \tau \in M$
and
 $IH: \bigwedge \sigma. \sigma \in \text{domain}(\tau) \Longrightarrow \text{val}(G, \pi) = \text{val}(G, \sigma) \Longrightarrow$
 $\exists p \in G. \text{forces_eq}(p, \pi, \sigma)$

shows
 $\exists p \in G. \text{forces_mem}(p, \pi, \tau)$

proof -
from $\langle \text{val}(G, \pi) \in \text{val}(G, \tau) \rangle$
obtain $\sigma \ r$ **where** $r \in G \ \langle \sigma, r \rangle \in \tau \ \text{val}(G, \pi) = \text{val}(G, \sigma)$ **by** auto
moreover from this and IH
obtain p' **where** $p' \in G \ \text{forces_eq}(p', \pi, \sigma)$ **by** blast
moreover
note $\langle M_generic(G) \rangle$
ultimately
obtain p **where** $p \preceq r \ p \in G \ \text{forces_eq}(p, \pi, \sigma)$
using $M_generic_compatD \ \text{strengthening_eq}$ [of p'] **by** blast
moreover

note $\langle M_generic(G) \rangle$
moreover from *calculation*
have $forces_eq(q,\pi,\sigma)$ **if** $q \in P$ $q \preceq p$ **for** q
using *that strengthening_eq* **by** *blast*
moreover
note $\langle \langle \sigma, r \rangle \in \tau \rangle \langle r \in G \rangle$
ultimately
have $r \in P \wedge \langle \sigma, r \rangle \in \tau \wedge q \preceq r \wedge forces_eq(q,\pi,\sigma)$ **if** $q \in P$ $q \preceq p$ **for** q
using *that leq_transD[of _ p r]* **by** *blast*
then
have $dense_below(\{q \in P. \exists s r. r \in P \wedge \langle s, r \rangle \in \tau \wedge q \preceq r \wedge forces_eq(q,\pi,s)\}, p)$
using *leq_refl* **by** *blast*
moreover
note $\langle M_generic(G) \rangle \langle p \in G \rangle$
moreover from *calculation*
have $forces_mem(p,\pi,\tau)$
using *forces_mem_iff_dense_below* **by** *blast*
ultimately
show *?thesis* **by** *blast*
qed
end

lemma *Collect_forces_eq_in_M*:
assumes $\tau \in M$ $\vartheta \in M$
shows $\{p \in P. forces_eq(p,\tau,\vartheta)\} \in M$
using *assms Collect_in_M_4p*[of *forces_eq_fm*(1,2,0,3,4) P *leq* τ ϑ
 $\lambda A x p l t1 t2. is_forces_eq(x,t1,t2)$
 $\lambda x p l t1 t2. forces_eq(x,t1,t2)$ P]
 $arity_forces_eq_fm$ P *in* M *leq* *in* M *sats* *forces_eq_fm* *forces_eq_abs*
forces_eq_fm_type
by (*simp* *add*: *nat_union_abs1* *Un_commute*)

lemma *IV240b_eq_Collects*:
assumes $\tau \in M$ $\vartheta \in M$
shows $\{p \in P. \exists \sigma \in domain(\tau) \cup domain(\vartheta). forces_mem(p,\sigma,\tau) \wedge forces_nmem(p,\sigma,\vartheta)\} \in M$
and
 $\{p \in P. \exists \sigma \in domain(\tau) \cup domain(\vartheta). forces_nmem(p,\sigma,\tau) \wedge forces_mem(p,\sigma,\vartheta)\} \in M$
proof -
let $?rel_pred = \lambda M x a1 a2 a3 a4.$
 $\exists \sigma[M]. \exists u[M]. \exists da3[M]. \exists da4[M]. is_domain(M, a3, da3) \wedge is_domain(M, a4, da4)$
 \wedge
 $union(M, da3, da4, u) \wedge \sigma \in u \wedge is_forces_mem'(M, a1, a2, x, \sigma, a3) \wedge$
 $is_forces_nmem'(M, a1, a2, x, \sigma, a4)$
let $? \varphi = Exists(Exists(Exists(Exists(And(domain_fm(7,1), And(domain_fm(8,0),$
 $And(union_fm(1,0,2), And(Member(3,2), And(forces_mem_fm(5,6,4,3,7),$
 $forces_nmem_fm(5,6,4,3,8))))))))))$
have $1: \sigma \in M$ **if** $\langle \sigma, y \rangle \in \delta$ $\delta \in M$ **for** σ δ y
using *that pair_in_M_iff_transitivity*[of $\langle \sigma, y \rangle$ δ] **by** *simp*

```

have abs1: ?rel_pred(##M,p,P,leq,τ,∅) ↔
  (∃σ∈domain(τ) ∪ domain(∅). forces_mem'(P,leq,p,σ,τ) ∧ forces_nmem'(P,leq,p,σ,∅))

  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 _ _ ∅])
  have abs2: ?rel_pred(##M,p,P,leq,∅,τ) ↔ (∃σ∈domain(τ) ∪ domain(∅).
    forces_nmem'(P,leq,p,σ,τ) ∧ forces_mem'(P,leq,p,σ,∅)) 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 _ _ ∅])
  have fsats1: (M,[p,P,leq,τ,∅] ⊨ ?φ) ↔ ?rel_pred(##M,p,P,leq,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 fsats2: (M,[p,P,leq,∅,τ] ⊨ ?φ) ↔ ?rel_pred(##M,p,P,leq,∅,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(∅). forces_mem(p, σ, τ) ∧ forces_nmem(p,
    σ, ∅)} ∈ M
  and {p ∈ P . ∃σ∈domain(τ) ∪ domain(∅). forces_nmem(p, σ, τ) ∧ forces_mem(p,
    σ, ∅)} ∈ 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 ∅ 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

$M_generic(G) \text{ val}(G, \tau) = \text{val}(G, \vartheta) \ \tau \in M \ \vartheta \in M$

and

$IH: \bigwedge \sigma. \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta) \implies$

$(\text{val}(G, \sigma) \in \text{val}(G, \tau) \longrightarrow (\exists q \in G. \text{forces_mem}(q, \sigma, \tau))) \wedge$

$(\text{val}(G, \sigma) \in \text{val}(G, \vartheta) \longrightarrow (\exists q \in G. \text{forces_mem}(q, \sigma, \vartheta)))$

shows

$\exists p \in G. \text{forces_eq}(p, \tau, \vartheta)$

proof -

let $?D1 = \{p \in P. \text{forces_eq}(p, \tau, \vartheta)\}$

let $?D2 = \{p \in P. \exists \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta). \text{forces_mem}(p, \sigma, \tau) \wedge \text{forces_nmem}(p, \sigma, \vartheta)\}$

let $?D3 = \{p \in P. \exists \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta). \text{forces_nmem}(p, \sigma, \tau) \wedge \text{forces_mem}(p, \sigma, \vartheta)\}$

let $?D = ?D1 \cup ?D2 \cup ?D3$

note *assms*

moreover from *this*

have $\text{domain}(\tau) \cup \text{domain}(\vartheta) \in M$ (**is** $?B \in M$) **using** *domain_closed Un_closed*

by *auto*

moreover from *calculation*

have $?D2 \in M$ **and** $?D3 \in M$ **using** *IV240b_eq_Collects* **by** *simp_all*

ultimately

have $?D \in M$ **using** *Collect_forces_eq_in_M Un_closed* **by** *auto*

moreover

have *dense*($?D$)

proof

fix p

assume $p \in P$

have $\exists d \in P. (\text{forces_eq}(d, \tau, \vartheta) \vee$

$(\exists \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta). \text{forces_mem}(d, \sigma, \tau) \wedge \text{forces_nmem}(d,$

$\sigma, \vartheta)) \vee$

$(\exists \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta). \text{forces_nmem}(d, \sigma, \tau) \wedge \text{forces_mem}(d,$

$\sigma, \vartheta))) \wedge$

$d \preceq p$

proof (*cases* $\text{forces_eq}(p, \tau, \vartheta)$)

case *True*

with $\langle p \in P \rangle$

show *?thesis* **using** *leq_refl* **by** *blast*

next

case *False*

moreover note $\langle p \in P \rangle$

moreover from *calculation*

obtain $\sigma \ q$ **where** $\sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta) \ q \in P \ q \preceq p$

$(\text{forces_mem}(q, \sigma, \tau) \wedge \neg \text{forces_mem}(q, \sigma, \vartheta)) \vee$

$(\neg \text{forces_mem}(q, \sigma, \tau) \wedge \text{forces_mem}(q, \sigma, \vartheta))$

using *def_forces_eq* **by** *blast*

moreover from *this*

obtain r **where** $r \preceq q \ r \in P$

$(\text{forces_mem}(r, \sigma, \tau) \wedge \text{forces_nmem}(r, \sigma, \vartheta)) \vee$

```

      (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  $\sigma$  where  $\sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta)$  forces_mem( $p, \sigma, \vartheta$ ) forces_nmem( $p, \sigma, \tau$ )

  by blast
  moreover from this and  $\langle p \in G \rangle$  and assms
  have  $\text{val}(G, \sigma) \in \text{val}(G, \vartheta)$ 
    using IV240a[of G σ ϑ] transitivity[OF _ domain_closed[simplified]] by blast
  moreover note IH  $\langle \text{val}(G, \tau) = \_ \rangle$ 
  ultimately
  obtain  $q$  where  $q \in G$  forces_mem( $q, \sigma, \tau$ ) by auto
  moreover from this and  $\langle p \in G \rangle$   $\langle M\_generic(G) \rangle$ 
  obtain  $r$  where  $r \in P$   $r \preceq p$   $r \preceq q$ 
    by blast
  moreover
  note  $\langle M\_generic(G) \rangle$ 
  ultimately
  have forces_mem( $r, \sigma, \tau$ )
    using strengthening_mem by blast
  with  $\langle r \preceq p \rangle$   $\langle \text{forces_nmem}(p, \sigma, \tau) \rangle$   $\langle r \in P \rangle$ 
  have False
    unfolding forces_nmem_def by blast
  then
  show ?thesis by simp
qed
qed

```

lemma *IV240b*:

```

assumes
   $M\_generic(G)$ 
shows
   $(\tau \in M \longrightarrow \vartheta \in M \longrightarrow \text{val}(G, \tau) = \text{val}(G, \vartheta) \longrightarrow (\exists p \in G. \text{forces\_eq}(p, \tau, \vartheta))) \wedge$ 
   $(\tau \in M \longrightarrow \vartheta \in M \longrightarrow \text{val}(G, \tau) \in \text{val}(G, \vartheta) \longrightarrow (\exists p \in G. \text{forces\_mem}(p, \tau, \vartheta)))$ 
  (is  $?Q(\tau, \vartheta) \wedge ?R(\tau, \vartheta)$ )
proof (intro forces_induction)
  fix  $\tau \vartheta p$ 
  assume  $\sigma \in \text{domain}(\vartheta) \implies ?Q(\tau, \sigma)$  for  $\sigma$ 
  with assms
  show  $?R(\tau, \vartheta)$ 
    using IV240b_mem domain_closed transitivity by (simp)
next
  fix  $\tau \vartheta p$ 
  assume  $\sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta) \implies ?R(\sigma, \tau) \wedge ?R(\sigma, \vartheta)$  for  $\sigma$ 

```

moreover from this
have $IH': \tau \in M \implies \vartheta \in M \implies \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta) \implies$
 $(\text{val}(G, \sigma) \in \text{val}(G, \tau) \longrightarrow (\exists q \in G. \text{forces_mem}(q, \sigma, \tau))) \wedge$
 $(\text{val}(G, \sigma) \in \text{val}(G, \vartheta) \longrightarrow (\exists q \in G. \text{forces_mem}(q, \sigma, \vartheta)))$ **for** σ
by (*blast intro:left_in_M*)
ultimately
show $?Q(\tau, \vartheta)$
using *IV240b_eq[OF assms(1)]* **by** (*auto*)
qed

lemma map_val_in_MG:
assumes
 $\text{env} \in \text{list}(M)$
shows
 $\text{map}(\text{val}(G), \text{env}) \in \text{list}(M[G])$
unfolding *GenExt_def* **using** *assms map_type2* **by** *simp*

lemma truth_lemma_mem:
assumes
 $\text{env} \in \text{list}(M)$ $M_generic(G)$
 $n \in \text{nat}$ $m \in \text{nat}$ $n < \text{length}(\text{env})$ $m < \text{length}(\text{env})$
shows
 $(\exists p \in G. p \Vdash \text{Member}(n, m) \text{ env}) \longleftrightarrow M[G], \text{map}(\text{val}(G), \text{env}) \models \text{Member}(n, m)$
using *assms IV240a[OF assms(2), of_nth(n, env) nth(m, env)]*
 $IV240b[OF assms(2), of_nth(n, env) nth(m, env)]$
 $P_in_M \text{ leq_in_M one_in_M}$
 $\text{Forces_Member}[of_ _ \text{nth}(n, \text{env}) \text{nth}(m, \text{env}) \text{env} n m]$ *map_val_in_MG*
by (*auto*)

lemma truth_lemma_eq:
assumes
 $\text{env} \in \text{list}(M)$ $M_generic(G)$
 $n \in \text{nat}$ $m \in \text{nat}$ $n < \text{length}(\text{env})$ $m < \text{length}(\text{env})$
shows
 $(\exists p \in G. p \Vdash \text{Equal}(n, m) \text{ env}) \longleftrightarrow M[G], \text{map}(\text{val}(G), \text{env}) \models \text{Equal}(n, m)$
using *assms IV240a(1)[OF assms(2), of_nth(n, env) nth(m, env)]*
 $IV240b(1)[OF assms(2), of_nth(n, env) nth(m, env)]$
 $P_in_M \text{ leq_in_M one_in_M}$
 $\text{Forces_Equal}[of_ _ \text{nth}(n, \text{env}) \text{nth}(m, \text{env}) \text{env} n m]$ *map_val_in_MG*
by (*auto*)

lemma arities_at_aux:
assumes
 $n \in \text{nat}$ $m \in \text{nat}$ $\text{env} \in \text{list}(M)$ $\text{succ}(n) \cup \text{succ}(m) \leq \text{length}(\text{env})$
shows
 $n < \text{length}(\text{env})$ $m < \text{length}(\text{env})$
using *assms succ_leE[OF Un_leD1, of n succ(m) length(env)]*
 $\text{succ_leE}[OF Un_leD2, of succ(n) m length(env)]$ **by** *auto*

18.13 The Strengthening Lemma

lemma *strengthening_lemma*:

assumes

$p \in P \ \varphi \in \text{formula} \ r \in P \ r \leq p$

shows

$\bigwedge \text{env. env} \in \text{list}(M) \implies \text{arity}(\varphi) \leq \text{length}(\text{env}) \implies p \Vdash \varphi \ \text{env} \implies r \Vdash \varphi \ \text{env}$

using *assms(2)*

proof (*induct*)

case (*Member n m*)

then

have $n < \text{length}(\text{env}) \ m < \text{length}(\text{env})$

using *arities_at_aux* **by** *simp_all*

moreover

assume $\text{env} \in \text{list}(M)$

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 $n < \text{length}(\text{env}) \ m < \text{length}(\text{env})$

using *arities_at_aux* **by** *simp_all*

moreover

assume $\text{env} \in \text{list}(M)$

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 $\varphi \ \psi$*)

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 $p \Vdash \varphi \ ([x] @ \text{env})$ **if** $x \in M$ **for** x

using *that Forces_Forall* **by** *simp*

with *Forall*

have $r \Vdash \varphi \ ([x] @ \text{env})$ **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 $\varphi \in \text{formula}$ $\psi \in \text{formula}$ $\text{arity}(\text{Nand}(\varphi, \psi)) \leq \text{length}(\text{env})$ $\text{env} \in \text{list}(A)$
shows $\text{arity}(\varphi) \leq \text{length}(\text{env})$ $\text{arity}(\psi) \leq \text{length}(\text{env})$
using *assms*
by (*rule_tac Un_leD1*, *rule_tac [5] Un_leD2*, *auto*)

lemma *dense_below_imp_forces*:

assumes
 $p \in P$ $\varphi \in \text{formula}$
shows
 $\bigwedge \text{env. env} \in \text{list}(M) \implies \text{arity}(\varphi) \leq \text{length}(\text{env}) \implies$
 $\text{dense_below}(\{q \in P. (q \Vdash \varphi \text{ env})\}, p) \implies (p \Vdash \varphi \text{ env})$
using *assms(2)*
proof (*induct*)
case (*Member n m*)
then
have $n < \text{length}(\text{env})$ $m < \text{length}(\text{env})$
using *arities_at_aux* by *simp_all*
moreover
assume $\text{env} \in \text{list}(M)$
moreover
note *assms Member*
ultimately
show *?case*
using *Forces_Member*[of $_ nth(n, \text{env}) nth(m, \text{env}) \text{ env } n m$]
density_mem[of $p nth(n, \text{env}) nth(m, \text{env})$] by *simp*
next
case (*Equal n m*)
then
have $n < \text{length}(\text{env})$ $m < \text{length}(\text{env})$
using *arities_at_aux* by *simp_all*
moreover
assume $\text{env} \in \text{list}(M)$
moreover
note *assms Equal*
ultimately
show *?case*
using *Forces_Equal*[of $_ nth(n, \text{env}) nth(m, \text{env}) \text{ env } n m$]
density_eq[of $p nth(n, \text{env}) nth(m, \text{env})$] by *simp*
next
case (*Nand $\varphi \psi$*)
{
fix q
assume $q \in M$ $q \in P$ $q \preceq p$ $q \Vdash \varphi \text{ env}$

```

moreover
note Nand
moreover from calculation
obtain d where  $d \in P$   $d \Vdash Nand(\varphi, \psi)$  env  $d \preceq q$ 
  using dense_below1 by auto
moreover from calculation
have  $\neg(d \Vdash \psi)$  env if  $d \Vdash \varphi$  env
  using that Forces_Nand leq_refl transitivity[OF _ P_in_M, of d] by auto
moreover
note arity_Nand_le[of  $\varphi$   $\psi$ ]
moreover from calculation
have  $d \Vdash \varphi$  env
  using strengthening_lemma[of  $q$   $\varphi$   $d$  env] Un_leD1 by auto
ultimately
have  $\neg(q \Vdash \psi)$  env
  using strengthening_lemma[of  $q$   $\psi$   $d$  env] by auto
}
with  $\langle p \in P \rangle$ 
show ?case
  using Forces_Nand[symmetric, OF _ Nand(5,1,3)] by blast
next
case (Forall  $\varphi$ )
have dense_below( $\{q \in P. q \Vdash \varphi ([a]@env)\}, p)$  if  $a \in M$  for a
proof
  fix r
  assume  $r \in P$   $r \preceq p$ 
  with  $\langle dense\_below(\_, p) \rangle$ 
  obtain q where  $q \in P$   $q \preceq r$   $q \Vdash Forall(\varphi)$  env
  by blast
moreover
note Forall  $\langle a \in M \rangle$ 
moreover from calculation
have  $q \Vdash \varphi ([a]@env)$ 
  using Forces_Forall by simp
ultimately
show  $\exists d \in \{q \in P. q \Vdash \varphi ([a]@env)\}. d \in P \wedge d \preceq r$ 
  by auto
qed
moreover
note Forall(2)[of Cons( $\_, env$ )] Forall(1,3-5)
ultimately
have  $p \Vdash \varphi ([a]@env)$  if  $a \in M$  for a
  using that pred_le2 by simp
with assms Forall
show ?case using Forces_Forall by simp
qed

lemma density_lemma:
assumes

```

```

    p ∈ P φ ∈ formula env ∈ list(M) arity(φ) ≤ length(env)
  shows
    p ⊨ φ env ↔ dense_below({q ∈ P. (q ⊨ φ env)}, p)
proof
  assume dense_below({q ∈ P. (q ⊨ φ env)}, p)
  with assms
  show (p ⊨ φ env)
    using dense_below_imp_forces by simp
next
  assume p ⊨ φ env
  with assms
  show dense_below({q ∈ P. q ⊨ φ env}, p)
    using strengthening_lemma leq_refl by auto
qed

```

18.15 The Truth Lemma

lemma *Forces_And*:

```

assumes
  p ∈ P env ∈ list(M) φ ∈ formula ψ ∈ formula
  arity(φ) ≤ length(env) arity(ψ) ≤ length(env)
shows
  p ⊨ And(φ, ψ) env ↔ (p ⊨ φ env) ∧ (p ⊨ ψ env)
proof
  assume p ⊨ And(φ, ψ) env
  with assms
  have dense_below({r ∈ P . (r ⊨ φ env) ∧ (r ⊨ ψ env)}, p)
    using Forces_And_iff_dense_below by simp
  then
  have dense_below({r ∈ P . (r ⊨ φ env)}, p) dense_below({r ∈ P . (r ⊨ ψ env)},
  p)
    by blast+
  with assms
  show (p ⊨ φ env) ∧ (p ⊨ ψ env)
    using density_lemma[symmetric] by simp
next
  assume (p ⊨ φ env) ∧ (p ⊨ ψ env)
  have dense_below({r ∈ P . (r ⊨ φ env) ∧ (r ⊨ ψ env)}, p)
proof (intro dense_belowI beqI conjI, assumption)
  fix q
  assume q ∈ P q ≤ p
  with assms ⟨(p ⊨ φ env) ∧ (p ⊨ ψ env)⟩
  show q ∈ {r ∈ P . (r ⊨ φ env) ∧ (r ⊨ ψ env)} q ≤ q
    using strengthening_lemma leq_refl by auto
qed
  with assms
  show p ⊨ And(φ, ψ) env
    using Forces_And_iff_dense_below by simp
qed

```

lemma *Forces_Nand_alt*:

assumes
 $p \in P$ $env \in list(M)$ $\varphi \in formula$ $\psi \in formula$
 $arity(\varphi) \leq length(env)$ $arity(\psi) \leq length(env)$

shows
 $(p \Vdash Nand(\varphi, \psi) env) \longleftrightarrow (p \Vdash Neg(And(\varphi, \psi)) env)$

using *assms Forces_Nand Forces_And Forces_Neg* **by** *auto*

lemma *truth_lemma_Neg*:

assumes
 $\varphi \in formula$ $M_generic(G)$ $env \in list(M)$ $arity(\varphi) \leq length(env)$ **and**
 $IH: (\exists p \in G. p \Vdash \varphi env) \longleftrightarrow M[G], map(val(G), env) \models \varphi$

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** $\langle M_generic(G) \rangle \langle p \in G \rangle$

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 ar-*
ity_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 ar-*
ity_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_refl 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( $\varphi$ ) $\leq$ length(env) arity( $\psi$ )  $\leq$  length(env) M_generic(G)
  and
  IH: ( $\exists p \in G. p \Vdash \varphi$  env)  $\longleftrightarrow$  M[G], map(val(G),env)  $\models$   $\varphi$ 
      ( $\exists p \in G. p \Vdash \psi$  env)  $\longleftrightarrow$  M[G], map(val(G),env)  $\models$   $\psi$ 
shows
  ( $\exists p \in G. (p \Vdash \text{And}(\varphi, \psi)$  env))  $\longleftrightarrow$  M[G], map(val(G),env)  $\models$   $\text{And}(\varphi, \psi)$ 
using assms map_val_in_MG Forces_And[OF M_genericD assms(1-5)]
proof (intro iffI, elim bexE)
  fix p
  assume p  $\in$  G p  $\Vdash$   $\text{And}(\varphi, \psi)$  env
  with assms
  show M[G], map(val(G),env)  $\models$   $\text{And}(\varphi, \psi)$ 
    using Forces_And[OF M_genericD, of _ _ _  $\varphi$   $\psi$ ] map_val_in_MG by auto
next
assume M[G], map(val(G),env)  $\models$   $\text{And}(\varphi, \psi)$ 
moreover
note assms
moreover from calculation
obtain q r where q  $\Vdash$   $\varphi$  env r  $\Vdash$   $\psi$  env q  $\in$  G r  $\in$  G
  using map_val_in_MG Forces_And[OF M_genericD assms(1-5)] by auto
moreover from calculation
obtain p where p  $\preceq$  q p  $\preceq$  r p  $\in$  G
  by blast
moreover from calculation
have (p  $\Vdash$   $\varphi$  env)  $\wedge$  (p  $\Vdash$   $\psi$  env)
  using strengthening_lemma by (blast)
ultimately
show  $\exists p \in G. (p \Vdash \text{And}(\varphi, \psi)$  env)
  using Forces_And[OF M_genericD assms(1-5)] by auto
qed

```

definition

```

ren_truth_lemma ::  $i \Rightarrow i$  where
ren_truth_lemma( $\varphi$ )  $\equiv$ 
  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( $\lambda p. \text{incr\_bv}(p)$ '5 , 6,  $\varphi$ ))))))))))

```

```

lemma ren_truth_lemma_type[TC] :
 $\varphi \in \text{formula} \Longrightarrow \text{ren\_truth\_lemma}(\varphi) \in \text{formula}$ 
unfolding ren_truth_lemma_def
by simp

```

```

lemma arity_ren_truth :
assumes  $\varphi \in \text{formula}$ 
shows  $\text{arity}(\text{ren\_truth\_lemma}(\varphi)) \leq 6 \cup \text{succ}(\text{arity}(\varphi))$ 
proof -
consider (lt)  $5 < \text{arity}(\varphi)$  | (ge)  $\neg 5 < \text{arity}(\varphi)$ 
  by auto

```

```

then
show ?thesis
proof cases
  case lt
  consider (a)  $5 < \text{arity}(\varphi) \# + 5$  | (b)  $\text{arity}(\varphi) \# + 5 \leq 5$ 
  using not_lt_iff_le ⟨ $\varphi \in \_$ ⟩ by force
  then
  show ?thesis
  proof cases
    case a
    with ⟨ $\varphi \in \_$ ⟩ lt
    have  $5 < \text{succ}(\text{arity}(\varphi))$   $5 < \text{arity}(\varphi) \# + 2$   $5 < \text{arity}(\varphi) \# + 3$   $5 < \text{arity}(\varphi) \# + 4$ 
    using succ_ltI by auto
    with ⟨ $\varphi \in \_$ ⟩
    have  $c:\text{arity}(\text{iterates}(\lambda p. \text{incr\_bv}(p) '5, 5, \varphi)) = 5 \# + \text{arity}(\varphi)$  (is  $\text{arity}(\varphi') =$ 
    —)
    using arity_incr_bv_lemma lt a
    by simp
    with ⟨ $\varphi \in \_$ ⟩
    have  $\text{arity}(\text{incr\_bv}(\varphi') '5) = 6 \# + \text{arity}(\varphi)$ 
    using arity_incr_bv_lemma[of  $\varphi'$  5] a by auto
    with ⟨ $\varphi \in \_$ ⟩
    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 ⟨ $\varphi \in \_$ ⟩ lt
  have  $5 < \text{succ}(\text{arity}(\varphi))$   $5 < \text{arity}(\varphi) \# + 2$   $5 < \text{arity}(\varphi) \# + 3$   $5 < \text{arity}(\varphi) \# + 4$ 
   $5 < \text{arity}(\varphi) \# + 5$ 
  using succ_ltI by auto
  with ⟨ $\varphi \in \_$ ⟩
  have  $\text{arity}(\text{iterates}(\lambda p. \text{incr\_bv}(p) '5, 6, \varphi)) = 6 \# + \text{arity}(\varphi)$  (is  $\text{arity}(\varphi') =$ 
  —)
  using arity_incr_bv_lemma lt
  by simp
  with ⟨ $\varphi \in \_$ ⟩
  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 ⟨ $\varphi \in \_$ ⟩
have  $\text{arity}(\varphi) \leq 5$   $\text{pred}^5(\text{arity}(\varphi)) \leq 5$ 
using not_lt_iff_le le_trans[OF le_pred]
by auto

```

```

with ⟨ $\varphi \in \_$ ⟩
have  $\text{arity}(\text{iterates}(\lambda p. \text{incr\_bv}(p) \text{ } \langle 5, 6, \varphi \rangle)) = \text{arity}(\varphi) \text{ } \text{arity}(\varphi) \leq 6 \text{ } \text{pred}^{\wedge 5}(\text{arity}(\varphi))$ 
 $\leq 6$ 
using  $\text{arity\_incr\_bv\_lemma}$   $\text{ge}$   $\text{le\_trans}[OF \text{ } \langle \text{arity}(\varphi) \leq 5 \rangle]$   $\text{le\_trans}[OF$ 
 $\langle \text{pred}^{\wedge 5}(\text{arity}(\varphi)) \leq 5 \rangle]$ 
by  $\text{auto}$ 
with ⟨ $\text{arity}(\varphi) \leq 5$ ⟩ ⟨ $\varphi \in \_$ ⟩ ⟨ $\text{pred}^{\wedge 5}(\_) \leq 5$ ⟩
show  $?thesis$ 
unfolding  $\text{ren\_truth\_lemma\_def}$ 
using  $\text{pred\_Un\_distrib}$   $\text{nat\_union\_abs1}$   $\text{Un\_assoc}[\text{symmetric}]$   $\text{nat\_union\_abs2}$ 

by  $\text{simp}$ 
qed
qed

```

```

lemma  $\text{sats\_ren\_truth\_lemma}$ :
 $[q, b, d, a1, a2, a3] @ \text{env} \in \text{list}(M) \implies \varphi \in \text{formula} \implies$ 
 $(M, [q, b, d, a1, a2, a3] @ \text{env} \models \text{ren\_truth\_lemma}(\varphi)) \longleftrightarrow$ 
 $(M, [q, a1, a2, a3, b] @ \text{env} \models \varphi)$ 
unfolding  $\text{ren\_truth\_lemma\_def}$ 
by  $(\text{insert } \text{sats\_incr\_bv\_iff} [\text{of } \_ \_ M \_ [q, a1, a2, a3, b]], \text{simp})$ 

```

```

lemma  $\text{truth\_lemma}'$  :
assumes
 $\varphi \in \text{formula}$   $\text{env} \in \text{list}(M)$   $\text{arity}(\varphi) \leq \text{succ}(\text{length}(\text{env}))$ 
shows
 $\text{separation}(\#\#M, \lambda d. \exists b \in M. \forall q \in P. q \preceq d \longrightarrow \neg(q \Vdash \varphi ([b] @ \text{env})))$ 
proof -
let  $?rel\_pred = \lambda M \ x \ a1 \ a2 \ a3. \exists b \in M. \forall q \in M. q \in a1 \wedge \text{is\_leq}(\#\#M, a2, q, x) \longrightarrow$ 
 $\neg(M, [q, a1, a2, a3, b] @ \text{env} \models \text{forces}(\varphi))$ 
let  $?psi = \text{Exists}(\text{Forall}(\text{Implies}(\text{And}(\text{Member}(0, 3), \text{leq\_fm}(4, 0, 2)),$ 
 $\text{Neg}(\text{ren\_truth\_lemma}(\text{forces}(\varphi))))))$ 
have  $q \in M$  if  $q \in P$  for  $q$  using  $\text{that}$   $\text{transitivity}[OF \_ P\_in\_M]$  by  $\text{simp}$ 
then
have  $1: \forall q \in M. q \in P \wedge R(q) \longrightarrow Q(q) \implies (\forall q \in P. R(q) \longrightarrow Q(q))$  for  $R \ Q$ 
by  $\text{auto}$ 
then
have  $\llbracket b \in M; \forall q \in M. q \in P \wedge q \preceq d \longrightarrow \neg(q \Vdash \varphi ([b] @ \text{env})) \rrbracket \implies$ 
 $\exists c \in M. \forall q \in P. q \preceq d \longrightarrow \neg(q \Vdash \varphi ([c] @ \text{env}))$  for  $b \ d$ 
by  $(\text{rule } \text{bexI}, \text{simp\_all})$ 
then
have  $?rel\_pred(M, d, P, \text{leq}, \text{one}) \longleftrightarrow (\exists b \in M. \forall q \in P. q \preceq d \longrightarrow \neg(q \Vdash \varphi ([b] @ \text{env})))$ 
if  $d \in M$  for  $d$ 
using  $\text{that}$   $\text{leq\_abs}$   $\text{leq\_in\_M}$   $P\_in\_M$   $\text{one\_in\_M}$   $\text{assms}$ 
by  $\text{auto}$ 
moreover
have  $?psi \in \text{formula}$  using  $\text{assms}$  by  $\text{simp}$ 
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  $\exists p \in G. (p \Vdash \text{Forall}(\varphi) \text{ env})$ 
    with ⟨M_generic(G)⟩
    obtain p where  $p \in G \ p \in M \ p \in P \ p \Vdash \text{Forall}(\varphi) \text{ env}$ 
    using transitivity[OF _ P_in_M] by auto
    with ⟨env∈list(M)⟩ ⟨φ∈formula⟩
    have  $p \Vdash \varphi ([x]@env)$  if  $x \in M$  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  $M[G], \text{map}(\text{val}(G), \text{env}) \models \text{Forall}(\varphi)$ 
    using pred_le2 map_val_in_MG
    by (auto iff:GenExtD)
  next
  assume  $M[G], \text{map}(\text{val}(G), \text{env}) \models \text{Forall}(\varphi)$ 
  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  $D \equiv ?D1 \cup ?D2$ 
  have  $\text{arity}(\varphi) \leq \text{succ}(\text{length}(\text{env}))$ 
  using assms ⟨arity(Forall(φ)) ≤ length(env)⟩ ⟨φ∈formula⟩ ⟨env∈list(M)⟩
  pred_le2
  by simp
  then
  have  $\text{arity}(\text{Forall}(\varphi)) \leq \text{length}(\text{env})$ 

```

```

    using pred_le ⟨ $\varphi \in \text{formula}$ ⟩ ⟨ $\text{env} \in \text{list}(M)$ ⟩ by simp
  then
  have ?D1 ∈ M using Collect_forces ar $\varphi$  ⟨ $\varphi \in \text{formula}$ ⟩ ⟨ $\text{env} \in \text{list}(M)$ ⟩ by simp
  moreover
  have ?D2 ∈ M using ⟨ $\text{env} \in \text{list}(M)$ ⟩ ⟨ $\varphi \in \text{formula}$ ⟩ truth_lemma' separation_closed
ar $\varphi$ 
      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( $\varphi$ ) env)
      case True
      with ⟨p ∈ P⟩
      show ?thesis unfolding D_def using leq_refl by blast
    next
      case False
      with Forall ⟨p ∈ P⟩
      obtain b where b ∈ M ¬(p ⊢  $\varphi$  ([b]@env))
      using Forces_Forall by blast
      moreover from this ⟨p ∈ P⟩ Forall
      have ¬dense_below({q ∈ P. q ⊢  $\varphi$  ([b]@env)}, p)
      using density_lemma pred_le2 by auto
      moreover from this
      obtain d where d ≤ p ∀ q ∈ P. q ≤ d ⟶ ¬(q ⊢  $\varphi$  ([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( $\varphi$ ) 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

$p \in P \ \varphi \in \text{formula} \ \text{env} \in \text{list}(M) \ \text{arity}(\varphi) \leq \text{length}(\text{env})$

shows

$(p \Vdash \varphi \ \text{env}) \longleftrightarrow$

$(\forall G. M_generic(G) \wedge p \in G \longrightarrow M[G], \text{map}(\text{val}(G), \text{env}) \models \varphi)$

proof (intro iffI allI impI, elim conjE)

fix G

assume $(p \Vdash \varphi \ \text{env}) \ M_generic(G) \ p \in G$

with *assms*

show $M[G], \text{map}(\text{val}(G), \text{env}) \models \varphi$

using *truth_lemma* by blast

next

assume 1: $\forall G. (M_generic(G) \wedge p \in G) \longrightarrow M[G], \text{map}(\text{val}(G), \text{env}) \models \varphi$

{

fix r

assume 2: $r \in P \ r \leq p$

then

obtain G where $r \in G \ M_generic(G)$

using *generic_filter_existence* by auto

moreover from *calculation 2* ⟨ $p \in P$ ⟩

have $p \in G$

```

    unfolding M_generic_def using filter_leqD by simp
  moreover note 1
  ultimately
  have  $M[G], \text{map}(\text{val}(G), \text{env}) \models \varphi$ 
    by simp
  with assms  $\langle M\_generic(G) \rangle$ 
  obtain s where  $s \in G$  ( $s \Vdash \varphi \text{ env}$ )
    using truth_lemma by blast
  moreover from this and  $\langle M\_generic(G) \rangle \langle r \in G \rangle$ 
  obtain q where  $q \in G$   $q \preceq s$   $q \preceq r$ 
    by blast
  moreover from calculation  $\langle s \in G \rangle \langle M\_generic(G) \rangle$ 
  have  $s \in P$   $q \in P$ 
    unfolding M_generic_def filter_def by auto
  moreover
  note assms
  ultimately
  have  $\exists q \in P. q \preceq r \wedge (q \Vdash \varphi \text{ env})$ 
    using strengthening_lemma by blast
}
then
have dense_below( $\{q \in P. (q \Vdash \varphi \text{ env})\}, p$ )
  unfolding dense_below_def by blast
with assms
show ( $p \Vdash \varphi \text{ env}$ )
  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 :  $[p, t] \in \text{list}(A) \implies \text{env} \in \text{list}(A) \implies \text{nth}(1 \# + \text{length}(\text{env}), [p] @ \text{env} @ [t]) = t$ 
  by(auto simp add:nth_append)

```

```

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

```

```

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

```

by(*auto simp add:nth_append*)

definition

sep_var :: $i \Rightarrow i$ **where**
sep_var(n) $\equiv \{\langle 0,1 \rangle, \langle 1,3 \rangle, \langle 2,4 \rangle, \langle 3,5 \rangle, \langle 4,0 \rangle, \langle 5\#+n,6 \rangle, \langle 6\#+n,2 \rangle\}$

definition

sep_env :: $i \Rightarrow i$ **where**
sep_env(n) $\equiv \lambda i \in (5\#+n)-5 . i\#+2$

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

weak(n,m) $\equiv \{i\#+m . i \in n\}$

lemma *weakD* :

assumes $n \in \text{nat } k \in \text{nat } x \in \text{weak}(n,k)$
shows $\exists i \in n . x = i\#+k$
using *assms unfolding weak_def by blast*

lemma *weak_equal* :

assumes $n \in \text{nat } m \in \text{nat}$
shows $\text{weak}(n,m) = (m\#+n) - m$

proof -

have $\text{weak}(n,m) \subseteq (m\#+n) - m$

proof(*intro subsetI*)

fix x

assume $x \in \text{weak}(n,m)$

with *assms*

obtain i **where**

$i \in n \ x = i\#+m$

using *weakD by blast*

then

have $m \leq i\#+m \ i < n$

using *add_le_self2*[*of m i*] $\langle m \in \text{nat} \rangle \langle n \in \text{nat} \rangle$ *ltI*[*OF* $\langle i \in n \rangle$] **by** *simp_all*

then

have $\neg i\#+m < m$

using *not_lt_iff_le_in_n_in_nat*[*OF* $\langle n \in \text{nat} \rangle \langle i \in n \rangle$] $\langle m \in \text{nat} \rangle$ **by** *simp*

with $\langle x = i\#+m \rangle$

have $x \notin m$

using *ltI* $\langle m \in \text{nat} \rangle$ **by** *auto*

moreover

from *assms* $\langle x = i\#+m \rangle \langle i < n \rangle$

have $x < m\#+n$

using *add_lt_mono1*[*OF* $\langle i < n \rangle \langle n \in \text{nat} \rangle$] **by** *simp*

ultimately

show $x \in (m\#+n) - m$

using *ltD DiffI* **by** *simp*

qed

moreover

have $(m\#+n) - m \subseteq \text{weak}(n,m)$

```

proof (intro subsetI)
  fix x
  assume  $x \in (m\#\!+\!n)\text{-}m$ 
  then
  have  $x \in m\#\!+\!n$   $x \notin m$ 
    using DiffD1[of x n#\!+\!m m] DiffD2[of x n#\!+\!m m] by simp_all
  then
  have  $x < m\#\!+\!n$   $x \in \text{nat}$ 
    using ltI in_n_in_nat[OF add_type[of m n]] by simp_all
  then
  obtain i where
     $m\#\!+\!n = \text{succ}(x\#\!+\!i)$ 
    using less_iff_succ_add[OF  $\langle x \in \text{nat} \rangle$ , of  $m\#\!+\!n$ ] add_type by auto
  then
  have  $x\#\!+\!i < m\#\!+\!n$  using succ_le_iff by simp
  with  $\langle x \notin m \rangle$ 
  have  $\neg x < m$  using ltD by blast
  with  $\langle m \in \text{nat} \rangle$   $\langle x \in \text{nat} \rangle$ 
  have  $m \leq x$  using not_lt_iff_le by simp
  with  $\langle x < m\#\!+\!n \rangle$   $\langle n \in \text{nat} \rangle$ 
  have  $x\#\!-\!m < m\#\!+\!n\#\!-\!m$ 
    using diff_mono[OF  $\langle x \in \text{nat} \rangle$  _  $\langle m \in \text{nat} \rangle$ ] by simp
  have  $m\#\!+\!n\#\!-\!m = n$  using diff_cancel2  $\langle m \in \text{nat} \rangle$   $\langle n \in \text{nat} \rangle$  by simp
  with  $\langle x\#\!-\!m < m\#\!+\!n\#\!-\!m \rangle$   $\langle x \in \text{nat} \rangle$ 
  have  $x\#\!-\!m \in n$   $x = x\#\!-\!m\#\!+\!m$ 
    using ltD add_diff_inverse2[OF  $\langle m \leq x \rangle$ ] by simp_all
  then
  show  $x \in \text{weak}(n, m)$ 
    unfolding weak_def by auto
qed
ultimately
show ?thesis by auto
qed

```

```

lemma weak_zero:
  shows  $\text{weak}(0, n) = 0$ 
  unfolding weak_def by simp

```

```

lemma weakening_diff :
  assumes  $n \in \text{nat}$ 
  shows  $\text{weak}(n, 7) - \text{weak}(n, 5) \subseteq \{5\#\!+\!n, 6\#\!+\!n\}$ 
  unfolding weak_def using assms
proof(auto)
  {
    fix i
    assume  $i \in n$   $\text{succ}(\text{succ}(\text{natty}(i))) \neq n$   $\forall w \in n. \text{succ}(\text{succ}(\text{natty}(i))) \neq \text{natty}(w)$ 
    then
    have  $i < n$ 
      using ltI  $\langle n \in \text{nat} \rangle$  by simp
  }

```

```

from ⟨ $n \in \text{nat}$ ⟩ ⟨ $i \in n$ ⟩ ⟨ $\text{succ}(\text{succ}(\text{nativify}(i))) \neq n$ ⟩
have  $i \in \text{nat}$   $\text{succ}(\text{succ}(i)) \neq n$  using  $\text{in\_n\_in\_nat}$  by  $\text{simp\_all}$ 
from ⟨ $i < n$ ⟩
have  $\text{succ}(i) \leq n$  using  $\text{succ\_leI}$  by  $\text{simp}$ 
with ⟨ $n \in \text{nat}$ ⟩
consider (a)  $\text{succ}(i) = n$  | (b)  $\text{succ}(i) < n$ 
  using  $\text{leD}$  by  $\text{auto}$ 
then have  $\text{succ}(i) = n$ 
proof cases
  case a
  then show  $?thesis$  .
next
  case b
  then
  have  $\text{succ}(\text{succ}(i)) \leq n$  using  $\text{succ\_leI}$  by  $\text{simp}$ 
  with ⟨ $n \in \text{nat}$ ⟩
  consider (a)  $\text{succ}(\text{succ}(i)) = n$  | (b)  $\text{succ}(\text{succ}(i)) < n$ 
    using  $\text{leD}$  by  $\text{auto}$ 
  then have  $\text{succ}(i) = n$ 
  proof cases
    case a
    with ⟨ $\text{succ}(\text{succ}(i)) \neq n$ ⟩ show  $?thesis$  by  $\text{blast}$ 
  next
    case b
    then
    have  $\text{succ}(\text{succ}(i)) \in n$  using  $\text{ltD}$  by  $\text{simp}$ 
    with ⟨ $i \in \text{nat}$ ⟩
    have  $\text{succ}(\text{succ}(\text{nativify}(i))) \neq \text{nativify}(\text{succ}(\text{succ}(i)))$ 
      using ⟨ $\forall w \in n. \text{succ}(\text{succ}(\text{nativify}(i))) \neq \text{nativify}(w)$ ⟩ by  $\text{auto}$ 
    then
    have  $\text{False}$  using ⟨ $i \in \text{nat}$ ⟩ by  $\text{auto}$ 
    then show  $?thesis$  by  $\text{blast}$ 
  qed
  then show  $?thesis$  .
qed
with ⟨ $i \in \text{nat}$ ⟩ have  $\text{succ}(\text{nativify}(i)) = n$  by  $\text{simp}$ 
}
then
show  $n \in \text{nat} \implies$ 
   $\text{succ}(\text{succ}(\text{nativify}(y))) \neq n \implies$ 
   $\forall x \in n. \text{succ}(\text{succ}(\text{nativify}(y))) \neq \text{nativify}(x) \implies$ 
   $y \in n \implies \text{succ}(\text{nativify}(y)) = n$  for  $y$ 
  by  $\text{blast}$ 
qed

lemma  $\text{in\_add\_del}$  :
  assumes  $x \in j \# + n$   $n \in \text{nat}$   $j \in \text{nat}$ 
  shows  $x < j \vee x \in \text{weak}(n, j)$ 
proof (cases  $x < j$ )

```

```

case True
then show ?thesis ..
next
case False
have  $x \in \text{nat } j \# + n \in \text{nat}$ 
  using  $\text{in\_n\_in\_nat}[OF \_ \langle x \in j \# + n \rangle]$  assms by simp_all
then
have  $j \leq x \ x < j \# + n$ 
  using  $\text{not\_lt\_iff\_le } False \ \langle j \in \text{nat} \rangle \ \langle n \in \text{nat} \rangle$   $\text{ltI}[OF \ \langle x \in j \# + n \rangle]$  by auto
then
have  $x \# - j < (j \# + n) \# - j \ x = j \# + (x \# - j)$ 
  using  $\text{diff\_mono} \ \langle x \in \text{nat} \rangle \ \langle j \# + n \in \text{nat} \rangle \ \langle j \in \text{nat} \rangle \ \langle n \in \text{nat} \rangle$ 
   $\text{add\_diff\_inverse}[OF \ \langle j \leq x \rangle]$  by simp_all
then
have  $x \# - j < n \ x = (x \# - j) \# + j$ 
  using  $\text{diff\_add\_inverse} \ \langle n \in \text{nat} \rangle$  add\_commute by simp_all
then
have  $x \# - j \in n$  using ltD by simp
then
have  $x \in \text{weak}(n, j)$ 
  unfolding weak_def
  using  $\langle x = (x \# - j) \# + j \rangle$  RepFunI[ $OF \ \langle x \# - j \in n \rangle$ ] add\_commute by force
then show ?thesis ..
qed

```

lemma *sep_env_action*:

```

assumes
   $[t, p, u, P, \text{leq}, o, pi] \in \text{list}(M)$ 
   $env \in \text{list}(M)$ 
shows  $\forall i . i \in \text{weak}(\text{length}(env), 5) \longrightarrow$ 
   $\text{nth}(\text{sep\_env}(\text{length}(env))) \ i, [t, p, u, P, \text{leq}, o, pi] @ env = \text{nth}(i, [p, P, \text{leq}, o, t] @ env$ 
   $@ [pi, u])$ 
proof -
from assms
have  $A: 5 \# + \text{length}(env) \in \text{nat} \ [p, P, \text{leq}, o, t] \in \text{list}(M)$ 
  by simp_all
let  $?f = \text{sep\_env}(\text{length}(env))$ 
have  $EQ: \text{weak}(\text{length}(env), 5) = 5 \# + \text{length}(env) - 5$ 
  using weak_equal_length_type[ $OF \ \langle env \in \text{list}(M) \rangle$ ] by simp
let  $?tgt = [t, p, u, P, \text{leq}, o, pi] @ env$ 
let  $?src = [p, P, \text{leq}, o, t] @ env @ [pi, u]$ 
have  $\text{nth}(\ ?f \ i, [t, p, u, P, \text{leq}, o, pi] @ env) = \text{nth}(i, [p, P, \text{leq}, o, t] @ env @ [pi, u])$ 
  if  $i \in (5 \# + \text{length}(env) - 5)$  for  $i$ 
proof -
from that
have  $2: i \in 5 \# + \text{length}(env) \ i \notin 5 \ i \in \text{nat} \ i \# - 5 \in \text{nat} \ i \# + 2 \in \text{nat}$ 
  using  $\text{in\_n\_in\_nat}[OF \ \langle 5 \# + \text{length}(env) \in \text{nat} \rangle]$  by simp_all
then

```

```

have 3:  $\neg i < 5$  using ltD by force
then
have  $5 \leq i \leq 5$ 
  using not_lt_iff_le  $\langle i \in \text{nat} \rangle$  by simp_all
then have  $2 \leq i$  using le_trans[OF  $\langle 2 \leq 5 \rangle$ ] by simp
from A  $\langle i \in 5 \# + \text{length}(env) \rangle$ 
have  $i < 5 \# + \text{length}(env)$  using ltI by simp
with  $\langle i \in \text{nat} \rangle \langle 2 \leq i \rangle$  A
have C:  $i \# + 2 < 7 \# + \text{length}(env)$  by simp
with that
have B:  $?f^i = i \# + 2$  unfolding sep_env_def by simp
from 3 assms(1)  $\langle i \in \text{nat} \rangle$ 
have  $\neg i \# + 2 < 7$  using not_lt_iff_le add_le_mono by simp
from  $\langle i < 5 \# + \text{length}(env) \rangle$  3  $\langle i \in \text{nat} \rangle$ 
have  $i \# - 5 < 5 \# + \text{length}(env) \# - 5$ 
  using diff_mono[of  $i \ 5 \# + \text{length}(env) \ 5, OF \_ \_ \_ \langle i < 5 \# + \text{length}(env) \rangle$ ]
  not_lt_iff_le[THEN iffD1] by force
with assms(2)
have  $i \# - 5 < \text{length}(env)$  using diff_add_inverse length_type by simp
have  $\text{nth}(i, ?src) = \text{nth}(i \# - 5, env@[pi, u])$ 
  using nth_append[OF A(2)  $\langle i \in \text{nat} \rangle$ ] 3 by simp
also
have ... =  $\text{nth}(i \# - 5, env)$ 
  using nth_append[OF  $\langle env \in \text{list}(M) \rangle \langle i \# - 5 \in \text{nat} \rangle \langle i \# - 5 < \text{length}(env) \rangle$ ] by
simp
also
have ... =  $\text{nth}(i \# + 2, ?tgt)$ 
  using nth_append[OF assms(1)  $\langle i \# + 2 \in \text{nat} \rangle \langle \neg i \# + 2 < 7 \rangle$ ] by simp
ultimately
have  $\text{nth}(i, ?src) = \text{nth}(?f^i, ?tgt)$ 
  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  $n \in \text{nat}$ 
  shows  $\text{sep\_env}(n) : (5 \# + n) - 5 \rightarrow (7 \# + n) - 7$ 
proof -
  let  $?h = \text{sep\_env}(n)$ 
  from  $\langle n \in \text{nat} \rangle$ 
  have  $(5 \# + n) \# + 2 = 7 \# + n$   $7 \# + n \in \text{nat}$   $5 \# + n \in \text{nat}$  by simp_all
  have
    D:  $\text{sep\_env}(n) 'x \in (7 \# + n) - 7$  if  $x \in (5 \# + n) - 5$  for  $x$ 
  proof -
    from  $\langle x \in 5 \# + n - 5 \rangle$ 
    have  $?h 'x = x \# + 2$   $x < 5 \# + n$   $x \in \text{nat}$ 
    unfolding sep_env_def using ltI in_n_in_nat[OF  $\langle 5 \# + n \in \text{nat} \rangle$ ] by simp_all

```

```

then
have  $x\#+2 < 7\#+n$  by simp
then
have  $x\#+2 \in 7\#+n$  using ltD by simp
from  $\langle x \in 5\#+n-5 \rangle$ 
have  $x \notin 5$  by simp
then have  $\neg x < 5$  using ltD by blast
then have  $5 \leq x$  using not_lt_iff_le  $\langle x \in \text{nat} \rangle$  by simp
then have  $7 \leq x\#+2$  using add_le_mono  $\langle x \in \text{nat} \rangle$  by simp
then have  $\neg x\#+2 < 7$  using not_lt_iff_le  $\langle x \in \text{nat} \rangle$  by simp
then have  $x\#+2 \notin 7$  using ltI  $\langle x \in \text{nat} \rangle$  by force
with  $\langle x\#+2 \in 7\#+n \rangle$  show ?thesis using  $\langle ?h'x = x\#+2 \rangle$  DiffI by simp
qed
then show ?thesis unfolding sep_env_def using lam_type by simp
qed

```

```

lemma sep_var_fn_type :
assumes  $n \in \text{nat}$ 
shows  $\text{sep\_var}(n) : 7\#+n -||> 7\#+n$ 
unfolding sep_var_def
using consI ltD emptyI by force

```

```

lemma sep_var_domain :
assumes  $n \in \text{nat}$ 
shows  $\text{domain}(\text{sep\_var}(n)) = 7\#+n - \text{weak}(n,5)$ 
proof -
let  $?A = \text{weak}(n,5)$ 
have  $A : \text{domain}(\text{sep\_var}(n)) \subseteq (7\#+n)$ 
unfolding sep_var_def
by(auto simp add: le_natE)
have  $C : x = 5\#+n \vee x = 6\#+n \vee x \leq 4$  if  $x \in \text{domain}(\text{sep\_var}(n))$  for  $x$ 
using that unfolding sep_var_def by auto
have  $D : x < n\#+7$  if  $x \in 7\#+n$  for  $x$ 
using that  $\langle n \in \text{nat} \rangle$  ltI by simp
have  $\neg 5\#+n < 5\#+n$  using  $\langle n \in \text{nat} \rangle$  lt_irrefl[of False] by force
have  $\neg 6\#+n < 5\#+n$  using  $\langle n \in \text{nat} \rangle$  by force
have  $R : x < 5\#+n$  if  $x \in ?A$  for  $x$ 
proof -
from that
obtain  $i$  where
 $i < n$   $x = 5\#+i$ 
unfolding weak_def
using ltI  $\langle n \in \text{nat} \rangle$  RepFun_iff by force
with  $\langle n \in \text{nat} \rangle$ 
have  $5\#+i < 5\#+n$  using add_lt_mono2 by simp
with  $\langle x = 5\#+i \rangle$ 
show  $x < 5\#+n$  by simp
qed
then

```

```

have 1:  $x \notin ?A$  if  $\neg x < 5\# + n$  for  $x$  using that by blast
have 5:  $5\# + n \notin ?A$  6:  $6\# + n \notin ?A$ 
proof -
  show 5:  $5\# + n \notin ?A$  using 1  $\langle \neg 5\# + n < 5\# + n \rangle$  by blast
  with 1 show 6:  $6\# + n \notin ?A$  using  $\langle \neg 6\# + n < 5\# + n \rangle$  by blast
qed
then
have E:  $x \notin ?A$  if  $x \in \text{domain}(\text{sep\_var}(n))$  for  $x$ 
  unfolding weak_def
  using C that by force
then
have F:  $\text{domain}(\text{sep\_var}(n)) \subseteq 7\# + n - ?A$  using A by auto
from assms
have  $x < 7 \vee x \in \text{weak}(n, 7)$  if  $x \in 7\# + n$  for  $x$ 
  using in_add_del[OF  $\langle x \in 7\# + n \rangle$ ] by simp
moreover
{
  fix  $x$ 
  assume asm:  $x \in 7\# + n$   $x \notin ?A$   $x \in \text{weak}(n, 7)$ 
  then
  have  $x \in \text{domain}(\text{sep\_var}(n))$ 
  proof -
    from  $\langle n \in \text{nat} \rangle$ 
    have  $\text{weak}(n, 7) - \text{weak}(n, 5) \subseteq \{n\# + 5, n\# + 6\}$ 
      using weakening_diff by simp
    with  $\langle x \notin ?A \rangle$  asm
    have  $x \in \{n\# + 5, n\# + 6\}$  using subsetD DiffI by blast
    then
    show ?thesis unfolding sep_var_def by simp
  qed
}
moreover
{
  fix  $x$ 
  assume asm:  $x \in 7\# + n$   $x \notin ?A$   $x < 7$ 
  then have  $x \in \text{domain}(\text{sep\_var}(n))$ 
  proof (cases  $2 \leq n$ )
    case True
    moreover
    have  $0 < n$  using leD[OF  $\langle n \in \text{nat} \rangle \langle 2 \leq n \rangle$ ] lt_imp_0_lt by auto
    ultimately
    have  $x < 5$ 
      using  $\langle x < 7 \rangle \langle x \notin ?A \rangle \langle n \in \text{nat} \rangle$  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  $w \in \text{domain}(\text{sep\_var}(n))$  if  $w \in 7\#+n - ?A$  for  $w$ 
  using that by blast
then
have  $7\#+n - ?A \subseteq \text{domain}(\text{sep\_var}(n))$  by blast
with F
show ?thesis by auto
qed

```

```

lemma sep_var_type :
  assumes  $n \in \text{nat}$ 
  shows  $\text{sep\_var}(n) : (7\#+n)\text{-weak}(n,5) \rightarrow 7\#+n$ 
  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
     $[t,p,u,P,leq,o,pi] \in \text{list}(M)$ 
     $env \in \text{list}(M)$ 
  shows  $\forall i . i \in (7\#+\text{length}(env)) - \text{weak}(\text{length}(env),5) \longrightarrow$ 
     $\text{nth}(\text{sep\_var}(\text{length}(env)))'i,[t,p,u,P,leq,o,pi]@env = \text{nth}(i,[p,P,leq,o,t] @ env$ 
     $@ [pi,u])$ 
  using assms
  proof (subst sep_var_domain[OF length_type[OF <env∈list(M)>],symmetric],auto)
    fix  $i y$ 
    assume  $\langle i, y \rangle \in \text{sep\_var}(\text{length}(env))$ 
    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 renssep_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_renssep: 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 renssep_type assms
  by simp

lemma type_renssep [TC]:
  assumes φ∈formula env∈list(M)
  shows sep_ren(length(env),φ) ∈ formula
  unfolding sep_ren_def
  using ren_tc renssep_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  $\exists nenv \in list(M). env = map(val(G), nenv)$ 
  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
       $l' \in list(M) \ l = map(val(G), l') \ a = val(G, a')$ 
       $Cons(a, l) = map(val(G), Cons(a', l')) \ Cons(a', l') \in list(M)$ 
    using ‹a ∈  $M[G]$ › GenExtD

```

by force
 then show ?case by force
 qed

lemma Collect_sats_in_MG :

assumes
 $c \in M[G]$
 $\varphi \in \text{formula } \text{env} \in \text{list}(M[G]) \text{ arity}(\varphi) \leq 1 \# + \text{length}(\text{env})$

shows
 $\{x \in c. (M[G], [x] @ \text{env} \models \varphi)\} \in M[G]$

proof -

from $\langle c \in M[G] \rangle$
 obtain π where $\pi \in M \text{ val}(G, \pi) = c$
 using GenExt_def by auto
 let $?X = \text{And}(\text{Member}(0, 1 \# + \text{length}(\text{env})), \varphi)$ and $?Pl1 = [P, \text{leq}, \text{one}]$
 let $?new_form = \text{sep_ren}(\text{length}(\text{env}), \text{forces}(?X))$
 let $?psi = \text{Exists}(\text{Exists}(\text{And}(\text{pair_fm}(0, 1, 2), ?new_form)))$
 note $phi = \langle \varphi \in \text{formula} \rangle \langle \text{arity}(\varphi) \leq 1 \# + \text{length}(\text{env}) \rangle$
 then
 have $?X \in \text{formula}$ by simp
 with $\langle \text{env} \in _ \rangle phi$
 have $\text{arity}(?X) \leq 2 \# + \text{length}(\text{env})$
 using nat_simp_union leI by simp
 with $\langle \text{env} \in \text{list}(_) \rangle phi$
 have $\text{arity}(\text{forces}(?X)) \leq 6 \# + \text{length}(\text{env})$
 using arity_forces_le by simp
 then
 have $\text{arity}(\text{forces}(?X)) \leq 7 \# + \text{length}(\text{env})$
 using nat_simp_union arity_forces leI by simp
 with $\langle \text{arity}(\text{forces}(?X)) \leq 7 \# + _ \rangle \langle \text{env} \in _ \rangle \langle \varphi \in \text{formula} \rangle$
 have $\text{arity}(?new_form) \leq 7 \# + \text{length}(\text{env}) \ ?new_form \in \text{formula}$
 using arity_resep[OF definability[of ?X]] definability[of ?X] type_resep
 by auto
 then
 have $\text{pred}(\text{pred}(\text{arity}(?new_form))) \leq 5 \# + \text{length}(\text{env}) \ ?psi \in \text{formula}$
 unfolding pair_fm_def upair_fm_def
 using nat_simp_union length_type[OF $\langle \text{env} \in \text{list}(M[G]) \rangle$]
 $\text{pred_mono}[OF _ \text{pred_mono}[OF _ \langle \text{arity}(?new_form) \leq _ \rangle]]$
 by auto
 with $\langle \text{arity}(?new_form) \leq _ \rangle \langle ?new_form \in \text{formula} \rangle$
 have $\text{arity}(?psi) \leq 5 \# + \text{length}(\text{env})$
 unfolding pair_fm_def upair_fm_def
 using nat_simp_union arity_forces
 by auto
 from $\langle \varphi \in \text{formula} \rangle$
 have $\text{forces}(?X) \in \text{formula}$
 using definability by simp
 from $\langle \pi \in M \rangle P_in_M$

```

have domain( $\pi$ ) $\in$ M domain( $\pi$ )  $\times$  P  $\in$  M
  by (simp_all flip:setclass_iff)
from  $\langle$ env  $\in$   $\_$  $\rangle$ 
  obtain nenv where nenv $\in$ list(M) env = map(val(G),nenv) length(nenv) =
length(env)
  using map_val by auto
from  $\langle$ arity( $\varphi$ )  $\leq$   $\_$  $\rangle$   $\langle$ env $\in$  $\_$  $\rangle$   $\langle$  $\varphi$  $\in$  $\_$  $\rangle$ 
have arity( $\varphi$ )  $\leq$  2# + length(env)
  using le_trans[OF  $\langle$ arity( $\varphi$ ) $\leq$  $\_$  $\rangle$ ] add_le_mono[of 1 2,OF  $\_$  le_refl]
  by auto
with  $\langle$ nenv $\in$  $\_$  $\rangle$   $\langle$ env $\in$  $\_$  $\rangle$   $\langle$  $\pi$  $\in$ M $\rangle$   $\langle$  $\varphi$  $\in$  $\_$  $\rangle$   $\langle$ length(nenv) = length(env) $\rangle$ 
have arity( $\varphi$ )  $\leq$  length( $\varphi$ ) @ nenv @  $\pi$  for  $\varphi$ 
  using nat_union_abs2[OF  $\_$   $\_$   $\langle$ arity( $\varphi$ )  $\leq$  2# +  $\_$  $\rangle$ ] nat_simp_union
  by simp
note in_M =  $\langle$  $\pi$  $\in$ M $\rangle$   $\langle$ domain( $\pi$ )  $\times$  P  $\in$  M $\rangle$  P_in_M one_in_M leq_in_M
{
  fix u
  assume u  $\in$  domain( $\pi$ )  $\times$  P u  $\in$  M
  with in_M  $\langle$ ?new_form  $\in$  formula $\rangle$   $\langle$ ? $\psi$  $\in$ formula $\rangle$   $\langle$ nenv  $\in$   $\_$  $\rangle$ 
  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  $\implies$  p $\in$ P  $\implies$ 
    (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  $\vartheta$  p
  proof -
    fix p  $\vartheta$ 
    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  $\langle$  $\vartheta$   $\in$  M $\rangle$   $\langle$ p $\in$ M $\rangle$   $\langle$ u  $\in$  domain( $\pi$ )  $\times$  P $\rangle$   $\langle$ u  $\in$  M $\rangle$ 
 $\langle$ nenv $\in$  $\_$  $\rangle$ 
    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, [∅, p, u]@?PI1@[π] @ nenv ⊨ ?new_form) ↔ (M, ?new_env ⊨
?new_form)
by simp
from in_M' ⟨env ∈ _⟩ Eq1' ⟨length(nenv) = length(env)⟩
⟨arity(forces(?χ)) ≤ 7 #+ length(env)⟩ ⟨forces(?χ) ∈ formula⟩
⟨[∅, p, u, π, leq, one, π] ∈ list(M)⟩
have ... ↔ M, ?env ⊨ forces(?χ)
using sepren_action[of forces(?χ) nenv, OF __ _ ⟨nenv ∈ list(M)⟩]
by simp
also from in_M'
have ... ↔ M, ([p, P, leq, one, ∅]@nenv@[π])@[u] ⊨ forces(?χ)
using app_assoc by simp
also
from in_M' ⟨env ∈ _⟩ phi ⟨length(nenv) = length(env)⟩
⟨arity(forces(?χ)) ≤ 6 #+ length(env)⟩ ⟨forces(?χ) ∈ formula⟩
have ... ↔ M, [p, P, leq, one, ∅]@ nenv @ [π] ⊨ forces(?χ)
by (rule_tac arity_sats_iff, auto)
also
from ⟨arity(forces(?χ)) ≤ 6 #+ length(env)⟩ ⟨forces(?χ) ∈ formula⟩ in_M'
phi
have ... ↔ (∀ F. M_generic(F) ∧ p ∈ F →
M[F], map(val(F), [∅] @ nenv @ [π]) ⊨ ?χ)
using definition_of_forcing
proof (intro iffI)
assume a1: M, [p, P, leq, one, ∅] @ nenv @ [π] ⊨ forces(?χ)
note definition_of_forcing ⟨arity(φ) ≤ 1 #+ _⟩
with ⟨nenv ∈ _⟩ ⟨arity(?χ) ≤ length([∅] @ nenv @ [π])⟩ ⟨env ∈ _⟩
have p ∈ P ⇒ ?χ ∈ formula ⇒ [∅, π] ∈ list(M) ⇒
M, [p, P, leq, one] @ [∅]@ nenv@[π] ⊨ forces(?χ) ⇒
∀ G. M_generic(G) ∧ p ∈ G → M[G], map(val(G), [∅] @ nenv @ [π])
⊨ ?χ
by auto
then
show ∀ F. M_generic(F) ∧ p ∈ F →
M[F], map(val(F), [∅] @ nenv @ [π]) ⊨ ?χ
using ⟨?χ ∈ formula⟩ ⟨p ∈ P⟩ a1 ⟨∅ ∈ M⟩ ⟨π ∈ M⟩ by simp
next
assume ∀ F. M_generic(F) ∧ p ∈ F →
M[F], map(val(F), [∅] @ nenv @ [π]) ⊨ ?χ
with definition_of_forcing [THEN iffD2] ⟨arity(?χ) ≤ length([∅] @ nenv
@ [π])⟩
show M, [p, P, leq, one, ∅] @ nenv @ [π] ⊨ forces(?χ)
using ⟨?χ ∈ formula⟩ ⟨p ∈ P⟩ in_M'
by auto
qed
finally
show (M, [∅, p, u]@?PI1@[π]@nenv ⊨ ?new_form) ↔ (∀ F. M_generic(F)
∧ p ∈ F →
M[F], map(val(F), [∅] @ nenv @ [π]) ⊨ ?χ)

```

```

    by simp
  qed
  with Eq1
  have (M, [u] @ ?Pl1 @ [π] @ nenv ⊨ ?ψ) ↔
    (∃ ϑ ∈ M. ∃ p ∈ P. u = ⟨ϑ, p⟩ ∧
     (∀ F. M_generic(F) ∧ p ∈ F → M[F], map(val(F), [ϑ] @ nenv @ [π])
    ⊨ ?χ))
  by auto
}
then
have Equivalence: u ∈ domain(π) × P ⇒ u ∈ M ⇒
  (M, [u] @ ?Pl1 @ [π] @ nenv ⊨ ?ψ) ↔
  (∃ ϑ ∈ M. ∃ p ∈ P. u = ⟨ϑ, p⟩ ∧
   (∀ F. M_generic(F) ∧ p ∈ F → M[F], map(val(F), [ϑ] @ nenv @ [π])
  ⊨ ?χ))
for u
by simp
moreover from ⟨env = _⟩ ⟨π ∈ M⟩ ⟨nenv ∈ list(M)⟩
have map_nenv: map(val(G), nenv @ [π]) = env @ [val(G, π)]
  using map_app_distrib append1_eq_iff by auto
ultimately
have aux: (∃ ϑ ∈ M. ∃ p ∈ P. u = ⟨ϑ, p⟩ ∧ (p ∈ G → M[G], [val(G, ϑ)] @ env @
[val(G, π)] ⊨ ?χ))
  (is (∃ ϑ ∈ M. ∃ p ∈ P. _ ( _ → _, ?vals(ϑ) ⊨ _)))
  if u ∈ domain(π) × P u ∈ M M, [u] @ ?Pl1 @ [π] @ nenv ⊨ ?ψ for u
  using Equivalence[THEN iffD1, OF that] generic by force
moreover
have ϑ ∈ M ⇒ val(G, ϑ) ∈ M[G] for ϑ
  using GenExt_def by auto
moreover
have ϑ ∈ M ⇒ [val(G, ϑ)] @ env @ [val(G, π)] ∈ list(M[G]) for ϑ
proof -
  from ⟨π ∈ M⟩
  have val(G, π) ∈ M[G] using GenExtI by simp
  moreover
  assume ϑ ∈ M
  moreover
  note ⟨env ∈ list(M[G])⟩
  ultimately
  show ?thesis
  using GenExtI by simp
qed
ultimately
have (∃ ϑ ∈ M. ∃ p ∈ P. u = ⟨ϑ, p⟩ ∧ (p ∈ G → val(G, ϑ) ∈ nth(1 #+ length(env), [val(G,
ϑ)] @ env @ [val(G, π)]))
  ∧ M[G], ?vals(ϑ) ⊨ ϕ)
  if u ∈ domain(π) × P u ∈ M M, [u] @ ?Pl1 @ [π] @ nenv ⊨ ?ψ for u
  using aux[OF that] by simp
moreover from ⟨env ∈ _⟩ ⟨π ∈ M⟩

```

have $nth:nth(1 \ \#\ + \ length(env), [val(G, \vartheta)] \ @ \ env \ @ \ [val(G, \pi)]) = val(G, \pi)$
if $\vartheta \in M$ **for** ϑ
using $nth_concat[of \ val(G, \vartheta) \ val(G, \pi) \ M[G]]$ **using that** $GenExtI$ **by simp**
ultimately
have $(\exists \vartheta \in M. \exists p \in P. u = \langle \vartheta, p \rangle \wedge (p \in G \longrightarrow val(G, \vartheta) \in val(G, \pi) \wedge M[G], \ ?vals(\vartheta) \models \varphi))$
 $\models \varphi)$
if $u \in domain(\pi) \times P$ $u \in M$ $M, [u] \ @ \ ?Pl1 \ @ [\pi] \ @ \ nenv \models \ ?\psi$ **for** u
using that $\langle \pi \in M \rangle \langle env \in _ \rangle$ **by simp**
with $\langle domain(\pi) \times P \in M \rangle$
have $\forall u \in domain(\pi) \times P . (M, [u] \ @ \ ?Pl1 \ @ [\pi] \ @ \ nenv \models \ ?\psi) \longrightarrow (\exists \vartheta \in M. \exists p \in P. u = \langle \vartheta, p \rangle \wedge (p \in G \longrightarrow val(G, \vartheta) \in val(G, \pi) \wedge M[G], \ ?vals(\vartheta) \models \varphi))$
 $\exists p \in P. u = \langle \vartheta, p \rangle \wedge (p \in G \longrightarrow val(G, \vartheta) \in val(G, \pi) \wedge M[G], \ ?vals(\vartheta) \models \varphi))$
by $(simp \ add:transitivity)$
then
have $\{u \in domain(\pi) \times P . (M, [u] \ @ \ ?Pl1 \ @ [\pi] \ @ \ nenv \models \ ?\psi)\} \subseteq \{u \in domain(\pi) \times P . \exists \vartheta \in M. \exists p \in P. u = \langle \vartheta, p \rangle \wedge (p \in G \longrightarrow val(G, \vartheta) \in val(G, \pi) \wedge (M[G], \ ?vals(\vartheta) \models \varphi))\}$
(is $\ ?n \subseteq \ ?m)$
by auto
with val_mono
have $first_incl: val(G, \ ?n) \subseteq val(G, \ ?m)$
by simp
note $\langle val(G, \pi) = c \rangle$
with $\langle \ ?\psi \in formula \rangle \langle arity(\ ?\psi) \leq _ \rangle$ **in** M $\langle nenv \in _ \rangle \langle env \in _ \rangle \langle length(nenv) = _ \rangle$
have $\ ?n \in M$
using $separation_ax \ leI \ separation_iff$ **by auto**
from $generic$
have $filter(G) \ G \subseteq P$
unfolding $M_generic_def \ filter_def$ **by simp_all**
from $\langle val(G, \pi) = c \rangle$
have $val(G, \ ?m) = \{val(G, t) .. t \in domain(\pi) , \exists q \in P . (\exists \vartheta \in M. \exists p \in P. \langle t, q \rangle = \langle \vartheta, p \rangle \wedge (p \in G \longrightarrow val(G, \vartheta) \in c \wedge (M[G], [val(G, \vartheta)] \ @ \ env \ @ \ [c] \models \varphi)) \wedge q \in G)\}$
using val_of_name **by auto**
also
have $\dots = \{val(G, t) .. t \in domain(\pi) , \exists q \in P. val(G, t) \in c \wedge (M[G], [val(G, t)] \ @ \ env \ @ \ [c] \models \varphi) \wedge q \in G\}$
proof -
have $t \in M \implies (\exists q \in P. (\exists \vartheta \in M. \exists p \in P. \langle t, q \rangle = \langle \vartheta, p \rangle \wedge (p \in G \longrightarrow val(G, \vartheta) \in c \wedge (M[G], [val(G, \vartheta)] \ @ \ env \ @ \ [c] \models \varphi)) \wedge q \in G)$
 $\iff (\exists q \in P. val(G, t) \in c \wedge (M[G], [val(G, t)] \ @ \ env \ @ \ [c] \models \varphi) \wedge q \in G)$ **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 ∃∅. ∃q∈G. ⟨∅,q⟩∈π ∧ val(G,∅) =x
    using elem_of_val_pair by auto
  then obtain ∅ q where
    ⟨∅,q⟩∈π q∈G val(G,∅)=x by auto
  from ⟨∅,q⟩∈π
  have ∅∈M
    using domain_trans[OF trans_M ⟨π∈_⟩] by auto
  with ⟨π∈M⟩ ⟨nenv ∈ _⟩ ⟨env = _⟩

```

```

have [val(G,∅), val(G,π)] @ env ∈ list(M[G])
  using GenExt_def by auto
with Eq4 ⟨val(G,∅)=x⟩ ⟨val(G,π) = c⟩ ⟨x ∈ val(G,π)⟩ nth ⟨∅∈M⟩
  have Eq5: M[G], [val(G,∅)] @ env @[val(G,π)] ⊨ And(Member(0,1 #+
length(env)),φ)
  by auto

with ⟨∅∈M⟩ ⟨π∈M⟩ Eq5 ⟨M_generic(G)⟩ ⟨φ∈formula⟩ ⟨nenv ∈ _⟩ ⟨env =
_⟩ map_nenv
  ⟨arity(?χ) ≤ length([∅] @ nenv @ [π])⟩
have (∃ r∈G. M, [r,P,leq,one,∅] @ nenv @ [π] ⊨ forces(?χ))
  using truth_lemma
  by auto
then obtain r where
  r∈G M, [r,P,leq,one,∅] @ nenv @ [π] ⊨ forces(?χ) by auto
with ⟨filter(G)⟩ and ⟨q∈G⟩ obtain p where
  p∈G p≤q p≤r
  unfolding filter_def compat_in_def by force
with ⟨r∈G⟩ ⟨q∈G⟩ ⟨G⊆P⟩
have p∈P r∈P q∈P p∈M
  using P_in_M by (auto simp add:transitivity)
with ⟨φ∈formula⟩ ⟨∅∈M⟩ ⟨π∈M⟩ ⟨p≤r⟩ ⟨nenv ∈ _⟩ ⟨arity(?χ) ≤ length([∅]
@ nenv @ [π])⟩
  ⟨M, [r,P,leq,one,∅] @ nenv @ [π] ⊨ forces(?χ)⟩ ⟨env∈_⟩
have M, [p,P,leq,one,∅] @ nenv @ [π] ⊨ forces(?χ)
  using strengthening_lemma
  by simp
with ⟨p∈P⟩ ⟨φ∈formula⟩ ⟨∅∈M⟩ ⟨π∈M⟩ ⟨nenv ∈ _⟩ ⟨arity(?χ) ≤ length([∅] @
nenv @ [π])⟩
have ∀ F. M_generic(F) ∧ p ∈ F →
  M[F], map(val(F), [∅] @ nenv @ [π]) ⊨ ?χ
  using definition_of_forcing
  by simp
with ⟨p∈P⟩ ⟨∅∈M⟩
have Eq6: ∃ ∅'∈M. ∃ p'∈P. ⟨∅,p⟩ = ⟨∅',p'⟩ ∧ (∀ F. M_generic(F) ∧ p' ∈ F
→
  M[F], map(val(F), [∅'] @ nenv @ [π]) ⊨ ?χ) by auto
from ⟨π∈M⟩ ⟨⟨∅,q⟩∈π⟩
have ⟨∅,q⟩ ∈ M by (simp add:transitivity)
from ⟨⟨∅,q⟩∈π⟩ ⟨∅∈M⟩ ⟨p∈P⟩ ⟨p∈M⟩
have ⟨∅,p⟩∈M ⟨∅,p⟩∈domain(π)×P
  using tuples_in_M by auto
with ⟨∅∈M⟩ Eq6 ⟨p∈P⟩
have M, [⟨∅,p⟩] @ ?Pl1 @ [π] @ nenv ⊨ ?ψ
  using Equivalence by auto
with ⟨⟨∅,p⟩∈domain(π)×P⟩
have ⟨∅,p⟩∈?n by simp
with ⟨p∈G⟩ ⟨p∈P⟩
have val(G,∅)∈val(G,?n)

```

```

    using val_of_elem[of  $\vartheta$   $p$ ] by simp
  with  $\langle \text{val}(G, \vartheta) = x \rangle$ 
  show  $x \in \text{val}(G, ?n)$  by simp
qed
with val_m_first_incl
have  $\text{val}(G, ?n) = \{x \in c. (M[G], [x] @ \text{env} @ [c] \models \varphi)\}$  by auto
also
have  $\dots = \{x \in c. (M[G], [x] @ \text{env} \models \varphi)\}$ 
proof -
  {
    fix  $x$ 
    assume  $x \in c$ 
    moreover from assms
    have  $c \in M[G]$ 
      unfolding GenExt_def by auto
    moreover from this and  $\langle x \in c \rangle$ 
    have  $x \in M[G]$ 
      using transitivity_MG
      by simp
    ultimately
    have  $(M[G], ([x] @ \text{env}) @ [c] \models \varphi) \longleftrightarrow (M[G], [x] @ \text{env} \models \varphi)$ 
      using phi  $\langle \text{env} \in \_ \rangle$  by (rule_tac arity_sats_iff, simp_all)
  }
  then show ?thesis by auto
qed
finally
show  $\{x \in c. (M[G], [x] @ \text{env} \models \varphi)\} \in M[G]$ 
  using  $\langle ?n \in M \rangle$  GenExt_def by force
qed

theorem separation_in_MG:
  assumes
     $\varphi \in \text{formula}$  and  $\text{arity}(\varphi) \leq 1 \ \#\ + \ \text{length}(\text{env})$  and  $\text{env} \in \text{list}(M[G])$ 
  shows
    separation( $\#\ \#\ M[G]$ ,  $\lambda x. (M[G], [x] @ \text{env} \models \varphi)$ )
proof -
  {
    fix  $c$ 
    assume  $c \in M[G]$ 
    moreover from  $\langle \text{env} \in \_ \rangle$ 
    obtain nenv where  $\text{nenv} \in \text{list}(M)$ 
       $\text{env} = \text{map}(\text{val}(G), \text{nenv})$   $\text{length}(\text{env}) = \text{length}(\text{nenv})$ 
      using GenExt_def map_val[of env] by auto
    moreover note  $\langle \varphi \in \_ \rangle$   $\langle \text{arity}(\varphi) \leq \_ \rangle$   $\langle \text{env} \in \_ \rangle$ 
    ultimately
    have Eq1:  $\{x \in c. (M[G], [x] @ \text{env} \models \varphi)\} \in M[G]$ 
      using Collect_sats_in_MG by auto
  }
  then

```

```

  show ?thesis
  using separation_iff rev_bexI unfolding is_Collect_def by force
qed

end

end

```

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

```

theory Pairing_Axiom imports Names begin

```

```

context forcing_data
begin

```

```

lemma val_Upair :

```

```

  one ∈ G ⇒ val(G, {⟨τ, one⟩, ⟨ρ, one⟩}) = {val(G, τ), val(G, ρ)}

```

```

  by (insert one_in_P, rule trans, subst def_val, auto simp add: Sep_and_Replace)

```

```

lemma pairing_in_MG :

```

```

  assumes M_generic(G)

```

```

  shows upair_ax(##M[G])

```

```

proof -

```

```

  {

```

```

    fix x y

```

```

    have one∈G using assms one_in_G by simp

```

```

    from assms

```

```

    have G⊆P unfolding M_generic_def and filter_def by simp

```

```

    with ⟨one∈G⟩

```

```

    have one∈P using subsetD by simp

```

```

    then

```

```

    have one∈M using transitivity[OF _ P_in_M] by simp

```

```

    assume x ∈ M[G] y ∈ M[G]

```

```

    then

```

```

    obtain τ ρ where

```

```

      0 : val(G, τ) = x val(G, ρ) = y ρ ∈ M τ ∈ M

```

```

      using GenExtD by blast

```

```

    with ⟨one∈M⟩

```

```

    have ⟨τ, one⟩ ∈ M ⟨ρ, one⟩ ∈ M using pair_in_M_iff by auto

```

```

    then

```

```

    have 1: {⟨τ, one⟩, ⟨ρ, one⟩} ∈ M (is ?σ ∈ _) using upair_in_M_iff by simp

```

```

    then

```

```

    have val(G, ?σ) ∈ M[G] using GenExtI by simp

```

```

    with 1

```

```

    have {val(G, τ), val(G, ρ)} ∈ M[G] using val_Upair assms one_in_G by simp

```

```

    with 0

```

```

    have {x, y} ∈ 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* :: $[i,i,i,i] \Rightarrow o$ where
 $Union_name_body(P',leq',\tau,\vartheta p) \equiv (\exists \sigma[\#\#M].$
 $\exists q[\#\#M]. (q \in P' \wedge (\langle \sigma, q \rangle \in \tau \wedge$
 $(\exists r[\#\#M]. r \in P' \wedge (\langle fst(\vartheta p), r \rangle \in \sigma \wedge \langle snd(\vartheta p), r \rangle \in leq' \wedge \langle snd(\vartheta p), q \rangle$
 $\in leq'))))$

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

lemma *Union_name_fm_type* [TC]:
 $Union_name_fm \in formula$
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* :
 $\llbracket a \in M; b \in M; P' \in M; p \in M; \vartheta \in M; \tau \in M; leq' \in M \rrbracket \implies$
 $sats(M, Union_name_fm, [\langle \vartheta, p \rangle, \tau, leq', P'] @ [a, b]) \longleftrightarrow$
 $Union_name_body(P', leq', \tau, \langle \vartheta, p \rangle)$
unfolding *Union_name_fm_def Union_name_body_def tuples_in_M*
by(*subgoal_tac* $\langle \vartheta, p \rangle \in M$, *auto simp add: tuples_in_M*)

```

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

definition Union_name ::  $i \Rightarrow i$  where
  Union_name( $\tau$ )  $\equiv$ 
   $\{u \in \text{domain}(\bigcup(\text{domain}(\tau))) \times P . \text{Union\_name\_body}(P, \text{leq}, \tau, u)\}$ 

lemma Union_name_M : assumes  $\tau \in M$ 
  shows  $\{u \in \text{domain}(\bigcup(\text{domain}(\tau))) \times P . \text{Union\_name\_body}(P, \text{leq}, \tau, u)\} \in M$ 
  unfolding Union_name_def
proof -
  let  $?P = \lambda x . \text{sats}(M, \text{Union\_name\_fm}, [x, \tau, \text{leq}] @ [P, \tau, \text{leq}])$ 
  let  $?Q = \lambda x . \text{Union\_name\_body}(P, \text{leq}, \tau, x)$ 
  from  $\langle \tau \in M \rangle$ 
  have  $\text{domain}(\bigcup(\text{domain}(\tau))) \in M$  (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  $\text{arity}(\text{Union\_name\_fm}) \leq 6$  using arity_Union_name_fm by simp
  from assms P_in_M leq_in_M arity_Union_name_fm
  have  $[\tau, \text{leq}] \in \text{list}(M)$   $[P, \tau, \text{leq}] \in \text{list}(M)$  by auto
  with assms assms P_in_M leq_in_M  $\langle \text{arity}(\text{Union\_name\_fm}) \leq 6 \rangle$ 
  have separation(##M, ?P)
  using separation_ax by simp
  with  $\langle ?d \times P \in M \rangle$ 
  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  $\langle x \in ?d \times P \rangle$ 
  have  $x = \langle \text{fst}(x), \text{snd}(x) \rangle$  using Pair_fst_snd_eq by simp
  with  $\langle x \in ?d \times P \rangle$   $\langle ?d \in M \rangle$ 
  have  $\text{fst}(x) \in M$   $\text{snd}(x) \in M$ 
  using mtrans fst_type snd_type P_in_M unfolding M_trans_def by auto
  then
  have  $?P(\langle \text{fst}(x), \text{snd}(x) \rangle) \longleftrightarrow ?Q(\langle \text{fst}(x), \text{snd}(x) \rangle)$ 
  using P_in_M sats_Union_name_fm P_in_M  $\langle \tau \in M \rangle$  leq_in_M by simp
  with  $\langle x = \langle \text{fst}(x), \text{snd}(x) \rangle \rangle$ 
  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 $a = \text{val}(G, \tau)$ and $\text{filter}(G)$ and $\tau \in M$

shows $\bigcup a = \text{val}(G, \text{Union_name}(\tau))$

proof -

```

{
  fix x
  assume  $x \in \bigcup (\text{val}(G, \tau))$ 
  then obtain i where  $i \in \text{val}(G, \tau)$   $x \in i$  by blast
  with  $\langle \tau \in M \rangle$  obtain  $\sigma$   $q$  where
     $q \in G$   $\langle \sigma, q \rangle \in \tau$   $\text{val}(G, \sigma) = i$   $\sigma \in M$ 
    using elem_of_val_pair domD by blast
  with  $\langle x \in i \rangle$  obtain  $\vartheta$   $r$  where
     $r \in G$   $\langle \vartheta, r \rangle \in \sigma$   $\text{val}(G, \vartheta) = x$   $\vartheta \in M$ 
    using elem_of_val_pair domD by blast
  with  $\langle \langle \sigma, q \rangle \in \tau \rangle$  have  $\vartheta \in \text{domain}(\bigcup (\text{domain}(\tau)))$  by auto
  with  $\langle \text{filter}(G) \rangle$   $\langle q \in G \rangle$   $\langle r \in G \rangle$  obtain  $p$  where
     $A: p \in G$   $\langle p, r \rangle \in \text{leq}$   $\langle p, q \rangle \in \text{leq}$   $p \in P$   $r \in P$   $q \in P$ 
    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  $A$   $\langle \langle \vartheta, r \rangle \in \sigma \rangle$   $\langle \langle \sigma, q \rangle \in \tau \rangle$   $\langle \vartheta \in M \rangle$   $\langle \vartheta \in \text{domain}(\bigcup (\text{domain}(\tau))) \rangle$   $\langle \sigma \in M \rangle$ 
  have
     $\langle \vartheta, p \rangle \in \text{Union\_name}(\tau)$  unfolding Union_name_def Union_name_body_def
    by auto
  with  $\langle p \in P \rangle$   $\langle p \in G \rangle$  have  $\text{val}(G, \vartheta) \in \text{val}(G, \text{Union\_name}(\tau))$ 
    using val_of_elem by simp
  with  $\langle \text{val}(G, \vartheta) = x \rangle$  have  $x \in \text{val}(G, \text{Union\_name}(\tau))$  by simp
}
with  $\langle a = \text{val}(G, \tau) \rangle$  have  $1: x \in \bigcup a \implies x \in \text{val}(G, \text{Union\_name}(\tau))$  for  $x$  by
simp
{
  fix x
  assume  $x \in (\text{val}(G, \text{Union\_name}(\tau)))$ 
  then obtain  $\vartheta$   $p$  where
     $p \in G$   $\langle \vartheta, p \rangle \in \text{Union\_name}(\tau)$   $\text{val}(G, \vartheta) = x$ 
    using elem_of_val_pair by blast
  with  $\langle \text{filter}(G) \rangle$  have  $p \in P$  using filterD by simp
  from  $\langle \langle \vartheta, p \rangle \in \text{Union\_name}(\tau) \rangle$  obtain  $\sigma$   $q$   $r$  where
     $\sigma \in \text{domain}(\tau)$   $\langle \sigma, q \rangle \in \tau$   $\langle \vartheta, r \rangle \in \sigma$   $r \in P$   $q \in P$   $\langle p, r \rangle \in \text{leq}$   $\langle p, q \rangle \in \text{leq}$ 
    unfolding Union_name_def Union_name_body_def by force
  with  $\langle p \in G \rangle$   $\langle \text{filter}(G) \rangle$  have  $r \in G$   $q \in G$ 
    using filter_leqD by auto
  with  $\langle \langle \vartheta, r \rangle \in \sigma \rangle$   $\langle \langle \sigma, q \rangle \in \tau \rangle$   $\langle q \in P \rangle$   $\langle r \in P \rangle$  have
     $\text{val}(G, \sigma) \in \text{val}(G, \tau)$   $\text{val}(G, \vartheta) \in \text{val}(G, \sigma)$ 
    using val_of_elem by simp+
  then have  $\text{val}(G, \vartheta) \in \bigcup \text{val}(G, \tau)$  by blast
  with  $\langle \text{val}(G, \vartheta) = x \rangle$   $\langle a = \text{val}(G, \tau) \rangle$  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

$Transset(M) \ b \in M$

shows

$\{x \in b . P(x)\} = \{x \in b . P(x)\} \cap M$

using *assms* **unfolding** *Transset_def*

by (*auto*)

context *G_generic* **begin**

lemma *name_components_in_M*:

assumes $\langle \sigma, p \rangle \in \vartheta \ \vartheta \in M$

shows $\sigma \in M \ p \in M$

proof -

from *assms* **obtain** *a* **where**

$\sigma \in a \ p \in a \ a \in \langle \sigma, p \rangle$

unfolding *Pair_def* **by** *auto*

moreover from *assms*

have $\langle \sigma, p \rangle \in M$

using *transitivity* **by** *simp*

moreover from *calculation*

have $a \in M$

using *transitivity* **by** *simp*

ultimately

show $\sigma \in M \ p \in M$

using *transitivity* **by** *simp_all*

qed

lemma *sats_fst_snd_in_M*:

assumes

$A \in M \ B \in M \ \varphi \in \text{formula} \ p \in M \ l \in M \ o \in M \ \chi \in M$

$\text{arity}(\varphi) \leq 6$

shows

$\{sq \in A \times B . \text{sats}(M, \varphi, [\text{snd}(sq), p, l, o, \text{fst}(sq), \chi])\} \in M$

(**is** $\vartheta \in M$)

proof -

have $6 \in \text{nat} \ 7 \in \text{nat}$ **by** *simp_all*

let $?\varphi' = \text{ren}(\varphi) \text{'6'7'perm_pow_fn}$

from $\langle A \in M \rangle \ \langle B \in M \rangle$ **have**

$A \times B \in M$

using *cartprod_closed* **by** *simp*

from $\langle \text{arity}(\varphi) \leq 6 \rangle \ \langle \varphi \in \text{formula} \rangle \ \langle 6 \in _ \rangle \ \langle 7 \in _ \rangle$

have $?\varphi' \in \text{formula} \ \text{arity}(\varphi') \leq 7$

unfolding *perm_pow_fn_def*

using *perm_pow_thm* *arity_ren* *ren_tc* *Nil_type*

by *auto*

with $\langle ?\varphi' \in \text{formula} \rangle$

have $1: \text{arity}(\text{Exists}(\text{Exists}(\text{And}(\text{pair_fm}(0, 1, 2), ?\varphi')))) \leq 5 \quad (\text{is } \text{arity}(\varphi) \leq 5)$

```

unfolding pair_fm_def upair_fm_def
using nat_simp_union pred_le arity_type by auto
{
  fix sp
  note  $\langle A \times B \in M \rangle$ 
  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  $\langle A \in M \rangle \langle B \in M \rangle$  transitivity
    by simp_all
  note  $inM = \langle A \in M \rangle \langle B \in M \rangle \langle p \in M \rangle \langle l \in M \rangle \langle o \in M \rangle \langle \chi \in M \rangle$ 
     $\langle sp \in M \rangle \langle fst(sp) \in M \rangle \langle snd(sp) \in M \rangle$ 
  with 1  $\langle sp \in M \rangle \langle ?\varphi' \in formula \rangle$ 
  have  $M, [sp, p, l, o, \chi]@[p] \models ?\psi \longleftrightarrow M, [sp, p, l, o, \chi] \models ?\psi$  (is  $M, ?env0 @ \_ \models \_$ 
 $\longleftrightarrow \_$ )
    using arity_sats_iff[of  $?\psi$   $[p]$   $M$   $?env0$ ] by auto
  also from  $inM \langle sp \in A \times B \rangle$ 
  have ...  $\longleftrightarrow sats(M, ?\varphi', [fst(sp), snd(sp), sp, p, l, o, \chi])$ 
    by auto
  also from  $inM \langle \varphi \in formula \rangle \langle arity(\varphi) \leq 6 \rangle$ 
  have ...  $\longleftrightarrow sats(M, \varphi, [snd(sp), p, l, o, fst(sp), \chi])$ 
    (is  $sats(\_, \_, ?env1) \longleftrightarrow sats(\_, \_, ?env2)$ )
  using sats_iff_sats_ren[of  $\varphi$  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
   $?\emptyset = \{sp \in A \times B . sats(M, ?\psi, [sp, p, l, o, \chi, p])\}$ 
  by auto
also from  $assms \langle A \times B \in M \rangle$  have
  ...  $\in M$ 
proof -
  from 1
  have  $arity(?\psi) \leq 6$ 
    using leI by simp
  moreover from  $\langle ?\varphi' \in formula \rangle$ 
  have  $?\psi \in formula$ 
    by simp
  moreover note  $assms \langle A \times B \in M \rangle$ 
  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  $\chi$  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  $\langle domain(\tau) \in M \rangle \langle \chi \in M \rangle$ 
  have  $?\vartheta \in M$ 
    using P_in_M one_in_M leq_in_M satsfst_snd_in_M
    by simp
  then
  have  $?Q \in ?Q$ 
    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  $\sigma$   $p$  where
       $1: \langle \sigma, p \rangle \in ?\vartheta$   $p \in G$   $val(G, \sigma) = x$ 
      using elem_of_val_pair
      by blast
    moreover from  $\langle \langle \sigma, p \rangle \in ?\vartheta \rangle \langle ?\vartheta \in M \rangle$ 
    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  $\langle val(G, \chi) = c \rangle$ 
    ultimately
    have  $sats(M[G], Member(0,1), [x, c])$ 
      using  $\langle \chi \in M \rangle$  generic_definition_of_forcing_nat_simp_union
      by auto
    moreover
    have  $x \in M[G]$ 
      using  $\langle val(G, \sigma) = x \rangle \langle \sigma \in M \rangle \langle \chi \in M \rangle$  GenExtI by blast
    ultimately
    show  $x \in c$ 
      using  $\langle c \in M[G] \rangle$  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⟩ ∈ ?∅
  using ⟨σ ∈ domain(τ)⟩ by simp
with ⟨val(G, σ) = x⟩ ⟨p ∈ G⟩
show x ∈ val(G, ?∅)
  using val_of_elem [of _ _ ?∅] by auto
qed
with ⟨val(G, ?∅) ∈ ?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  $\langle x \in M[G] \rangle$  have
       $z \in x \longleftrightarrow z \in M[G] \wedge z \in x$ 
    using transitivity_MG by auto
    also have
       $\dots \longleftrightarrow z \in y$ 
    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  $x \in M[G] \exists y \in M[G] . y \in x$ 
    then
    have  $\exists y \in M[G] . y \in x \cap M[G]$  by simp
    then
    obtain y where  $y \in x \cap M[G] \forall z \in y . z \notin x \cap M[G]$ 
      using foundation[of x ∩ M[G]] by blast
    then
    have  $\exists y \in M[G] . y \in x \wedge (\forall z \in M[G] . z \notin x \vee z \notin y)$  by auto
  }
  then show ?thesis
    unfolding foundation_ax_def by auto
qed

end
end

```

26 The binder *Least*

```

theory Least
  imports
    Names

```

```

begin

```

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

```

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

```

```

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

```

```

definition

```

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

definition

$least_fm :: [i, i] \Rightarrow i$ **where**
 $least_fm(q, i) \equiv And(ordinal_fm(i),$
 $Or(And(empty_fm(i), Forall(Implies(ordinal_fm(0), Neg(q)))),$
 $And(Exists(And(q, Equal(0, succ(i))),$
 $Forall(Implies(And(ordinal_fm(0), Member(0, succ(i))), Neg(q))))))$

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

lemmas $basic_fm_simps = sats_subset_fm' sats_transset_fm' sats_ordinal_fm'$

lemma $sats_least_fm :$

assumes $p_iff_sats:$

$\bigwedge a. a \in A \Longrightarrow P(a) \longleftrightarrow sats(A, p, Cons(a, env))$

shows

$\llbracket y \in nat; env \in list(A) ; 0 \in A \rrbracket$
 $\Longrightarrow sats(A, least_fm(p, y), env) \longleftrightarrow$
 $least(\#\#A, P, nth(y, env))$

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

$\llbracket nth(j, env) = y; j \in nat; env \in list(A); 0 \in A \rrbracket$
 $\Longrightarrow least(\#\#A, is_Q, y) \longleftrightarrow sats(A, least_fm(q, j), env)$

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 $\bigwedge x. Q(x) \implies M(x) \ M(a)$

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 $\langle \bigwedge x. Q(x) \implies M(x) \rangle$

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 $\langle M(a) \rangle$

moreover from $\langle Q(i) \rangle \ \langle Ord(i) \rangle$

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 $\langle \bigwedge x. Q(x) \implies M(x) \rangle$

have $Ord(b) \implies b \in a \implies \neg Q(b)$ for *b*

by *blast*

moreover from *this* and $\langle Ord(a) \rangle$

have $b < a \implies \neg Q(b)$ 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 $\bigwedge x. Q(x) \implies M(x)$
shows $M(\mu x. Q(x))$
using *assms* *LeastI*[of *Q*] *Least_0* **by** (*cases* ($\exists i. \text{Ord}(i) \wedge Q(i)$), *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**

$\text{renrep_fn}(env) \equiv \text{sum}(\text{renrep1_fn}, \text{id}(\text{length}(env)), 6, 8, \text{length}(env))$

definition

renrep :: [i, i] $\Rightarrow i$ **where**

$\text{renrep}(\varphi, env) = \text{ren}(\varphi)^{(6\# + \text{length}(env))} (8\# + \text{length}(env)) \text{renrep_fn}(env)$

lemma *renrep_type* [*TC*]:

assumes $\varphi \in \text{formula}$ $env \in \text{list}(M)$

shows $\text{renrep}(\varphi, env) \in \text{formula}$

unfolding *renrep_def* *renrep_fn_def* *renrep1_fn_def*

using *assms* *renrep1_thm(1)* *ren_tc*

by *simp*

lemma *arity_renrep*:

assumes $\varphi \in \text{formula}$ $\text{arity}(\varphi) \leq 6\# + \text{length}(env)$ $env \in \text{list}(M)$

shows $\text{arity}(\text{renrep}(\varphi, env)) \leq 8\# + \text{length}(env)$

unfolding *renrep_def* *renrep_fn_def* *renrep1_fn_def*

using *assms* *renrep1_thm(1)* *arity_ren*

by *simp*

lemma *renrep_sats* :

assumes $\text{arity}(\varphi) \leq 6\# + \text{length}(env)$

$[P, leq, o, p, \rho, \tau] @ env \in \text{list}(M)$

$V \in M \ \alpha \in M$

$\varphi \in \text{formula}$
shows $\text{sats}(M, \varphi, [p, P, \text{leq}, o, \varrho, \tau] \text{ @ } \text{env}) \longleftrightarrow \text{sats}(M, \text{renrep}(\varphi, \text{env}), [V, \tau, \varrho, p, \alpha, P, \text{leq}, o] \text{ @ } \text{env})$
unfolding renrep_def renrep_fn_def renrep1_fn_def
by (rule sats_iff_sats_ren , insert assms , auto simp add: $\text{renrep1_thm}(1)$) [of $_$ $M, \text{simplified}$]
 $\text{renrep1_thm}(2)$ [simplified, where $p=p$ and $\alpha=\alpha$]

rename renpbdy1 **src** $[\varrho, p, \alpha, P, \text{leq}, o]$ **tgt** $[\varrho, p, x, \alpha, P, \text{leq}, o]$

definition $\text{renpbdy_fn} :: i \Rightarrow i$ **where**
 $\text{renpbdy_fn}(\text{env}) \equiv \text{sum}(\text{renpbdy1_fn}, \text{id}(\text{length}(\text{env})), 6, 7, \text{length}(\text{env}))$

definition
 $\text{renpbdy} :: [i, i] \Rightarrow i$ **where**
 $\text{renpbdy}(\varphi, \text{env}) = \text{ren}(\varphi) '(6 \# + \text{length}(\text{env})) '(7 \# + \text{length}(\text{env})) ' \text{renpbdy_fn}(\text{env})$

lemma
 renpbdy_type [TC]: $\varphi \in \text{formula} \Longrightarrow \text{env} \in \text{list}(M) \Longrightarrow \text{renpbdy}(\varphi, \text{env}) \in \text{formula}$
unfolding renpbdy_def renpbdy_fn_def renpbdy1_fn_def
using $\text{renpbdy1_thm}(1)$ ren_tc
by simp

lemma arity_renpbdy : $\varphi \in \text{formula} \Longrightarrow \text{arity}(\varphi) \leq 6 \# + \text{length}(\text{env}) \Longrightarrow \text{env} \in \text{list}(M) \Longrightarrow \text{arity}(\text{renpbdy}(\varphi, \text{env})) \leq 7 \# + \text{length}(\text{env})$
unfolding renpbdy_def renpbdy_fn_def renpbdy1_fn_def
using $\text{renpbdy1_thm}(1)$ arity_ren
by simp

lemma
 sats_renpbdy : $\text{arity}(\varphi) \leq 6 \# + \text{length}(\text{nenv}) \Longrightarrow [\varrho, p, x, \alpha, P, \text{leq}, o, \pi] \text{ @ } \text{nenv} \in \text{list}(M) \Longrightarrow \varphi \in \text{formula} \Longrightarrow \text{sats}(M, \varphi, [\varrho, p, \alpha, P, \text{leq}, o] \text{ @ } \text{nenv}) \longleftrightarrow \text{sats}(M, \text{renpbdy}(\varphi, \text{nenv}), [\varrho, p, x, \alpha, P, \text{leq}, o] \text{ @ } \text{nenv})$
unfolding renpbdy_def renpbdy_fn_def renpbdy1_fn_def
by (rule sats_iff_sats_ren , auto simp add: $\text{renpbdy1_thm}(1)$) [of $_$ $M, \text{simplified}$]
 $\text{renpbdy1_thm}(2)$ [simplified, where $\alpha=\alpha$ and $x=x$]

rename renbody1 **src** $[x, \alpha, P, \text{leq}, o]$ **tgt** $[\alpha, x, m, P, \text{leq}, o]$

definition $\text{renbody_fn} :: i \Rightarrow i$ **where**
 $\text{renbody_fn}(\text{env}) \equiv \text{sum}(\text{renbody1_fn}, \text{id}(\text{length}(\text{env})), 5, 6, \text{length}(\text{env}))$

definition
 $\text{renbody} :: [i, i] \Rightarrow i$ **where**
 $\text{renbody}(\varphi, \text{env}) = \text{ren}(\varphi) '(5 \# + \text{length}(\text{env})) '(6 \# + \text{length}(\text{env})) ' \text{renbody_fn}(\text{env})$

lemma

renbody_type [TC]: $\varphi \in \text{formula} \implies \text{env} \in \text{list}(M) \implies \text{renbody}(\varphi, \text{env}) \in \text{formula}$
unfolding *renbody_def renbody_fn_def renbody1_fn_def*
using *renbody1_thm(1) ren_tc*
by *simp*

lemma *arity_renbody*: $\varphi \in \text{formula} \implies \text{arity}(\varphi) \leq 5 \# + \text{length}(\text{env}) \implies \text{env} \in \text{list}(M)$
 \implies

$\text{arity}(\text{renbody}(\varphi, \text{env})) \leq 6 \# + \text{length}(\text{env})$
unfolding *renbody_def renbody_fn_def renbody1_fn_def*
using *renbody1_thm(1) arity_ren*
by *simp*

lemma

sats_renbody: $\text{arity}(\varphi) \leq 5 \# + \text{length}(\text{nenv}) \implies [\alpha, x, m, P, \text{leq}, o] @ \text{nenv} \in \text{list}(M) \implies \varphi \in \text{formula} \implies$
 $\text{sats}(M, \varphi, [\alpha, \alpha, P, \text{leq}, o] @ \text{nenv}) \longleftrightarrow \text{sats}(M, \text{renbody}(\varphi, \text{nenv}), [\alpha, x, m, P, \text{leq}, o] @ \text{nenv})$
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 $\alpha = \alpha$ and $m = m$, simplified])

context *G_generic*

begin

lemma *pow_inter_M*:

assumes
 $x \in M \ y \in M$
shows
 $\text{powerset}(\#\#M, x, y) \longleftrightarrow y = \text{Pow}(x) \cap M$
using *assms by auto*

schematic_goal *sats_prebody_fm_auto*:

assumes
 $\varphi \in \text{formula} [P, \text{leq}, \text{one}, p, \varrho, \pi] @ \text{nenv} \in \text{list}(M) \ \alpha \in M \ \text{arity}(\varphi) \leq 2 \# + \text{length}(\text{nenv})$
shows
 $(\exists \tau \in M. \exists V \in M. \text{is_Vset}(\#\#M, \alpha, V) \wedge \tau \in V \wedge \text{sats}(M, \text{forces}(\varphi), [p, P, \text{leq}, \text{one}, \varrho, \tau] @ \text{nenv}))$
 $\longleftrightarrow \text{sats}(M, ?\text{prebody_fm}, [\varrho, p, \alpha, P, \text{leq}, \text{one}] @ \text{nenv})$
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( $\varphi$ ) ≤  $\_$ ⟩ ⟨ $\varphi$  ∈  $\_$ ⟩
  show arity(forces( $\varphi$ )) ≤ succ(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  $\varphi$  ∈ formula
    env ∈ list(M)
  shows prebody_fm( $\varphi$ , env) ∈ formula
proof -
  from ⟨ $\varphi$  ∈ formula⟩
  have forces( $\varphi$ ) ∈ formula by simp
  then
  have renrep(forces( $\varphi$ ), 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,  $\varrho$ ] @ nenv ∈ list(M)  $\varphi$  ∈ formula  $\alpha$  ∈ M arity( $\varphi$ ) ≤ 2 #+ length(nenv)
  shows
    sats(M, prebody_fm( $\varphi$ , nenv), [ $\varrho$ , p,  $\alpha$ , P, leq, one] @ nenv)  $\longleftrightarrow$ 
    ( $\exists \tau$  ∈ M.  $\exists V$  ∈ M. is_Vset( $\#\#\mathit{M}$ ,  $\alpha$ , V)  $\wedge \tau$  ∈ V  $\wedge$  sats(M, forces( $\varphi$ ), [p, P, leq, one,  $\varrho$ ,  $\tau$ ]
    @ nenv))
  unfolding prebody_fm_def using assms sats_prebody_fm_auto by force

```

lemma *arity_prebody_fm*:

assumes

$\varphi \in \text{formula}$ $\alpha \in M$ $\text{env} \in \text{list}(M)$ $\text{arity}(\varphi) \leq 2 \ \#\ + \ \text{length}(\text{env})$

shows

$\text{arity}(\text{prebody_fm}(\varphi, \text{env})) \leq 6 \ \#\ + \ \text{length}(\text{env})$

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' :: $[i, i] \Rightarrow i$ **where**

$\text{body_fm}'(\varphi, \text{env}) \equiv \text{Exists}(\text{Exists}(\text{And}(\text{pair_fm}(0, 1, 2), \text{renpbdy}(\text{prebody_fm}(\varphi, \text{env}), \text{env}))))$

lemma *body_fm'_type* [TC]: $\varphi \in \text{formula} \Longrightarrow \text{env} \in \text{list}(M) \Longrightarrow \text{body_fm}'(\varphi, \text{env}) \in \text{formula}$

unfolding *body_fm'_def* **using** *prebody_fm_type*

by *simp*

lemma *arity_body_fm'*:

assumes

$\varphi \in \text{formula}$ $\alpha \in M$ $\text{env} \in \text{list}(M)$ $\text{arity}(\varphi) \leq 2 \ \#\ + \ \text{length}(\text{env})$

shows

$\text{arity}(\text{body_fm}'(\varphi, \text{env})) \leq 5 \ \#\ + \ \text{length}(\text{env})$

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

$\exists t p. x = \langle t, p \rangle$ $x \in M$ $[\alpha, P, \text{leq}, \text{one}, p, \varrho]$ @ $\text{nenv} \in \text{list}(M)$ $\varphi \in \text{formula}$ $\text{arity}(\varphi) \leq 2$
 $\#\ + \ \text{length}(\text{nenv})$

shows

$\text{sats}(M, \text{body_fm}'(\varphi, \text{nenv}), [x, \alpha, P, \text{leq}, \text{one}] \ @ \ \text{nenv}) \longleftrightarrow$

$\text{sats}(M, \text{renpbdy}(\text{prebody_fm}(\varphi, \text{nenv}), \text{nenv}), [\text{fst}(x), \text{snd}(x), x, \alpha, P, \text{leq}, \text{one}] \ @ \ \text{nenv})$

using *assms_fst_snd_closed*[*OF* $\langle x \in M \rangle$] **unfolding** *body_fm'_def*

by (*auto*)

definition

body_fm :: $[i, i] \Rightarrow i$ **where**

$\text{body_fm}(\varphi, \text{env}) \equiv \text{renbody}(\text{body_fm}'(\varphi, \text{env}), \text{env})$

lemma *body_fm_type* [TC]: $\text{env} \in \text{list}(M) \Longrightarrow \varphi \in \text{formula} \Longrightarrow \text{body_fm}(\varphi, \text{env}) \in \text{formula}$

unfolding *body_fm_def* **by** *simp*

lemma *sats_body_fm*:

assumes

$\exists t p. x = \langle t, p \rangle$ $[\alpha, x, m, P, \text{leq}, \text{one}] \ @ \ \text{nenv} \in \text{list}(M)$

$\varphi \in \text{formula}$ $\text{arity}(\varphi) \leq 2 \ \#\ + \ \text{length}(\text{nenv})$

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

lemma *sats_renpbdy_prebody_fm*:

assumes
 $\exists t p. x = \langle t, p \rangle \ x \in M \ [\alpha, m, P, leq, one] @ nenv \in list(M)$
 $\varphi \in formula \ arity(\varphi) \leq 2 \ \#\ + \ length(nenv)$
shows
 $sats(M, renpbdy(prebody_fm(\varphi, nenv), nenv), [fst(x), snd(x), x, \alpha, P, leq, one] @ nenv)$
 \longleftrightarrow
 $sats(M, prebody_fm(\varphi, nenv), [fst(x), snd(x), \alpha, P, leq, one] @ nenv)$
using *assms* *fst_snd_closed*[*OF* $\langle x \in M \rangle$]
sats_renpbdy[*OF* *arity_prebody_fm* *_prebody_fm_type*, *of* *concl*:*M*, *symmet-*
ric]
by *force*

lemma *body_lemma*:

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

lemma *Replace_sats_in_MG*:

assumes
 $c \in M[G] \ env \in list(M[G])$
 $\varphi \in formula \ arity(\varphi) \leq 2 \ \#\ + \ length(env)$
 $univalent(\#\#M[G], c, \lambda x v. (M[G], [x, v] @ env \models \varphi))$
shows
 $\{v. x \in c, v \in M[G] \wedge (M[G], [x, v] @ env \models \varphi)\} \in M[G]$
proof -
let $?R = \lambda x v. v \in M[G] \wedge (M[G], [x, v] @ env \models \varphi)$
from $\langle c \in M[G] \rangle$
obtain π' **where** $val(G, \pi') = c \ \pi' \in M$
using *GenExt_def* **by** *auto*
then
have $domain(\pi') \times P \in M$ (**is** $? \pi \in M$)
using *cartprod_closed* *P_in_M* *domain_closed* **by** *simp*
from $\langle val(G, \pi') = c \rangle$
have $c \subseteq val(G, ?\pi)$
using *def_val*[*of* $G \ ?\pi$] *one_in_P* *one_in_G*[*OF* *generic*] *elem_of_val*

```

    domain_of_prod[OF one_in_P, of domain( $\pi'$ )] by force
  from  $\langle env \in \_ \rangle$ 
  obtain nenv where nenv $\in$ list(M) env = map(val(G),nenv)
    using map_val by auto
  then
  have length(nenv) = length(env) by simp
  define f where f( $\varrho p$ )  $\equiv$   $\mu \alpha. \alpha \in M \wedge (\exists \tau \in M. \tau \in Vset(\alpha) \wedge$ 
    (snd( $\varrho p$ )  $\Vdash$   $\varphi$  ([fst( $\varrho p$ ), $\tau$ ] @ nenv))) (is  $\_ \equiv \mu \alpha. ?P(\varrho p, \alpha)$ ) for  $\varrho p$ 
  have f( $\varrho p$ ) = ( $\mu \alpha. \alpha \in M \wedge (\exists \tau \in M. \exists V \in M. is\_Vset(\#\#M, \alpha, V) \wedge \tau \in V \wedge$ 
    (snd( $\varrho p$ )  $\Vdash$   $\varphi$  ([fst( $\varrho p$ ), $\tau$ ] @ nenv)))) (is  $\_ = (\mu \alpha. \alpha \in M \wedge ?Q(\varrho p, \alpha))$ ) for
 $\varrho p$ 
  unfolding f_def using Vset_abs Vset_closed Ord_Least_cong[of ?P( $\varrho p$ )  $\lambda \alpha. \alpha \in M \wedge ?Q(\varrho p, \alpha)$ ]
    by (simp, simp del:setclass_iff)
  moreover
  have f( $\varrho p$ )  $\in$  M for  $\varrho p$ 
    unfolding f_def using Least_closed[of ?P( $\varrho p$ )] by simp
  ultimately
  have 1:least( $\#\#M, \lambda \alpha. ?Q(\varrho p, \alpha), f(\varrho p)$ ) for  $\varrho p$ 
    using least_abs[of  $\lambda \alpha. \alpha \in M \wedge ?Q(\varrho p, \alpha) f(\varrho p)$ ] least_conj
    by (simp flip:setclass_iff)
  have Ord(f( $\varrho p$ )) for  $\varrho p$  unfolding f_def by simp
  define QQ where QQ $\equiv$ ?Q
  from 1
  have least( $\#\#M, \lambda \alpha. QQ(\varrho p, \alpha), f(\varrho p)$ ) for  $\varrho p$ 
    unfolding QQ_def .
  from  $\langle arity(\varphi) \leq \_ \rangle \langle length(nenv) = \_ \rangle$ 
  have arity( $\varphi$ )  $\leq$  2  $\#\+$  length(nenv)
    by simp
  moreover
  note assms  $\langle nenv \in list(M) \rangle \langle ?\pi \in M \rangle$ 
  moreover
  have  $\varrho p \in ?\pi \implies \exists t p. \varrho p = \langle t, p \rangle$  for  $\varrho p$ 
    by auto
  ultimately
  have body:M , [ $\alpha, \varrho p, m, P, leq, one$ ] @ nenv  $\models$  body_fm( $\varphi, nenv$ )  $\longleftrightarrow$  ?Q( $\varrho p, \alpha$ )
    if  $\varrho p \in ?\pi \varrho p \in M m \in M \alpha \in M$  for  $\alpha \varrho p m$ 
    using that P_in_M leq_in_M one_in_M body_lemma[of  $\varrho p \alpha m nenv \varphi$ ] by
simp
  let ?f_fm=least_fm(body_fm( $\varphi, nenv$ ),1)
  {
    fix  $\varrho p m$ 
    assume asm:  $\varrho p \in M \varrho p \in ?\pi m \in M$ 
    note inM = this P_in_M leq_in_M one_in_M  $\langle nenv \in list(M) \rangle$ 
    with body
    have body': $\wedge \alpha. \alpha \in M \implies (\exists \tau \in M. \exists V \in M. is\_Vset(\lambda a. (\#\#M)(a), \alpha, V)$ 
 $\wedge \tau \in V \wedge$ 
    (snd( $\varrho p$ )  $\Vdash$   $\varphi$  ([fst( $\varrho p$ ), $\tau$ ] @ nenv)))  $\longleftrightarrow$ 
    M, Cons( $\alpha, [\varrho p, m, P, leq, one]$  @ nenv)  $\models$  body_fm( $\varphi, nenv$ ) by simp

```

```

from inM
have M , [op,m,P,leq,one] @ nenv ⊨ ?f_fm ↔ least(##M, QQ(op), m)
  using sats_least_fm[OF body', of 1] unfolding QQ_def
  by (simp, simp flip: setclass_iff)
}
then
have M , [op,m,P,leq,one] @ nenv ⊨ ?f_fm ↔ least(##M, QQ(op), m)
  if op∈M op∈?π m∈M for op m using that by simp
then
have univalent(##M, ?π, λop m. M , [op,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 ↔ (∃ op∈M. op ∈ ?π ∧ M, [op,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(op)∈Y if op∈?π for op
  using that transitivity[OF _ ⟨?π∈M⟩]
  by (clarsimp, rule_tac x=⟦x,y⟧ in bestI, 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  $\sigma$  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  $\langle \langle \varrho, p \rangle \in \pi' \rangle \langle \langle \varrho, q \rangle \in ?\pi \implies f(\langle \varrho, q \rangle) \in Y \rangle$ 
  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  $\tau$  where  $\tau \in M$   $\tau \in Vset(f(\langle \varrho, q \rangle))$   $q \Vdash \varphi ([\varrho, \tau] @ nenv)$ 

```

```

    using LeastI[of  $\lambda \alpha. ?P(\langle \varrho, q \rangle, \alpha) ?\alpha$ ] by auto
  with  $\langle q \in G \rangle \langle \varrho \in M \rangle \langle nenv \in \_ \rangle \langle \text{arity}(\varphi) \leq 2 \ \#\ + \ \text{length}(nenv) \rangle$ 
  have  $M[G], \text{map}(\text{val}(G), [\varrho, \tau] @ nenv) \models \varphi$ 
    using truth_lemma[OF  $\langle \varphi \in \_ \rangle$  generic, of  $[\varrho, \tau] @ nenv$ ] by auto
  moreover from  $\langle x \in c \rangle \langle c \in M[G] \rangle$ 
  have  $x \in M[G]$  using transitivity_MG by simp
  moreover
  note  $\langle M[G], [x, v] @ env \models \varphi \rangle \langle env = \text{map}(\text{val}(G), nenv) \rangle \langle \tau \in M \rangle \langle \text{val}(G, \varrho) = x \rangle$ 
     $\langle \text{univalent}(\#\#M[G], \_, \_) \rangle \langle x \in c \rangle \langle v \in M[G] \rangle$ 
  ultimately
  have  $v = \text{val}(G, \tau)$ 
    using GenExtI[of  $\tau \ G$ ] unfolding univalent_def by (auto)
  from  $\langle \tau \in \text{Vset}(f(\langle \varrho, q \rangle)) \rangle \langle \text{Ord}(f(\_)) \rangle \langle f(\langle \varrho, q \rangle) \in Y \rangle$ 
  have  $\tau \in \text{Vset}(?sup)$ 
    using Vset_Ord_rank_iff_lt_Union_iff[of  $\_ \ \text{rank}(\tau)$ ] by auto
  with  $\langle \tau \in M \rangle$ 
  have  $\text{val}(G, \tau) \in \text{val}(G, ?big\_name)$ 
    using domain_of_prod[of one  $\{one\}$   $\{x \in \text{Vset}(?sup). x \in M\}$ ] def_val[of  $G$ 
 $?big\_name$ ]
    one_in_G[OF generic] one_in_P by (auto simp del: Vset_rank_iff)
  with  $\langle v = \text{val}(G, \tau) \rangle$ 
  have  $v \in \text{val}(G, \{x \in \text{Vset}(?sup). x \in M\} \times \{one\})$ 
    by simp
}
then
have  $\{v. x \in c, ?R(x, v)\} \subseteq \text{val}(G, ?big\_name)$  (is  $?repl \subseteq ?big$ )
  by blast
with  $\langle ?big\_name \in M \rangle$ 
have  $?repl = \{v \in ?big. \exists x \in c. \text{sats}(M[G], \varphi, [x, v] @ env)\}$  (is  $\_ = ?rhs$ )
proof (intro equalityI subsetI)
  fix v
  assume  $v \in ?repl$ 
  with  $\langle ?repl \subseteq ?big \rangle$ 
  obtain x where  $x \in c \ M[G], [x, v] @ env \models \varphi \ v \in ?big$ 
    using subsetD by auto
  with  $\langle \text{univalent}(\#\#M[G], \_, \_) \rangle \langle c \in M[G] \rangle$ 
  show  $v \in ?rhs$ 
    unfolding univalent_def
    using transitivity_MG ReplaceI[of  $\lambda x v. \exists x \in c. M[G], [x, v] @ env \models \varphi$ ] by
blast
next
  fix v
  assume  $v \in ?rhs$ 
  then
  obtain x where
     $v \in \text{val}(G, ?big\_name) \ M[G], [x, v] @ env \models \varphi \ x \in c$ 
    by blast
  moreover from this  $\langle c \in M[G] \rangle$ 
  have  $v \in M[G] \ x \in M[G]$ 

```

```

    using transitivity_MG GenExtI[OF ‹?big_name∈_›,of G] by auto
  moreover from calculation ‹univalent(##M[G],_,_)›
  have  $?R(x,y) \implies y = v$  for  $y$ 
    unfolding univalent_def by auto
  ultimately
  show  $v \in ?repl$ 
    using ReplaceI[of ?R x v c]
    by blast
qed
moreover
let  $?\psi = \text{Exists}(\text{And}(\text{Member}(0, 2 \# + \text{length}(\text{env})), \varphi))$ 
have  $v \in M[G] \implies (\exists x \in c. M[G], [x, v] @ \text{env} \models \varphi) \longleftrightarrow M[G], [v] @ \text{env} @ [c]$ 
 $\models ?\psi$ 
  arity( $?\psi$ )  $\leq 2 \# + \text{length}(\text{env})$   $?\psi \in \text{formula}$ 
  for  $v$ 
proof -
  fix  $v$ 
  assume  $v \in M[G]$ 
  with  $\langle c \in M[G] \rangle$ 
  have  $\text{nth}(\text{length}(\text{env}) \# + 1, [v] @ \text{env} @ [c]) = c$ 
    using  $\langle \text{env} \in \_ \rangle \text{nth\_concat}[of v c M[G] \text{env}]$ 
    by auto
  note  $\text{inMG} = \langle \text{nth}(\text{length}(\text{env}) \# + 1, [v] @ \text{env} @ [c]) = c \rangle \langle c \in M[G] \rangle \langle v \in M[G] \rangle$ 
 $\langle \text{env} \in \_ \rangle$ 
  show  $(\exists x \in c. M[G], [x, v] @ \text{env} \models \varphi) \longleftrightarrow M[G], [v] @ \text{env} @ [c] \models ?\psi$ 
  proof
    assume  $\exists x \in c. M[G], [x, v] @ \text{env} \models \varphi$ 
    then obtain  $x$  where
       $x \in c$   $M[G], [x, v] @ \text{env} \models \varphi$   $x \in M[G]$ 
      using transitivity_MG[OF _ ‹c ∈ M[G]›]
      by auto
    with  $\langle \varphi \in \_ \rangle \langle \text{arity}(\varphi) \leq 2 \# + \text{length}(\text{env}) \rangle$  inMG
    show  $M[G], [v] @ \text{env} @ [c] \models \text{Exists}(\text{And}(\text{Member}(0, 2 \# + \text{length}(\text{env})),$ 
 $\varphi))$ 
      using arity_sats_iff[of  $\varphi [c] \_ [x, v] @ \text{env}$ ]
      by auto
  next
  assume  $M[G], [v] @ \text{env} @ [c] \models \text{Exists}(\text{And}(\text{Member}(0, 2 \# + \text{length}(\text{env})),$ 
 $\varphi))$ 
    with inMG
    obtain  $x$  where
       $x \in M[G]$   $x \in c$   $M[G], [x, v] @ \text{env} @ [c] \models \varphi$ 
      by auto
    with  $\langle \varphi \in \_ \rangle \langle \text{arity}(\varphi) \leq 2 \# + \text{length}(\text{env}) \rangle$  inMG
    show  $\exists x \in c. M[G], [x, v] @ \text{env} \models \varphi$ 
      using arity_sats_iff[of  $\varphi [c] \_ [x, v] @ \text{env}$ ]
      by auto
  qed
next

```

```

from ⟨env∈_⟩ ⟨φ∈_⟩
show  $\text{arity}(\varphi) \leq 2 \# + \text{length}(\text{env})$ 
  using  $\text{pred\_mono}[OF\_ \langle \text{arity}(\varphi) \leq 2 \# + \text{length}(\text{env}) \rangle]$   $\text{lt\_trans}[OF\_ \text{le\_refl}]$ 
  by (auto simp add:nat_simp_union)
next
  from ⟨φ∈_⟩
  show  $\varphi \in \text{formula}$  by simp
qed
moreover from this
have  $\{v \in ?big. \exists x \in c. M[G], [x, v] @ \text{env} \models \varphi\} = \{v \in ?big. M[G], [v] @ \text{env} @ [c]$ 
 $\models \varphi\}$ 
  using  $\text{transitivity\_MG}[OF\_ \text{GenExtI}, OF\_ \langle ?big\_name \in M \rangle]$ 
  by simp
moreover from calculation and ⟨env∈_⟩ ⟨c∈_⟩ ⟨?big∈M[G]⟩
have  $\{v \in ?big. M[G], [v] @ \text{env} @ [c] \models \varphi\} \in M[G]$ 
  using  $\text{Collect\_sats\_in\_MG}$  by auto
ultimately
show  $\varphi$  thesis by simp
qed

theorem strong_replacement_in_MG:
assumes
   $\varphi \in \text{formula}$  and  $\text{arity}(\varphi) \leq 2 \# + \text{length}(\text{env})$   $\text{env} \in \text{list}(M[G])$ 
shows
   $\text{strong\_replacement}(\#\#M[G], \lambda x v. \text{sats}(M[G], \varphi, [x, v] @ \text{env}))$ 
proof -
let  $?R = \lambda x y. M[G], [x, y] @ \text{env} \models \varphi$ 
{
  fix A
  let  $?Y = \{v. x \in A, v \in M[G] \wedge ?R(x, v)\}$ 
  assume 1:  $(\#\#M[G])(A)$ 
   $\forall x[\#\#M[G]]. x \in A \longrightarrow (\forall y[\#\#M[G]]. \forall z[\#\#M[G]]. ?R(x, y) \wedge ?R(x, z)$ 
 $\longrightarrow y = z)$ 
  then
  have  $\text{univalent}(\#\#M[G], A, ?R)$   $A \in M[G]$ 
  unfolding  $\text{univalent\_def}$  by simp_all
  with  $\text{assms}$  ⟨A∈_⟩
  have  $(\#\#M[G])(?Y)$ 
  using  $\text{Replace\_sats\_in\_MG}$  by auto
  have  $b \in ?Y \longleftrightarrow (\exists x[\#\#M[G]]. x \in A \wedge ?R(x, b))$  if  $(\#\#M[G])(b)$  for b
  proof(rule)
  from ⟨A∈_⟩
  show  $\exists x[\#\#M[G]]. x \in A \wedge ?R(x, b)$  if  $b \in ?Y$ 
  using  $\text{that transitivity\_MG}$  by auto
  next
  show  $b \in ?Y$  if  $\exists x[\#\#M[G]]. x \in A \wedge ?R(x, b)$ 
  proof -
  from  $(\#\#M[G])(b)$ 
  have  $b \in M[G]$  by simp

```

```

with that
obtain  $x$  where  $(\#\#M[G])(x) \ x \in A \ b \in M[G] \wedge ?R(x,b)$ 
  by blast
moreover from this 1  $\langle (\#\#M[G])(b) \rangle$ 
have  $x \in M[G] \ z \in M[G] \wedge ?R(x,z) \implies b = z$  for  $z$ 
  by auto
ultimately
show ?thesis
  using ReplaceI[of  $\lambda \ x \ y. \ y \in M[G] \wedge ?R(x,y)$ ] 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  $\langle (\#\#M[G])(?Y) \rangle$ 
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-1a⟩
  have restrict(f, f-1a) 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-1a⟩
  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-1a ⇒ induced_surj(f, a, e) x = f x
proof -
  let ?f1 = f-1((range(f) - a) × {e}) and ?f2 = restrict(f, f-1a)
  have domain(?f2) = domain(f) ∩ f-1a
    using domain_restrict by simp
  moreover from assms
  have 1: domain(?f1) = f-1((range(f)) - f-1a)
    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-1a)] 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  $\text{domain}(?f1) \cup \text{domain}(?f2) = \text{domain}(f)$ 
  using  $\text{domain\_restrict domain\_of\_prod}$  by auto
ultimately
show  $\text{induced\_surj}(f,a,e): \text{domain}(f) \rightarrow \{e\} \cup a$ 
  by simp
assume  $x \in f^{-1}a$ 
then
have  $?f2'x = f'x$ 
  using restrict by simp
moreover from  $\langle x \in f^{-1}a \rangle$  and 1
have  $x \notin \text{domain}(?f1)$ 
  by simp
ultimately
show  $\text{induced\_surj}(f,a,e)'x = f'x$ 
  unfolding induced\_surj\_def using fun\_disjoint\_apply2[ $of\ x\ ?f1\ ?f2$ ] by simp
qed

```

```

lemma induced\_surj\_is\_surj :
assumes
   $e \in a$   $\text{function}(f)$   $\text{domain}(f) = \alpha$   $\bigwedge y. y \in a \implies \exists x \in \alpha. f'x = y$ 
shows
   $\text{induced\_surj}(f,a,e) \in \text{surj}(\alpha,a)$ 
unfolding surj\_def
proof (intro CollectI ballI)
from  $\text{assms}$ 
show  $\text{induced\_surj}(f,a,e): \alpha \rightarrow a$ 
  using induced\_surj\_type[ $of\ f\ a\ e$ ]  $\text{cons\_eq cons\_absorb}$  by simp
fix  $y$ 
assume  $y \in a$ 
with  $\text{assms}$ 
have  $\exists x \in \alpha. f'x = y$ 
  by simp
then
obtain  $x$  where  $x \in \alpha$   $f'x = y$  by auto
with  $\langle y \in a \rangle$   $\text{assms}$ 
have  $x \in f^{-1}a$ 
  using vimage\_iff function\_apply\_Pair[ $of\ f\ x$ ] by auto
with  $\langle f'x = y \rangle$   $\text{assms}$ 
have  $\text{induced\_surj}(f, a, e)'x = y$ 
  using induced\_surj\_type by simp
with  $\langle x \in \alpha \rangle$  show
   $\exists x \in \alpha. \text{induced\_surj}(f, a, e)'x = y$  by auto
qed

```

```

context  $G\_generic$ 
begin

```

```

definition

```

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

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 $upair_name_closed :$

$\llbracket x \in M; y \in M \rrbracket \Longrightarrow upair_name(x, y) \in M$
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 $upair_name_fm_type[TC] :$

$\llbracket s \in nat; x \in nat; y \in nat; o \in nat \rrbracket \Longrightarrow upair_name_fm(s, x, y, o) \in formula$
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**
 $opair_name(\tau, \rho) \equiv upair_name(upair_name(\tau, \tau), upair_name(\tau, \rho))$

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* :

$\llbracket x \in M; y \in M \rrbracket \implies \text{opair_name}(x,y) \in M$

unfolding *opair_name_def* **using** *upair_name_closed* **by** *simp*

definition

opair_name_fm :: $[i,i,i,i] \Rightarrow i$ **where**

$\text{opair_name_fm}(x,y,o,z) \equiv \text{Exists}(\text{Exists}(\text{And}(\text{upair_name_fm}(x\#+2,x\#+2,o\#+2,1),$

$\text{And}(\text{upair_name_fm}(x\#+2,y\#+2,o\#+2,0),\text{upair_name_fm}(1,0,o\#+2,z\#+2))))))$

lemma *opair_name_fm_type*[*TC*] :

$\llbracket s \in \text{nat}; x \in \text{nat}; y \in \text{nat}; o \in \text{nat} \rrbracket \implies \text{opair_name_fm}(s,x,y,o) \in \text{formula}$

unfolding *opair_name_fm_def* **by** *simp*

lemma *sats_opair_name_fm* :

assumes $x \in \text{nat } y \in \text{nat } z \in \text{nat } o \in \text{nat } \text{env} \in \text{list}(M) \text{nth}(o,\text{env}) = \text{one}$

shows

$\text{sats}(M, \text{opair_name_fm}(x,y,o,z), \text{env}) \longleftrightarrow \text{is_opair_name}(\text{nth}(x,\text{env}), \text{nth}(y,\text{env}), \text{nth}(z,\text{env}))$

unfolding *opair_name_fm_def* *is_opair_name_def* **using** *assms* *sats_upair_name_fm*

by *simp*

lemma *val_upair_name* : $\text{val}(G, \text{upair_name}(\tau, \rho)) = \{\text{val}(G, \tau), \text{val}(G, \rho)\}$

unfolding *upair_name_def* **using** *val_Upair_generic_one_in_G_one_in_P* **by** *simp*

lemma *val_opair_name* : $\text{val}(G, \text{opair_name}(\tau, \rho)) = \langle \text{val}(G, \tau), \text{val}(G, \rho) \rangle$

unfolding *opair_name_def* *Pair_def* **using** *val_upair_name* **by** *simp*

lemma *val_RepFun_one*: $\text{val}(G, \{\langle f(x), \text{one} \rangle . x \in a\}) = \{\text{val}(G, f(x)) . x \in a\}$

proof -

let $?A = \{f(x) . x \in a\}$

let $?Q = \lambda \langle x, p \rangle . p = \text{one}$

have $\text{one} \in P \cap G$ **using** *generic_one_in_G_one_in_P* **by** *simp*

have $\{\langle f(x), \text{one} \rangle . x \in a\} = \{t \in ?A \times P . ?Q(t)\}$

using *one_in_P* **by** *force*

then

have $\text{val}(G, \{\langle f(x), \text{one} \rangle . x \in a\}) = \text{val}(G, \{t \in ?A \times P . ?Q(t)\})$

by *simp*

also

have $\dots = \{\text{val}(G, t) .. t \in ?A , \exists p \in P \cap G . ?Q(\langle t, p \rangle)\}$

using *val_of_name_alt* **by** *simp*

also

have $\dots = \{\text{val}(G, t) . t \in ?A\}$

using $\langle \text{one} \in P \cap G \rangle$ **by** *force*

also

have $\dots = \{\text{val}(G, f(x)) . x \in a\}$

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]* :

$\llbracket s \in nat; x \in nat; y \in nat; o \in nat \rrbracket \Longrightarrow opname_check_fm(s,x,y,o) \in formula$
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  $x \in A \implies \text{opair\_name}(\text{check}(x), f \cdot x) \in M$  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)
     $\lambda x. \text{opair\_name}(\text{check}(x), f \cdot x)$ ]
  by simp
qed

```

```

theorem choice_in_MG:
  assumes choice_ax(##M)
  shows choice_ax(##M[G])
proof -
  {
  fix a
  assume  $a \in M[G]$ 
  then
  obtain  $\tau$  where  $\tau \in M$   $\text{val}(G, \tau) = a$ 
    using GenExt_def by auto
  with  $\langle \tau \in M \rangle$ 
  have  $\text{domain}(\tau) \in M$ 
    using domain_closed by simp
  then
  obtain  $s \ \alpha$  where  $s \in \text{surj}(\alpha, \text{domain}(\tau))$   $\text{Ord}(\alpha)$   $s \in M$   $\alpha \in M$ 
    using assms choice_ax_abs by auto
  then
  have  $\alpha \in M[G]$ 
    using M_subset_MG generic_one_in_G subsetD by blast
  let ?A =  $\text{domain}(\tau) \times P$ 
  let ?g =  $\{\text{opair\_name}(\text{check}(\beta), s \cdot \beta). \beta \in \alpha\}$ 
  have ?g  $\in M$  using  $\langle s \in M \rangle$   $\langle \alpha \in M \rangle$  repl_opname_check by simp
  let ?f_dot =  $\{\langle \text{opair\_name}(\text{check}(\beta), s \cdot \beta), \text{one} \rangle. \beta \in \alpha\}$ 
  have ?f_dot = ?g  $\times$  {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  $\langle \{one\} \in M \rangle \langle ?g \in M \rangle \langle ?f\_dot = ?g \times \{one\} \rangle$ 
have  $?f\_dot \in M$ 
  using cartprod_closed by simp
then
have  $f \in M[G]$ 
  unfolding f_def by (blast intro: GenExtI)
have  $f = \{val(G, opair\_name(check(\beta), s'\beta)) . \beta \in \alpha\}$ 
  unfolding f_def using val_RepFun_one by simp
also
have  $\dots = \{\langle \beta, val(G, s'\beta) \rangle . \beta \in \alpha\}$ 
  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  $2: y \in a \implies \exists x \in \alpha. f' x = y$  for y
proof -
  fix y
  assume
     $y \in a$ 
  with  $\langle val(G, \tau) = a \rangle$ 
  obtain  $\sigma$  where  $\sigma \in domain(\tau)$   $val(G, \sigma) = y$ 
    using elem_of_val[of y _  $\tau$ ] by blast
  with  $\langle s \in surj(\alpha, domain(\tau)) \rangle$ 
  obtain  $\beta$  where  $\beta \in \alpha$   $s'\beta = \sigma$ 
    unfolding surj_def by auto
  with  $\langle val(G, \sigma) = y \rangle$ 
  have  $val(G, s'\beta) = y$ 
    by simp
  with  $\langle f = \{\langle \beta, val(G, s'\beta) \rangle . \beta \in \alpha\} \rangle \langle \beta \in \alpha \rangle$ 
  have  $\langle \beta, y \rangle \in f$ 
    by auto
  with  $\langle function(f) \rangle$ 
  have  $f'\beta = y$ 
    using function_apply_equality by simp
  with  $\langle \beta \in \alpha \rangle$  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 ( $\bigcup u \in \{t \in \text{domain}(x). \exists p \in P. \langle t, p \rangle \in x \wedge p \in G\}. \text{succ}(\text{rank}(\text{val}(G, u))) \leq$ 
 $\text{rank}(x)$ )
    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  $\alpha \in M \implies \alpha \in M[G]$ 
    using generic[THEN one_in_G, THEN M_subset_MG] ..
next
  assume  $\alpha \in M[G]$ 
  then
  obtain  $x$  where  $x \in M$   $\text{val}(G, x) = \alpha$ 
    using GenExtD by auto
  then
  have  $\text{rank}(\alpha) \leq \text{rank}(x)$ 
    using rank_val by blast
  with assms
  have  $\alpha \leq \text{rank}(x)$ 
    using rank_of_Ord by simp
  then
  have  $\alpha \in \text{succ}(\text{rank}(x))$  using ltD by simp
  with  $\langle x \in M \rangle$ 
  show  $\alpha \in M$ 
    using cons_closed_transitivity[of  $\alpha$   $\text{succ}(\text{rank}(x))$ ]
    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*: $p \in P \implies \exists q \in P. \exists r \in P. q \preceq p \wedge r \preceq p \wedge q \perp r$
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 $\langle p \in P \rangle$ *assms*
moreover
obtain $q r$ **where** $q \preceq p r \preceq p q \perp r q \in P r \in P$
using *separative*[*OF* $\langle p \in P \rangle$]
by *force*
with $\langle \text{filter}(G) \rangle$
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 $\langle p \in P \rangle$
show *?thesis* **using** *leq_refl* **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 \implies 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 successions

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 finite binary sequences.

definition
seqspace $:: i \Rightarrow i \langle _ \hat{<} \omega \rangle$ [100]100) **where**
seqspace(B) $\equiv \bigcup_{n \in \text{nat.}} (n \rightarrow B)$

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

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

lemma *seqspace_type*:
 $f \in B^{<\omega} \implies \exists n \in \text{nat. } f: n \rightarrow B$
unfolding *seqspace_def* **by** *auto*

schematic_goal *seqspace_fm_auto*:
assumes
 $\text{nth}(i, \text{env}) = n$ $\text{nth}(j, \text{env}) = z$ $\text{nth}(h, \text{env}) = B$
 $i \in \text{nat}$ $j \in \text{nat}$ $h \in \text{nat}$ $\text{env} \in \text{list}(A)$
shows
 $(\exists om \in A. \text{omega}(\#\#A, om) \wedge n \in om \wedge \text{is_funspace}(\#\#A, n, B, z)) \longleftrightarrow (A, \text{env} \models (?sqsprp(i, j, h)))$
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:  $M(B) \implies \text{strong\_replacement}(M, \lambda n z. n \in \text{nat} \wedge \text{is\_funspace}(M, n, B, z))$ 
begin

lemma seqspace_closed:
   $M(B) \implies M(B^{\omega})$ 
  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) 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,  $\lambda x y. M, [x, y, B] \models \text{seqspace\_rep\_fm}(0, 1, 2)$ )
  using replacement_ax[of seqspace_rep_fm(0,1,2)]
  by simp
  moreover
  note  $\langle B \in M \rangle$ 
  moreover from this
  have univalent(##M, A,  $\lambda x y. M, [x, y, B] \models \text{seqspace\_rep\_fm}(0, 1, 2)$ )
  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,  $\lambda n z. \exists om[##M]. \text{omega}(##M, om) \wedge n \in$ 
   $om \wedge \text{is\_funspace}(##M, n, B, z)$ )
  using seqspace_fm_auto[of 0 [_,_,B] _ 1 _ 2 B M] unfolding seqspace_rep_fm_def
  strong_replacement_def
  by simp
  with  $\langle B \in M \rangle$ 
  show strong_replacement(##M,  $\lambda n z. n \in \text{nat} \wedge \text{is\_funspace}(##M, n, B, z)$ )
  using M_nat by simp
qed

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

```

$seq_upd(f,a) \equiv \lambda j \in succ(domain(f)) . \text{if } j < domain(f) \text{ then } f'j \text{ else } a$

```

lemma seq_upd_succ_type :
  assumes  $n \in nat$   $f \in n \rightarrow A$   $a \in A$ 
  shows  $seq\_upd(f,a) \in succ(n) \rightarrow A$ 
proof -
  from assms
  have equ:  $domain(f) = n$  using domain_of_fun by simp
  {
    fix  $j$ 
    assume  $j \in succ(domain(f))$ 
    with equ  $\langle n \in \_ \rangle$ 
    have  $j \leq n$  using ltI by auto
    with  $\langle n \in \_ \rangle$ 
    consider (lt)  $j < n$  | (eq)  $j = n$  using leD by auto
    then
    have ( $\text{if } j < n \text{ then } f'j \text{ else } a$ )  $\in A$ 
    proof cases
      case lt
      with  $\langle f \in \_ \rangle$ 
      show ?thesis using apply_type ltD[OF lt] by simp
    next
      case eq
      with  $\langle a \in \_ \rangle$ 
      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  $f \in A^{<\omega}$   $a \in A$ 
  shows  $seq\_upd(f,a) \in A^{<\omega}$ 
proof -
  from  $\langle f \in \_ \rangle$ 
  obtain  $y$  where  $y \in nat$   $f \in y \rightarrow A$ 
    unfolding seqspace_def by blast
  with  $\langle a \in A \rangle$ 
  have  $seq\_upd(f,a) \in succ(y) \rightarrow A$ 
    using seq_upd_succ_type by simp
  with  $\langle y \in \_ \rangle$ 
  show ?thesis
    unfolding seqspace_def by auto
qed

```

```

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

lemma zero_in_seqspace :
  shows  $0 \in A^{\omega}$ 
  unfolding seqspace_def
  by force

definition
  seqleR ::  $i \Rightarrow i \Rightarrow o$  where
  seqleR(f,g)  $\equiv g \subseteq f$ 

definition
  seqlerel ::  $i \Rightarrow i$  where
  seqlerel(A)  $\equiv Rrel(\lambda x y. y \subseteq x, A^{\omega})$ 

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

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

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

lemma upd_leI :
  assumes  $f \in 2^{\omega}$   $a \in 2$ 
  shows  $\langle \text{seq\_upd}(f,a), f \rangle \in \text{seqle}$  (is  $\langle ?f, \_ \rangle \in \_$ )
proof
  show  $\langle ?f, f \rangle \in 2^{\omega} \times 2^{\omega}$ 
  using assms seq_upd_type by auto
next
  show  $f \subseteq \text{seq\_upd}(f,a)$ 
proof
  fix  $x$ 
  assume  $x \in f$ 
  moreover from  $\langle f \in 2^{\omega} \rangle$ 
  obtain  $n$  where  $n \in \text{nat}$   $f : n \rightarrow 2$ 
  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 $is_seqleR :: [i \Rightarrow o, i, i] \Rightarrow o$ **where**

$is_seqleR(Q, f, g) \equiv g \subseteq f$

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

$seqleR_fm(fg) \equiv \text{Exists}(\text{Exists}(\text{And}(\text{pair_fm}(0, 1, fg\# + 2), \text{subset_fm}(1, 0))))$

lemma $type_seqleR_fm :$

$fg \in nat \Longrightarrow seqleR_fm(fg) \in formula$

unfolding $seqleR_fm_def$

by $simp$

lemma $arity_seqleR_fm :$

$fg \in nat \Longrightarrow arity(seqleR_fm(fg)) = succ(fg)$

unfolding $seqleR_fm_def$

using $arity_pair_fm$ $arity_subset_fm$ nat_simp_union **by** $simp$

lemma (**in** M_basic) $seqleR_abs :$

assumes $M(f)$ $M(g)$

shows $seqleR(f, g) \longleftrightarrow is_seqleR(M, f, g)$

unfolding $seqleR_def$ is_seqleR_def

using $assms$ $apply_abs$ $domain_abs$ $domain_closed[OF \langle M(f) \rangle]$ $domain_closed[OF \langle M(g) \rangle]$

by $auto$

definition

$relP :: [i \Rightarrow o, [i \Rightarrow o, i, i] \Rightarrow o, i] \Rightarrow o$ **where**

$relP(M, r, xy) \equiv (\exists x[M]. \exists y[M]. \text{pair}(M, x, y, xy) \wedge r(M, x, y))$

lemma (**in** M_ctm) $seqleR_fm_sats :$

assumes $fg \in nat$ $env \in list(M)$

shows $sats(M, seqleR_fm(fg), env) \longleftrightarrow relP(\#\#M, is_seqleR, nth(fg, env))$

unfolding $seqleR_fm_def$ is_seqleR_def $relP_def$

using $assms$ $trans_M$ $sats_subset_fm$ $pair_iff_sats$

by $auto$

lemma (**in** M_basic) $is_related_abs :$

assumes $\bigwedge f g . M(f) \Longrightarrow M(g) \Longrightarrow rel(f, g) \longleftrightarrow is_rel(M, f, g)$

shows $\bigwedge z . M(z) \Longrightarrow relP(M, is_rel, z) \longleftrightarrow (\exists x y . z = \langle x, y \rangle \wedge rel(x, y))$

unfolding $relP_def$ **using** $pair_in_M_iff$ $assms$ **by** $auto$

definition

$is_RRel :: [i \Rightarrow o, [i \Rightarrow o, i, i] \Rightarrow o, i, i] \Rightarrow o$ **where**

$is_RRel(M, is_r, A, r) \equiv \exists A2[M]. \text{cartprod}(M, A, A, A2) \wedge is_Collect(M, A2, relP(M, is_r), r)$

lemma (**in** M_basic) $is_Rrel_abs :$

assumes $M(A)$ $M(r)$

$\bigwedge f g . M(f) \implies M(g) \implies \text{rel}(f,g) \longleftrightarrow \text{is_rel}(M,f,g)$
shows $\text{is_RRel}(M,\text{is_rel},A,r) \longleftrightarrow r = \text{Rrel}(\text{rel},A)$
proof -
from $\langle M(A) \rangle$
have $M(z)$ **if** $z \in A \times A$ **for** z
using $\text{cartprod_closed transM}$ [of $z A \times A$] **that by simp**
then
have $A : \text{relP}(M, \text{is_rel}, z) \longleftrightarrow (\exists x y. z = \langle x, y \rangle \wedge \text{rel}(x, y))$ $M(z)$ **if** $z \in A \times A$
for z
using $\text{that is_related_abs}$ [of $\text{rel is_rel}, OF \text{ assms}(3)$] **by auto**
then
have $\text{Collect}(A \times A, \text{relP}(M, \text{is_rel})) = \text{Collect}(A \times A, \lambda z. (\exists x y. z = \langle x, y \rangle \wedge \text{rel}(x, y)))$
using Collect_cong [of $A \times A A \times A \text{relP}(M, \text{is_rel}), OF _ A(1)$] $\text{assms}(1) \text{ assms}(2)$
by auto
with assms
show $?thesis$ **unfolding** $\text{is_RRel_def Rrel_def}$ **using** cartprod_closed
by auto
qed

definition

$\text{is_seqlel} :: [i \Rightarrow o, i, i] \Rightarrow o$ **where**
 $\text{is_seqlel}(M, A, r) \equiv \text{is_RRel}(M, \text{is_seqleR}, A, r)$

lemma (in M_basic) $\text{seqlel_abs} :$

assumes $M(A) M(r)$
shows $\text{is_seqlel}(M, A, r) \longleftrightarrow r = \text{Rrel}(\text{seqleR}, A)$
unfolding is_seqlel_def
using is_Rrel_abs [OF $\langle M(A) \rangle \langle M(r) \rangle$, of seqleR is_seqleR] seqleR_abs
by auto

definition $\text{RrelP} :: [i \Rightarrow i \Rightarrow o, i] \Rightarrow i$ **where**

$\text{RrelP}(R, A) \equiv \{z \in A \times A. \exists x y. z = \langle x, y \rangle \wedge R(x, y)\}$

lemma $\text{Rrel_eq} : \text{RrelP}(R, A) = \text{Rrel}(R, A)$

unfolding $\text{Rrel_def RrelP_def}$ **by auto**

context M_ctm

begin

lemma $\text{Rrel_closed} :$

assumes $A \in M$
 $\bigwedge a. a \in \text{nat} \implies \text{rel_fm}(a) \in \text{formula}$
 $\bigwedge f g . (\#\#M)(f) \implies (\#\#M)(g) \implies \text{rel}(f,g) \longleftrightarrow \text{is_rel}(\#\#M, f, g)$
 $\text{arity}(\text{rel_fm}(0)) = 1$
 $\bigwedge a . a \in M \implies \text{sats}(M, \text{rel_fm}(0), [a]) \longleftrightarrow \text{relP}(\#\#M, \text{is_rel}, a)$
shows $(\#\#M)(\text{Rrel}(\text{rel}, A))$

proof -

have $z \in M \implies \text{relP}(\#\#M, \text{is_rel}, z) \longleftrightarrow (\exists x y. z = \langle x, y \rangle \wedge \text{rel}(x, y))$ **for** z
using $\text{assms}(3) \text{ is_related_abs}$ [of rel is_rel]

```

    by auto
  with assms
  have  $\text{Collect}(A \times A, \lambda z. (\exists x y. z = \langle x, y \rangle \wedge \text{rel}(x, y))) \in M$ 
    using  $\text{Collect\_in\_M\_Op}[\text{of rel\_fm}(0) \lambda A z . \text{relP}(A, \text{is\_rel}, z) \lambda z. \exists x y. z = \langle x, y \rangle \wedge \text{rel}(x, y) ]$ 
      cartprod\_closed
    by simp
  then show ?thesis
    unfolding Rrel\_def by simp
qed

```

```

lemma seqle_in_M:  $\text{seqle} \in M$ 
  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  $?q \in 2^{<\omega} \quad ?r \in 2^{<\omega}$ 
    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 beI)+ — 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  $\text{seqle} \in M$  using seqle_in_M .
qed

```

```

lemma cohen_extension_is_proper:  $\exists G. M\_generic(G) \wedge M \neq \text{GenExt}(G)$ 
  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  $\langle$ well_ord( $A, r$ ) $\rangle$ 
have wf[ $A$ ]( $r$ )
  using well_ord_is_wf[OF  $\langle$ well_ord( $A, r$ ) $\rangle$ ] by simp
with  $\langle B \subseteq A \rangle$ 
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  $\langle B \neq 0 \rangle$ 
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  $\langle$ well_ord( $A, r$ ) $\rangle$   $B$   $\langle B \subseteq A \rangle$ 
  have  $\langle z, y \rangle \in r \mid \langle y, z \rangle \in r \mid y = z$ 
    unfolding well_ord_def tot_ord_def linear_def by auto
  with  $B$   $\langle y \in B \rangle$   $\langle y \neq z \rangle$ 
  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 :
   $\exists f. f \in \text{surj}(\text{nat}, M[G])$ 
proof -
  let ?f =  $\lambda n \in \text{nat}. \text{val}(G, \text{enum}'n)$ 
  have  $x \in \text{nat} \implies \text{val}(G, \text{enum}'x) \in M[G]$  for  $x$ 
    using GenExtD[THEN iffD2, of _ G] bij_is_fun[OF M_countable] by force
  then
  have ?f:  $\text{nat} \rightarrow M[G]$ 
    using lam_type[of nat  $\lambda n. \text{val}(G, \text{enum}'n)$   $\lambda_. M[G]$ ] by simp
  moreover
  have  $\exists n \in \text{nat}. ?f'n = x$  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:  $M[G] \approx \text{nat}$ 
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  $f \in \text{surj}(\text{nat}, M[G])$ 
    using surj_nat_MG by blast
  then
  have  $M[G] \lesssim \text{nat}$ 
    using well_ord_surj_imp_lepoll well_ord_Memrel[of nat]
    by simp
  moreover
  have  $\text{nat} \lesssim M[G]$ 
    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

```

```

     $M \approx \text{nat Transset}(M) \quad M \models ZF$ 
shows
     $\exists N.$ 
     $M \subseteq N \wedge N \approx \text{nat} \wedge \text{Transset}(N) \wedge N \models ZF \wedge M \neq N \wedge$ 
     $(\forall \alpha. \text{Ord}(\alpha) \longrightarrow (\alpha \in M \longleftrightarrow \alpha \in N)) \wedge$ 
     $(M, \models AC \longrightarrow N \models ZFC)$ 
proof -
  from  $\langle M \approx \text{nat} \rangle$ 
  obtain enum where  $\text{enum} \in \text{bij}(\text{nat}, M)$ 
  using eqpoll_sym unfolding eqpoll_def by blast
  with assms
  interpret  $M\_ctm \ M \ \text{enum}$ 
  using  $M\_ZF\_iff \ M\_satT$ 
  by intro locales (simp_all add: M_ctm_axioms_def)
  interpret  $\text{ctm\_separative } 2^{\lt \omega} \ \text{seqle } 0 \ M \ \text{enum}$ 
  proof (unfold_locales)
    fix  $f$ 
    let  $?q = \text{seq\_upd}(f, 0)$  and  $?r = \text{seq\_upd}(f, 1)$ 
    assume  $f \in 2^{\lt \omega}$ 
    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  $?q \in 2^{\lt \omega} \quad ?r \in 2^{\lt \omega}$ 
    using seq_upd_type[of f 2] by auto
    ultimately
    show  $\exists q \in 2^{\lt \omega}. \exists r \in 2^{\lt \omega}. q \preceq_s f \wedge r \preceq_s f \wedge q \perp_s r$ 
    by (rule_tac bexI) + — why the heck auto-tools don't solve this?
  next
    show  $2^{\lt \omega} \in M$  using nat_into_M seqspace_closed by simp
  next
    show  $\text{seqle} \in M$  using seqle_in_M .
  qed
  from cohen_extension_is_proper
  obtain  $G$  where  $M\_generic(G)$ 
     $M \neq \text{GenExt}(G)$  (is  $M \neq ?N$ )
    by blast
  then
  interpret  $G\_generic \ 2^{\lt \omega} \ \text{seqle } 0 \ \_ \ \text{enum } G$  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  $?N \models ZF$ 
    using  $M\_ZF\_iff \ M\_satT$  [of ?N]  $MG.M\_ZF\_axioms$  by simp
  moreover
  have  $M, \models AC \implies ?N \models ZFC$ 

```

```

proof -
  assume  $M, [] \models AC$ 
  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  $\langle ?N \models ZF \rangle$ 
  show  $?N \models ZFC$ 
    using  $ZF\_choice\_auto$   $sats\_ZFC\_iff\_sats\_ZF\_AC$ 
    unfolding  $ZF\_choice\_fm\_def$  by simp
qed
moreover
note  $\langle M \neq ?N \rangle$ 
moreover
have  $Transset(?N)$  using  $Transset\_MG$  .
moreover
have  $M \subseteq ?N$  using  $M\_subset\_MG[OF\ one\_in\_G]$  generic by simp
ultimately
show  $?thesis$ 
  using  $Ord\_MG\_iff\ MG\_eqpoll\_nat$ 
  by (rule_tac  $x=?N$  in exI, simp)
qed

end

```

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).