Formalization of Forcing in Isabelle/ZF

Emmanuel Gunther[*] Miguel Pagano[*] Pedro Sánchez Terraf[*†]

March 17, 2025

Abstract

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

Contents

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

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

1 Introduction

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

Release notes

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

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

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

2 Forcing notions

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

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

2.1 Basic concepts

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

definition *compat_in* :: $i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow o$ **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* :
  $\llbracket\ d{\in}A\ ;\ \langle d,p\rangle{\in}r\ ;\ \langle d,g\rangle{\in}r\ \rrbracket \Longrightarrow compat\_in(A,r,p,g)$
  $\langle proof\rangle$

lemma *refl_compat*:
  $\llbracket\ refl(A,r)\ ;\ \langle p,q\rangle \in r\ |\ p{=}q\ |\ \langle q,p\rangle \in r\ ;\ p{\in}A\ ;\ q{\in}A\rrbracket \Longrightarrow compat\_in(A,r,p,q)$
  $\langle proof\rangle$

lemma *chain_compat*:
  $refl(A,r) \Longrightarrow linear(A,r) \Longrightarrow\ (\forall p{\in}A.\forall q{\in}A.\ compat\_in(A,r,p,q))$
  $\langle proof\rangle$

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

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

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

definition
  ccc :: $i \Rightarrow i \Rightarrow o$ **where**
  $ccc(P,leq) \equiv \forall A.\ antichain(P,leq,A) \longrightarrow |A| \le nat$

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

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

lemma *refl_leq*:
  $r{\in}P \Longrightarrow r{\preceq}r$
  $\langle proof\rangle$

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

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

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

definition

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

lemma P_dense: $dense(P)$
$\langle proof \rangle$

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

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

lemma leq_transD: $a \preceq b \Longrightarrow b \preceq c \Longrightarrow a \in P \Longrightarrow b \in P \Longrightarrow c \in P \Longrightarrow a \preceq c$
$\langle proof \rangle$

lemma leq_transD': $A \subseteq P \Longrightarrow a \preceq b \Longrightarrow b \preceq c \Longrightarrow a \in A \Longrightarrow b \in P \Longrightarrow c \in P \Longrightarrow a \preceq c$
$\langle proof \rangle$

lemma leq_reflI: $p \in P \Longrightarrow p \preceq p$
$\langle proof \rangle$

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

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

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

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

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

lemma $dense\_belowD$ $[dest]$:
assumes $dense\_below(D,p)$ $q \in P$ $q \preceq p$
shows $\exists d \in D.\ d \in P \wedge d \preceq q$
$\langle proof \rangle$

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 $dense\_below(D,p)$

$\langle proof \rangle$

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

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

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

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

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

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

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

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

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

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

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

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

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

shows $\exists\, r{\in}G.\ r{\preceq}p \wedge r{\preceq}q$
$\langle proof \rangle$

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

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

lemma *upclosureI* [*intro*] : $p{\in}P \Longrightarrow a{\in}A \Longrightarrow a{\preceq}p \Longrightarrow p{\in}upclosure(A)$
$\langle proof \rangle$

lemma *upclosureE* [*elim*] :
$p{\in}upclosure(A) \Longrightarrow (\bigwedge x\ a.\ x{\in}P \Longrightarrow a{\in}A \Longrightarrow a{\preceq}x \Longrightarrow R) \Longrightarrow R$
$\langle proof \rangle$

lemma *upclosureD* [*dest*] :
$p{\in}upclosure(A) \Longrightarrow \exists\, a{\in}A.(a{\preceq}p) \wedge p{\in}P$
$\langle proof \rangle$

lemma *upclosure_increasing* :
assumes $A{\subseteq}P$
shows $increasing(upclosure(A))$
$\langle proof \rangle$

lemma *upclosure_in_P*: $A \subseteq P \Longrightarrow upclosure(A) \subseteq P$
$\langle proof \rangle$

lemma *A_sub_upclosure*: $A \subseteq P \Longrightarrow A{\subseteq}upclosure(A)$
$\langle proof \rangle$

lemma *elem_upclosure*: $A{\subseteq}P \Longrightarrow x{\in}A \Longrightarrow x{\in}upclosure(A)$
$\langle proof \rangle$

lemma *closure_compat_filter*:
assumes $A{\subseteq}P$ $(\forall\, p{\in}A.\forall\, q{\in}A.\ compat\_in(A,leq,p,q))$
shows $filter(upclosure(A))$
$\langle proof \rangle$

lemma *aux_RS1*: $f \in N \rightarrow P \Longrightarrow n{\in}N \Longrightarrow f`n \in upclosure(f\ ``N)$
$\langle proof \rangle$

lemma *decr_succ_decr*:
assumes $f \in nat \rightarrow P$ $preorder\_on(P,leq)$
$\forall\, n{\in}nat.\ \langle f\ `\ succ(n), f\ `\ n\rangle \in leq$
$m{\in}nat$
shows $n{\in}nat \Longrightarrow n{\le}m \Longrightarrow \langle f\ `\ m, f\ `\ n\rangle \in leq$
$\langle proof \rangle$

lemma *decr_seq_linear*:
  assumes $refl(P,leq)$ $f \in nat \to P$
    $\forall n{\in}nat.\ \langle f\ `\ succ(n),\ f\ `\ n\rangle \in leq$
    $trans[P](leq)$
  shows $linear(f\ ``\ nat,\ leq)$
$\langle proof\rangle$

end

2.2 Towards Rasiowa-Sikorski Lemma (RSL)

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

begin

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

The next lemma identifies a sufficient condition for obtaining RSL.

lemma *RS_sequence_imp_rasiowa_sikorski*:
  assumes
    $p{\in}P$ $f : nat{\to}P$ $f\ `\ 0 = p$
    $\bigwedge n.\ n{\in}nat \Longrightarrow f\ `\ succ(n){\preceq}\ f\ `\ n \wedge f\ `\ succ(n) \in \mathcal{D}\ `\ n$
  shows
    $\exists G.\ p{\in}G \wedge D\_generic(G)$
$\langle proof\rangle$

end

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

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

context *countable_generic*
begin

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

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

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

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

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

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

end

3 A pointed version of DC

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

begin

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

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

lemma *witness_into_A* [*TC*]:
  assumes $a \in A$

$(\forall X . X \neq 0 \wedge X \subseteq A \longrightarrow s`X \in X)$
$\forall y \in A. \{x \in A. \langle y,x\rangle \in R\} \neq 0$ $n \in nat$
shows $dc\_witness(n, A, a, s, R) \in A$
⟨*proof*⟩

lemma *witness_related* :
assumes $a \in A$
$(\forall X . X \neq 0 \wedge X \subseteq A \longrightarrow s`X \in X)$
$\forall y \in A. \{x \in A. \langle y,x\rangle \in R\} \neq 0$ $n \in nat$
shows $\langle dc\_witness(n, A, a, s, R), dc\_witness(succ(n), A, a, s, R)\rangle \in R$
⟨*proof*⟩

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

lemma *witness_to_fun*: **assumes** $a \in A$
$(\forall X . X \neq 0 \wedge X \subseteq A \longrightarrow s`X \in X)$
$\forall y \in A. \{x \in A. \langle y,x\rangle \in R\} \neq 0$
shows $\exists f \in nat \rightarrow A. \forall n \in nat. f`n = dc\_witness(n,A,a,s,R)$
⟨*proof*⟩

theorem *pointed_DC* :
assumes $(\forall x \in A. \exists y \in A. \langle x,y\rangle \in R)$
shows $\forall a \in A. (\exists f \in nat \rightarrow A. f`0 = a \wedge (\forall n \in nat. \langle f`n, f`succ(n)\rangle \in R))$
⟨*proof*⟩

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

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

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

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

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

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

end

4 The general Rasiowa-Sikorski lemma

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

context *countable_generic*
begin

lemma *RS_relation*:
assumes $p{\in}P$ $n{\in}nat$
shows $\exists y{\in}P.\ \langle p,y\rangle \in (\lambda m{\in}nat.\ \{\langle x,y\rangle{\in}P{\times}P.\ y{\preceq}x \wedge y{\in}\mathcal{D}`(pred(m))\})`n$
⟨*proof*⟩

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

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

end

end

5 Auxiliary results on arithmetic

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *funcI* : $f \in A \to B \Longrightarrow a \in A \Longrightarrow b{=}\ f\ `\ a \Longrightarrow \langle a,\ b\rangle \in f$
⟨*proof*⟩

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

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

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

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

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

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

lemma *Un_leD2* : $Ord(i){\Longrightarrow}\ Ord(j){\Longrightarrow}\ Ord(k){\Longrightarrow}\ i \cup j \le k \Longrightarrow j \le k$

$\langle proof \rangle$

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

lemma *pred_mono* : $m \in nat \Longrightarrow n \leq m \Longrightarrow pred(n) \leq pred(m)$
$\langle proof \rangle$

lemma *succ_mono* : $m \in nat \Longrightarrow n \leq m \Longrightarrow succ(n) \leq succ(m)$
$\langle proof \rangle$

lemma *pred2_Un*:
assumes $j \in nat$ $m \leq j$ $n \leq j$
shows $pred(pred(m \cup n)) \leq pred(pred(j))$
$\langle proof \rangle$

lemma *nat_union_abs1* :
$[\![Ord(i) ; Ord(j) ; i \leq j]\!] \Longrightarrow i \cup j = j$
$\langle proof \rangle$

lemma *nat_union_abs2* :
$[\![Ord(i) ; Ord(j) ; i \leq j]\!] \Longrightarrow j \cup i = j$
$\langle proof \rangle$

lemma *nat_un_max* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow i \cup j = max(i,j)$
$\langle proof \rangle$

lemma *nat_max_ty* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow Ord(max(i,j))$
$\langle proof \rangle$

lemma *le_not_lt_nat* : $Ord(p) \Longrightarrow Ord(q) \Longrightarrow \neg\ p \leq q \Longrightarrow q \leq p$
$\langle proof \rangle$

lemmas *nat_simp_union* = *nat_un_max nat_max_ty max_def*

lemma *le_succ* : $x \in nat \Longrightarrow x \leq succ(x)$ $\langle proof \rangle$
lemma *le_pred* : $x \in nat \Longrightarrow pred(x) \leq x$
$\langle proof \rangle$

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

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

lemma *Un_leI3* : $o \leq r \Longrightarrow p \leq r \Longrightarrow q \leq r \Longrightarrow$

$Ord(o) \Longrightarrow Ord(p) \Longrightarrow Ord(q) \Longrightarrow Ord(r) \Longrightarrow$
$o \cup p \cup q \leq r$
⟨proof⟩

lemma *diff_mono* :
assumes $m \in nat$ $n{\in}nat$ $p \in nat$ $m < n$ $p{\leq}m$
shows $m\#\text{-}p < n\#\text{-}p$
⟨proof⟩

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

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

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

lemma *diff_cancel* :
assumes $m \in nat$ $n{\in}nat$ $m < n$
shows $m\#\text{-}n = 0$
⟨proof⟩

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

5.1 Some results in ordinal arithmetic

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

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

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

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

lemma *max_D1* :
assumes $x = y$ $w < z$ $Ord(x)$ $Ord(w)$ $Ord(z)$ $max(x,w) = max(y,z)$

shows $z \le y$
⟨*proof*⟩

lemma *max_D2* :
assumes $w = y \vee w = z$ $x < y$ $Ord(x)$ $Ord(w)$ $Ord(y)$ $Ord(z)$ $max(x,w) = max(y,z)$
shows $x < w$
⟨*proof*⟩

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

6 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 $nth(n,env) \in A$
⟨*proof*⟩

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

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

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

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

lemmas *fm_defs* = *omega_fm_def limit_ordinal_fm_def empty_fm_def typed_function_fm_def pair_fm_def upair_fm_def domain_fm_def function_fm_def succ_fm_def cons_fm_def fun_apply_fm_def image_fm_def big_union_fm_def union_fm_def*

relation_fm_def composition_fm_def field_fm_def ordinal_fm_def range_fm_def
transset_fm_def subset_fm_def Replace_fm_def

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 $R\hat{}*$

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

lemma *fld_restrict_mono* : $relation(r) \Longrightarrow A \subseteq B \Longrightarrow r \cap A{\times}A \subseteq r \cap B{\times}B$
⟨*proof*⟩

lemma *fld_restrict_dom* :
assumes $relation(r)$ $domain(r) \subseteq A$ $range(r) \subseteq A$
shows $r \cap A{\times}A = r$
⟨*proof*⟩

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

lemma *tr_downD* : $x \in tr\_down(r,a) \Longrightarrow \langle x,a\rangle \in r\hat{}+$
⟨*proof*⟩

lemma *pred_down* : $relation(r) \Longrightarrow r\text{-``}\{a\} \subseteq tr\_down(r,a)$
⟨*proof*⟩

lemma *tr_down_mono* : $relation(r) \Longrightarrow x \in r\text{-``}\{a\} \Longrightarrow tr\_down(r,x) \subseteq tr\_down(r,a)$
⟨*proof*⟩

lemma *rest_eq* :
assumes $relation(r)$ **and** $r\text{-``}\{a\} \subseteq B$ **and** $a \in B$
shows $r\text{-``}\{a\} = (r \cap B{\times}B)\text{-``}\{a\}$
⟨*proof*⟩

lemma *wfrec_restr_eq* : $r' = r \cap A{\times}A \Longrightarrow wfrec[A](r,a,H) = wfrec(r',a,H)$
⟨*proof*⟩

lemma *wfrec_restr* :
assumes *rr*: $relation(r)$ **and** *wfr*:$wf(r)$
shows $a \in A \Longrightarrow tr\_down(r,a) \subseteq A \Longrightarrow wfrec(r,a,H) = wfrec[A](r,a,H)$
⟨*proof*⟩

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

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

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

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

lemma *Rrel_mem*: $Rrel(mem,x) = Memrel(x)$
$\langle proof \rangle$

lemma *relation_Rrel*: $relation(Rrel(R,d))$
$\langle proof \rangle$

lemma *field_Rrel*: $field(Rrel(R,d)) \subseteq d$
$\langle proof \rangle$

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

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

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

$\langle proof \rangle$

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

lemma *restrict_trans_eq*:
assumes $w \in y$
shows $restrict(f,Memrel(eclose(\{x\}))\text{-``}\{y\})\text{`}w$
$= restrict(f,(Memrel(eclose(\{x\}))\hat{}+)\text{-``}\{y\})\text{`}w$
$\langle proof \rangle$

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

end

8 Relativization of the cumulative hierarchy

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

begin

lemma (**in** *M_trivial*) *powerset_abs'* [*simp*]:
assumes
$M(x)\ M(y)$
shows
$powerset(M,x,y) \longleftrightarrow y = \{a{\in}Pow(x)\ .\ M(a)\}$
⟨*proof*⟩

lemma *Collect_inter_Transset*:
assumes
$Transset(M)\ \ b \in M$
shows
$\{x{\in}b\ .\ P(x)\} = \{x{\in}b\ .\ P(x)\} \cap M$
⟨*proof*⟩

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

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*: $is\_powapply(M,f,y,z) \Longrightarrow M(z)$
$\langle proof \rangle$

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
$is\_powapply(\#\#A,nth(f,\ env),nth(y,\ env),nth(z,\ env))$
$\longleftrightarrow sats(A,?ipa\_fm(f,y,z),env)$
$\langle proof \rangle$

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
$is\_powapply(\#\#A,ff,yy,zz) \longleftrightarrow sats(A,\ ?is\_one\_fm(a,r),\ env)$
$\langle proof \rangle$

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

lemma (**in** *M_eclose*) *wf_rrank* : $M(x) \Longrightarrow wf(rrank(x))$
$\langle proof \rangle$

lemma (**in** *M_eclose*) *trans_rrank* : $M(x) \Longrightarrow trans(rrank(x))$
$\langle proof \rangle$

lemma (**in** *M_eclose*) *relation_rrank* : $M(x) \Longrightarrow relation(rrank(x))$
$\langle proof \rangle$

lemma (**in** *M_eclose*) *rrank_in_M* : $M(x) \Longrightarrow M(rrank(x))$
$\langle proof \rangle$

8.2 Absoluteness results

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

begin

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

lemma $[\![M(A); M(x); M(f); M(h)]\!] \Longrightarrow$
is_HVfrom$(M,A,x,f,h) \longleftrightarrow$
$(\exists R[M].\ h = A \cup \bigcup R \wedge$ *is_Replace*$(M,\ x, \lambda x\ y.\ y = \{x \in Pow(f\ `\ x)\ .\ M(x)\},$
$R))$
$\langle proof \rangle$

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*$(M,f))$
$\langle proof \rangle$

lemma *powapply_closed*:

$[\![\ M(y)\ ;\ M(f)\ ]\!] \Longrightarrow M(\{x \in Pow(f\ `\ y)\ .\ M(x)\})$
$\langle proof \rangle$

lemma *RepFun_is_powapply*:
assumes
$M(R)\ M(A)\ M(f)$
shows
$Replace(A, is\_powapply(M,f)) = RepFun(A, \lambda y.\{x \in Pow(f`y).\ M(x)\})$
$\langle proof \rangle$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *Replace_PHrank_abs*:
assumes
$M(z)\ M(f)\ M(hr)$
shows
$is\_Replace(M,z,PHrank(M,f),hr) \longleftrightarrow hr = Replace(z,PHrank(M,f))$
⟨*proof*⟩

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

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

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

lemma *Union_PHrank_closed*:
assumes
$M(x)\ M(f)$
shows

$M(\bigcup y\in x.\ succ(f`y))$
⟨*proof*⟩

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

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

lemma *M_into_Vset*:
assumes $M(a)$
shows $\exists\, i[M].\ \exists\, V[M].\ ordinal(M,i) \wedge is\_Vfrom(M,0,i,V) \wedge a\in V$
⟨*proof*⟩

end
end

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*⟩

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
  $\_sats$ :: $[i, i, i] \Rightarrow o$ (‹(_, _ $\models$ _)› $[36,36,36]$ 60)
syntax_consts
  $\_sats \rightleftharpoons sats$
translations
  $(M,env \models \varphi) \rightleftharpoons CONST\ sats(M,\varphi,env)$

abbreviation
  $dec10$ :: i (‹10›) **where** $10 \equiv succ(9)$

abbreviation
  $dec11$ :: i (‹11›) **where** $11 \equiv succ(10)$

abbreviation
  $dec12$ :: i (‹12›) **where** $12 \equiv succ(11)$

abbreviation
  $dec13$ :: i (‹13›) **where** $13 \equiv succ(12)$

abbreviation
  $dec14$ :: i (‹14›) **where** $14 \equiv succ(13)$

definition
  $infinity\_ax$ :: $(i \Rightarrow o) \Rightarrow o$ **where**
  $infinity\_ax(M) \equiv$
    $(\exists I[M].\ (\exists z[M].\ empty(M,z) \wedge z{\in}I) \wedge (\forall y[M].\ y{\in}I \longrightarrow (\exists sy[M].\ successor(M,y,sy) \wedge sy{\in}I)))$

definition
  $choice\_ax$ :: $(i{\Rightarrow}o) \Rightarrow o$ **where**
  $choice\_ax(M) \equiv \forall x[M].\ \exists a[M].\ \exists f[M].\ ordinal(M,a) \wedge surjection(M,a,x,f)$

context M_basic **begin**

lemma $choice\_ax\_abs$:
  $choice\_ax(M) \longleftrightarrow (\forall x[M].\ \exists a[M].\ \exists f[M].\ Ord(a) \wedge f \in surj(a,x))$
  $\langle proof \rangle$

end

definition
  $wellfounded\_trancl$ :: $[i{=}{>}o,i,i,i]$ => o **where**
  $wellfounded\_trancl(M,Z,r,p) \equiv$
    $\exists w[M].\ \exists wx[M].\ \exists rp[M].$
      $w \in Z\ \&\ pair(M,w,p,wx)\ \&\ tran\_closure(M,r,rp)\ \&\ wx \in rp$

lemma *empty_intf* :
  $infinity\_ax(M) \Longrightarrow$
  $(\exists z[M].\ empty(M,z))$
  ⟨*proof*⟩

lemma *Transset_intf* :
  $Transset(M) \Longrightarrow\ y{\in}x \Longrightarrow x \in M \Longrightarrow y \in M$
  ⟨*proof*⟩

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*: $\varphi{\in}formula \Longrightarrow env{\in}list(M) \Longrightarrow arity(\varphi) \leq 1\ \#\!+$
$length(env) \Longrightarrow$
      $separation(\#\#M,\lambda x.\ sats(M,\varphi,[x]\ @\ env))$
    and *replacement_ax*: $\varphi{\in}formula \Longrightarrow env{\in}list(M) \Longrightarrow arity(\varphi) \leq 2\ \#\!+$
$length(env) \Longrightarrow$
      $strong\_replacement(\#\#M,\lambda x\ y.\ sats(M,\varphi,[x,y]\ @\ env))$
    and *trans_M*: $Transset(M)$
begin

lemma *TranssetI* :
  $(\bigwedge y\ x.\ y{\in}x \Longrightarrow x{\in}M \Longrightarrow y{\in}M) \Longrightarrow Transset(M)$
  ⟨*proof*⟩

lemma *zero_in_M*: $0 \in M$
⟨*proof*⟩

10.1 Interface with *M_trivial*

lemma *mtrans* :
  $M\_trans(\#\#M)$
  ⟨*proof*⟩

lemma *mtriv* :
  $M\_trivial(\#\#M)$
  ⟨*proof*⟩

end

sublocale *M_ZF_trans* ⊆ *M_trivial* ##*M*

⟨*proof*⟩

context *M_ZF_trans*
begin

10.2 Interface with *M_basic*

schematic_goal *inter_fm_auto*:
assumes
$nth(i,env) = x$ $nth(j,env) = B$
$i \in nat$ $j \in nat$ $env \in list(A)$
shows
$(\forall y\in A . y\in B \longrightarrow x\in y) \longleftrightarrow sats(A,?ifm(i,j),env)$
⟨*proof*⟩

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

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

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

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

lemma *cartprod_sep_intf* :

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

schematic_goal *im_fm_auto*:
assumes
$nth(i,env) = y$ $nth(j,env) = r$ $nth(h,env) = B$
$i \in nat$ $j \in nat$ $h \in nat$ $env \in list(A)$
shows
$(\exists p \in A.\ p \in r\ \&\ (\exists x \in A.\ x \in B\ \&\ pair(\#\#A,x,y,p))) \longleftrightarrow sats(A,?imfm(i,j,h),env)$
⟨*proof*⟩

lemma *image_sep_intf* :
assumes
$A \in M$
and
$r \in M$
shows
separation$(\#\#M,\ \lambda y.\ \exists p \in M.\ p \in r\ \&\ (\exists x \in M.\ x \in A\ \&\ pair(\#\#M,x,y,p)))$
⟨*proof*⟩

schematic_goal *con_fm_auto*:
assumes
$nth(i,env) = z$ $nth(j,env) = R$
$i \in nat$ $j \in nat$ $env \in list(A)$
shows
$(\exists p \in A.\ p \in R\ \&\ (\exists x \in A.\exists y \in A.\ pair(\#\#A,x,y,p)\ \&\ pair(\#\#A,y,x,z)))$
$\longleftrightarrow sats(A,?cfm(i,j),env)$
⟨*proof*⟩

lemma *converse_sep_intf* :
assumes
$R \in M$
shows
separation$(\#\#M,\lambda z.\ \exists p \in M.\ p \in R\ \&\ (\exists x \in M.\exists y \in M.\ pair(\#\#M,x,y,p)\ \&\ pair(\#\#M,y,x,z)))$
⟨*proof*⟩

schematic_goal *rest_fm_auto*:
assumes
$nth(i,env) = z$ $nth(j,env) = C$
$i \in nat$ $j \in nat$ $env \in list(A)$
shows

$(\exists x{\in}A.\ x{\in}C\ \&\ (\exists y{\in}A.\ pair(\#\#A,x,y,z)))$
$\longleftrightarrow sats(A,?rfm(i,j),env)$
$\langle proof \rangle$

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

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
$(\exists x{\in}A.\ \exists y{\in}A.\ \exists z{\in}A.\ \exists xy{\in}A.\ \exists yz{\in}A.$
$pair(\#\#A,x,z,xz)\ \&\ pair(\#\#A,x,y,xy)\ \&\ pair(\#\#A,y,z,yz)\ \&\ xy{\in}S\ \&$
$yz{\in}R)$
$\longleftrightarrow sats(A,?cfm(i,j,h),env)$
$\langle proof \rangle$

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

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
$(\exists p{\in}A.\ p{\in}R\ \&\ pair(\#\#A,y,X,p)) \longleftrightarrow sats(A,?pfm(i,j,h),env)$
$\langle proof \rangle$

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*⟩

schematic_goal *mem_fm_auto*:
assumes
$nth(i,env) = z\ i \in nat\ env \in list(A)$
shows
$(\exists\, x{\in}A.\ \exists\, y{\in}A.\ pair(\#\#A,x,y,z)\ \&\ x \in y) \longleftrightarrow sats(A,?mfm(i),env)$
⟨*proof*⟩

lemma *memrel_sep_intf*:
separation$(\#\#M,\ \lambda z.\ \exists\, x{\in}M.\ \exists\, y{\in}M.\ pair(\#\#M,x,y,z)\ \&\ x \in y)$
⟨*proof*⟩

schematic_goal *recfun_fm_auto*:
assumes
$nth(i1,env) = x\ nth(i2,env) = r\ nth(i3,env) = f\ nth(i4,env) = g\ nth(i5,env) = a$
$nth(i6,env) = b\ i1{\in}nat\ i2{\in}nat\ i3{\in}nat\ i4{\in}nat\ i5{\in}nat\ i6{\in}nat\ env \in list(A)$
shows
$(\exists\, xa{\in}A.\ \exists\, xb{\in}A.\ pair(\#\#A,x,a,xa)\ \&\ xa \in r\ \&\ pair(\#\#A,x,b,xb)\ \&\ xb \in r\ \&$
$(\exists\, fx{\in}A.\ \exists\, gx{\in}A.\ fun\_apply(\#\#A,f,x,fx)\ \&\ fun\_apply(\#\#A,g,x,gx)\ \&\ fx \neq gx))$
$\longleftrightarrow sats(A,?rffm(i1,i2,i3,i4,i5,i6),env)$
⟨*proof*⟩

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*⟩

schematic_goal *funsp_fm_auto*:
assumes
$nth(i,env) = p\ nth(j,env) = z\ nth(h,env) = n$
$i \in nat\ j \in nat\ h \in nat\ env \in list(A)$
shows

($\exists f\in A.\ \exists b\in A.\ \exists nb\in A.\ \exists cnbf\in A.$ *pair*(##A,f,b,p) & *pair*(##A,n,b,nb) & *is_cons*(##A,nb,f,cnbf) &
upair(##A,cnbf,cnbf,z)) $\longleftrightarrow$ *sats*(A,?fsfm(i,j,h),env)
⟨*proof*⟩

lemma *funspace_succ_rep_intf* :
assumes
$n\in M$
shows
strong_replacement(##M,
$\lambda p\ z.\ \exists f\in M.\ \exists b\in M.\ \exists nb\in M.\ \exists cnbf\in M.$
pair(##M,f,b,p) & *pair*(##M,n,b,nb) & *is_cons*(##M,nb,f,cnbf) &
upair(##M,cnbf,cnbf,z))
⟨*proof*⟩

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)
⟨*proof*⟩

end

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

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)
⟨*proof*⟩

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*⟩

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
$rtran\_closure(\#\#A,r,rp) \longleftrightarrow sats(A,?rtc(i,j),env)$
⟨*proof*⟩

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
$tran\_closure(\#\#A,r,rp) \longleftrightarrow sats(A,?tc(i,j),env)$
⟨*proof*⟩

⟨*ML*⟩

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

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

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

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

lemma (**in** M_ZF_trans) nat_subset_I :
$\exists\, I{\in}M.\ nat \subseteq I$
$\langle proof \rangle$

lemma (**in** M_ZF_trans) nat_in_M :
$nat \in M$
$\langle proof \rangle$

lemma (**in** M_ZF_trans) $mtrancl$: $M\_trancl(\#\#M)$
$\langle proof \rangle$

sublocale $M\_ZF\_trans \subseteq M\_trancl\ \#\#M$
$\langle proof \rangle$

10.4 Interface with M_eclose

lemma $repl\_sats$:
assumes
$sat{:}\bigwedge x\ z.\ x{\in}M \Longrightarrow z{\in}M \Longrightarrow sats(M,\varphi,Cons(x,Cons(z,env))) \longleftrightarrow P(x,z)$
shows
$strong\_replacement(\#\#M,\lambda x\ z.\ sats(M,\varphi,Cons(x,Cons(z,env)))) \longleftrightarrow$
$strong\_replacement(\#\#M,P)$
$\langle proof \rangle$

lemma (**in** M_ZF_trans) nat_trans_M :
$n{\in}M$ **if** $n{\in}nat$ **for** n
$\langle proof \rangle$

lemma (**in** M_ZF_trans) $list\_repl1\_intf$:
assumes
$A{\in}M$
shows
$iterates\_replacement(\#\#M,\ is\_list\_functor(\#\#M,A),\ 0)$
$\langle proof \rangle$

lemma (**in** M_ZF_trans) $iterates\_repl\_intf$:
assumes
$v{\in}M$ **and**
$isfm{:}is\_F\_fm \in formula$ **and**
$arty{:}arity(is\_F\_fm){=}2$ **and**
$satsf{:}\ \bigwedge a\ b\ env'.\ [\![\ a{\in}M\ ;\ b{\in}M\ ;\ env'{\in}list(M)\ ]\!]$
$\Longrightarrow is\_F(a,b) \longleftrightarrow sats(M,\ is\_F\_fm,\ [b,a]@env')$

shows
iterates_replacement(##*M*,*is_F*,*v*)
⟨*proof*⟩

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

lemma (**in** *M_ZF_trans*) *nth_repl_intf*:
assumes
$l \in M$
shows
iterates_replacement(##*M*,λ*l'* *t*. *is_tl*(##*M*,*l'*,*t*),*l*)
⟨*proof*⟩

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

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

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

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

lemma (**in** *M_ZF_trans*) *mdatatypes* : *M_datatypes*(##*M*)

⟨*proof*⟩

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

lemma (**in** *M_ZF_trans*) *meclose* : *M_eclose*(*##M*)
⟨*proof*⟩

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

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

lemma *powerset_type* [*TC*]:
⟦ *x* ∈ *nat*; *y* ∈ *nat* ⟧ ⟹ *powerset_fm*(*x*,*y*) ∈ *formula*
⟨*proof*⟩

definition
is_powapply_fm :: [*i*,*i*,*i*] ⇒ *i* **where**
is_powapply_fm(*f*,*y*,*z*) ≡
Exists(*And*(*fun_apply_fm*(*succ*(*f*), *succ*(*y*), *0*),
Forall(*Iff*(*Member*(*0*, *succ*(*succ*(*z*))),
Forall(*Implies*(*Member*(*0*, *1*), *Member*(*0*, *2*)))))))

lemma *is_powapply_type* [*TC*] :
⟦*f*∈*nat* ; *y*∈*nat*; *z*∈*nat*⟧ ⟹ *is_powapply_fm*(*f*,*y*,*z*)∈*formula*
⟨*proof*⟩

lemma *sats_is_powapply_fm* :
assumes
f∈*nat* *y*∈*nat* *z*∈*nat* *env*∈*list*(*A*) *0*∈*A*
shows
is_powapply(*##A*,*nth*(*f*, *env*),*nth*(*y*, *env*),*nth*(*z*, *env*))
⟷ *sats*(*A*,*is_powapply_fm*(*f*,*y*,*z*),*env*)
⟨*proof*⟩

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

definition
  PHrank_fm :: $[i,i,i] \Rightarrow i$ **where**
  $PHrank\_fm(f,y,z) \equiv Exists(And(fun\_apply\_fm(succ(f),succ(y),0)$
  $,succ\_fm(0,succ(z))))$

lemma *PHrank_type* [*TC*]:
  $[\![x \in nat; y \in nat; z \in nat]\!] \Longrightarrow PHrank\_fm(x,y,z) \in formula$
  ⟨*proof*⟩

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

lemma (**in** *M_ZF_trans*) *phrank_repl* :
  assumes
  $f \in M$
  shows
  $strong\_replacement(\#\#M,PHrank(\#\#M,f))$
⟨*proof*⟩

definition
  is_Hrank_fm :: $[i,i,i] \Rightarrow i$ **where**
  $is\_Hrank\_fm(x,f,hc) \equiv Exists(And(big\_union\_fm(0,succ(hc)),$
  $Replace\_fm(succ(x),PHrank\_fm(succ(succ(succ(f))),0,1),0)))$

lemma *is_Hrank_type* [*TC*]:
  $[\![x \in nat; y \in nat; z \in nat]\!] \Longrightarrow is\_Hrank\_fm(x,y,z) \in formula$
  ⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *sats_is_Hrank_fm* [*simp*]:
  $[\![x \in nat; y \in nat; z \in nat; env \in list(M)]\!]$
  $\Longrightarrow sats(M,is\_Hrank\_fm(x,y,z),env) \longleftrightarrow$
  $is\_Hrank(\#\#M,nth(x,env),nth(y,env),nth(z,env))$
  ⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *wfrec_rank* :
  assumes
  $X \in M$
  shows
  $wfrec\_replacement(\#\#M,is\_Hrank(\#\#M),rrank(X))$

⟨proof⟩

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

lemma *is_HVfrom_type* [*TC*]:
  ⟦ *A*∈*nat*; *x* ∈ *nat*; *f* ∈ *nat*; *h* ∈ *nat* ⟧ ⟹ *is_HVfrom_fm*(*A*,*x*,*f*,*h*) ∈ *formula*
  ⟨proof⟩

lemma *sats_is_HVfrom_fm* :
  ⟦ *a*∈*nat*; *x* ∈ *nat*; *f* ∈ *nat*; *h* ∈ *nat*; *env* ∈ *list*(*A*); *0*∈*A*⟧
    ⟹ *sats*(*A*,*is_HVfrom_fm*(*a*,*x*,*f*,*h*),*env*) ⟷
      is_HVfrom(##*A*,*nth*(*a*,*env*),*nth*(*x*,*env*),*nth*(*f*,*env*),*nth*(*h*,*env*))
  ⟨proof⟩

lemma *is_HVfrom_iff_sats*:
  assumes
    nth(*a*,*env*) = *aa* *nth*(*x*,*env*) = *xx* *nth*(*f*,*env*) = *ff* *nth*(*h*,*env*) = *hh*
    a∈*nat* *x*∈*nat* *f*∈*nat* *h*∈*nat* *env*∈*list*(*A*) *0*∈*A*
  shows
    is_HVfrom(##*A*,*aa*,*xx*,*ff*,*hh*) ⟷ *sats*(*A*, *is_HVfrom_fm*(*a*,*x*,*f*,*h*), *env*)
  ⟨proof⟩

schematic_goal *sats_is_Vset_fm_auto*:
  assumes
    i∈*nat* *v*∈*nat* *env*∈*list*(*A*) *0*∈*A*
    i < *length*(*env*) *v* < *length*(*env*)
  shows
    is_Vset(##*A*,*nth*(*i*, *env*),*nth*(*v*, *env*))
    ⟷ *sats*(*A*,*?ivs_fm*(*i*,*v*),*env*)
  ⟨proof⟩

schematic_goal *is_Vset_iff_sats*:
  assumes
    nth(*i*,*env*) = *ii* *nth*(*v*,*env*) = *vv*
    i∈*nat* *v*∈*nat* *env*∈*list*(*A*) *0*∈*A*
    i < *length*(*env*) *v* < *length*(*env*)
  shows
    is_Vset(##*A*,*ii*,*vv*) ⟷ *sats*(*A*, *?ivs_fm*(*i*,*v*), *env*)
  ⟨proof⟩

lemma (**in** *M_ZF_trans*) *memrel_eclose_sing* :
  a∈*M* ⟹ ∃ *sa*∈*M*. ∃ *esa*∈*M*. ∃ *mesa*∈*M*.

$upair(\#\#M,a,a,sa)$ & $is\_eclose(\#\#M,sa,esa)$ & $membership(\#\#M,esa,mesa)$
⟨*proof*⟩

lemma (**in** M_ZF_trans) $trans\_repl\_HVFrom$:
assumes
$A{\in}M$ $i{\in}M$
shows
$transrec\_replacement(\#\#M,is\_HVfrom(\#\#M,A),i)$
⟨*proof*⟩

lemma (**in** M_ZF_trans) $meclose\_pow$: $M\_eclose\_pow(\#\#M)$
⟨*proof*⟩

sublocale $M\_ZF\_trans \subseteq M\_eclose\_pow$ $\#\#M$
⟨*proof*⟩

lemma (**in** M_ZF_trans) $repl\_gen$:
assumes
f_abs: $\bigwedge x\ y.\ [\![\ x{\in}M;\ y{\in}M\ ]\!] \Longrightarrow is\_F(\#\#M,x,y) \longleftrightarrow y = f(x)$
and
f_sats: $\bigwedge x\ y.\ [\![x{\in}M\ ;\ y{\in}M\ ]\!] \Longrightarrow$
$sats(M,f\_fm,Cons(x,Cons(y,env))) \longleftrightarrow is\_F(\#\#M,x,y)$
and
f_form: $f\_fm \in formula$
and
f_arty: $arity(f\_fm) = 2$
and
$env{\in}list(M)$
shows
$strong\_replacement(\#\#M,\ \lambda x\ y.\ y = f(x))$
⟨*proof*⟩

lemma (**in** M_ZF_trans) sep_in_M :
assumes
$\varphi \in formula$ $env{\in}list(M)$
$arity(\varphi) \leq 1\ \#{+}\ length(env)$ $A{\in}M$ **and**
$satsQ$: $\bigwedge x.\ x{\in}M \Longrightarrow sats(M,\varphi,[x]@env) \longleftrightarrow Q(x)$
shows
$\{y{\in}A\ .\ Q(y)\}{\in}M$
⟨*proof*⟩

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* :
  $Transset(M) \Longrightarrow\ y{\in}x \Longrightarrow x \in M \Longrightarrow y \in M$
  ⟨*proof*⟩

locale *M_ZF* =
  fixes *M*
  assumes
    upair_ax: *upair_ax*(##*M*)
    and *Union_ax*: *Union_ax*(##*M*)
    and *power_ax*: *power_ax*(##*M*)
    and *extensionality*: *extensionality*(##*M*)
    and *foundation_ax*: *foundation_ax*(##*M*)
    and *infinity_ax*: *infinity_ax*(##*M*)
    and *separation_ax*: $\varphi{\in}formula \Longrightarrow env{\in}list(M) \Longrightarrow arity(\varphi) \leq 1$ #+ $length(env) \Longrightarrow$
          separation(##*M*,λ*x*. *sats*(*M*,φ,[*x*] @ *env*))
    and *replacement_ax*: $\varphi{\in}formula \Longrightarrow env{\in}list(M) \Longrightarrow arity(\varphi) \leq 2$ #+ $length(env) \Longrightarrow$
          strong_replacement(##*M*,λ*x y*. *sats*(*M*,φ,[*x*,*y*] @ *env*))

locale *M_ctm* = *M_ZF* +
  fixes *enum*
  assumes *M_countable*: $enum{\in}bij(nat,M)$
    and *trans_M*: $Transset(M)$

begin
interpretation *intf*: *M_ZF_trans M*
  ⟨*proof*⟩

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

lemma *zero_in_M*: $0 \in M$
  ⟨*proof*⟩

lemma *tuples_in_M*: $A{\in}M \Longrightarrow B{\in}M \Longrightarrow \langle A,B\rangle{\in}M$
  ⟨*proof*⟩

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

lemma *n_in_M* : $n \in nat \Longrightarrow n \in M$
⟨*proof*⟩

lemma *mtriv*: *M_trivial*(##*M*)
⟨*proof*⟩

lemma *mtrans*: *M_trans*(##*M*)
⟨*proof*⟩

lemma *mbasic*: *M_basic*(##*M*)
⟨*proof*⟩

lemma *mtrancl*: *M_trancl*(##*M*)
⟨*proof*⟩

lemma *mdatatypes*: *M_datatypes*(##*M*)
⟨*proof*⟩

lemma *meclose*: *M_eclose*(##*M*)
⟨*proof*⟩

lemma *meclose_pow*: *M_eclose_pow*(##*M*)
⟨*proof*⟩

end

sublocale *M_ctm* ⊆ *M_trivial* ##*M*
⟨*proof*⟩

sublocale *M_ctm* ⊆ *M_trans* ##*M*
⟨*proof*⟩

sublocale *M_ctm* ⊆ *M_basic* ##*M*
⟨*proof*⟩

sublocale *M_ctm* ⊆ *M_trancl* ##*M*
⟨*proof*⟩

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

sublocale *M_ctm* ⊆ *M_eclose* ##*M*

⟨*proof*⟩

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

context *M_ctm*
begin

11.1 *Collects* in *M*

lemma *Collect_in_M_0p* :
assumes
Qfm : *Q_fm* ∈ *formula* **and**
Qarty : *arity*(*Q_fm*) = *1* **and**
Qsats : $\bigwedge$*x*. *x*∈*M* ⟹ *sats*(*M*,*Q_fm*,[*x*]) ⟷ *is_Q*(*##M*,*x*) **and**
Qabs : $\bigwedge$*x*. *x*∈*M* ⟹ *is_Q*(*##M*,*x*) ⟷ *Q*(*x*) **and**
A∈*M*
shows
Collect(*A*,*Q*)∈*M*
⟨*proof*⟩

lemma *Collect_in_M_2p* :
assumes
Qfm : *Q_fm* ∈ *formula* **and**
Qarty : *arity*(*Q_fm*) = *3* **and**
params_M : *y*∈*M* *z*∈*M* **and**
Qsats : $\bigwedge$*x*. *x*∈*M* ⟹ *sats*(*M*,*Q_fm*,[*x*,*y*,*z*]) ⟷ *is_Q*(*##M*,*x*,*y*,*z*) **and**
Qabs : $\bigwedge$*x*. *x*∈*M* ⟹ *is_Q*(*##M*,*x*,*y*,*z*) ⟷ *Q*(*x*,*y*,*z*) **and**
A∈*M*
shows
Collect(*A*,λ*x*. *Q*(*x*,*y*,*z*))∈*M*
⟨*proof*⟩

lemma *Collect_in_M_4p* :
assumes
Qfm : *Q_fm* ∈ *formula* **and**
Qarty : *arity*(*Q_fm*) = *5* **and**
params_M : *a1*∈*M* *a2*∈*M* *a3*∈*M* *a4*∈*M* **and**
Qsats : $\bigwedge$*x*. *x*∈*M* ⟹ *sats*(*M*,*Q_fm*,[*x*,*a1*,*a2*,*a3*,*a4*]) ⟷ *is_Q*(*##M*,*x*,*a1*,*a2*,*a3*,*a4*)
and
Qabs : $\bigwedge$*x*. *x*∈*M* ⟹ *is_Q*(*##M*,*x*,*a1*,*a2*,*a3*,*a4*) ⟷ *Q*(*x*,*a1*,*a2*,*a3*,*a4*) **and**
A∈*M*
shows
Collect(*A*,λ*x*. *Q*(*x*,*a1*,*a2*,*a3*,*a4*))∈*M*
⟨*proof*⟩

lemma *Repl_in_M* :

assumes
  f_fm: $f\_fm \in formula$ **and**
  f_ar: $arity(f\_fm) \le 2 \;\#+\; length(env)$ **and**
  fsats: $\bigwedge x\ y.\ x \in M \Longrightarrow y \in M \Longrightarrow sats(M, f\_fm, [x,y]@env) \longleftrightarrow is\_f(x,y)$ **and**
  fabs: $\bigwedge x\ y.\ x \in M \Longrightarrow y \in M \Longrightarrow is\_f(x,y) \longleftrightarrow y = f(x)$ **and**
  fclosed: $\bigwedge x.\ x \in A \Longrightarrow f(x) \in M$ **and**
  $A \in M$ $env \in list(M)$
shows $\{f(x).\ x \in A\} \in M$
⟨*proof*⟩

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* : $Transset(M) \Longrightarrow y \in M \Longrightarrow y \subseteq M$
  ⟨*proof*⟩

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

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

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

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

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

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

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

lemma *one_in_G* :

assumes *M_generic*(G)
shows *one* $\in$ G
⟨*proof*⟩

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

declare *iff_trans* [*trans*]

lemma *generic_filter_existence*:
$p{\in}P \Longrightarrow \exists\, G.\ p{\in}G \wedge$ *M_generic*(G)
⟨*proof*⟩

lemma *compat_in_abs* :
assumes
$A{\in}M$ $r{\in}M$ $p{\in}M$ $q{\in}M$
shows
is_compat_in(##M,A,r,p,q) $\longleftrightarrow$ *compat_in*(A,r,p,q)
⟨*proof*⟩

definition
compat_in_fm :: [i,i,i,i] $\Rightarrow$ i **where**
compat_in_fm(A,r,p,q) $\equiv$
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*] :
⟦ $A{\in}nat$;$r{\in}nat$;$p{\in}nat$;$q{\in}nat$⟧ $\Longrightarrow$ *compat_in_fm*(A,r,p,q)$\in$*formula*
⟨*proof*⟩

lemma *sats_compat_in_fm*:
assumes
$A{\in}nat$ $r{\in}nat$ $p{\in}nat$ $q{\in}nat$ *env*$\in$*list*(M)
shows
sats(M,*compat_in_fm*(A,r,p,q),*env*) $\longleftrightarrow$
is_compat_in(##M,*nth*(A, *env*),*nth*(r, *env*),*nth*(p, *env*),*nth*(q, *env*))
⟨*proof*⟩

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*)
⟨*proof*⟩

⟨*ML*⟩

schematic_goal *ZF_power_auto*:
power_ax(##*A*) $\longleftrightarrow$ (*A*, [] $\models$ *?zfpow*)
⟨*proof*⟩

⟨*ML*⟩

schematic_goal *ZF_pairing_auto*:
upair_ax(##*A*) $\longleftrightarrow$ (*A*, [] $\models$ *?zfpair*)
⟨*proof*⟩

⟨*ML*⟩

schematic_goal *ZF_foundation_auto*:
foundation_ax(##*A*) $\longleftrightarrow$ (*A*, [] $\models$ *?zfpow*)
⟨*proof*⟩

⟨*ML*⟩

schematic_goal *ZF_extensionality_auto*:
extensionality(##*A*) $\longleftrightarrow$ (*A*, [] $\models$ *?zfpow*)
⟨*proof*⟩

⟨*ML*⟩

schematic_goal *ZF_infinity_auto*:
infinity_ax(##*A*) $\longleftrightarrow$ (*A*, [] $\models$ (*?*φ(*i*,*j*,*h*)))
⟨*proof*⟩

⟨*ML*⟩

schematic_goal *ZF_choice_auto*:
choice_ax(##*A*) $\longleftrightarrow$ (*A*, [] $\models$ (*?*φ(*i*,*j*,*h*)))
⟨*proof*⟩

⟨*ML*⟩

syntax
_choice :: *i* (‹*AC*›)
syntax_consts

_choice ⇌ *ZF_choice_fm*
translations
AC ⇀ *CONST ZF_choice_fm*

lemmas *ZFC_fm_defs* = *ZF_extensionality_fm_def ZF_foundation_fm_def ZF_pairing_fm_def ZF_union_fm_def ZF_infinity_fm_def ZF_power_fm_def ZF_choice_fm_def*

lemmas *ZFC_fm_sats* = *ZF_extensionality_auto ZF_foundation_auto ZF_pairing_auto ZF_union_auto ZF_infinity_auto ZF_power_auto ZF_choice_auto*

definition
ZF_fin :: *i* **where**
ZF_fin ≡ { *ZF_extensionality_fm*, *ZF_foundation_fm*, *ZF_pairing_fm*, *ZF_union_fm*, *ZF_infinity_fm*, *ZF_power_fm* }

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

lemma *ZFC_fin_type* : *ZFC_fin* ⊆ *formula*
⟨*proof*⟩

12.1 The Axiom of Separation, internalized

lemma *iterates_Forall_type* [*TC*]:
⟦ *n* ∈ *nat*; *p* ∈ *formula* ⟧ ⟹ *Forall^n*(*p*) ∈ *formula*
⟨*proof*⟩

lemma *last_init_eq* :
assumes *l* ∈ *list*(*A*) *length*(*l*) = *succ*(*n*)
shows ∃ *a*∈*A*. ∃ *l'*∈*list*(*A*). *l* = *l'*@[*a*]
⟨*proof*⟩

lemma *take_drop_eq* :
assumes *l*∈*list*(*M*)
shows ⋀ *n* . *n* < *succ*(*length*(*l*)) ⟹ *l* = *take*(*n*,*l*) @ *drop*(*n*,*l*)
⟨*proof*⟩

lemma *list_split* :
assumes *n* ≤ *succ*(*length*(*rest*)) *rest* ∈ *list*(*M*)
shows ∃ *re*∈*list*(*M*). ∃ *st*∈*list*(*M*). *rest* = *re* @ *st* ∧ *length*(*re*) = *pred*(*n*)
⟨*proof*⟩

lemma *sats_nForall*:
assumes
φ ∈ *formula*
shows
n∈*nat* ⟹ *ms* ∈ *list*(*M*) ⟹
M, *ms* ⊨ (*Forall^n*(*φ*)) ⟷

$(\forall$ *rest* $\in$ *list*(M). *length*(*rest*) $=$ *n* $\longrightarrow$ M, *rest* @ *ms* $\models \varphi)$
⟨*proof*⟩

definition
sep_body_fm :: *i* ⇒ *i* **where**
sep_body_fm(*p*) ≡ *Forall*(*Exists*(*Forall*(
Iff(*Member*(*0*,*1*),*And*(*Member*(*0*,*2*),
incr_bv1^*2*(*p*))))))

lemma *sep_body_fm_type* [*TC*]: *p* ∈ *formula* ⟹ *sep_body_fm*(*p*) ∈ *formula*
⟨*proof*⟩

lemma *sats_sep_body_fm*:
assumes
φ ∈ *formula* *ms*∈*list*(*M*) *rest*∈*list*(*M*)
shows
M, *rest* @ *ms* ⊨ *sep_body_fm*(φ) ⟷
separation(##*M*,λ*x*. *M*, [*x*] @ *rest* @ *ms* ⊨ φ)
⟨*proof*⟩

definition
ZF_separation_fm :: *i* ⇒ *i* **where**
ZF_separation_fm(*p*) ≡ *Forall*^(*pred*(*arity*(*p*)))(*sep_body_fm*(*p*))

lemma *ZF_separation_fm_type* [*TC*]: *p* ∈ *formula* ⟹ *ZF_separation_fm*(*p*) ∈ *formula*
⟨*proof*⟩

lemma *sats_ZF_separation_fm_iff*:
assumes
φ∈*formula*
shows
(*M*, [] ⊨ (*ZF_separation_fm*(φ)))
⟷
(∀ *env*∈*list*(*M*). *arity*(φ) ≤ *1* #+ *length*(*env*) ⟶
separation(##*M*,λ*x*. *M*, [*x*] @ *env* ⊨ φ))
⟨*proof*⟩

12.2 The Axiom of Replacement, internalized

schematic_goal *sats_univalent_fm_auto*:
assumes

Q_iff_sats:⋀*x* *y* *z*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *z*∈*A* ⟹
Q(*x*,*z*) ⟷ (*A*,*Cons*(*z*,*Cons*(*y*,*Cons*(*x*,*env*))) ⊨ *Q1_fm*)
⋀*x* *y* *z*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *z*∈*A* ⟹
Q(*x*,*y*) ⟷ (*A*,*Cons*(*z*,*Cons*(*y*,*Cons*(*x*,*env*))) ⊨ *Q2_fm*)
and
asms: *nth*(*i*,*env*) = *B* *i* ∈ *nat* *env* ∈ *list*(*A*)

shows
  $univalent(\#\#A,B,Q) \longleftrightarrow A,env \models ?ufm(i)$
  ⟨*proof*⟩

⟨*ML*⟩

lemma *univalent_fm_type* [*TC*]: $q1 \in formula \Longrightarrow q2 \in formula \Longrightarrow i \in nat \Longrightarrow$
  $univalent\_fm(q2,q1,i) \in formula$
  ⟨*proof*⟩

lemma *sats_univalent_fm* :
  assumes
    $Q\_iff\_sats: \bigwedge x\ y\ z.\ x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow$
      $Q(x,z) \longleftrightarrow (A,Cons(z,Cons(y,Cons(x,env))) \models Q1\_fm)$
    $\bigwedge x\ y\ z.\ x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow$
      $Q(x,y) \longleftrightarrow (A,Cons(z,Cons(y,Cons(x,env))) \models Q2\_fm)$
    and
    $asms: nth(i,env) = B\ i \in nat\ env \in list(A)$
  shows
    $A,env \models univalent\_fm(Q1\_fm,Q2\_fm,i) \longleftrightarrow univalent(\#\#A,B,Q)$
  ⟨*proof*⟩

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

lemma *swap_vars_type*[*TC*] :
  $\varphi \in formula \Longrightarrow swap\_vars(\varphi) \in formula$
  ⟨*proof*⟩

lemma *sats_swap_vars* :
  $[x,y]\ @\ env \in list(M) \Longrightarrow \varphi \in formula \Longrightarrow$
    $M,\ [x,y]\ @\ env \models swap\_vars(\varphi) \longleftrightarrow M,[y,x]\ @\ env \models \varphi$
  ⟨*proof*⟩

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

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

lemma *univalent_Qs_type* [*TC*]:
  assumes $\varphi \in formula$
  shows $univalent\_Q1(\varphi) \in formula\ univalent\_Q2(\varphi) \in formula$
  ⟨*proof*⟩

lemma *sats_univalent_fm_assm*:
  assumes
    $x \in A$ $y \in A$ $z{\in}A$ $env{\in}$ $list(A)$ $\varphi \in formula$
  shows
    $(A, ([x,z]$ @ $env) \models \varphi) \longleftrightarrow (A,\ Cons(z,Cons(y,Cons(x,env))) \models (univalent\_Q1(\varphi)))$
    $(A, ([x,y]$ @ $env) \models \varphi) \longleftrightarrow (A,\ Cons(z,Cons(y,Cons(x,env))) \models (univalent\_Q2(\varphi)))$
  ⟨*proof*⟩

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*]: $p \in formula \Longrightarrow rep\_body\_fm(p) \in formula$
  ⟨*proof*⟩

lemmas *ZF_replacement_simps* = *formula_add_params1*[*of* φ *2* _ *M* [_,_]]
  sats_incr_bv_iff[*of* _ _ *M* _ []] — simplifies iterates of $\lambda x.\ incr\_bv(x)$ ` *0*
  sats_incr_bv_iff[*of* _ _ *M* _ [_,_]]— simplifies $\lambda x.\ incr\_bv(x)$ ` *2*
  sats_incr_bv1_iff[*of* _ _ *M*] *sats_swap_vars* **for** φ *M*

lemma *sats_rep_body_fm*:
  assumes
    $\varphi \in formula$ $ms{\in}list(M)$ $rest{\in}list(M)$
  shows
    $M,\ rest$ @ $ms \models rep\_body\_fm(\varphi) \longleftrightarrow$
     $strong\_replacement(\#\#M,\lambda x\ y.\ M,\ [x,y]$ @ $rest$ @ $ms \models \varphi)$
  ⟨*proof*⟩

definition
  $ZF\_replacement\_fm :: i \Rightarrow i$ **where**
  $ZF\_replacement\_fm(p) \equiv Forall\hat{}(pred(pred(arity(p))))(rep\_body\_fm(p))$

lemma *ZF_replacement_fm_type* [*TC*]: $p \in formula \Longrightarrow ZF\_replacement\_fm(p)$
$\in formula$
  ⟨*proof*⟩

lemma *sats_ZF_replacement_fm_iff*:
  assumes
    $\varphi{\in}formula$
  shows
  $(M,\ [] \models (ZF\_replacement\_fm(\varphi)))$
   $\longleftrightarrow$
   $(\forall\ env{\in}list(M).\ arity(\varphi) \leq 2\ \#{+}\ length(env) \longrightarrow$
     $strong\_replacement(\#\#M,\lambda x\ y.\ M,[x,y]$ @ $env \models \varphi))$
⟨*proof*⟩

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

lemma *Un_subset_formula*: *A*⊆*formula* ∧ *B*⊆*formula* ⟹ *A*∪*B* ⊆ *formula*
⟨*proof*⟩

lemma *ZF_inf_subset_formula* : *ZF_inf* ⊆ *formula*
⟨*proof*⟩

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

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

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

lemma *ZFC_subset_formula*: *ZFC* ⊆ *formula*
⟨*proof*⟩

Satisfaction of a set of sentences

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

lemma *satTI* [*intro*!]:
assumes ⋀φ. φ∈Φ ⟹ *A*,[] ⊨ φ
shows *A* ⊨ Φ
⟨*proof*⟩

lemma *satTD* [*dest*] :*A* ⊨ Φ ⟹ φ∈Φ ⟹ *A*,[] ⊨ φ
⟨*proof*⟩

lemma *sats_ZFC_iff_sats_ZF_AC*:
(*N* ⊨ *ZFC*) ⟷ (*N* ⊨ *ZF*) ∧ (*N*, [] ⊨ *AC*)
⟨*proof*⟩

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

end

13 Renaming of variables in internalized formulas

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

lemma *app_nm* :
  assumes $n{\in}nat$ $m{\in}nat$ $f{\in}n{\to}m$ $x \in nat$
  shows $f`x \in nat$
$\langle proof \rangle$

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 $f \in m \to n$
    $g \in p \to q$
  shows $union\_fun(f,g,m,p) \in m \cup p \to n \cup q$
$\langle proof \rangle$

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

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

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

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

$sum(f,g,m,n,p) \equiv \lambda j \in m\#{+}p$. *if* $j{<}m$ *then* $f`j$ *else* $(g`(j\#{-}m))\#{+}n$

lemma *sum_inl*:
assumes $m \in nat$ $n{\in}nat$
$f \in m{\to}n$ $x \in m$
shows $sum(f,g,m,n,p)`x = f`x$
⟨*proof*⟩

lemma *sum_inr*:
assumes $m \in nat$ $n{\in}nat$ $p{\in}nat$
$g{\in}p{\to}q$ $m \le x$ $x < m\#{+}p$
shows $sum(f,g,m,n,p)`x = g`(x\#{-}m)\#{+}n$
⟨*proof*⟩

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

lemma *sum_type* :
assumes $m \in nat$ $n{\in}nat$ $p{\in}nat$ $q{\in}nat$
$f \in m{\to}n$ $g{\in}p{\to}q$
shows $sum(f,g,m,n,p) \in (m\#{+}p) \to (n\#{+}q)$
⟨*proof*⟩

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

lemma *sum_type_id_aux2* :

assumes
$f \in m \rightarrow n$
$m \in nat$ $n \in nat$
$env1 \in list(M)$
shows
$sum(f,id(length(env1)),m,n,length(env1)) \in$
$(m\#+length(env1)) \rightarrow (n\#+length(env1))$
$\langle proof \rangle$

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

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

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

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

lemma *sum_id_tc_aux* :
$p \in nat \Longrightarrow q \in nat \Longrightarrow f \in p \rightarrow q \Longrightarrow sum\_id(p,f) \in 1\#+p \rightarrow 1\#+q$
$\langle proof \rangle$

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

13.2 Renaming of formulas

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

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

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

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

lemma *arity_meml* : $l \in nat \Longrightarrow Member(x,y) \in formula \Longrightarrow arity(Member(x,y)) \le l \Longrightarrow x \in l$
$\langle proof \rangle$
lemma *arity_memr* : $l \in nat \Longrightarrow Member(x,y) \in formula \Longrightarrow arity(Member(x,y)) \le l \Longrightarrow y \in l$
$\langle proof \rangle$
lemma *arity_eql* : $l \in nat \Longrightarrow Equal(x,y) \in formula \Longrightarrow arity(Equal(x,y)) \le l \Longrightarrow x \in l$
$\langle proof \rangle$
lemma *arity_eqr* : $l \in nat \Longrightarrow Equal(x,y) \in formula \Longrightarrow arity(Equal(x,y)) \le l \Longrightarrow y \in l$
$\langle proof \rangle$
lemma *nand_ar1* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(p) \le arity(Nand(p,q))$
$\langle proof \rangle$
lemma *nand_ar2* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(q) \le arity(Nand(p,q))$
$\langle proof \rangle$

lemma *nand_ar1D* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(Nand(p,q)) \le n \Longrightarrow arity(p) \le n$
$\langle proof \rangle$
lemma *nand_ar2D* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(Nand(p,q)) \le n \Longrightarrow arity(q) \le n$
$\langle proof \rangle$

lemma *ren_tc* : $p \in formula \Longrightarrow$
$(\bigwedge n\ m\ f\ .\ n \in nat \Longrightarrow m \in nat \Longrightarrow f \in n \to m \Longrightarrow ren(p)`n`m`f \in formula)$
$\langle proof \rangle$

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

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

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

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

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*⟩

end

14 Names and generic extensions

theory *Names*
  imports
    Forcing_Data
    Interface
    Recursion_Thms
    Synthetic_Definition
begin

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

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

lemma *Sep_and_Replace*: $\{b(x) .. x{\in}A, P(x) \} = \{b(x) . x{\in}\{y{\in}A. P(y)\}\}$
  ⟨*proof*⟩

lemma *SepReplace_subset* : $A{\subseteq}A' \Longrightarrow \{b .. x{\in}A, Q\}{\subseteq}\{b .. x{\in}A', Q\}$
  ⟨*proof*⟩

lemma *SepReplace_iff* [*simp*]: $y{\in}\{b(x) .. x{\in}A, P(x)\} \longleftrightarrow (\exists x{\in}A. y{=}b(x) \ \& \ P(x))$
  ⟨*proof*⟩

lemma *SepReplace_dom_implies* :
  $(\bigwedge x . x \in A \Longrightarrow b(x) = b'(x)) \Longrightarrow \{b(x) .. x{\in}A, Q(x)\}{=}\{b'(x) .. x{\in}A, Q(x)\}$
  ⟨*proof*⟩

lemma *SepReplace_pred_implies* :
  $\forall x. Q(x) \longrightarrow b(x) = b'(x) \Longrightarrow \{b(x) .. x{\in}A, Q(x)\}{=}\{b'(x) .. x{\in}A, Q(x)\}$
  ⟨*proof*⟩

14.1 The well-founded relation *ed*

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

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

$\langle proof \rangle$

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

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

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

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

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

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

lemma *edI*[*intro*!]: $t \in domain(x) \Longrightarrow ed(t,x)$
$\langle proof \rangle$

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

lemma *rank_ed*:
assumes $ed(y,x)$
shows $succ(rank(y)) \leq rank(x)$
$\langle proof \rangle$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

context *M_ctm*
begin

lemma *upairM* : $x \in M \Longrightarrow y \in M \Longrightarrow \{x,y\} \in M$
⟨*proof*⟩

lemma *singletonM* : $a \in M \Longrightarrow \{a\} \in M$
⟨*proof*⟩

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

end

14.2 Values and check-names

context *forcing_data*
begin

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

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

lemma *checkD*:
$check(x) = \ wfrec(Memrel(eclose(\{x\})),\ x,\ Hcheck)$
⟨*proof*⟩

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

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

lemma *check_trancl*: $check(x) = wfrec(rcheck(x),\ x,\ Hcheck)$
⟨*proof*⟩

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

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

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

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

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

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

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

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

lemma *aux_def_val*:
  assumes $z \in domain(x)$
  shows $wfrec(edrel(eclose(\{x\})),z,Hv(G)) = wfrec(edrel(eclose(\{z\})),z,Hv(G))$
$\langle proof\rangle$

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

lemma *def_val*: $val(G,x) = \{val(G,t) .. t{\in}domain(x) , \exists p{\in}P . \langle t,p\rangle{\in}x \wedge p \in G \}$
$\langle proof\rangle$

lemma *val_mono* : $x{\subseteq}y \Longrightarrow val(G,x) \subseteq val(G,y)$
  $\langle proof\rangle$

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

lemma *valcheck* : $one \in G \Longrightarrow one \in P \Longrightarrow val(G,check(y)) = y$
$\langle proof\rangle$

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

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

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

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

lemma *val_subset_domain_times_range*: $val(F,\tau) \subseteq val(F,domain(\tau)\times range(\tau))$
$\langle proof\rangle$

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

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

lemma *val_of_elem*: $\langle\vartheta,p\rangle \in \pi \Longrightarrow p{\in}G \Longrightarrow p{\in}P \Longrightarrow val(G,\vartheta) \in val(G,\pi)$
$\langle proof\rangle$

lemma *elem_of_val*: $x{\in}val(G,\pi) \Longrightarrow \exists\,\vartheta{\in}domain(\pi).\ val(G,\vartheta) = x$
$\langle proof\rangle$

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

lemma *elem_of_val_pair'*:
assumes $\pi{\in}M$ $x{\in}val(G,\pi)$
shows $\exists\,\vartheta{\in}M.\ \exists\, p{\in}G.\ \ \langle\vartheta,p\rangle{\in}\pi \wedge val(G,\vartheta) = x$
$\langle proof\rangle$

lemma *GenExtD*:
$x \in M[G] \Longrightarrow \exists\,\tau{\in}M.\ x = val(G,\tau)$
$\langle proof\rangle$

lemma *GenExtI*:
$x \in M \Longrightarrow val(G,x) \in M[G]$
$\langle proof\rangle$

lemma *Transset_MG* : $Transset(M[G])$
$\langle proof\rangle$

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

lemma *check_n_M* :
fixes n

assumes $n \in nat$
shows $check(n) \in M$
$\langle proof \rangle$

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: $one \in M$
$\langle proof \rangle$

lemma $def\_PHcheck$:
assumes
$z{\in}M\ f{\in}M$
shows
$Hcheck(z,f) = Replace(z,PHcheck(one,f))$
$\langle proof \rangle$

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 $PHcheck\_type$ $[TC]$:
$[\![x \in nat;\ y \in nat;\ z \in nat;\ u \in nat]\!] \Longrightarrow PHcheck\_fm(x,y,z,u) \in formula$
$\langle proof \rangle$

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

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 $[TC]$:
$[\![x \in nat;\ y \in nat;\ z \in nat;\ u \in nat]\!] \Longrightarrow is\_Hcheck\_fm(x,y,z,u) \in formula$
$\langle proof \rangle$

lemma $sats\_is\_Hcheck\_fm$ $[simp]$:

$[\![x \in nat; y \in nat; z \in nat; u \in nat ; env \in list(M)]\!]$
$\Longrightarrow sats(M,is\_Hcheck\_fm(x,y,z,u),env) \longleftrightarrow$
$is\_Hcheck(nth(x,env),nth(y,env),nth(z,env),nth(u,env))$
⟨proof⟩

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

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

lemma *univ_PHcheck* : $[\![z \in M ; f \in M]\!] \Longrightarrow univalent(\#\#M,z,PHcheck(one,f))$
⟨proof⟩

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

lemma *PHcheck_closed* :
$[\![z \in M ; f \in M ; x \in z; PHcheck(one,f,x,y)]\!] \Longrightarrow (\#\#M)(y)$
⟨proof⟩

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

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

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

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

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

end

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

lemma (**in** *M_trivial*) *singleton_abs*[*simp*] : $[\![\ M(x)\ ;\ M(s)\ ]\!] \Longrightarrow is\_singleton(M,x,s) \longleftrightarrow s = \{x\}$
  ⟨*proof*⟩

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

lemma *singleton_type*[*TC*] : $[\![\ x \in nat;\ y \in nat\ ]\!] \Longrightarrow singleton\_fm(x,y) \in formula$
  ⟨*proof*⟩

lemma *is_singleton_iff_sats*:
  $[\![\ nth(i,env) = x;\ nth(j,env) = y;$
      $i \in nat;\ j{\in}nat\ ;\ env \in list(A)]\!]$
    $\Longrightarrow is\_singleton(\#\#A,x,y) \longleftrightarrow sats(A,\ singleton\_fm(i,j),\ env)$
  ⟨*proof*⟩

context *forcing_data* **begin**

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

lemma *rcheck_abs* :
  $[\![\ x{\in}M\ ;\ r{\in}M\ ]\!] \Longrightarrow is\_rcheck(x,r) \longleftrightarrow r = rcheck(x)$
  ⟨*proof*⟩

schematic_goal *rcheck_fm_auto*:
  assumes
    $i \in nat\ j \in nat\ env \in list(M)$
  shows
    $is\_rcheck(nth(i,env),nth(j,env)) \longleftrightarrow sats(M,?rch(i,j),env)$
  ⟨*proof*⟩

⟨*ML*⟩

definition
  is_check :: $[i,i] \Rightarrow o$ **where**
  $is\_check(x,z) \equiv \exists\, rch{\in}M.\ is\_rcheck(x,rch) \wedge is\_wfrec(\#\#M,is\_Hcheck(one),rch,x,z)$

lemma *check_abs* :
  assumes
    $x\in M$ $z\in M$
  shows
    $is\_check(x,z) \longleftrightarrow z = check(x)$
$\langle proof \rangle$

definition
  $check\_fm :: [i,i,i] \Rightarrow i$ **where**
  $check\_fm(x,o,z) \equiv Exists(And(rcheck\_fm(1\#\#+x,0),$
    $is\_wfrec\_fm(is\_Hcheck\_fm(6\#\#+o,2,1,0),0,1\#\#+x,1\#\#+z)))$

lemma *check_fm_type*[*TC*] :
  $[\![x\in nat;o\in nat;z\in nat]\!] \Longrightarrow check\_fm(x,o,z)\in formula$
  $\langle proof \rangle$

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
    $sats(M,\ check\_fm(x,o,z),\ env) \longleftrightarrow is\_check(nth(x,env),nth(z,env))$
$\langle proof \rangle$

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

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

lemma *M_subset_MG* : $one \in G \Longrightarrow M \subseteq M[G]$
  $\langle proof \rangle$

The name for the generic filter

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

lemma *G_dot_in_M* :
  $G\_dot \in M$
$\langle proof \rangle$

lemma *val_G_dot* :
  assumes $G \subseteq P$

  one ∈ *G*
 shows *val*(*G*,*G_dot*) = *G*
⟨*proof*⟩

lemma *G_in_Gen_Ext* :
 assumes *G* ⊆ *P* **and** *one* ∈ *G*
 shows *G* ∈ *M*[*G*]
 ⟨*proof*⟩

lemma *fst_snd_closed*: *p*∈*M* ⟹ *fst*(*p*) ∈ *M* ∧ *snd*(*p*)∈ *M*
⟨*proof*⟩

end

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

lemma *zero_in_MG* :
 0 ∈ *M*[*G*]
⟨*proof*⟩

lemma *G_nonempty*: *G*≠*0*
⟨*proof*⟩

end
end

15 Well-founded relation on names

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

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

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

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

lemma (**in** *M_trivial*) *hcomp_abs*:
 assumes
  is_f_abs:⋀*a z*. *M*(*a*) ⟹ *M*(*z*) ⟹ *is_f*(*a*,*z*) ⟷ *z* = *f*(*a*) **and**
  is_g_abs:⋀*a z*. *M*(*a*) ⟹ *M*(*z*) ⟹ *is_g*(*a*,*z*) ⟷ *z* = *g*(*a*) **and**
  g_closed:⋀*a*. *M*(*a*) ⟹ *M*(*g*(*a*))

$M(a)$ $M(w)$
shows
$is\_hcomp(M,is\_f,is\_g,a,w) \longleftrightarrow w = f(g(a))$
$\langle proof \rangle$

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

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

$\langle proof \rangle$

definition
$ftype :: i{\Rightarrow}i$ **where**
$ftype \equiv fst$

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

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

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

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

definition $eclose\_n :: [i{\Rightarrow}i,i] \Rightarrow i$ **where**

eclose_n(*name*,*x*) = *eclose*({*name*(*x*)})

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

lemma *components_in_eclose* :
n1 ∈ *ecloseN*(⟨*f*,*n1*,*n2*,*c*⟩)
n2 ∈ *ecloseN*(⟨*f*,*n1*,*n2*,*c*⟩)
⟨*proof*⟩

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

lemma *ecloseNI1* :
assumes *x* ∈ *eclose*(*n1*) ∨ *x*∈*eclose*(*n2*)
shows *x* ∈ *ecloseN*(⟨*f*,*n1*,*n2*,*c*⟩)
⟨*proof*⟩

lemmas *ecloseNI* = *ecloseNI1*

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

definition
is_fst :: (*i*⇒*o*)⇒*i*⇒*i*⇒*o* **where**
is_fst(*M*,*x*,*t*) ≡ (∃ *z*[*M*]. *pair*(*M*,*t*,*z*,*x*)) ∨
(¬(∃ *z*[*M*]. ∃ *w*[*M*]. *pair*(*M*,*w*,*z*,*x*)) ∧ *empty*(*M*,*t*))

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

lemma *sats_fst_fm* :
⟦ *x* ∈ *nat*; *y* ∈ *nat*;*env* ∈ *list*(*A*) ⟧
⟹ *sats*(*A*, *fst_fm*(*x*,*y*), *env*) ⟷
is_fst(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
⟨*proof*⟩

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

definition

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

lemma $sats\_ftype\_fm$:
$[\![x \in nat; y \in nat; env \in list(A)]\!]$
$\Longrightarrow sats(A, ftype\_fm(x,y), env) \longleftrightarrow$
$is\_ftype(\#\#A, nth(x,env), nth(y,env))$
$\langle proof \rangle$

lemma $is\_ftype\_iff\_sats$:
assumes
$nth(a,env) = aa$ $nth(b,env) = bb$ $a \in nat$ $b \in nat$ $env \in list(A)$
shows
$is\_ftype(\#\#A,aa,bb) \longleftrightarrow sats(A,ftype\_fm(a,b), env)$
$\langle proof \rangle$

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 $sats\_snd\_fm$:
$[\![x \in nat; y \in nat; env \in list(A)]\!]$
$\Longrightarrow sats(A, snd\_fm(x,y), env) \longleftrightarrow$
$is\_snd(\#\#A, nth(x,env), nth(y,env))$
$\langle proof \rangle$

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 $sats\_name1\_fm$:
$[\![x \in nat; y \in nat; env \in list(A)]\!]$
$\Longrightarrow sats(A, name1\_fm(x,y), env) \longleftrightarrow$
$is\_name1(\#\#A, nth(x,env), nth(y,env))$
$\langle proof \rangle$

lemma $is\_name1\_iff\_sats$:
assumes
$nth(a,env) = aa$ $nth(b,env) = bb$ $a \in nat$ $b \in nat$ $env \in list(A)$

shows
$is\_name1(\#\#A,aa,bb) \longleftrightarrow sats(A,name1\_fm(a,b), env)$
⟨*proof*⟩

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* :
$[\![x \in nat; y \in nat; env \in list(A)]\!]$
$\Longrightarrow sats(A,snd\_snd\_fm(x,y), env) \longleftrightarrow$
$is\_snd\_snd(\#\#A, nth(x,env), nth(y,env))$
⟨*proof*⟩

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* :
$[\![x \in nat; y \in nat; env \in list(A)]\!]$
$\Longrightarrow sats(A,name2\_fm(x,y), env) \longleftrightarrow$
$is\_name2(\#\#A, nth(x,env), nth(y,env))$
⟨*proof*⟩

lemma *is_name2_iff_sats*:
assumes
$nth(a,env) = aa$ $nth(b,env) = bb$ $a \in nat$ $b \in nat$ $env \in list(A)$
shows
$is\_name2(\#\#A,aa,bb) \longleftrightarrow sats(A,name2\_fm(a,b), env)$
⟨*proof*⟩

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* :
$[\![x \in nat; y \in nat; env \in list(A)]\!]$

$\Longrightarrow$ *sats*(*A*,*cond_of_fm*(*x*,*y*), *env*) $\longleftrightarrow$
is_cond_of(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
⟨*proof*⟩

lemma *is_cond_of_iff_sats*:
assumes
nth(*a*,*env*) = *aa* *nth*(*b*,*env*) = *bb* *a*∈*nat* *b*∈*nat* *env* ∈ *list*(*A*)
shows
is_cond_of(##*A*,*aa*,*bb*) $\longleftrightarrow$ *sats*(*A*,*cond_of_fm*(*a*,*b*), *env*)
⟨*proof*⟩

lemma *components_type*[*TC*] :
assumes *a*∈*nat* *b*∈*nat*
shows
ftype_fm(*a*,*b*)∈*formula*
name1_fm(*a*,*b*)∈*formula*
name2_fm(*a*,*b*)∈*formula*
cond_of_fm(*a*,*b*)∈*formula*
⟨*proof*⟩

lemmas *sats_components_fm*[*simp*] = *sats_ftype_fm* *sats_name1_fm* *sats_name2_fm* *sats_cond_of_fm*

lemmas *components_iff_sats* = *is_ftype_iff_sats* *is_name1_iff_sats* *is_name2_iff_sats*
is_cond_of_iff_sats

lemmas *components_defs* = *fst_fm_def* *ftype_fm_def* *snd_fm_def* *snd_snd_fm_def* *hcomp_fm_def*
name1_fm_def *name2_fm_def* *cond_of_fm_def*

definition
is_eclose_n :: [*i*⇒*o*,[*i*⇒*o*,*i*,*i*]⇒*o*,*i*,*i*] ⇒ *o* **where**
is_eclose_n(*N*,*is_name*,*en*,*t*) ≡
∃ *n1*[*N*].∃ *s1*[*N*]. *is_name*(*N*,*t*,*n1*) ∧ *is_singleton*(*N*,*n1*,*s1*) ∧ *is_eclose*(*N*,*s1*,*en*)

definition
eclose_n1_fm :: [*i*,*i*] ⇒ *i* **where**
eclose_n1_fm(*m*,*t*) ≡ *Exists*(*Exists*(*And*(*And*(*name1_fm*(*t*#+*2*,*0*),*singleton_fm*(*0*,*1*)),
is_eclose_fm(*1*,*m*#+*2*))))

definition
eclose_n2_fm :: [*i*,*i*] ⇒ *i* **where**
eclose_n2_fm(*m*,*t*) ≡ *Exists*(*Exists*(*And*(*And*(*name2_fm*(*t*#+*2*,*0*),*singleton_fm*(*0*,*1*)),
is_eclose_fm(*1*,*m*#+*2*))))

definition
is_ecloseN :: [*i*⇒*o*,*i*,*i*] ⇒ *o* **where**
is_ecloseN(*N*,*en*,*t*) ≡ ∃ *en1*[*N*].∃ *en2*[*N*].

$is\_eclose\_n(N,is\_name1,en1,t) \wedge is\_eclose\_n(N,is\_name2,en2,t) \wedge$
$union(N,en1,en2,en)$

definition
$ecloseN\_fm :: [i,i] \Rightarrow i$ **where**
$ecloseN\_fm(en,t) \equiv Exists(Exists(And(eclose\_n1\_fm(1,t\#+2),$
$And(eclose\_n2\_fm(0,t\#+2),union\_fm(1,0,en\#+2)))))$

lemma $ecloseN\_fm\_type$ [TC] :
$[\![en \in nat ; t \in nat]\!] \Longrightarrow ecloseN\_fm(en,t) \in formula$
$\langle proof \rangle$

lemma $sats\_ecloseN\_fm$ [$simp$]:
$[\![en \in nat; t \in nat ; env \in list(A)]\!]$
$\Longrightarrow sats(A, ecloseN\_fm(en,t), env) \longleftrightarrow is\_ecloseN(\#\#A,nth(en,env),nth(t,env))$
$\langle proof \rangle$

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 $(ftype(x) = 0 \wedge ftype(y) = 1) \vee (ftype(x) = 1 \wedge ftype(y) = 0)$
$\langle proof \rangle$

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

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

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

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

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

$\langle proof \rangle$

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

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

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

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

lemma *frecR_DI* :
assumes $frecR(\langle a,b,c,d\rangle,\langle ftype(y),name1(y),name2(y),cond\_of(y)\rangle)$
shows $frecR(\langle a,b,c,d\rangle,y)$
$\langle proof \rangle$

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
$is\_frecR(\#\#A,a,b) \longleftrightarrow sats(A,?fr\_fm(i,j),env)$
$\langle proof \rangle$

⟨*ML*⟩

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

definition
  $rank\_names :: i \Rightarrow i$ **where**
  $rank\_names(x) \equiv max(rank(name1(x)),rank(name2(x)))$

lemma *rank_names_types* [*TC*]:
  shows $Ord(rank\_names(x))$
  ⟨*proof*⟩

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 $type\_form(x) \in 3$
  ⟨*proof*⟩

lemma *frecR_le_rnk_names* :
  assumes $frecR(x,y)$
  shows $rank\_names(x) \le rank\_names(y)$
⟨*proof*⟩

definition
  $\Gamma :: i \Rightarrow i$ **where**
  $\Gamma(x) = 3 ** rank\_names(x) ++ type\_form(x)$

lemma Γ*_type* [*TC*]:
  shows $Ord(\Gamma(x))$
  ⟨*proof*⟩

lemma Γ*_mono* :
  assumes $frecR(x,y)$
  shows $\Gamma(x) < \Gamma(y)$
⟨*proof*⟩

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 $\langle x,y\rangle\in frecrel(A)$
  $\langle proof\rangle$

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

lemma *wf_frecrel* :
  shows $wf(frecrel(A))$
$\langle proof\rangle$

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

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

lemma *frecrel_fst_snd*:
  $frecrel(A) = \{z \in A\times A\ .$
      $ftype(fst(z)) = 1\ \wedge$
      $ftype(snd(z)) = 0 \wedge name1(fst(z)) \in domain(name1(snd(z))) \cup domain(name2(snd(z))) \wedge$
      $(name2(fst(z)) = name1(snd(z)) \vee name2(fst(z)) = name2(snd(z)))$
     $\vee\ (ftype(fst(z)) = 0\ \wedge$
    $ftype(snd(z)) = 1 \wedge\ \ name1(fst(z)) = name1(snd(z)) \wedge name2(fst(z)) \in domain(name2(snd(z))))\}$
  $\langle proof\rangle$

end

16 Arities of internalized formulas

theory *Arities*
imports *FrecR*
begin

lemma *arity_upair_fm* : $[\![$ $t1 \in nat$; $t2 \in nat$; $up \in nat$ $]\!] \Longrightarrow$
$arity(upair\_fm(t1,t2,up)) = \bigcup \{succ(t1),succ(t2),succ(up)\}$
⟨*proof*⟩

lemma *arity_pair_fm* : $[\![$ $t1 \in nat$; $t2 \in nat$; $p \in nat$ $]\!] \Longrightarrow$
$arity(pair\_fm(t1,t2,p)) = \bigcup \{succ(t1),succ(t2),succ(p)\}$
⟨*proof*⟩

lemma *arity_composition_fm* :
$[\![$ $r \in nat$; $s \in nat$; $t \in nat$ $]\!] \Longrightarrow arity(composition\_fm(r,s,t)) = \bigcup \{succ(r), succ(s), succ(t)\}$
⟨*proof*⟩

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

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

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

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

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

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

lemma *arity_fun_apply_fm* :

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

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

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

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

lemma *number1arity__fm* :
$[\![r{\in}nat]\!] \Longrightarrow arity(number1\_fm(r)) = succ(r)$
⟨*proof*⟩

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

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

lemma *arity_restriction_fm* :
$[\![r{\in}nat ; z{\in}nat ; A{\in}nat]\!] \Longrightarrow arity(restriction\_fm(A,z,r)) = succ(A) \cup succ(r) \cup succ(z)$
⟨*proof*⟩

lemma *arity_typed_function_fm* :
$[\![x{\in}nat ; y{\in}nat ; f{\in}nat]\!] \Longrightarrow$
$arity(typed\_function\_fm(f,x,y)) = \bigcup \{succ(f), succ(x), succ(y)\}$
⟨*proof*⟩

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

lemma *arity_transset_fm* :
$[\![x{\in}nat]\!] \Longrightarrow arity(transset\_fm(x)) = succ(x)$
⟨*proof*⟩

lemma *arity_ordinal_fm* :
$[\![x\in nat]\!] \Longrightarrow arity(ordinal\_fm(x)) = succ(x)$
$\langle proof \rangle$

lemma *arity_limit_ordinal_fm* :
$[\![x\in nat]\!] \Longrightarrow arity(limit\_ordinal\_fm(x)) = succ(x)$
$\langle proof \rangle$

lemma *arity_finite_ordinal_fm* :
$[\![x\in nat]\!] \Longrightarrow arity(finite\_ordinal\_fm(x)) = succ(x)$
$\langle proof \rangle$

lemma *arity_omega_fm* :
$[\![x\in nat]\!] \Longrightarrow arity(omega\_fm(x)) = succ(x)$
$\langle proof \rangle$

lemma *arity_cartprod_fm* :
$[\![A\in nat ; B\in nat ; z\in nat]\!] \Longrightarrow arity(cartprod\_fm(A,B,z)) = succ(A) \cup succ(B) \cup succ(z)$
$\langle proof \rangle$

lemma *arity_fst_fm* :
$[\![x\in nat ; t\in nat]\!] \Longrightarrow arity(fst\_fm(x,t)) = succ(x) \cup succ(t)$
$\langle proof \rangle$

lemma *arity_snd_fm* :
$[\![x\in nat ; t\in nat]\!] \Longrightarrow arity(snd\_fm(x,t)) = succ(x) \cup succ(t)$
$\langle proof \rangle$

lemma *arity_snd_snd_fm* :
$[\![x\in nat ; t\in nat]\!] \Longrightarrow arity(snd\_snd\_fm(x,t)) = succ(x) \cup succ(t)$
$\langle proof \rangle$

lemma *arity_ftype_fm* :
$[\![x\in nat ; t\in nat]\!] \Longrightarrow arity(ftype\_fm(x,t)) = succ(x) \cup succ(t)$
$\langle proof \rangle$

lemma *name1arity__fm* :
$[\![x\in nat ; t\in nat]\!] \Longrightarrow arity(name1\_fm(x,t)) = succ(x) \cup succ(t)$
$\langle proof \rangle$

lemma *name2arity__fm* :
$[\![x\in nat ; t\in nat]\!] \Longrightarrow arity(name2\_fm(x,t)) = succ(x) \cup succ(t)$
$\langle proof \rangle$

lemma *arity_cond_of_fm* :
$[\![x\in nat ; t\in nat]\!] \Longrightarrow arity(cond\_of\_fm(x,t)) = succ(x) \cup succ(t)$
$\langle proof \rangle$

lemma *arity_singleton_fm* :
$[\![x\in nat\ ;\ t\in nat]\!] \Longrightarrow arity(singleton\_fm(x,t)) = succ(x) \cup succ(t)$
$\langle proof \rangle$

lemma *arity_Memrel_fm* :
$[\![x\in nat\ ;\ t\in nat]\!] \Longrightarrow arity(Memrel\_fm(x,t)) = succ(x) \cup succ(t)$
$\langle proof \rangle$

lemma *arity_quasinat_fm* :
$[\![x\in nat]\!] \Longrightarrow arity(quasinat\_fm(x)) = succ(x)$
$\langle proof \rangle$

lemma *arity_is_recfun_fm* :
$[\![p\in formula\ ;\ v\in nat\ ;\ n\in nat;\ Z\in nat;i\in nat]\!] \Longrightarrow\ arity(p) = i \Longrightarrow$
$arity(is\_recfun\_fm(p,v,n,Z)) = succ(v) \cup succ(n) \cup succ(Z) \cup pred(pred(pred(pred(i))))$
$\langle proof \rangle$

lemma *arity_is_wfrec_fm* :
$[\![p\in formula\ ;\ v\in nat\ ;\ n\in nat;\ Z\in nat\ ;\ i\in nat]\!] \Longrightarrow arity(p) = i \Longrightarrow$
$arity(is\_wfrec\_fm(p,v,n,Z)) = succ(v) \cup succ(n) \cup succ(Z) \cup pred(pred(pred(pred(pred(i)))))$
$\langle proof \rangle$

lemma *arity_is_nat_case_fm* :
$[\![p\in formula\ ;\ v\in nat\ ;\ n\in nat;\ Z\in nat;\ i\in nat]\!] \Longrightarrow arity(p) = i \Longrightarrow$
$arity(is\_nat\_case\_fm(v,p,n,Z)) = succ(v) \cup succ(n) \cup succ(Z) \cup pred(pred(i))$
$\langle proof \rangle$

lemma *arity_iterates_MH_fm* :
assumes $isF\in formula\ v\in nat\ n\in nat\ g\in nat\ z\in nat\ i\in nat$
$arity(isF) = i$
shows $arity(iterates\_MH\_fm(isF,v,n,g,z)) =$
$succ(v) \cup succ(n) \cup succ(g) \cup succ(z) \cup pred(pred(pred(pred(i))))$
$\langle proof \rangle$

lemma *arity_is_iterates_fm* :
assumes $p\in formula\ v\in nat\ n\in nat\ Z\in nat\ i\in nat$
$arity(p) = i$
shows $arity(is\_iterates\_fm(p,v,n,Z)) = succ(v) \cup succ(n) \cup succ(Z) \cup$
$pred(pred(pred(pred(pred(pred(pred(pred(pred(pred(pred(i)))))))))))$
$\langle proof \rangle$

lemma *arity_eclose_n_fm* :
assumes $A\in nat\ x\in nat\ t\in nat$
shows $arity(eclose\_n\_fm(A,x,t)) = succ(A) \cup succ(x) \cup succ(t)$
$\langle proof \rangle$

lemma *arity_mem_eclose_fm* :
assumes $x\in nat\ t\in nat$
shows $arity(mem\_eclose\_fm(x,t)) = succ(x) \cup succ(t)$

⟨*proof*⟩

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

lemma *eclose_n1arity___fm* :
  $[\![x\in nat ; t\in nat]\!] \Longrightarrow arity(eclose\_n1\_fm(x,t)) = succ(x) \cup succ(t)$
  ⟨*proof*⟩

lemma *eclose_n2arity___fm* :
  $[\![x\in nat ; t\in nat]\!] \Longrightarrow arity(eclose\_n2\_fm(x,t)) = succ(x) \cup succ(t)$
  ⟨*proof*⟩

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

lemma *arity_frecR_fm* :
  $[\![a\in nat;b\in nat]\!] \Longrightarrow arity(frecR\_fm(a,b)) = succ(a) \cup succ(b)$
  ⟨*proof*⟩

lemma *arity_Collect_fm* :
  assumes $x \in nat$ $y \in nat$ $p\in formula$
  shows $arity(Collect\_fm(x,p,y)) = succ(x) \cup succ(y) \cup pred(arity(p))$
  ⟨*proof*⟩

end

17 The definition of *forces*

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

This is the core of our development.

17.1 The relation *frecrel*

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

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

lemma *arity_frecrelP_fm* :
  $a\in nat \Longrightarrow arity(frecrelP\_fm(a)) = succ(a)$
  ⟨*proof*⟩

lemma *frecrelP_fm_type*[*TC*] :
  $a \in nat \Longrightarrow frecrelP\_fm(a) \in formula$
  ⟨*proof*⟩

lemma *sats_frecrelP_fm* :
  assumes $a \in nat$ $env \in list(A)$
  shows $sats(A, frecrelP\_fm(a), env) \longleftrightarrow frecrelP(\#\#A, nth(a, env))$
  ⟨*proof*⟩

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

definition
  $is\_frecrel :: [i \Rightarrow o,i,i] \Rightarrow o$ **where**
  $is\_frecrel(M,A,r) \equiv \exists A2[M].\ cartprod(M,A,A,A2) \wedge is\_Collect(M,A2, frecrelP(M)$
$,r)$

definition
  $frecrel\_fm :: [i,i] \Rightarrow i$ **where**
  $frecrel\_fm(a,r) \equiv Exists(And(cartprod\_fm(a\#+1,a\#+1,0),Collect\_fm(0,frecrelP\_fm(0),r\#+1)))$

lemma *frecrel_fm_type*[*TC*] :
  $[\![a \in nat; b \in nat]\!] \Longrightarrow frecrel\_fm(a,b) \in formula$
  ⟨*proof*⟩

lemma *arity_frecrel_fm* :
  assumes $a \in nat$ $b \in nat$
  shows $arity(frecrel\_fm(a,b)) = succ(a) \cup succ(b)$
  ⟨*proof*⟩

lemma *sats_frecrel_fm* :
  assumes
    $a \in nat$ $r \in nat$ $env \in list(A)$
  shows
    $sats(A,frecrel\_fm(a,r),env)$
    $\longleftrightarrow is\_frecrel(\#\#A,nth(a, env),nth(r, env))$
  ⟨*proof*⟩

lemma *is_frecrel_iff_sats*:
  assumes
    $nth(a,env) = aa$ $nth(r,env) = rr$ $a \in nat$ $r \in nat$ $env \in list(A)$
  shows
    $is\_frecrel(\#\#A, aa,rr) \longleftrightarrow sats(A, frecrel\_fm(a,r), env)$
  ⟨*proof*⟩

definition
  $names\_below :: i \Rightarrow i \Rightarrow i$ **where**
  $names\_below(P,x) \equiv 2\times ecloseN(x)\times ecloseN(x)\times P$

lemma $names\_belowsD$:
  assumes $x \in names\_below(P,z)$
  obtains $f\ n1\ n2\ p$ **where**
    $x = \langle f,n1,n2,p\rangle\ f\in 2\ n1\in ecloseN(z)\ n2\in ecloseN(z)\ p\in P$
  $\langle proof\rangle$

definition
  $is\_names\_below :: [i\Rightarrow o,i,i,i] \Rightarrow o$ **where**
  $is\_names\_below(M,P,x,nb) \equiv \exists p1[M].\ \exists p0[M].\ \exists t[M].\ \exists ec[M].$
    $is\_ecloseN(M,ec,x) \wedge number2(M,t) \wedge cartprod(M,ec,P,p0) \wedge cartprod(M,ec,p0,p1)$
    $\wedge\ cartprod(M,t,p1,nb)$

definition
  $number2\_fm :: i\Rightarrow i$ **where**
  $number2\_fm(a) \equiv Exists(And(number1\_fm(0),\ succ\_fm(0,succ(a))))$

lemma $number2\_fm\_type[TC]$:
  $a\in nat \Longrightarrow number2\_fm(a) \in formula$
  $\langle proof\rangle$

lemma $number2arity\_\_fm$:
  $a\in nat \Longrightarrow arity(number2\_fm(a)) = succ(a)$
  $\langle proof\rangle$

lemma $sats\_number2\_fm$ [$simp$]:
  $[\![\ x \in nat;\ env \in list(A)\ ]\!]$
    $\Longrightarrow sats(A,\ number2\_fm(x),\ env) \longleftrightarrow number2(\#\#A,\ nth(x,env))$
  $\langle proof\rangle$

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 $arity\_is\_names\_below\_fm$:
  $[\![P\in nat;x\in nat;nb\in nat]\!] \Longrightarrow arity(is\_names\_below\_fm(P,x,nb)) = succ(P) \cup succ(x) \cup succ(nb)$
  $\langle proof\rangle$

lemma $is\_names\_below\_fm\_type[TC]$:
  $[\![P\in nat;x\in nat;nb\in nat]\!] \Longrightarrow is\_names\_below\_fm(P,x,nb)\in formula$

⟨*proof*⟩

lemma *sats_is_names_below_fm* :
  assumes
    $P \in nat$ $x \in nat$ $nb \in nat$ $env \in list(A)$
  shows
    $sats(A,is\_names\_below\_fm(P,x,nb),env)$
    $\longleftrightarrow is\_names\_below(\#\#A,nth(P, env),nth(x, env),nth(nb, env))$
  ⟨*proof*⟩

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

definition
  $is\_tuple\_fm :: [i,i,i,i,i] \Rightarrow i$ **where**
  $is\_tuple\_fm(z,t1,t2,p,tup) = Exists(Exists(And(pair\_fm(t2 \#+ 2,p \#+ 2,0),$
    $And(pair\_fm(t1 \#+ 2,0,1),pair\_fm(z \#+ 2,1,tup \#+ 2)))))$

lemma *arity_is_tuple_fm* : $[\![z \in nat ; t1 \in nat ; t2 \in nat ; p \in nat ; tup \in nat]\!] \Longrightarrow$
  $arity(is\_tuple\_fm(z,t1,t2,p,tup)) = \bigcup \{succ(z),succ(t1),succ(t2),succ(p),succ(tup)\}$
  ⟨*proof*⟩

lemma *is_tuple_fm_type*[*TC*] :
  $z \in nat \Longrightarrow t1 \in nat \Longrightarrow t2 \in nat \Longrightarrow p \in nat \Longrightarrow tup \in nat \Longrightarrow is\_tuple\_fm(z,t1,t2,p,tup) \in formula$
  ⟨*proof*⟩

lemma *sats_is_tuple_fm* :
  assumes
    $z \in nat$ $t1 \in nat$ $t2 \in nat$ $p \in nat$ $tup \in nat$ $env \in list(A)$
  shows
    $sats(A,is\_tuple\_fm(z,t1,t2,p,tup),env)$
    $\longleftrightarrow is\_tuple(\#\#A,nth(z, env),nth(t1, env),nth(t2, env),nth(p, env),nth(tup,$
$env))$
  ⟨*proof*⟩

lemma *is_tuple_iff_sats*:
  assumes
    $nth(a,env) = aa$ $nth(b,env) = bb$ $nth(c,env) = cc$ $nth(d,env) = dd$ $nth(e,env)$
$= ee$
    $a \in nat$ $b \in nat$ $c \in nat$ $d \in nat$ $e \in nat$ $env \in list(A)$
  shows
    $is\_tuple(\#\#A,aa,bb,cc,dd,ee) \longleftrightarrow sats(A, is\_tuple\_fm(a,b,c,d,e), env)$
  ⟨*proof*⟩

17.2 Definition of *forces* for equality and membership

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

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

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

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

schematic_goal *sats_is_mem_case_fm_auto*:
assumes
$n1\in nat\ n2\in nat\ p\in nat\ P\in nat\ leq\in nat\ f\in nat\ env\in list(A)$
shows
$is\_mem\_case(\#\#A,\ nth(n1,\ env),nth(n2,\ env),nth(p,\ env),nth(P,\ env),\ nth(leq,\ env),nth(f,env))$
$\longleftrightarrow sats(A,?imc\_fm(n1,n2,p,P,leq,f),env)$
⟨*proof*⟩

⟨*ML*⟩

lemma *arity_mem_case_fm* :

assumes
n1∈*nat* *n2*∈*nat* *p*∈*nat* *P*∈*nat* *leq*∈*nat* *f*∈*nat*
shows
arity(*mem_case_fm*(*n1*,*n2*,*p*,*P*,*leq*,*f*)) =
succ(*n1*) ∪ *succ*(*n2*) ∪ *succ*(*p*) ∪ *succ*(*P*) ∪ *succ*(*leq*) ∪ *succ*(*f*)
⟨*proof*⟩

schematic_goal *sats_is_eq_case_fm_auto*:
assumes
n1∈*nat* *n2*∈*nat* *p*∈*nat* *P*∈*nat* *leq*∈*nat* *f*∈*nat* *env*∈*list*(*A*)
shows
is_eq_case(##*A*, *nth*(*n1*, *env*),*nth*(*n2*, *env*),*nth*(*p*, *env*),*nth*(*P*, *env*), *nth*(*leq*, *env*),*nth*(*f*,*env*))
⟷ *sats*(*A*,?*iec_fm*(*n1*,*n2*,*p*,*P*,*leq*,*f*),*env*)
⟨*proof*⟩

⟨*ML*⟩

lemma *arity_eq_case_fm* :
assumes
n1∈*nat* *n2*∈*nat* *p*∈*nat* *P*∈*nat* *leq*∈*nat* *f*∈*nat*
shows
arity(*eq_case_fm*(*n1*,*n2*,*p*,*P*,*leq*,*f*)) =
succ(*n1*) ∪ *succ*(*n2*) ∪ *succ*(*p*) ∪ *succ*(*P*) ∪ *succ*(*leq*) ∪ *succ*(*f*)
⟨*proof*⟩

definition
Hfrc :: [*i*,*i*,*i*,*i*] ⇒ *o* **where**
Hfrc(*P*,*leq*,*fnnc*,*f*) ≡ ∃*ft*. ∃*n1*. ∃*n2*. ∃*c*. *c*∈*P* ∧ *fnnc* = ⟨*ft*,*n1*,*n2*,*c*⟩ ∧
(*ft* = *0* ∧ *eq_case*(*n1*,*n2*,*c*,*P*,*leq*,*f*)
∨ *ft* = *1* ∧ *mem_case*(*n1*,*n2*,*c*,*P*,*leq*,*f*))

definition
is_Hfrc :: [*i*⇒*o*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_Hfrc(*M*,*P*,*leq*,*fnnc*,*f*) ≡
∃*ft*[*M*]. ∃*n1*[*M*]. ∃*n2*[*M*]. ∃*co*[*M*].
co∈*P* ∧ *is_tuple*(*M*,*ft*,*n1*,*n2*,*co*,*fnnc*) ∧
((*empty*(*M*,*ft*) ∧ *is_eq_case*(*M*,*n1*,*n2*,*co*,*P*,*leq*,*f*))
∨ (*number1*(*M*,*ft*) ∧ *is_mem_case*(*M*,*n1*,*n2*,*co*,*P*,*leq*,*f*)))

definition
Hfrc_fm :: [*i*,*i*,*i*,*i*] ⇒ *i* **where**
Hfrc_fm(*P*,*leq*,*fnnc*,*f*) ≡
Exists(*Exists*(*Exists*(*Exists*(
And(*Member*(*0*,*P* #+ *4*),*And*(*is_tuple_fm*(*3*,*2*,*1*,*0*,*fnnc* #+ *4*),
Or(*And*(*empty_fm*(*3*),*eq_case_fm*(*2*,*1*,*0*,*P* #+ *4*,*leq* #+ *4*,*f* #+ *4*)),
And(*number1_fm*(*3*),*mem_case_fm*(*2*,*1*,*0*,*P* #+ *4*,*leq* #+ *4*,*f* #+ *4*)))))))))

lemma *Hfrc_fm_type*[*TC*] :
  ⟦*P*∈*nat*;*leq*∈*nat*;*fnnc*∈*nat*;*f*∈*nat*⟧ ⟹ *Hfrc_fm*(*P*,*leq*,*fnnc*,*f*)∈*formula*
  ⟨*proof*⟩

lemma *arity_Hfrc_fm* :
  assumes
    P∈*nat* *leq*∈*nat* *fnnc*∈*nat* *f*∈*nat*
  shows
    arity(*Hfrc_fm*(*P*,*leq*,*fnnc*,*f*)) = *succ*(*P*) ∪ *succ*(*leq*) ∪ *succ*(*fnnc*) ∪ *succ*(*f*)
  ⟨*proof*⟩

lemma *sats_Hfrc_fm*:
  assumes
    P∈*nat* *leq*∈*nat* *fnnc*∈*nat* *f*∈*nat* *env*∈*list*(*A*)
  shows
    sats(*A*,*Hfrc_fm*(*P*,*leq*,*fnnc*,*f*),*env*)
    ⟷ *is_Hfrc*(##*A*,*nth*(*P*, *env*), *nth*(*leq*, *env*), *nth*(*fnnc*, *env*),*nth*(*f*, *env*))
  ⟨*proof*⟩

lemma *Hfrc_iff_sats*:
  assumes
    P∈*nat* *leq*∈*nat* *fnnc*∈*nat* *f*∈*nat* *env*∈*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*)
    ⟷ *sats*(*A*,*Hfrc_fm*(*P*,*leq*,*fnnc*,*f*),*env*)
  ⟨*proof*⟩

definition
  is_Hfrc_at :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
  is_Hfrc_at(*M*,*P*,*leq*,*fnnc*,*f*,*z*) ≡
            (*empty*(*M*,*z*) ∧ ¬ *is_Hfrc*(*M*,*P*,*leq*,*fnnc*,*f*))
          ∨ (*number1*(*M*,*z*) ∧ *is_Hfrc*(*M*,*P*,*leq*,*fnnc*,*f*))

definition
  Hfrc_at_fm :: [*i*,*i*,*i*,*i*,*i*] ⇒ *i* **where**
  Hfrc_at_fm(*P*,*leq*,*fnnc*,*f*,*z*) ≡ *Or*(*And*(*empty_fm*(*z*),*Neg*(*Hfrc_fm*(*P*,*leq*,*fnnc*,*f*))),
                                    And(*number1_fm*(*z*),*Hfrc_fm*(*P*,*leq*,*fnnc*,*f*)))

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

lemma *Hfrc_at_fm_type*[*TC*] :

⟦$P\in nat;leq\in nat;fnnc\in nat;f\in nat;z\in nat$⟧ $\Longrightarrow$ *Hfrc_at_fm*(P,*leq*,*fnnc*,f,z)$\in$*formula*
⟨*proof*⟩

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*))
⟨*proof*⟩

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*) = *ffnnc*
nth(f,*env*) = *ff* *nth*(z,*env*) = *zz*
shows
is_Hfrc_at(##A, *PP*, *lleq*,*ffnnc*,*ff*,*zz*)
$\longleftrightarrow$ *sats*(A,*Hfrc_at_fm*(P,*leq*,*fnnc*,f,z),*env*)
⟨*proof*⟩

lemma *arity_tran_closure_fm* :
⟦$x\in nat;f\in nat$⟧ $\Longrightarrow$ *arity*(*trans_closure_fm*(x,f)) = *succ*(x) $\cup$ *succ*(f)
⟨*proof*⟩

17.3 The well-founded relation *forcerel*

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

definition
is_forcerel :: [$i\Rightarrow o$,i,i,i] $\Rightarrow$ o **where**
is_forcerel(M,P,x,z) $\equiv$ $\exists$ r[M]. $\exists$ *nb*[M]. *tran_closure*(M,r,z) $\wedge$
(*is_names_below*(M,P,x,*nb*) $\wedge$ *is_frecrel*(M,*nb*,r))

definition
forcerel_fm :: $i\Rightarrow$ $i \Rightarrow i \Rightarrow i$ **where**
forcerel_fm(p,x,z) $\equiv$ *Exists*(*Exists*(*And*(*trans_closure_fm*(*1*, z#+*2*),
And(*is_names_below_fm*(p#+*2*,x#+*2*,*0*),*frecrel_fm*(*0*,*1*)))))

lemma *arity_forcerel_fm*:
⟦$p\in nat;x\in nat;z\in nat$⟧ $\Longrightarrow$ *arity*(*forcerel_fm*(p,x,z)) = *succ*(p) $\cup$ *succ*(x) $\cup$ *succ*(z)
⟨*proof*⟩

lemma *forcerel_fm_type*[*TC*]:
⟦$p\in nat;x\in nat;z\in nat$⟧ $\Longrightarrow$ *forcerel_fm*(p,x,z)$\in$*formula*
⟨*proof*⟩

lemma *sats_forcerel_fm*:
  assumes
    $p \in nat$ $x \in nat$ $z \in nat$ $env \in list(A)$
  shows
    $sats(A,forcerel\_fm(p,x,z),env) \longleftrightarrow is\_forcerel(\#\#A,nth(p,env),nth(x, env),nth(z, env))$
$\langle proof \rangle$

17.4 *frc_at*, forcing for atomic formulas

definition
  $frc\_at :: [i,i,i] \Rightarrow i$ **where**
  $frc\_at(P,leq,fnnc) \equiv wfrec(frecrel(names\_below(P,fnnc)),fnnc,$
                        $\lambda x\, f.\ bool\_of\_o(Hfrc(P,leq,x,f)))$

definition
  $is\_frc\_at :: [i \Rightarrow o,i,i,i,i] \Rightarrow o$ **where**
  $is\_frc\_at(M,P,leq,x,z) \equiv \exists\, r[M].\ is\_forcerel(M,P,x,r) \wedge$
                        $is\_wfrec(M,is\_Hfrc\_at(M,P,leq),r,x,z)$

definition
  $frc\_at\_fm :: [i,i,i,i] \Rightarrow i$ **where**
  $frc\_at\_fm(p,l,x,z) \equiv Exists(And(forcerel\_fm(succ(p),succ(x),0),$
        $is\_wfrec\_fm(Hfrc\_at\_fm(6\#\!+p,6\#\!+l,2,1,0),0,succ(x),succ(z))))$

lemma *frc_at_fm_type* [*TC*] :
  $[\![p \in nat;l \in nat;x \in nat;z \in nat]\!] \Longrightarrow frc\_at\_fm(p,l,x,z) \in formula$
  $\langle proof \rangle$

lemma *arity_frc_at_fm* :
  assumes $p \in nat$ $l \in nat$ $x \in nat$ $z \in nat$
  shows $arity(frc\_at\_fm(p,l,x,z)) = succ(p) \cup succ(l) \cup succ(x) \cup succ(z)$
$\langle proof \rangle$

lemma *sats_frc_at_fm* :
  assumes
    $p \in nat$ $l \in nat$ $i \in nat$ $j \in nat$ $env \in list(A)$ $i < length(env)$ $j < length(env)$
  shows
    $sats(A,frc\_at\_fm(p,l,i,j),env) \longleftrightarrow$
     $is\_frc\_at(\#\#A,nth(p,env),nth(l,env),nth(i,env),nth(j,env))$
$\langle proof \rangle$

definition
  $forces\_eq' :: [i,i,i,i,i] \Rightarrow o$ **where**
  $forces\_eq'(P,l,p,t1,t2) \equiv frc\_at(P,l,\langle 0,t1,t2,p\rangle) = 1$

definition

$forces\_mem' :: [i,i,i,i,i] \Rightarrow o$ **where**
$forces\_mem'(P,l,p,t1,t2) \equiv frc\_at(P,l,\langle 1,t1,t2,p\rangle) = 1$

definition
$forces\_neq' :: [i,i,i,i,i] \Rightarrow o$ **where**
$forces\_neq'(P,l,p,t1,t2) \equiv \neg\ (\exists\, q{\in}P.\ \langle q,p\rangle{\in}l \wedge forces\_eq'(P,l,q,t1,t2))$

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
$is\_forces\_neq' :: [i{\Rightarrow}o,i,i,i,i,i] \Rightarrow o$ **where**
$is\_forces\_neq'(M,P,l,p,t1,t2) \equiv$
$\neg\ (\exists\, q[M].\ q{\in}P \wedge (\exists\, qp[M].\ pair(M,q,p,qp) \wedge qp{\in}l \wedge is\_forces\_eq'(M,P,l,q,t1,t2)))$

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
$forces\_neq\_fm :: [i,i,i,i,i] \Rightarrow i$ **where**
$forces\_neq\_fm(p,l,q,t1,t2) \equiv Neg(Exists(Exists(And(Member(1,p\#{+}2),$
$And(pair\_fm(1,q\#{+}2,0),And(Member(0,l\#{+}2),forces\_eq\_fm(p\#{+}2,l\#{+}2,1,t1\#{+}2,t2\#{+}2)))))))$

definition
forces_nmem_fm :: [*i*,*i*,*i*,*i*,*i*] ⇒ *i* **where**
forces_nmem_fm(*p*,*l*,*q*,*t1*,*t2*) ≡ *Neg*(*Exists*(*Exists*(*And*(*Member*(*1*,*p*#+*2*),
And(*pair_fm*(*1*,*q*#+*2*,*0*),*And*(*Member*(*0*,*l*#+*2*),*forces_mem_fm*(*p*#+*2*,*l*#+*2*,*1*,*t1*#+*2*,*t2*#+*2*)))))))

lemma *forces_eq_fm_type* [*TC*]:
⟦ *p*∈*nat*;*l*∈*nat*;*q*∈*nat*;*t1*∈*nat*;*t2*∈*nat*⟧ ⟹ *forces_eq_fm*(*p*,*l*,*q*,*t1*,*t2*) ∈ *formula*
⟨*proof*⟩

lemma *forces_mem_fm_type* [*TC*]:
⟦ *p*∈*nat*;*l*∈*nat*;*q*∈*nat*;*t1*∈*nat*;*t2*∈*nat*⟧ ⟹ *forces_mem_fm*(*p*,*l*,*q*,*t1*,*t2*) ∈ *formula*
⟨*proof*⟩

lemma *forces_neq_fm_type* [*TC*]:
⟦ *p*∈*nat*;*l*∈*nat*;*q*∈*nat*;*t1*∈*nat*;*t2*∈*nat*⟧ ⟹ *forces_neq_fm*(*p*,*l*,*q*,*t1*,*t2*) ∈ *formula*
⟨*proof*⟩

lemma *forces_nmem_fm_type* [*TC*]:
⟦ *p*∈*nat*;*l*∈*nat*;*q*∈*nat*;*t1*∈*nat*;*t2*∈*nat*⟧ ⟹ *forces_nmem_fm*(*p*,*l*,*q*,*t1*,*t2*) ∈ *formula*
⟨*proof*⟩

lemma *arity_forces_eq_fm* :
p∈*nat* ⟹ *l*∈*nat* ⟹ *q*∈*nat* ⟹ *t1* ∈ *nat* ⟹ *t2* ∈ *nat* ⟹
arity(*forces_eq_fm*(*p*,*l*,*q*,*t1*,*t2*)) = *succ*(*t1*) ∪ *succ*(*t2*) ∪ *succ*(*q*) ∪ *succ*(*p*) ∪ *succ*(*l*)
⟨*proof*⟩

lemma *arity_forces_mem_fm* :
p∈*nat* ⟹ *l*∈*nat* ⟹ *q*∈*nat* ⟹ *t1* ∈ *nat* ⟹ *t2* ∈ *nat* ⟹
arity(*forces_mem_fm*(*p*,*l*,*q*,*t1*,*t2*)) = *succ*(*t1*) ∪ *succ*(*t2*) ∪ *succ*(*q*) ∪ *succ*(*p*) ∪ *succ*(*l*)
⟨*proof*⟩

lemma *sats_forces_eq'_fm*:
assumes *p*∈*nat* *l*∈*nat* *q*∈*nat* *t1*∈*nat* *t2*∈*nat* *env*∈*list*(*M*)
shows *sats*(*M*,*forces_eq_fm*(*p*,*l*,*q*,*t1*,*t2*),*env*) ⟷
is_forces_eq'(##*M*,*nth*(*p*,*env*),*nth*(*l*,*env*),*nth*(*q*,*env*),*nth*(*t1*,*env*),*nth*(*t2*,*env*))
⟨*proof*⟩

lemma *sats_forces_mem'_fm*:
assumes *p*∈*nat* *l*∈*nat* *q*∈*nat* *t1*∈*nat* *t2*∈*nat* *env*∈*list*(*M*)
shows *sats*(*M*,*forces_mem_fm*(*p*,*l*,*q*,*t1*,*t2*),*env*) ⟷
is_forces_mem'(##*M*,*nth*(*p*,*env*),*nth*(*l*,*env*),*nth*(*q*,*env*),*nth*(*t1*,*env*),*nth*(*t2*,*env*))
⟨*proof*⟩

lemma *sats_forces_neq'_fm*:
assumes *p*∈*nat* *l*∈*nat* *q*∈*nat* *t1*∈*nat* *t2*∈*nat* *env*∈*list*(*M*)

shows $sats(M,forces\_neq\_fm(p,l,q,t1,t2),env) \longleftrightarrow$
$is\_forces\_neq'(\#\#M,nth(p,env),nth(l,env),nth(q,env),nth(t1,env),nth(t2,env))$
$\langle proof \rangle$

lemma $sats\_forces\_nmem'\_fm$:
assumes $p \in nat$ $l \in nat$ $q \in nat$ $t1 \in nat$ $t2 \in nat$ $env \in list(M)$
shows $sats(M,forces\_nmem\_fm(p,l,q,t1,t2),env) \longleftrightarrow$
$is\_forces\_nmem'(\#\#M,nth(p,env),nth(l,env),nth(q,env),nth(t1,env),nth(t2,env))$
$\langle proof \rangle$

context *forcing_data*
begin

lemma fst_abs $[simp]$:
$[\![x \in M;\ y \in M\ ]\!] \Longrightarrow is\_fst(\#\#M,x,y) \longleftrightarrow y = fst(x)$
$\langle proof \rangle$

lemma snd_abs $[simp]$:
$[\![x \in M;\ y \in M\ ]\!] \Longrightarrow is\_snd(\#\#M,x,y) \longleftrightarrow y = snd(x)$
$\langle proof \rangle$

lemma $ftype\_abs[simp]$:
$[\![x \in M;\ y \in M\ ]\!] \Longrightarrow is\_ftype(\#\#M,x,y) \longleftrightarrow y = ftype(x)$ $\langle proof \rangle$

lemma $name1\_abs[simp]$:
$[\![x \in M;\ y \in M\ ]\!] \Longrightarrow is\_name1(\#\#M,x,y) \longleftrightarrow y = name1(x)$
$\langle proof \rangle$

lemma snd_snd_abs:
$[\![x \in M;\ y \in M\ ]\!] \Longrightarrow is\_snd\_snd(\#\#M,x,y) \longleftrightarrow y = snd(snd(x))$
$\langle proof \rangle$

lemma $name2\_abs[simp]$:
$[\![x \in M;\ y \in M\ ]\!] \Longrightarrow is\_name2(\#\#M,x,y) \longleftrightarrow y = name2(x)$
$\langle proof \rangle$

lemma $cond\_of\_abs[simp]$:
$[\![x \in M;\ y \in M\ ]\!] \Longrightarrow is\_cond\_of(\#\#M,x,y) \longleftrightarrow y = cond\_of(x)$
$\langle proof \rangle$

lemma $tuple\_abs[simp]$:
$[\![z \in M;t1 \in M;t2 \in M;p \in M;t \in M]\!] \Longrightarrow$
$is\_tuple(\#\#M,z,t1,t2,p,t) \longleftrightarrow t = \langle z,t1,t2,p \rangle$
$\langle proof \rangle$

lemma $oneN\_in\_M[simp]$: $1 \in M$
$\langle proof \rangle$

lemma *twoN_in_M* : $2{\in}M$
⟨*proof*⟩

lemma *comp_in_M*:
$p \preceq q \Longrightarrow p{\in}M$
$p \preceq q \Longrightarrow q{\in}M$
⟨*proof*⟩

lemma *eq_case_abs* [*simp*]:
assumes
$t1{\in}M$ $t2{\in}M$ $p{\in}M$ $f{\in}M$
shows
is_eq_case(##M,t1,t2,p,P,leq,f) ⟷ *eq_case*(t1,t2,p,P,leq,f)
⟨*proof*⟩

lemma *mem_case_abs* [*simp*]:
assumes
$t1{\in}M$ $t2{\in}M$ $p{\in}M$ $f{\in}M$
shows
is_mem_case(##M,t1,t2,p,P,leq,f) ⟷ *mem_case*(t1,t2,p,P,leq,f)
⟨*proof*⟩

lemma *Hfrc_abs*:
⟦*fnnc*∈*M*; *f*∈*M*⟧ ⟹
is_Hfrc(##M,P,leq,fnnc,f) ⟷ *Hfrc*(P,leq,fnnc,f)
⟨*proof*⟩

lemma *Hfrc_at_abs*:
⟦*fnnc*∈*M*; *f*∈*M* ; *z*∈*M*⟧ ⟹
is_Hfrc_at(##M,P,leq,fnnc,f,z) ⟷ z = *bool_of_o*(*Hfrc*(P,leq,fnnc,f))
⟨*proof*⟩

lemma *components_closed* :
x∈*M* ⟹ *ftype*(x)∈M
x∈*M* ⟹ *name1*(x)∈M
x∈*M* ⟹ *name2*(x)∈M
x∈*M* ⟹ *cond_of*(x)∈M
⟨*proof*⟩

lemma *ecloseN_closed*:
(##M)(A) ⟹ (##M)(*ecloseN*(A))
(##M)(A) ⟹ (##M)(*eclose_n*(*name1*,A))
(##M)(A) ⟹ (##M)(*eclose_n*(*name2*,A))
⟨*proof*⟩

lemma *is_eclose_n_abs* :

assumes $x{\in}M$ $ec{\in}M$
shows $is\_eclose\_n({\#\#}M,is\_name1,ec,x) \longleftrightarrow ec = eclose\_n(name1,x)$
$is\_eclose\_n({\#\#}M,is\_name2,ec,x) \longleftrightarrow ec = eclose\_n(name2,x)$
⟨*proof*⟩

lemma *is_ecloseN_abs* :
$[\![x{\in}M;ec{\in}M]\!] \Longrightarrow is\_ecloseN({\#\#}M,ec,x) \longleftrightarrow ec = ecloseN(x)$
⟨*proof*⟩

lemma *frecR_abs* :
$x{\in}M \Longrightarrow y{\in}M \Longrightarrow frecR(x,y) \longleftrightarrow is\_frecR({\#\#}M,x,y)$
⟨*proof*⟩

lemma *frecrelP_abs* :
$z{\in}M \Longrightarrow frecrelP({\#\#}M,z) \longleftrightarrow (\exists\, x\ y.\ z = \langle x,y\rangle \wedge frecR(x,y))$
⟨*proof*⟩

lemma *frecrel_abs*:
assumes
$A{\in}M$ $r{\in}M$
shows
$is\_frecrel({\#\#}M,A,r) \longleftrightarrow\ \ r = frecrel(A)$
⟨*proof*⟩

lemma *frecrel_closed*:
assumes
$x{\in}M$
shows
$frecrel(x){\in}M$
⟨*proof*⟩

lemma *field_frecrel* : $field(frecrel(names\_below(P,x))) \subseteq names\_below(P,x)$
⟨*proof*⟩

lemma *forcerelD* : $uv \in forcerel(P,x) \Longrightarrow uv{\in}\ names\_below(P,x) \times names\_below(P,x)$
⟨*proof*⟩

lemma *wf_forcerel* :
$wf(forcerel(P,x))$
⟨*proof*⟩

lemma *restrict_trancl_forcerel*:
assumes $frecR(w,y)$
shows $restrict(f,frecrel(names\_below(P,x))\text{-}``\{y\})`w$
$= restrict(f,forcerel(P,x)\text{-}``\{y\})`w$
⟨*proof*⟩

lemma *names_belowI* :

assumes $frecR(\langle ft,n1,n2,p\rangle,\langle a,b,c,d\rangle)$ $p\in P$
shows $\langle ft,n1,n2,p\rangle \in names\_below(P,\langle a,b,c,d\rangle)$ (**is** $?x \in names\_below(\_,?y)$)
$\langle proof\rangle$

lemma *names_below_tr* :
assumes $x\in names\_below(P,y)$
$y\in names\_below(P,z)$
shows $x\in names\_below(P,z)$
$\langle proof\rangle$

lemma *arg_into_names_below2* :
assumes $\langle x,y\rangle \in frecrel(names\_below(P,z))$
shows $x \in names\_below(P,y)$
$\langle proof\rangle$

lemma *arg_into_names_below* :
assumes $\langle x,y\rangle \in frecrel(names\_below(P,z))$
shows $x \in names\_below(P,x)$
$\langle proof\rangle$

lemma *forcerel_arg_into_names_below* :
assumes $\langle x,y\rangle \in forcerel(P,z)$
shows $x \in names\_below(P,x)$
$\langle proof\rangle$

lemma *names_below_mono* :
assumes $\langle x,y\rangle \in frecrel(names\_below(P,z))$
shows $names\_below(P,x) \subseteq names\_below(P,y)$
$\langle proof\rangle$

lemma *frecrel_mono* :
assumes $\langle x,y\rangle \in frecrel(names\_below(P,z))$
shows $frecrel(names\_below(P,x)) \subseteq frecrel(names\_below(P,y))$
$\langle proof\rangle$

lemma *forcerel_mono2* :
assumes $\langle x,y\rangle \in frecrel(names\_below(P,z))$
shows $forcerel(P,x) \subseteq forcerel(P,y)$
$\langle proof\rangle$

lemma *forcerel_mono_aux* :
assumes $\langle x,y\rangle \in frecrel(names\_below(P, w))\hat{}+$
shows $forcerel(P,x) \subseteq forcerel(P,y)$
$\langle proof\rangle$

lemma *forcerel_mono* :
assumes $\langle x,y\rangle \in forcerel(P,z)$
shows $forcerel(P,x) \subseteq forcerel(P,y)$
$\langle proof\rangle$

lemma *aux*: $x \in names\_below(P,\ w) \Longrightarrow \langle x,y\rangle \in forcerel(P,z) \Longrightarrow$
$(y \in names\_below(P,\ w) \longrightarrow \langle x,y\rangle \in forcerel(P,w))$
⟨*proof*⟩

lemma *forcerel_eq* :
assumes $\langle z,x\rangle \in forcerel(P,x)$
shows $forcerel(P,z) = forcerel(P,x) \cap names\_below(P,z) \times names\_below(P,z)$
⟨*proof*⟩

lemma *forcerel_below_aux* :
assumes $\langle z,x\rangle \in forcerel(P,x)$ $\langle u,z\rangle \in forcerel(P,x)$
shows $u \in names\_below(P,z)$
⟨*proof*⟩

lemma *forcerel_below* :
assumes $\langle z,x\rangle \in forcerel(P,x)$
shows $forcerel(P,x)$ -`` $\{z\} \subseteq names\_below(P,z)$
⟨*proof*⟩

lemma *relation_forcerel* :
shows $relation(forcerel(P,z))$ $trans(forcerel(P,z))$
⟨*proof*⟩

lemma *Hfrc_restrict_trancl*: $bool\_of\_o(Hfrc(P,\ leq,\ y,\ restrict(f,frecrel(names\_below(P,x))$-``$\{y\})))$
$= bool\_of\_o(Hfrc(P,\ leq,\ y,\ restrict(f,(frecrel(names\_below(P,x))\hat{}+)$-``$\{y\})))$
⟨*proof*⟩

lemma *frc_at_trancl*: $frc\_at(P,leq,z) = wfrec(forcerel(P,z),z,\lambda x\,f.\ bool\_of\_o(Hfrc(P,leq,x,f)))$
⟨*proof*⟩

lemma *forcerelI1* :
assumes $n1 \in domain(b) \lor n1 \in domain(c)$ $p \in P$ $d \in P$
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*⟩

lemma *forcerelI2* :
assumes $n1 \in domain(b) \lor n1 \in domain(c)$ $p \in P$ $d \in P$
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*⟩

lemma *forcerelI3* :
assumes $\langle n2,\ r\rangle \in c$ $p \in P$ $d \in P$ $r \in P$
shows $\langle\langle 0,\ b,\ n2,\ p\rangle,\langle 1,\ b,\ c,\ d\rangle\rangle \in forcerel(P,\langle 1,b,c,d\rangle)$
⟨*proof*⟩

lemmas *forcerelI* = *forcerelI1*[*THEN vimage_singleton_iff*[*THEN iffD2*]]

forcerelI2[*THEN vimage_singleton_iff*[*THEN iffD2*]]
forcerelI3[*THEN vimage_singleton_iff*[*THEN iffD2*]]

lemma *aux_def_frc_at*:
assumes $z \in forcerel(P,x)$ -‘‘ $\{x\}$
shows $wfrec(forcerel(P,x),\ z,\ H) =\ wfrec(forcerel(P,z),\ z,\ H)$
⟨*proof*⟩

17.5 Recursive expression of *frc_at*

lemma *def_frc_at* :
assumes $p \in P$
shows
$frc\_at(P,leq,\langle ft,n1,n2,p\rangle) =$
$bool\_of\_o(\ p \in P \wedge$
$(\ \ ft = 0 \wedge\ (\forall s.\ s \in domain(n1) \cup domain(n2) \longrightarrow$
$(\forall q.\ q \in P \wedge q \preceq p \longrightarrow (frc\_at(P,leq,\langle 1,s,n1,q\rangle) = 1 \longleftrightarrow frc\_at(P,leq,\langle 1,s,n2,q\rangle)$
$= 1)))$
$\vee ft = 1 \wedge (\ \forall v \in P.\ v \preceq p \longrightarrow$
$(\exists q.\ \exists s.\ \exists r.\ r \in P \wedge q \in P \wedge q \preceq v \wedge \langle s,r\rangle \in n2 \wedge q \preceq r \wedge\ frc\_at(P,leq,\langle 0,n1,s,q\rangle)$
$= 1))))$
⟨*proof*⟩

17.6 Absoluteness of *frc_at*

lemma *trans_forcerel_t* : $trans(forcerel(P,x))$
⟨*proof*⟩

lemma *relation_forcerel_t* : $relation(forcerel(P,x))$
⟨*proof*⟩

lemma *forcerel_in_M* :
assumes
$x \in M$
shows
$forcerel(P,x) \in M$
⟨*proof*⟩

lemma *relation2_Hfrc_at_abs*:
$relation2(\#\#M,is\_Hfrc\_at(\#\#M,P,leq),\lambda x\ f.\ bool\_of\_o(Hfrc(P,leq,x,f)))$
⟨*proof*⟩

lemma *Hfrc_at_closed* :
$\forall x \in M.\ \forall g \in M.\ function(g) \longrightarrow bool\_of\_o(Hfrc(P,leq,x,g)) \in M$
⟨*proof*⟩

lemma *wfrec_Hfrc_at* :
assumes
$X \in M$

shows
wfrec_replacement(##*M*,*is_Hfrc_at*(##*M*,*P*,*leq*),*forcerel*(*P*,*X*))
⟨*proof*⟩

lemma *names_below_abs* :
⟦*Q*∈*M*;*x*∈*M*;*nb*∈*M*⟧ ⟹ *is_names_below*(##*M*,*Q*,*x*,*nb*) ⟷ *nb* = *names_below*(*Q*,*x*)
⟨*proof*⟩

lemma *names_below_closed*:
⟦*Q*∈*M*;*x*∈*M*⟧ ⟹ *names_below*(*Q*,*x*) ∈ *M*
⟨*proof*⟩

lemma *names_below_productE* :
assumes *Q* ∈ *M* *x* ∈ *M*
⋀*A1 A2 A3 A4*. *A1* ∈ *M* ⟹ *A2* ∈ *M* ⟹ *A3* ∈ *M* ⟹ *A4* ∈ *M* ⟹ *R*(*A1* × *A2* × *A3* × *A4*)
shows *R*(*names_below*(*Q*,*x*))
⟨*proof*⟩

lemma *forcerel_abs* :
⟦*x*∈*M*;*z*∈*M*⟧ ⟹ *is_forcerel*(##*M*,*P*,*x*,*z*) ⟷ *z* = *forcerel*(*P*,*x*)
⟨*proof*⟩

lemma *frc_at_abs*:
assumes *fnnc*∈*M* *z*∈*M*
shows *is_frc_at*(##*M*,*P*,*leq*,*fnnc*,*z*) ⟷ *z* = *frc_at*(*P*,*leq*,*fnnc*)
⟨*proof*⟩

lemma *forces_eq'_abs* :
⟦*p*∈*M* ; *t1*∈*M* ; *t2*∈*M*⟧ ⟹ *is_forces_eq'*(##*M*,*P*,*leq*,*p*,*t1*,*t2*) ⟷ *forces_eq'*(*P*,*leq*,*p*,*t1*,*t2*)
⟨*proof*⟩

lemma *forces_mem'_abs* :
⟦*p*∈*M* ; *t1*∈*M* ; *t2*∈*M*⟧ ⟹ *is_forces_mem'*(##*M*,*P*,*leq*,*p*,*t1*,*t2*) ⟷ *forces_mem'*(*P*,*leq*,*p*,*t1*,*t2*)
⟨*proof*⟩

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*⟩

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*⟩

end

17.7 Forcing for general formulas

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

lemma *ren_forces_nand_type*[*TC*] :
φ∈*formula* ⟹ *ren_forces_nand*(φ) ∈*formula*
⟨*proof*⟩

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

lemma *sats_ren_forces_nand*:
[*q*,*P*,*leq*,*o*,*p*] @ *env* ∈ *list*(*M*) ⟹ φ∈*formula* ⟹
sats(*M*, *ren_forces_nand*(φ),[*q*,*p*,*P*,*leq*,*o*] @ *env*) ⟷ *sats*(*M*, φ,[*q*,*P*,*leq*,*o*] @ *env*)
⟨*proof*⟩

definition
ren_forces_forall :: *i*⇒*i* **where**
ren_forces_forall(φ) ≡
Exists(*Exists*(*Exists*(*Exists*(*Exists*(
And(*Equal*(*0*,*6*),*And*(*Equal*(*1*,*7*),*And*(*Equal*(*2*,*8*),*And*(*Equal*(*3*,*9*),
And(*Equal*(*4*,*5*),*iterates*(λp. *incr_bv*(*p*)‘*5* , *5*, φ)))))))))))

lemma *arity_ren_forces_all* :
assumes φ∈*formula*
shows *arity*(*ren_forces_forall*(φ)) = *5* ∪ *arity*(φ)
⟨*proof*⟩

lemma *ren_forces_forall_type*[*TC*] :
φ∈*formula* ⟹ *ren_forces_forall*(φ) ∈*formula*
⟨*proof*⟩

lemma *sats_ren_forces_forall* :
[*x*,*P*,*leq*,*o*,*p*] @ *env* ∈ *list*(*M*) ⟹ φ∈*formula* ⟹
sats(*M*, *ren_forces_forall*(φ),[*x*,*p*,*P*,*leq*,*o*] @ *env*) ⟷ *sats*(*M*, φ,[*p*,*P*,*leq*,*o*,*x*] @ *env*)
⟨*proof*⟩

definition
is_leq :: $[i \Rightarrow o,i,i,i] \Rightarrow o$ **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*]:
$[\![\ l \in M\ ;\ q \in M\ ;\ p \in M\ ]\!] \Longrightarrow is\_leq(\#\#M,l,q,p) \longleftrightarrow \langle q,p\rangle \in l$
⟨*proof*⟩

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* :
$[\![leq \in nat;q \in nat;p \in nat]\!] \Longrightarrow arity(leq\_fm(leq,q,p)) = succ(q) \cup succ(p) \cup succ(leq)$
⟨*proof*⟩

lemma *leq_fm_type*[*TC*] :
$[\![leq \in nat;q \in nat;p \in nat]\!] \Longrightarrow leq\_fm(leq,q,p) \in formula$
⟨*proof*⟩

lemma *sats_leq_fm* :
$[\![\ leq \in nat;q \in nat;p \in nat;env \in list(A)\ ]\!] \Longrightarrow$
$sats(A,leq\_fm(leq,q,p),env) \longleftrightarrow is\_leq(\#\#A,nth(leq,env),nth(q,env),nth(p,env))$
⟨*proof*⟩

17.7.1 The primitive recursion

consts *forces'* :: $i \Rightarrow i$
primrec
$forces'(Member(x,y)) = forces\_mem\_fm(1,2,0,x\#+4,y\#+4)$
$forces'(Equal(x,y)) = forces\_eq\_fm(1,2,0,x\#+4,y\#+4)$
$forces'(Nand(p,q)) =$
$Neg(Exists(And(Member(0,2),And(leq\_fm(3,0,1),And(ren\_forces\_nand(forces'(p)),$
$ren\_forces\_nand(forces'(q)))))))$
$forces'(Forall(p)) = Forall(ren\_forces\_forall(forces'(p)))$

definition
forces :: $i \Rightarrow i$ **where**
$forces(\varphi) \equiv And(Member(0,1),forces'(\varphi))$

lemma *forces'_type* [*TC*]: $\varphi \in formula \Longrightarrow forces'(\varphi) \in formula$
⟨*proof*⟩

lemma *forces_type*[*TC*] : $\varphi \in formula \Longrightarrow forces(\varphi) \in formula$
⟨*proof*⟩

context *forcing_data*

begin

17.8 Forcing for atomic formulas in context

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

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

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

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

lemma *def_forces_eq*: $p\in P \Longrightarrow forces\_eq(p,t1,t2) \longleftrightarrow$
  $(\forall s\in domain(t1) \cup domain(t2).\ \forall q.\ q\in P \wedge q \preceq p \longrightarrow$
  $(forces\_mem(q,s,t1) \longleftrightarrow forces\_mem(q,s,t2)))$
  ⟨*proof*⟩

lemma *def_forces_mem*: $p\in P \Longrightarrow forces\_mem(p,t1,t2) \longleftrightarrow$
  $(\forall v\in P.\ v \preceq p \longrightarrow$
  $(\exists q.\ \exists s.\ \exists r.\ r\in P \wedge q\in P \wedge q \preceq v \wedge \langle s,r\rangle \in t2 \wedge q \preceq r \wedge forces\_eq(q,t1,s)))$
  ⟨*proof*⟩

lemma *forces_eq_abs* :
  $[\![p\in M\ ;\ t1\in M\ ;\ t2\in M]\!] \Longrightarrow is\_forces\_eq(p,t1,t2) \longleftrightarrow forces\_eq(p,t1,t2)$
  ⟨*proof*⟩

lemma *forces_mem_abs* :
  $[\![p\in M\ ;\ t1\in M\ ;\ t2\in M]\!] \Longrightarrow is\_forces\_mem(p,t1,t2) \longleftrightarrow forces\_mem(p,t1,t2)$
  ⟨*proof*⟩

lemma *sats_forces_eq_fm*:
  assumes $p\in nat$ $l\in nat$ $q\in nat$ $t1\in nat$ $t2\in nat$ $env\in list(M)$
    $nth(p,env)=P$ $nth(l,env)=leq$
  shows $sats(M,forces\_eq\_fm(p,l,q,t1,t2),env) \longleftrightarrow$
    $is\_forces\_eq(nth(q,env),nth(t1,env),nth(t2,env))$
  ⟨*proof*⟩

lemma *sats_forces_mem_fm*:

assumes $p\in nat$ $l\in nat$ $q\in nat$ $t1\in nat$ $t2\in nat$ $env\in list(M)$
$nth(p,env)=P$ $nth(l,env)=leq$
shows $sats(M,forces\_mem\_fm(p,l,q,t1,t2),env) \longleftrightarrow$
$is\_forces\_mem(nth(q,env),nth(t1,env),nth(t2,env))$
$\langle proof\rangle$

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* :
$forces\_neq(p,t1,t2) \longleftrightarrow forces\_neq'(P,leq,p,t1,t2)$
$\langle proof\rangle$

lemma *forces_nmem* :
$forces\_nmem(p,t1,t2) \longleftrightarrow forces\_nmem'(P,leq,p,t1,t2)$
$\langle proof\rangle$

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 $sats(M,forces(Member(x,y)),[q,P,leq,one]@env) \longleftrightarrow$
$(q\in P \wedge is\_forces\_mem(q,xx,yy))$
$\langle proof\rangle$

lemma *sats_forces_Equal* :
assumes $x\in nat$ $y\in nat$ $env\in list(M)$
$nth(x,env)=xx$ $nth(y,env)=yy$ $q\in M$
shows $sats(M,forces(Equal(x,y)),[q,P,leq,one]@env) \longleftrightarrow$
$(q\in P \wedge is\_forces\_eq(q,xx,yy))$
$\langle proof\rangle$

lemma *sats_forces_Nand* :
assumes $\varphi\in formula$ $\psi\in formula$ $env\in list(M)$ $p\in M$
shows $sats(M,forces(Nand(\varphi,\psi)),[p,P,leq,one]@env) \longleftrightarrow$
$(p\in P \wedge \neg(\exists q\in M.\ q\in P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
$(sats(M,forces'(\varphi),[q,P,leq,one]@env) \wedge sats(M,forces'(\psi),[q,P,leq,one]@env))))$
$\langle proof\rangle$

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

$(p{\in}P \wedge \neg(\exists\, q{\in}M.\ q{\in}P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
$(sats(M,forces'(\varphi),[q,P,leq,one]@env))))$
⟨proof⟩

lemma *sats_forces_Forall* :
assumes $\varphi{\in}formula$ $env{\in}list(M)$ $p{\in}M$
shows $sats(M,forces(Forall(\varphi)),[p,P,leq,one]@env) \longleftrightarrow$
$p{\in}P \wedge (\forall\, x{\in}M.\ sats(M,forces'(\varphi),[p,P,leq,one,x]@env))$
⟨proof⟩

end

17.9 The arity of *forces*

lemma *arity_forces_at*:
assumes $x \in nat$ $y \in nat$
shows $arity(forces(Member(x,\ y))) = (succ(x) \cup succ(y))\ \#+\ 4$
$arity(forces(Equal(x,\ y))) = (succ(x) \cup succ(y))\ \#+\ 4$
⟨proof⟩

lemma *arity_forces'*:
assumes $\varphi{\in}formula$
shows $arity(forces'(\varphi)) \le arity(\varphi)\ \#+\ 4$
⟨proof⟩

lemma *arity_forces* :
assumes $\varphi{\in}formula$
shows $arity(forces(\varphi)) \le 4\#{+}arity(\varphi)$
⟨proof⟩

lemma *arity_forces_le* :
assumes $\varphi{\in}formula$ $n{\in}nat$ $arity(\varphi) \le n$
shows $arity(forces(\varphi)) \le 4\#{+}n$
⟨proof⟩

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**
$p \Vdash \varphi\ env \quad \equiv \quad M,\ ([p,P,leq,one]\ @\ env) \models forces(\varphi)$

lemma *Collect_forces* :
assumes
fty: $\varphi \in formula$ **and**
far: $arity(\varphi) \le length(env)$ **and**
envty: $env \in list(M)$
shows
$\{p \in P\ .\ p \Vdash \varphi\ env\} \in M$
$\langle proof \rangle$

lemma *forces_mem_iff_dense_below*: $p \in P \Longrightarrow forces\_mem(p,t1,t2) \longleftrightarrow dense\_below($
$\{q \in P.\ \exists s.\ \exists r.\ r \in P \wedge \langle s,r\rangle \in t2 \wedge q \preceq r \wedge forces\_eq(q,t1,s)\}$
$,p)$
$\langle proof \rangle$

18.2 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_eq*:
assumes $p \in P$ $r \in P$ $r \preceq p$ $forces\_eq(p,t1,t2)$
shows $forces\_eq(r,t1,t2)$
$\langle proof \rangle$

18.3 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_mem*:
assumes $p \in P$ $r \in P$ $r \preceq p$ $forces\_mem(p,t1,t2)$
shows $forces\_mem(r,t1,t2)$
$\langle proof \rangle$

18.4 Kunen 2013, Lemma IV.2.37(b)

lemma *density_mem*:
assumes $p \in P$
shows $forces\_mem(p,t1,t2) \longleftrightarrow dense\_below(\{q \in P.\ forces\_mem(q,t1,t2)\},p)$
$\langle proof \rangle$

lemma *aux_density_eq*:
assumes
$dense\_below($
$\{q' \in P.\ \forall q.\ q \in P \wedge q \preceq q' \longrightarrow forces\_mem(q,s,t1) \longleftrightarrow forces\_mem(q,s,t2)\}$
$,p)$
$forces\_mem(q,s,t1)$ $q \in P$ $p \in P$ $q \preceq p$
shows
$dense\_below(\{r \in P.\ forces\_mem(r,s,t2)\},q)$
$\langle proof \rangle$

lemma *density_eq*:
  assumes $p \in P$
  shows $forces\_eq(p,t1,t2) \longleftrightarrow dense\_below(\{q \in P.\ forces\_eq(q,t1,t2)\},p)$
$\langle proof \rangle$

18.5 Kunen 2013, Lemma IV.2.38

lemma *not_forces_neq*:
  assumes $p \in P$
  shows $forces\_eq(p,t1,t2) \longleftrightarrow \neg\ (\exists\, q \in P.\ q \preceq p \wedge forces\_neq(q,t1,t2))$
  $\langle proof \rangle$

lemma *not_forces_nmem*:
  assumes $p \in P$
  shows $forces\_mem(p,t1,t2) \longleftrightarrow \neg\ (\exists\, q \in P.\ q \preceq p \wedge forces\_nmem(q,t1,t2))$
  $\langle proof \rangle$

lemma *sats_forces_Nand'*:
  assumes
    $p \in P\ \varphi \in formula\ \psi \in formula\ env \in list(M)$
  shows
    $M,\ [p,P,leq,one]\ @\ env \models forces(Nand(\varphi,\psi)) \longleftrightarrow$
     $\neg(\exists\, q \in M.\ q \in P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
        $M,\ [q,P,leq,one]\ @\ env \models forces(\varphi) \wedge$
        $M,\ [q,P,leq,one]\ @\ env \models forces(\psi))$
  $\langle proof \rangle$

lemma *sats_forces_Neg'*:
  assumes
    $p \in P\ env \in list(M)\ \varphi \in formula$
  shows
    $M,\ [p,P,leq,one]\ @\ env \models forces(Neg(\varphi)) \longleftrightarrow$
     $\neg(\exists\, q \in M.\ q \in P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
        $M,\ [q,P,leq,one]@env \models forces(\varphi))$
  $\langle proof \rangle$

lemma *sats_forces_Forall'*:
  assumes
    $p \in P\ env \in list(M)\ \varphi \in formula$
  shows
    $M,[p,P,leq,one]\ @\ env \models forces(Forall(\varphi)) \longleftrightarrow$
     $(\forall\, x \in M.\ \ M,\ [p,P,leq,one,x]\ @\ env \models forces(\varphi))$
  $\langle proof \rangle$

18.6 The relation of forcing and atomic formulas

lemma *Forces_Equal*:
  assumes
    $p{\in}P$ $t1{\in}M$ $t2{\in}M$ $env{\in}list(M)$ $nth(n,env) = t1$ $nth(m,env) = t2$ $n{\in}nat$ $m{\in}nat$
  shows
    $(p \Vdash Equal(n,m)\ env) \longleftrightarrow forces\_eq(p,t1,t2)$
    ⟨*proof*⟩

lemma *Forces_Member*:
  assumes
    $p{\in}P$ $t1{\in}M$ $t2{\in}M$ $env{\in}list(M)$ $nth(n,env) = t1$ $nth(m,env) = t2$ $n{\in}nat$ $m{\in}nat$
  shows
    $(p \Vdash Member(n,m)\ env) \longleftrightarrow forces\_mem(p,t1,t2)$
    ⟨*proof*⟩

lemma *Forces_Neg*:
  assumes
    $p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$
  shows
    $(p \Vdash Neg(\varphi)\ env) \longleftrightarrow \neg(\exists\, q{\in}M.\ q{\in}P \wedge q{\preceq}p \wedge (q \Vdash \varphi\ env))$
    ⟨*proof*⟩

18.7 The relation of forcing and connectives

lemma *Forces_Nand*:
  assumes
    $p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$ $\psi{\in}formula$
  shows
    $(p \Vdash Nand(\varphi,\psi)\ env) \longleftrightarrow \neg(\exists\, q{\in}M.\ q{\in}P \wedge q{\preceq}p \wedge (q \Vdash \varphi\ env) \wedge (q \Vdash \psi\ env))$
    ⟨*proof*⟩

lemma *Forces_And_aux*:
  assumes
    $p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$ $\psi{\in}formula$
  shows
    $p \Vdash And(\varphi,\psi)\ env \quad \longleftrightarrow$
    $(\forall\, q{\in}M.\ q{\in}P \wedge q{\preceq}p \longrightarrow (\exists\, r{\in}M.\ r{\in}P \wedge r{\preceq}q \wedge (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)))$
  ⟨*proof*⟩

lemma *Forces_And_iff_dense_below*:
  assumes
    $p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$ $\psi{\in}formula$
  shows
    $(p \Vdash And(\varphi,\psi)\ env) \longleftrightarrow dense\_below(\{r{\in}P.\ (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)\ \},p)$
  ⟨*proof*⟩

lemma *Forces_Forall*:
  assumes
    $p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$

shows
  $(p \Vdash \mathit{Forall}(\varphi)\ \mathit{env}) \longleftrightarrow (\forall x{\in}M.\ (p \Vdash \varphi\ ([x]\ @\ \mathit{env})))$
  ⟨*proof*⟩

bundle *some_rules* = *elem_of_val_pair* [*dest*] *SepReplace_iff* [*simp del*] *SepReplace_iff*[*iff*]

context
  includes *some_rules*
begin

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

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

lemma *left_in_M* : $tau{\in}M \Longrightarrow \langle a,b\rangle{\in}tau \Longrightarrow a{\in}M$
  ⟨*proof*⟩

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*⟩

18.9 Auxiliary results for Lemma IV.2.40(a)

lemma *IV240a_mem_Collect*:
  assumes
    $\pi{\in}M$ $\tau{\in}M$
  shows
    $\{q{\in}P.\ \exists\sigma.\ \exists r.\ r{\in}P \wedge \langle\sigma,r\rangle \in \tau \wedge q{\preceq}r \wedge forces\_eq(q,\pi,\sigma)\}{\in}M$
⟨*proof*⟩

lemma *IV240a_mem*:
  assumes
    $M\_generic(G)$ $p{\in}G$ $\pi{\in}M$ $\tau{\in}M$ $forces\_mem(p,\pi,\tau)$
    $\bigwedge q\ \sigma.\ q{\in}P \Longrightarrow q{\in}G \Longrightarrow \sigma{\in}domain(\tau) \Longrightarrow forces\_eq(q,\pi,\sigma) \Longrightarrow$
      $val(G,\pi) = val(G,\sigma)$
  shows
    $val(G,\pi){\in}val(G,\tau)$
⟨*proof*⟩

lemma *refl_forces_eq*: $p{\in}P \Longrightarrow forces\_eq(p,x,x)$
  ⟨*proof*⟩

lemma *forces_memI*: $\langle\sigma,r\rangle\in\tau \Longrightarrow p\in P \Longrightarrow r\in P \Longrightarrow p\preceq r \Longrightarrow forces\_mem(p,\sigma,\tau)$
⟨*proof*⟩

lemma *IV240a_eq_1st_incl*:
assumes
$M\_generic(G)$ $p\in G$ $forces\_eq(p,\tau,\vartheta)$
and
$IH: \bigwedge q\ \sigma.\ q\in P \Longrightarrow q\in G \Longrightarrow \sigma\in domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(forces\_mem(q,\sigma,\tau) \longrightarrow val(G,\sigma) \in val(G,\tau)) \wedge$
$(forces\_mem(q,\sigma,\vartheta) \longrightarrow val(G,\sigma) \in val(G,\vartheta))$

shows
$val(G,\tau) \subseteq val(G,\vartheta)$
⟨*proof*⟩

lemma *IV240a_eq_2nd_incl*:
assumes
$M\_generic(G)$ $p\in G$ $forces\_eq(p,\tau,\vartheta)$
and
$IH: \bigwedge q\ \sigma.\ q\in P \Longrightarrow q\in G \Longrightarrow \sigma\in domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(forces\_mem(q,\sigma,\tau) \longrightarrow val(G,\sigma) \in val(G,\tau)) \wedge$
$(forces\_mem(q,\sigma,\vartheta) \longrightarrow val(G,\sigma) \in val(G,\vartheta))$
shows
$val(G,\vartheta) \subseteq val(G,\tau)$
⟨*proof*⟩

lemma *IV240a_eq*:
assumes
$M\_generic(G)$ $p\in G$ $forces\_eq(p,\tau,\vartheta)$
and
$IH: \bigwedge q\ \sigma.\ q\in P \Longrightarrow q\in G \Longrightarrow \sigma\in domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(forces\_mem(q,\sigma,\tau) \longrightarrow val(G,\sigma) \in val(G,\tau)) \wedge$
$(forces\_mem(q,\sigma,\vartheta) \longrightarrow val(G,\sigma) \in val(G,\vartheta))$
shows
$val(G,\tau) = val(G,\vartheta)$
⟨*proof*⟩

18.10 Induction on names

lemma *core_induction*:
assumes
$\bigwedge\tau\ \vartheta\ p.\ p \in P \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![q\in P\ ;\ \sigma\in domain(\vartheta)]\!] \Longrightarrow Q(0,\tau,\sigma,q)]\!] \Longrightarrow Q(1,\tau,\vartheta,p)$
$\bigwedge\tau\ \vartheta\ p.\ p \in P \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![q\in P\ ;\ \sigma\in domain(\tau) \cup domain(\vartheta)]\!] \Longrightarrow Q(1,\sigma,\tau,q)$
$\wedge\ Q(1,\sigma,\vartheta,q)]\!] \Longrightarrow Q(0,\tau,\vartheta,p)$

$ft \in 2$ $p \in P$
shows
$Q(ft,\tau,\vartheta,p)$
$\langle proof \rangle$

lemma *forces_induction_with_conds*:
assumes
$\bigwedge\tau\ \vartheta\ p.\ p \in P \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![q{\in}P\ ;\ \sigma{\in}domain(\vartheta)]\!] \Longrightarrow Q(q,\tau,\sigma)]\!] \Longrightarrow R(p,\tau,\vartheta)$
$\bigwedge\tau\ \vartheta\ p.\ p \in P \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![q{\in}P\ ;\ \sigma{\in}domain(\tau) \cup domain(\vartheta)]\!] \Longrightarrow R(q,\sigma,\tau) \wedge R(q,\sigma,\vartheta)]\!] \Longrightarrow Q(p,\tau,\vartheta)$
$p \in P$
shows
$Q(p,\tau,\vartheta) \wedge R(p,\tau,\vartheta)$
$\langle proof \rangle$

lemma *forces_induction*:
assumes
$\bigwedge\tau\ \vartheta.\ [\![\bigwedge\sigma.\ \sigma{\in}domain(\vartheta) \Longrightarrow Q(\tau,\sigma)]\!] \Longrightarrow R(\tau,\vartheta)$
$\bigwedge\tau\ \vartheta.\ [\![\bigwedge\sigma.\ \sigma{\in}domain(\tau) \cup domain(\vartheta) \Longrightarrow R(\sigma,\tau) \wedge R(\sigma,\vartheta)]\!] \Longrightarrow Q(\tau,\vartheta)$
shows
$Q(\tau,\vartheta) \wedge R(\tau,\vartheta)$
$\langle proof \rangle$

18.11 Lemma IV.2.40(a), in full

lemma *IV240a*:
assumes
$M\_generic(G)$
shows
$(\tau{\in}M \longrightarrow \vartheta{\in}M \longrightarrow (\forall p{\in}G.\ forces\_eq(p,\tau,\vartheta) \longrightarrow val(G,\tau) = val(G,\vartheta))) \wedge$
$(\tau{\in}M \longrightarrow \vartheta{\in}M \longrightarrow (\forall p{\in}G.\ forces\_mem(p,\tau,\vartheta) \longrightarrow val(G,\tau) \in val(G,\vartheta)))$
(**is** $?Q(\tau,\vartheta) \wedge ?R(\tau,\vartheta)$)
$\langle proof \rangle$

18.12 Lemma IV.2.40(b)

lemma *IV240b_mem*:
assumes
$M\_generic(G)$ $val(G,\pi){\in}val(G,\tau)$ $\pi{\in}M$ $\tau{\in}M$
and
$IH{:}\bigwedge\sigma.\ \sigma{\in}domain(\tau) \Longrightarrow val(G,\pi) = val(G,\sigma) \Longrightarrow$
$\exists p{\in}G.\ forces\_eq(p,\pi,\sigma)$
shows
$\exists p{\in}G.\ forces\_mem(p,\pi,\tau)$
$\langle proof \rangle$

end

lemma *Collect_forces_eq_in_M*:
assumes $\tau \in M$ $\vartheta \in M$

shows $\{p{\in}P.\ forces\_eq(p,\tau,\vartheta)\} \in M$
⟨*proof*⟩

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*⟩

lemma *IV240b_eq*:
assumes
$M\_generic(G)\ val(G,\tau) = val(G,\vartheta)\ \tau{\in}M\ \vartheta{\in}M$
and
$IH{:}\bigwedge\sigma.\ \sigma{\in}domain(\tau){\cup}domain(\vartheta) \Longrightarrow$
$(val(G,\sigma){\in}val(G,\tau) \longrightarrow (\exists q{\in}G.\ forces\_mem(q,\sigma,\tau))) \wedge$
$(val(G,\sigma){\in}val(G,\vartheta) \longrightarrow (\exists q{\in}G.\ forces\_mem(q,\sigma,\vartheta)))$

shows
$\exists p{\in}G.\ forces\_eq(p,\tau,\vartheta)$
⟨*proof*⟩

lemma *IV240b*:
assumes
$M\_generic(G)$
shows
$(\tau{\in}M{\longrightarrow}\vartheta{\in}M{\longrightarrow}val(G,\tau) = val(G,\vartheta) \longrightarrow (\exists p{\in}G.\ forces\_eq(p,\tau,\vartheta))) \wedge$
$(\tau{\in}M{\longrightarrow}\vartheta{\in}M{\longrightarrow}val(G,\tau) \in val(G,\vartheta) \longrightarrow (\exists p{\in}G.\ forces\_mem(p,\tau,\vartheta)))$
(**is** $?Q(\tau,\vartheta) \wedge\ ?R(\tau,\vartheta)$)
⟨*proof*⟩

lemma *map_val_in_MG*:
assumes
$env{\in}list(M)$
shows
$map(val(G),env){\in}list(M[G])$
⟨*proof*⟩

lemma *truth_lemma_mem*:
assumes
$env{\in}list(M)\ M\_generic(G)$
$n{\in}nat\ m{\in}nat\ n{<}length(env)\ m{<}length(env)$
shows
$(\exists p{\in}G.\ p \Vdash Member(n,m)\ env) \longleftrightarrow M[G],\ map(val(G),env) \models Member(n,m)$
⟨*proof*⟩

lemma *truth_lemma_eq*:

assumes
$env{\in}list(M)$ $M\_generic(G)$
$n{\in}nat$ $m{\in}nat$ $n{<}length(env)$ $m{<}length(env)$
shows
$(\exists\, p{\in}G.\ p \Vdash Equal(n,m)\ env) \longleftrightarrow M[G],\ map(val(G),env) \models Equal(n,m)$
$\langle proof\rangle$

lemma *arities_at_aux*:
assumes
$n \in nat$ $m \in nat$ $env \in list(M)$ $succ(n) \cup succ(m) \leq length(env)$
shows
$n < length(env)$ $m < length(env)$
$\langle proof\rangle$

18.13 The Strenghtening Lemma

lemma *strengthening_lemma*:
assumes
$p{\in}P$ $\varphi{\in}formula$ $r{\in}P$ $r{\preceq}p$
shows
$\bigwedge env.\ env{\in}list(M) \Longrightarrow arity(\varphi){\leq}length(env) \Longrightarrow p \Vdash \varphi\ env \Longrightarrow r \Vdash \varphi\ env$
$\langle proof\rangle$

18.14 The Density Lemma

lemma *arity_Nand_le*:
assumes $\varphi \in formula$ $\psi \in formula$ $arity(Nand(\varphi, \psi)) \leq length(env)$ $env{\in}list(A)$
shows $arity(\varphi) \leq length(env)$ $arity(\psi) \leq length(env)$
$\langle proof\rangle$

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

lemma *density_lemma*:
assumes
$p{\in}P$ $\varphi{\in}formula$ $env{\in}list(M)$ $arity(\varphi){\leq}length(env)$
shows
$p \Vdash \varphi\ env \longleftrightarrow dense\_below(\{q{\in}P.\ (q \Vdash \varphi\ env)\},p)$
$\langle proof\rangle$

18.15 The Truth Lemma

lemma *Forces_And*:
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
$p \Vdash And(\varphi,\psi)\ env \longleftrightarrow (p \Vdash \varphi\ env) \wedge (p \Vdash \psi\ env)$
⟨*proof*⟩

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
$(p \Vdash Nand(\varphi,\psi)\ env) \longleftrightarrow (p \Vdash Neg(And(\varphi,\psi))\ env)$
⟨*proof*⟩

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*⟩

lemma *truth_lemma_And*:
assumes
$env{\in}list(M)$ $\varphi{\in}formula$ $\psi{\in}formula$
$arity(\varphi){\leq}length(env)$ $arity(\psi) \leq length(env)$ $M\_generic(G)$
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
$(\exists\, p{\in}G.\ (p \Vdash And(\varphi,\psi)\ env)) \longleftrightarrow M[G]\ ,\ map(val(G),env) \models And(\varphi,\psi)$
⟨*proof*⟩

definition
$ren\_truth\_lemma :: i{\Rightarrow}i$ **where**
$ren\_truth\_lemma(\varphi) \equiv$
$Exists(Exists(Exists(Exists(Exists($
$And(Equal(0,5),And(Equal(1,8),And(Equal(2,9),And(Equal(3,10),And(Equal(4,6),$
$iterates(\lambda p.\ incr\_bv(p)`5\ ,\ 6,\ \varphi))))))))))$

lemma *ren_truth_lemma_type*[*TC*] :
$\varphi{\in}formula \Longrightarrow ren\_truth\_lemma(\varphi) \in formula$
⟨*proof*⟩

lemma *arity_ren_truth* :
assumes $\varphi{\in}formula$
shows $arity(ren\_truth\_lemma(\varphi)) \leq 6 \cup succ(arity(\varphi))$
⟨*proof*⟩

lemma *sats_ren_truth_lemma*:

$[q,b,d,a1,a2,a3]$ @ $env \in list(M) \Longrightarrow \varphi \in formula \Longrightarrow$
$(M,\ [q,b,d,a1,a2,a3]$ @ $env \models ren\_truth\_lemma(\varphi)\ ) \longleftrightarrow$
$(M,\ [q,a1,a2,a3,b]$ @ $env \models \varphi)$
⟨*proof*⟩

lemma *truth_lemma'* :
assumes
$\varphi{\in}formula\ env{\in}list(M)\ arity(\varphi) \leq succ(length(env))$
shows
$separation(\#\#M,\lambda d.\ \exists\, b{\in}M.\ \forall\, q{\in}P.\ q{\preceq}d \longrightarrow \neg(q \Vdash \varphi\ ([b]@env)))$
⟨*proof*⟩

lemma *truth_lemma*:
assumes
$\varphi{\in}formula\ M\_generic(G)$
shows
$\bigwedge env.\ env{\in}list(M) \Longrightarrow arity(\varphi){\leq}length(env) \Longrightarrow$
$(\exists\, p{\in}G.\ p \Vdash \varphi\ env) \quad \longleftrightarrow \quad M[G],\ map(val(G),env) \models \varphi$
⟨*proof*⟩

18.16 The "Definition of forcing"

lemma *definition_of_forcing*:
assumes
$p{\in}P\ \varphi{\in}formula\ env{\in}list(M)\ arity(\varphi){\leq}length(env)$
shows
$(p \Vdash \varphi\ env) \longleftrightarrow$
$(\forall\, G.\ M\_generic(G) \wedge p{\in}G \ \longrightarrow\ M[G],\ map(val(G),env) \models \varphi)$
⟨*proof*⟩

lemmas *definability* = *forces_type*
end

end

19 Auxiliary renamings for Separation

theory *Separation_Rename*
imports *Interface Renaming*
begin

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

lemma *nth_concat* : $[p,t] \in list(A) \Longrightarrow env{\in}\ list(A) \Longrightarrow nth(1\ \#{+}\ length(env),[p]$@ env @ $[t]) = t$
⟨*proof*⟩

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

$\langle proof \rangle$

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

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**
$sep\_env(n) \equiv \lambda\ i \in (5\#+n)\text{-}5\ .\ i\#+2$

definition *weak* :: $[i,\ i] \Rightarrow i$ **where**
$weak(n,m) \equiv \{i\#+m\ .\ i \in n\}$

lemma *weakD* :
assumes $n \in nat$ $k \in nat$ $x \in weak(n,k)$
shows $\exists\ i \in n\ .\ x = i\#+k$
$\langle proof \rangle$

lemma *weak_equal* :
assumes $n \in nat$ $m \in nat$
shows $weak(n,m) = (m\#+n)$ - m
$\langle proof \rangle$

lemma *weak_zero*:
shows $weak(0,n) = 0$
$\langle proof \rangle$

lemma *weakening_diff* :
assumes $n \in nat$
shows $weak(n,7)$ - $weak(n,5) \subseteq \{5\#+n,\ 6\#+n\}$
$\langle proof \rangle$

lemma *in_add_del* :
assumes $x \in j\#+n$ $n \in nat$ $j \in nat$
shows $x < j \vee x \in weak(n,j)$
$\langle proof \rangle$

lemma *sep_env_action*:
assumes
$[t,p,u,P,leq,o,pi] \in list(M)$
$env \in list(M)$
shows $\forall\ i\ .\ i \in weak(length(env),5) \longrightarrow$
$nth(sep\_env(length(env))\text{`}i,[t,p,u,P,leq,o,pi]@env) = nth(i,[p,P,leq,o,t]\ @\ env$
$@\ [pi,u])$
$\langle proof \rangle$

lemma *sep_env_type* :
assumes $n \in nat$
shows $sep\_env(n) : (5\#+n)\text{-}5 \rightarrow (7\#+n)\text{-}7$
$\langle proof \rangle$

lemma *sep_var_fin_type* :
assumes $n \in nat$
shows $sep\_var(n) : 7\#+n \ \text{-||>} \ 7\#+n$
$\langle proof \rangle$

lemma *sep_var_domain* :
assumes $n \in nat$
shows $domain(sep\_var(n)) = \ 7\#+n \text{ - } weak(n,5)$
$\langle proof \rangle$

lemma *sep_var_type* :
assumes $n \in nat$
shows $sep\_var(n) : (7\#+n)\text{-}weak(n,5) \rightarrow 7\#+n$
$\langle proof \rangle$

lemma *sep_var_action* :
assumes
$[t,p,u,P,leq,o,pi] \in list(M)$
$env \in list(M)$
shows $\forall \ i . \ i \in (7\#+length(env)) \text{ - } weak(length(env),5) \longrightarrow$
$nth(sep\_var(length(env))`i,[t,p,u,P,leq,o,pi]@env) = nth(i,[p,P,leq,o,t] \ @ \ env$
$@ \ [pi,u])$
$\langle proof \rangle$

definition
$rensep :: i \Rightarrow i$ **where**
$rensep(n) \equiv union\_fun(sep\_var(n),sep\_env(n),7\#+n\text{-}weak(n,5),weak(n,5))$

lemma *rensep_aux* :
assumes $n{\in}nat$
shows $(7\#+n\text{-}weak(n,5)) \cup weak(n,5) = 7\#+n \ 7\#+n \cup (\ 7 \ \#+ \ n \text{ - } 7) =$
$7\#+n$
$\langle proof \rangle$

lemma *rensep_type* :
assumes $n{\in}nat$
shows $rensep(n) \in 7\#+n \rightarrow 7\#+n$
$\langle proof \rangle$

lemma *rensep_action* :
assumes $[t,p,u,P,leq,o,pi] \ @ \ env \in list(M)$
shows $\forall \ i . \ i < 7\#+length(env) \longrightarrow nth(rensep(length(env))`i,[t,p,u,P,leq,o,pi]@env)$
$= nth(i,[p,P,leq,o,t] \ @ \ env \ @ \ [pi,u])$

⟨*proof*⟩

definition *sep_ren* :: $[i,i] \Rightarrow i$ **where**
sep_ren$(n,\varphi) \equiv$ *ren*(φ)`$(7\#+n)$`$(7\#+n)$`*rensep*(n)

lemma *arity_rensep*: **assumes** $\varphi\in$*formula* *env* $\in$ *list*(M)
arity$(\varphi) \leq 7\#+$*length*(env)
shows *arity*(*sep_ren*(*length*$(env),\varphi$)) $\leq 7\#+$*length*(env)
⟨*proof*⟩

lemma *type_rensep* [*TC*]:
assumes $\varphi\in$*formula* *env*$\in$*list*(M)
shows *sep_ren*(*length*$(env),\varphi$) $\in$ *formula*
⟨*proof*⟩

lemma *sepren_action*:
assumes *arity*$(\varphi) \leq 7 \#+$ *length*(env)
$[t,p,u,P,leq,o,pi] \in$ *list*(M)
env$\in$*list*(M)
$\varphi\in$*formula*
shows *sats*$(M,$ *sep_ren*(*length*$(env),\varphi),[t,p,u,P,leq,o,pi]$ @ *env*$) \longleftrightarrow$ *sats*$(M,$
$\varphi,[p,P,leq,o,t]$ @ *env* @ $[pi,u])$
⟨*proof*⟩

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*$\in$*list*$(M[G])$
shows $\exists$ *nenv*$\in$*list*(M). *env* = *map*(*val*(G),*nenv*)
⟨*proof*⟩

lemma *Collect_sats_in_MG* :
assumes
$c\in M[G]$
$\varphi \in$ *formula* *env*$\in$*list*$(M[G])$ *arity*$(\varphi) \leq 1 \#+$ *length*(env)
shows
$\{x\in c.\ (M[G],\ [x]$ @ *env* $\models \varphi)\}\in M[G]$
⟨*proof*⟩

theorem *separation_in_MG*:
assumes
$\varphi \in$ *formula* **and** *arity*$(\varphi) \leq 1$ #+ *length*(*env*) **and** *env*$\in$*list*($M[G]$)
shows
separation(##$M[G]$,$\lambda x.$ ($M[G]$, $[x]$ @ *env* $\models \varphi$))
⟨*proof*⟩

end

end

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

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

context *forcing_data*
begin

lemma *val_Upair* :
one $\in G \Longrightarrow$ *val*(G,$\{\langle\tau,one\rangle,\langle\varrho,one\rangle\}$) = $\{val(G,\tau),val(G,\varrho)\}$
⟨*proof*⟩

lemma *pairing_in_MG* :
assumes *M_generic*(G)
shows *upair_ax*(##$M[G]$)
⟨*proof*⟩

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',τ,ϑp) $\equiv$ ($\exists$ σ[##M].
$\exists$ q[##M]. ($q\in P'$ $\wedge$ ($\langle\sigma,q\rangle \in \tau$ $\wedge$
($\exists$ r[##M].$r\in P'$ $\wedge$ ($\langle fst(\vartheta p),r\rangle \in \sigma$ $\wedge$ $\langle snd(\vartheta p),r\rangle \in leq'$ $\wedge$ $\langle snd(\vartheta p),q\rangle$ $\in leq'$)))))

definition *Union_name_fm* :: i **where**
Union_name_fm $\equiv$

Exists(
Exists(*And*(*pair_fm*(*1*,*0*,*2*),
Exists (
Exists (*And*(*Member*(*0*,*7*),
Exists (*And*(*And*(*pair_fm*(*2*,*1*,*0*),*Member*(*0*,*6*)),
Exists (*And*(*Member*(*0*,*9*),
Exists (*And*(*And*(*pair_fm*(*6*,*1*,*0*),*Member*(*0*,*4*)),
Exists (*And*(*And*(*pair_fm*(*6*,*2*,*0*),*Member*(*0*,*10*)),
Exists (*And*(*pair_fm*(*7*,*5*,*0*),*Member*(*0*,*11*))))))))))))))))

lemma *Union_name_fm_type* [*TC*]:
Union_name_fm ∈*formula*
⟨*proof*⟩

lemma *arity_Union_name_fm* :
arity(*Union_name_fm*) = *4*
⟨*proof*⟩

lemma *sats_Union_name_fm* :
$[\![a \in M;\ b \in M\ ;\ P' \in M\ ;\ p \in M\ ;\ \vartheta \in M\ ;\ \tau \in M\ ;\ leq' \in M\ ]\!] \Longrightarrow$
$sats(M,Union\_name\_fm,[\langle\vartheta,p\rangle,\tau,leq',P']@[a,b]) \longleftrightarrow$
$Union\_name\_body(P',leq',\tau,\langle\vartheta,p\rangle)$
⟨*proof*⟩

lemma *domD* :
assumes $\tau \in M$ $\sigma \in domain(\tau)$
shows $\sigma \in M$
⟨*proof*⟩

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

lemma *Union_name_M* : **assumes** $\tau \in M$
shows $\{u \in domain(\bigcup(domain(\tau))) \times P\ .\ Union\_name\_body(P,leq,\tau,u)\} \in M$
⟨*proof*⟩

lemma *Union_MG_Eq* :
assumes $a \in M[G]$ **and** $a = val(G,\tau)$ **and** $filter(G)$ **and** $\tau \in M$
shows $\bigcup\ a = val(G,Union\_name(\tau))$
⟨*proof*⟩

lemma *union_in_MG* : **assumes** *filter*(*G*)
shows *Union_ax*(##*M*[*G*])

⟨*proof*⟩

theorem *Union_MG* : $M\_generic(G) \Longrightarrow Union\_ax(\#\#M[G])$
  ⟨*proof*⟩

end
end

23 The Powerset Axiom in $M[G]$

theory *Powerset_Axiom*
  imports *Renaming_Auto Separation_Axiom Pairing_Axiom Union_Axiom*
begin

⟨*ML*⟩

lemma *Collect_inter_Transset*:
  assumes
    $Transset(M)\ \ b \in M$
  shows
    $\{x{\in}b\ .\ P(x)\} = \{x{\in}b\ .\ P(x)\} \cap M$
  ⟨*proof*⟩

context *G_generic* **begin**

lemma *name_components_in_M*:
  assumes $<\sigma,p>\in\vartheta\ \ \vartheta \in M$
  shows $\sigma{\in}M\ \ p{\in}M$
⟨*proof*⟩

lemma *sats_fst_snd_in_M*:
  assumes
    $A{\in}M\ \ B{\in}M\ \ \varphi \in formula\ \ p{\in}M\ \ l{\in}M\ \ o{\in}M\ \ \chi{\in}M$
    $arity(\varphi) \le 6$
  shows
    $\{sq \in A{\times}B\ .\ sats(M,\varphi,[snd(sq),p,l,o,fst(sq),\chi])\} \in M$
    (**is** $?\vartheta \in M$)
⟨*proof*⟩

lemma *Pow_inter_MG*:
  assumes
    $a{\in}M[G]$
  shows
    $Pow(a) \cap M[G] \in M[G]$
⟨*proof*⟩
end

context *G_generic* **begin**

interpretation *mgtriv*: *M_trivial* $\#\#M[G]$
  ⟨*proof*⟩

theorem *power_in_MG* : *power_ax*$(\#\#(M[G]))$
  ⟨*proof*⟩
end
end

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*⟩

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*$(\#\#(M[G]))$
  ⟨*proof*⟩

lemma *foundation_ax*$(\#\#(M[G]))$
⟨*proof*⟩

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

lemma *Ord_Least_cong*:
  assumes $\bigwedge y.\ Ord(y) \Longrightarrow R(y) \longleftrightarrow Q(y)$
  shows $(\mu\ \alpha.\ R(\alpha)) = (\mu\ \alpha.\ Q(\alpha))$
⟨*proof*⟩

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

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

lemma *least_fm_type*[*TC*] :$i \in nat \Longrightarrow q{\in}formula \Longrightarrow least\_fm(q,i) \in formula$
  ⟨*proof*⟩

lemmas *basic_fm_simps* = *sats_subset_fm′ sats_transset_fm′ sats_ordinal_fm′*

lemma *sats_least_fm* :
  assumes *p_iff_sats*:
    $\bigwedge a.\ a \in A \Longrightarrow P(a) \longleftrightarrow sats(A,\ p,\ Cons(a,\ env))$
  shows
    $[\![y \in nat;\ env \in list(A)\ ;\ 0{\in}A]\!]$
    $\Longrightarrow sats(A,\ least\_fm(p,y),\ env) \longleftrightarrow$
      $least(\#\#A,\ P,\ nth(y,env))$
  ⟨*proof*⟩

lemma *least_iff_sats*:
  assumes *is_Q_iff_sats*:
    $\bigwedge a.\ a \in A \Longrightarrow is\_Q(a) \longleftrightarrow sats(A,\ q,\ Cons(a,env))$
  shows

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

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

lemma (**in** *M_ctm*) *unique_least*: $a{\in}M \Longrightarrow b{\in}M \Longrightarrow least(\#\#M,Q,a) \Longrightarrow least(\#\#M,Q,b) \Longrightarrow a=b$
⟨proof⟩

context *M_trivial*
begin

26.1 Absoluteness and closure under *Least*

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

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

end

end

27 The Axiom of Replacement in $M[G]$

theory *Replacement_Axiom*
imports
Least Relative_Univ Separation_Axiom Renaming_Auto
begin

⟨ML⟩

definition *renrep_fn* :: $i \Rightarrow i$ **where**
$renrep\_fn(env) \equiv sum(renrep1\_fn,id(length(env)),6,8,length(env))$

definition
renrep :: $[i,i] \Rightarrow i$ **where**
$renrep(\varphi,env) = ren(\varphi)`(6\#{+}length(env))`(8\#{+}length(env))`renrep\_fn(env)$

lemma *renrep_type* [*TC*]:
assumes $\varphi{\in}formula\ env \in list(M)$

shows $renrep(\varphi,env) \in formula$
$\langle proof \rangle$

lemma *arity_renrep*:
assumes $\varphi\in formula$ $arity(\varphi)\leq 6\#+length(env)$ $env \in list(M)$
shows $arity(renrep(\varphi,env)) \leq 8\#+length(env)$
$\langle proof \rangle$

lemma *renrep_sats* :
assumes $arity(\varphi) \leq 6 \#+ length(env)$
$[P,leq,o,p,\varrho,\tau]$ @ $env \in list(M)$
$V \in M$ $\alpha \in M$
$\varphi\in formula$
shows $sats(M, \varphi, [p,P,leq,o,\varrho,\tau]$ @ $env) \longleftrightarrow sats(M, renrep(\varphi,env), [V,\tau,\varrho,p,\alpha,P,leq,o]$ @ $env)$
$\langle proof \rangle$

$\langle ML \rangle$

definition *renpbdy_fn* :: $i \Rightarrow i$ **where**
$renpbdy\_fn(env) \equiv sum(renpbdy1\_fn,id(length(env)),6,7,length(env))$

definition
renpbdy :: $[i,i] \Rightarrow i$ **where**
$renpbdy(\varphi,env) = ren(\varphi)`(6\#+length(env))`(7\#+length(env))`renpbdy\_fn(env)$

lemma
renpbdy_type $[TC]$: $\varphi\in formula \Longrightarrow env\in list(M) \Longrightarrow renpbdy(\varphi,env) \in formula$
$\langle proof \rangle$

lemma *arity_renpbdy*: $\varphi\in formula \Longrightarrow arity(\varphi) \leq 6 \#+ length(env) \Longrightarrow env\in list(M) \Longrightarrow arity(renpbdy(\varphi,env)) \leq 7 \#+ length(env)$
$\langle proof \rangle$

lemma
sats_renpbdy: $arity(\varphi) \leq 6 \#+ length(nenv) \Longrightarrow [\varrho,p,x,\alpha,P,leq,o,\pi]$ @ $nenv \in list(M) \Longrightarrow \varphi\in formula \Longrightarrow$
$sats(M, \varphi, [\varrho,p,\alpha,P,leq,o]$ @ $nenv) \longleftrightarrow sats(M, renpbdy(\varphi,nenv), [\varrho,p,x,\alpha,P,leq,o]$ @ $nenv)$
$\langle proof \rangle$

$\langle ML \rangle$

definition *renbody_fn* :: $i \Rightarrow i$ **where**
$renbody\_fn(env) \equiv sum(renbody1\_fn,id(length(env)),5,6,length(env))$

definition

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

lemma
renbody_type [*TC*]: φ∈*formula* ⟹ *env*∈*list*(*M*) ⟹ *renbody*(φ,*env*) ∈ *formula*
⟨*proof*⟩

lemma *arity_renbody*: φ∈*formula* ⟹ *arity*(φ) ≤ 5 #+ *length*(*env*) ⟹ *env*∈*list*(*M*) ⟹
arity(*renbody*(φ,*env*)) ≤ 6 #+ *length*(*env*)
⟨*proof*⟩

lemma
sats_renbody: *arity*(φ) ≤ 5 #+ *length*(*nenv*) ⟹ [α,*x*,*m*,*P*,*leq*,*o*] @ *nenv* ∈ *list*(*M*) ⟹ φ∈*formula* ⟹
sats(*M*, φ, [*x*,α,*P*,*leq*,*o*] @ *nenv*) ⟷ *sats*(*M*, *renbody*(φ,*nenv*), [α,*x*,*m*,*P*,*leq*,*o*] @ *nenv*)
⟨*proof*⟩

context *G_generic*
begin

lemma *pow_inter_M*:
assumes
x∈*M* *y*∈*M*
shows
powerset(##*M*,*x*,*y*) ⟷ *y* = *Pow*(*x*) ∩ *M*
⟨*proof*⟩

schematic_goal *sats_prebody_fm_auto*:
assumes
φ∈*formula* [*P*,*leq*,*one*,*p*,ϱ,π] @ *nenv* ∈*list*(*M*) α∈*M* *arity*(φ) ≤ 2 #+ *length*(*nenv*)
shows
(∃τ∈*M*. ∃ *V*∈*M*. *is_Vset*(##*M*,α,*V*) ∧ τ∈*V* ∧ *sats*(*M*,*forces*(φ),[*p*,*P*,*leq*,*one*,ϱ,τ] @ *nenv*))
⟷ *sats*(*M*,?*prebody_fm*,[ϱ,*p*,α,*P*,*leq*,*one*] @ *nenv*)
⟨*proof*⟩

⟨*ML*⟩

lemma *prebody_fm_type* [*TC*]:
assumes φ∈*formula*
env ∈ *list*(*M*)
shows *prebody_fm*(φ,*env*)∈*formula*
⟨*proof*⟩

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
$[P,leq,one,p,\varrho]$ @ $nenv \in list(M)$ $\varphi\in formula$ $\alpha\in M$ $arity(\varphi) \leq 2$ #+ $length(nenv)$
shows
$sats(M,prebody\_fm(\varphi,nenv),[\varrho,p,\alpha,P,leq,one]$ @ $nenv) \longleftrightarrow$
$(\exists \tau\in M. \exists V\in M. is\_Vset(\#\#M,\alpha,V) \wedge \tau\in V \wedge sats(M,forces(\varphi),[p,P,leq,one,\varrho,\tau]$ @ $nenv))$
⟨*proof*⟩

lemma *arity_prebody_fm*:
assumes
$\varphi\in formula$ $\alpha\in M$ $env \in list(M)$ $arity(\varphi) \leq 2$ #+ $length(env)$
shows
$arity(prebody\_fm(\varphi,env))\leq 6$ #+ $length(env)$
⟨*proof*⟩

definition
$body\_fm'$ :: $[i,i]\Rightarrow i$ **where**
$body\_fm'(\varphi,env) \equiv Exists(Exists(And(pair\_fm(0,1,2),renpbdy(prebody\_fm(\varphi,env),env))))$

lemma *body_fm'_type*$[TC]$: $\varphi\in formula \Longrightarrow env\in list(M) \Longrightarrow body\_fm'(\varphi,env)\in formula$
⟨*proof*⟩

lemma *arity_body_fm'*:
assumes
$\varphi\in formula$ $\alpha\in M$ $env\in list(M)$ $arity(\varphi) \leq 2$ #+ $length(env)$
shows
$arity(body\_fm'(\varphi,env))\leq 5$ #+ $length(env)$
⟨*proof*⟩

lemma *sats_body_fm'*:
assumes
$\exists t\ p.\ x=\langle t,p\rangle$ $x\in M$ $[\alpha,P,leq,one,p,\varrho]$ @ $nenv \in list(M)$ $\varphi\in formula$ $arity(\varphi) \leq 2$ #+ $length(nenv)$
shows
$sats(M,body\_fm'(\varphi,nenv),[x,\alpha,P,leq,one]$ @ $nenv) \longleftrightarrow$
$sats(M,renpbdy(prebody\_fm(\varphi,nenv),nenv),[fst(x),snd(x),x,\alpha,P,leq,one]$ @ $nenv)$
⟨*proof*⟩

definition
$body\_fm$ :: $[i,i]\Rightarrow i$ **where**
$body\_fm(\varphi,env) \equiv renbody(body\_fm'(\varphi,env),env)$

lemma *body_fm_type* [*TC*]: $env \in list(M) \Longrightarrow \varphi \in formula \Longrightarrow body\_fm(\varphi,env) \in formula$
⟨*proof*⟩

lemma *sats_body_fm*:
assumes
$\exists\, t\ p.\ x=\langle t,p\rangle$ $[\alpha,x,m,P,leq,one]$ @ $nenv \in list(M)$
$\varphi \in formula$ $arity(\varphi) \leq 2 \ \#+\ length(nenv)$
shows
$sats(M,body\_fm(\varphi,nenv),[\alpha,x,m,P,leq,one]$ @ $nenv) \longleftrightarrow$
$sats(M,renpbdy(prebody\_fm(\varphi,nenv),nenv),[fst(x),snd(x),x,\alpha,P,leq,one]$ @ $nenv)$
⟨*proof*⟩

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
$sats(M,renpbdy(prebody\_fm(\varphi,nenv),nenv),[fst(x),snd(x),x,\alpha,P,leq,one]$ @ $nenv)$
$\longleftrightarrow$
$sats(M,prebody\_fm(\varphi,nenv),[fst(x),snd(x),\alpha,P,leq,one]$ @ $nenv)$
⟨*proof*⟩

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
$sats(M,body\_fm(\varphi,nenv),[\alpha,x,m,P,leq,one]$ @ $nenv) \longleftrightarrow$
$(\exists \tau \in M.\ \exists\, V \in M.\ is\_Vset(\lambda a.\ (\#\#M)(a),\alpha,V) \wedge \tau \in V \wedge (snd(x) \Vdash \varphi\ ([fst(x),\tau]@nenv)))$
⟨*proof*⟩

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*⟩

theorem *strong_replacement_in_MG*:
assumes
$\varphi \in formula$ **and** $arity(\varphi) \leq 2 \ \#+\ length(env)$ $env \in list(M[G])$
shows
$strong\_replacement(\#\#M[G],\lambda x\ v.\ sats(M[G],\varphi,[x,v]$ @ $env))$
⟨*proof*⟩

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*##$M[G]$
  ⟨*proof*⟩

lemma *infinity_in_MG* : *infinity_ax*(##$M[G]$)
⟨*proof*⟩

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*)
  ⟨*proof*⟩

lemma *range_restrict_vimage*:
  assumes *function*(*f*)
  shows *range*(*restrict*(*f*,*f*-‘‘*a*)) ⊆ *a*
⟨*proof*⟩

lemma *induced_surj_type*:
  assumes
    function(*f*)
  shows
    induced_surj(*f*,*a*,*e*): *domain*(*f*) → {*e*} ∪ *a*
    and
    x ∈ *f*-‘‘*a* ⟹ *induced_surj*(*f*,*a*,*e*)‘*x* = *f*‘*x*

$\langle proof \rangle$

lemma *induced_surj_is_surj* :
  assumes
    $e{\in}a$ $function(f)$ $domain(f) = \alpha$ $\bigwedge y.\ y \in a \Longrightarrow \exists x{\in}\alpha.\ f\ `\ x = y$
  shows
    $induced\_surj(f,a,e) \in surj(\alpha,a)$
  $\langle proof \rangle$

context *G_generic*
begin

definition
  upair_name :: $i \Rightarrow i \Rightarrow i$ **where**
  $upair\_name(\tau,\varrho) \equiv \{\langle\tau,one\rangle,\langle\varrho,one\rangle\}$

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

lemma *upair_name_abs* :
  assumes $x{\in}M$ $y{\in}M$ $z{\in}M$
  shows $is\_upair\_name(x,y,z) \longleftrightarrow z = upair\_name(x,y)$
  $\langle proof \rangle$

lemma *upair_name_closed* :
  $[\![\ x{\in}M;\ y{\in}M\ ]\!] \Longrightarrow upair\_name(x,y){\in}M$
  $\langle proof \rangle$

definition
  upair_name_fm :: $[i,i,i,i] \Rightarrow i$ **where**
  $upair\_name\_fm(x,y,o,z) \equiv Exists(Exists(And(pair\_fm(x\#{+}2,o\#{+}2,1),$
  $And(pair\_fm(y\#{+}2,o\#{+}2,0),upair\_fm(1,0,z\#{+}2)))))$

lemma *upair_name_fm_type*[*TC*] :
  $[\![\ s{\in}nat;x{\in}nat;y{\in}nat;o{\in}nat]\!] \Longrightarrow upair\_name\_fm(s,x,y,o){\in}formula$
  $\langle proof \rangle$

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
  $sats(M,upair\_name\_fm(x,y,o,z),env) \longleftrightarrow is\_upair\_name(nth(x,env),nth(y,env),nth(z,env))$
  $\langle proof \rangle$

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

$opair\_name(\tau,\varrho) \equiv upair\_name(upair\_name(\tau,\tau),upair\_name(\tau,\varrho))$

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 $is\_opair\_name(x,y,z) \longleftrightarrow z = opair\_name(x,y)$
$\langle proof \rangle$

lemma *opair_name_closed* :
$[\![x\in M;\ y\in M]\!] \Longrightarrow opair\_name(x,y)\in M$
$\langle proof \rangle$

definition
$opair\_name\_fm :: [i,i,i,i] \Rightarrow i$ **where**
$opair\_name\_fm(x,y,o,z) \equiv Exists(Exists(And(upair\_name\_fm(x\#+2,x\#+2,o\#+2,1),$
$And(upair\_name\_fm(x\#+2,y\#+2,o\#+2,0),upair\_name\_fm(1,0,o\#+2,z\#+2)))))$

lemma *opair_name_fm_type*[*TC*] :
$[\![s\in nat;x\in nat;y\in nat;o\in nat]\!] \Longrightarrow opair\_name\_fm(s,x,y,o)\in formula$
$\langle proof \rangle$

lemma *sats_opair_name_fm* :
assumes $x\in nat$ $y\in nat$ $z\in nat$ $o\in nat$ $env\in list(M)nth(o,env)=one$
shows
$sats(M,opair\_name\_fm(x,y,o,z),env) \longleftrightarrow is\_opair\_name(nth(x,env),nth(y,env),nth(z,env))$
$\langle proof \rangle$

lemma *val_upair_name* : $val(G,upair\_name(\tau,\varrho)) = \{val(G,\tau),val(G,\varrho)\}$
$\langle proof \rangle$

lemma *val_opair_name* : $val(G,opair\_name(\tau,\varrho)) = \langle val(G,\tau),val(G,\varrho)\rangle$
$\langle proof \rangle$

lemma *val_RepFun_one*: $val(G,\{\langle f(x),one\rangle\ .\ x\in a\}) = \{val(G,f(x))\ .\ x\in a\}$
$\langle proof \rangle$

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

interpretation *mgzf*: $M\_ZF\_trans\ M[G]$
$\langle proof \rangle$

definition
$is\_opname\_check :: [i,i,i] \Rightarrow o$ **where**

$is\_opname\_check(s,x,y) \equiv \exists\, chx{\in}M.\ \exists\, sx{\in}M.\ is\_check(x,chx) \wedge fun\_apply(\#\#M,s,x,sx)$
$\wedge$
$is\_opair\_name(chx,sx,y)$

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

lemma *opname_check_fm_type*[*TC*] :
$[\![\ s{\in}nat;x{\in}nat;y{\in}nat;o{\in}nat]\!] \Longrightarrow opname\_check\_fm(s,x,y,o){\in}formula$
⟨*proof*⟩

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
$sats(M,opname\_check\_fm(x,y,z,o),env) \longleftrightarrow is\_opname\_check(nth(x,env),nth(y,env),nth(z,env))$
⟨*proof*⟩

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

lemma *repl_opname_check* :
assumes
$A{\in}M\ f{\in}M$
shows
$\{opair\_name(check(x),f`x).\ x{\in}A\}{\in}M$
⟨*proof*⟩

theorem *choice_in_MG*:
assumes $choice\_ax(\#\#M)$
shows $choice\_ax(\#\#M[G])$
⟨*proof*⟩

end

end

30 Ordinals in generic extensions

theory *Ordinals_In_MG*
imports
Forcing_Theorems Relative_Univ

begin

context *G_generic*
begin

lemma *rank_val*: $rank(val(G,x)) \leq rank(x)$ (**is** $?Q(x)$)
⟨*proof*⟩

lemma *Ord_MG_iff*:
  assumes $Ord(\alpha)$
  shows $\alpha \in M \longleftrightarrow \alpha \in M[G]$
⟨*proof*⟩

end

end

31 Separative notions and proper extensions

theory *Proper_Extension*
  imports
    Names

begin

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

locale *separative_notion* = *forcing_notion* +
  assumes *separative*: $p \in P \Longrightarrow \exists\, q \in P.\ \exists\, r \in P.\ q \preceq p \wedge r \preceq p \wedge q \perp r$
begin

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

lemma *filter_complement_dense*:
  assumes $filter(G)$ **shows** $dense(P - G)$
⟨*proof*⟩

end

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

lemma *generic_not_in_M*: **assumes** $M\_generic(G)$ **shows** $G \notin M$
⟨*proof*⟩

theorem *proper_extension*: **assumes** $M\_generic(G)$ **shows** $M \neq M[G]$
  ⟨*proof*⟩

end

end

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 of finite binary sequences.

definition
  $seqspace :: i \Rightarrow i$ (‹_^<ω› [100]100) **where**
  $seqspace(B) \equiv \bigcup n\in nat.\ (n\to B)$

lemma $seqspaceI[intro]$: $n\in nat \Longrightarrow f{:}n\to B \Longrightarrow f\in seqspace(B)$
  ⟨*proof*⟩

lemma $seqspaceD[dest]$: $f\in seqspace(B) \Longrightarrow \exists n\in nat.\ f{:}n\to B$
  ⟨*proof*⟩

lemma *seqspace_type*:
  $f \in B\hat{}{<}\omega \Longrightarrow \exists n\in nat.\ f{:}n\to B$
  ⟨*proof*⟩

schematic_goal *seqspace_fm_auto*:
  assumes
    $nth(i,env) = n$ $nth(j,env) = z$ $nth(h,env) = B$
    $i \in nat$ $j \in nat$ $h\in nat$ $env \in list(A)$
  shows
    $(\exists om\in A.\ omega(\#\#A,om) \wedge n \in om \wedge is\_funspace(\#\#A, n, B, z)) \longleftrightarrow (A, env \models (?sqsprp(i,j,h)))$
  ⟨*proof*⟩

⟨*ML*⟩

locale *M_seqspace* = *M_trancl* +
  assumes
   seqspace_replacement: $M(B) \Longrightarrow strong\_replacement(M,\lambda n\ z.\ n\in nat \wedge is\_funspace(M,n,B,z))$
begin

lemma *seqspace_closed*:
  $M(B) \Longrightarrow M(B\hat{}{<}\omega)$

⟨*proof*⟩

end

sublocale *M_ctm* ⊆ *M_seqspace* ##*M*
⟨*proof*⟩

definition *seq_upd* :: *i* ⇒ *i* ⇒ *i* **where**
seq_upd(*f*,*a*) ≡ λ *j* ∈ *succ*(*domain*(*f*)) . *if j* < *domain*(*f*) *then f`j else a*

lemma *seq_upd_succ_type* :
assumes *n*∈*nat f*∈*n*→*A a*∈*A*
shows *seq_upd*(*f*,*a*)∈ *succ*(*n*) → *A*
⟨*proof*⟩

lemma *seq_upd_type* :
assumes *f*∈*A*^<ω *a*∈*A*
shows *seq_upd*(*f*,*a*) ∈ *A*^<ω
⟨*proof*⟩

lemma *seq_upd_apply_domain* [*simp*]:
assumes *f*:*n*→*A n*∈*nat*
shows *seq_upd*(*f*,*a*)`*n* = *a*
⟨*proof*⟩

lemma *zero_in_seqspace* :
shows *0* ∈ *A*^<ω
⟨*proof*⟩

definition
seqleR :: *i* ⇒ *i* ⇒ *o* **where**
seqleR(*f*,*g*) ≡ *g* ⊆ *f*

definition
seqlerel :: *i* ⇒ *i* **where**
seqlerel(*A*) ≡ *Rrel*(λ*x y*. *y* ⊆ *x*,*A*^<ω)

definition
seqle :: *i* **where**
seqle ≡ *seqlerel*(*2*)

lemma *seqleI*[*intro*!]:
⟨*f*,*g*⟩ ∈ *2*^<ω×*2*^<ω ⟹ *g* ⊆ *f* ⟹ ⟨*f*,*g*⟩ ∈ *seqle*
⟨*proof*⟩

lemma *seqleD*[*dest*!]:
z ∈ *seqle* ⟹ ∃ *x y*. ⟨*x*,*y*⟩ ∈ *2*^<ω×*2*^<ω ∧ *y* ⊆ *x* ∧ *z* = ⟨*x*,*y*⟩
⟨*proof*⟩

lemma *upd_leI* :
  assumes $f{\in}2^{<}\omega$ $a{\in}2$
  shows $\langle seq\_upd(f,a),f\rangle{\in}seqle$ (**is** $\langle ?f,\_\rangle{\in}\_$)
$\langle proof\rangle$

lemma *preorder_on_seqle*: $preorder\_on(2^{<}\omega,seqle)$
  $\langle proof\rangle$

lemma *zero_seqle_max*: $x{\in}2^{<}\omega \Longrightarrow \langle x,0\rangle \in seqle$
  $\langle proof\rangle$

interpretation *forcing_notion* $2^{<}\omega$ *seqle 0*
  $\langle proof\rangle$

abbreviation *SEQle* :: $[i,\ i] \Rightarrow o$ (**infixl** $‹\preceq s›$ *50*)
  where $x \preceq s\ y \equiv Leq(x,y)$

abbreviation *SEQIncompatible* :: $[i,\ i] \Rightarrow o$ (**infixl** $‹\bot s›$ *50*)
  where $x \bot s\ y \equiv Incompatible(x,y)$

lemma *seqspace_separative*:
  assumes $f{\in}2^{<}\omega$
  shows $seq\_upd(f,0) \bot s\ seq\_upd(f,1)$ (**is** $?f \bot s\ ?g$)
$\langle proof\rangle$

definition *is_seqleR* :: $[i{\Rightarrow}o,i,i] \Rightarrow o$ **where**
  $is\_seqleR(Q,f,g) \equiv g \subseteq f$

definition *seqleR_fm* :: $i \Rightarrow i$ **where**
  $seqleR\_fm(fg) \equiv Exists(Exists(And(pair\_fm(0,1,fg\#{+}2),subset\_fm(1,0))))$

lemma *type_seqleR_fm* :
  $fg \in nat \Longrightarrow seqleR\_fm(fg) \in formula$
  $\langle proof\rangle$

lemma *arity_seqleR_fm* :
  $fg \in nat \Longrightarrow arity(seqleR\_fm(fg)) = succ(fg)$
  $\langle proof\rangle$

lemma (**in** *M_basic*) *seqleR_abs*:
  assumes $M(f)$ $M(g)$
  shows $seqleR(f,g) \longleftrightarrow is\_seqleR(M,f,g)$
  $\langle proof\rangle$

definition
  relP :: $[i{\Rightarrow}o,[i{\Rightarrow}o,i,i]{\Rightarrow}o,i] \Rightarrow o$ **where**
  $relP(M,r,xy) \equiv (\exists\, x[M].\ \exists\, y[M].\ pair(M,x,y,xy) \land r(M,x,y))$

lemma (**in** *M_ctm*) *seqleR_fm_sats* :
  assumes *fg*∈*nat* *env*∈*list*(*M*)
  shows *sats*(*M*,*seqleR_fm*(*fg*),*env*) ⟷ *relP*(##*M*,*is_seqleR*,*nth*(*fg*, *env*))
  ⟨*proof*⟩

lemma (**in** *M_basic*) *is_related_abs* :
  assumes ⋀ *f g* . *M*(*f*) ⟹ *M*(*g*) ⟹ *rel*(*f*,*g*) ⟷ *is_rel*(*M*,*f*,*g*)
  shows ⋀*z* . *M*(*z*) ⟹ *relP*(*M*,*is_rel*,*z*) ⟷ (∃ *x y*. *z* = ⟨*x*,*y*⟩ ∧ *rel*(*x*,*y*))
  ⟨*proof*⟩

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

lemma (**in** *M_basic*) *is_Rrel_abs* :
  assumes *M*(*A*) *M*(*r*)
    ⋀ *f g* . *M*(*f*) ⟹ *M*(*g*) ⟹ *rel*(*f*,*g*) ⟷ *is_rel*(*M*,*f*,*g*)
  shows *is_RRel*(*M*,*is_rel*,*A*,*r*) ⟷ *r* = *Rrel*(*rel*,*A*)
⟨*proof*⟩

definition
  is_seqlerel :: [*i*⇒*o*,*i*,*i*] ⇒ *o* **where**
  is_seqlerel(*M*,*A*,*r*) ≡ *is_RRel*(*M*,*is_seqleR*,*A*,*r*)

lemma (**in** *M_basic*) *seqlerel_abs* :
  assumes *M*(*A*) *M*(*r*)
  shows *is_seqlerel*(*M*,*A*,*r*) ⟷ *r* = *Rrel*(*seqleR*,*A*)
  ⟨*proof*⟩

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

lemma *Rrel_eq* : *RrelP*(*R*,*A*) = *Rrel*(*R*,*A*)
  ⟨*proof*⟩

context *M_ctm*
begin

lemma *Rrel_closed*:
  assumes *A*∈*M*
    ⋀ *a*. *a* ∈ *nat* ⟹ *rel_fm*(*a*)∈*formula*
    ⋀ *f g* . (##*M*)(*f*) ⟹ (##*M*)(*g*) ⟹ *rel*(*f*,*g*) ⟷ *is_rel*(##*M*,*f*,*g*)
    arity(*rel_fm*(*0*)) = *1*
    ⋀ *a* . *a* ∈ *M* ⟹ *sats*(*M*,*rel_fm*(*0*),[*a*]) ⟷ *relP*(##*M*,*is_rel*,*a*)
  shows (##*M*)(*Rrel*(*rel*,*A*))
⟨*proof*⟩

lemma *seqle_in_M*: *seqle* ∈ *M*

⟨*proof*⟩

32.2 Cohen extension is proper

interpretation *ctm_separative* $2^{<\omega}$ *seqle 0*
⟨*proof*⟩

lemma *cohen_extension_is_proper*: $\exists\, G.\ M\_generic(G) \wedge M \neq GenExt(G)$
⟨*proof*⟩

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*⟩

lemma *well_ord_surj_imp_lepoll*:
assumes $well\_ord(A,r)\ h \in surj(A,B)$
shows $B \lesssim A$
⟨*proof*⟩

lemma (**in** *forcing_data*) *surj_nat_MG* :
$\exists f.\ f \in surj(nat,M[G])$
⟨*proof*⟩

lemma (**in** *G_generic*) *MG_eqpoll_nat*: $M[G] \approx nat$
⟨*proof*⟩

33.2 The main result

theorem *extensions_of_ctms*:
assumes
$M \approx nat$ $Transset(M)$ $M \models ZF$
shows
$\exists N.$
$M \subseteq N \wedge N \approx nat \wedge Transset(N) \wedge N \models ZF \wedge M{\neq}N \wedge$
$(\forall \alpha.\ Ord(\alpha) \longrightarrow (\alpha \in M \longleftrightarrow \alpha \in N)) \wedge$
$(M, [] {\models} AC \longrightarrow N \models ZFC)$
$\langle proof \rangle$

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).