Formalization of Forcing in Isabelle/ZF

Emmanuel Gunther   Miguel Pagano   Pedro Sánchez Terraf

April 18, 2020

Contents

1 Forcing notions

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

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

1.1 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**
$\mathit{compat\_in}(A,r,p,q) == \exists\, d \in A\ .\ \langle d,p \rangle \in r \wedge \langle d,q \rangle \in r$

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

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

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

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

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

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

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

definition
ccc :: $i \Rightarrow i \Rightarrow o$ **where**
$\mathit{ccc}(P,\mathit{leq}) == \forall\, A.\ \mathit{antichain}(P,\mathit{leq},A) \longrightarrow |A| \leq \mathit{nat}$

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

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

lemma *refl_leq*:

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

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

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

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**
$dense\_below(D,q) == \forall p{\in}P.\ p{\preceq}q \longrightarrow (\exists d{\in}D.\ d{\in}P \land d{\preceq}p)$

lemma *P_dense*: $dense(P)$
⟨*proof*⟩

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

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

lemma *leq_transD*: $a{\preceq}b \Longrightarrow b{\preceq}c \Longrightarrow a \in P{\Longrightarrow} b \in P{\Longrightarrow} c \in P{\Longrightarrow} a{\preceq}c$
⟨*proof*⟩

lemma *leq_reflI*: $p{\in}P \Longrightarrow p{\preceq}p$
⟨*proof*⟩

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

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

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

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

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

lemma *dense_belowD* [*dest*]:
assumes $dense\_below(D,p)\ q{\in}P\ q{\preceq}p$
shows $\exists d{\in}D.\ d{\in}P \land d{\preceq}q$

⟨*proof*⟩

lemma *dense_belowI* [*intro*!]:
 assumes $\bigwedge q.\ q{\in}P \Longrightarrow q{\preceq}p \Longrightarrow \exists\, d{\in}D.\ d{\in}P \wedge d{\preceq}q$
 shows $\mathit{dense\_below}(D,p)$
 ⟨*proof*⟩

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

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

lemma *dense_below_mono*: $p{\in}P \Longrightarrow D \subseteq D' \Longrightarrow \mathit{dense\_below}(D,p) \Longrightarrow \mathit{dense\_below}(D',p)$
 ⟨*proof*⟩

lemma *dense_below_under*:
 assumes $\mathit{dense\_below}(D,p)$ $p{\in}P$ $q{\in}P$ $q{\preceq}p$
 shows $\mathit{dense\_below}(D,q)$
 ⟨*proof*⟩

lemma *ideal_dense_below*:
 assumes $\bigwedge q.\ q{\in}P \Longrightarrow q{\preceq}p \Longrightarrow q{\in}D$
 shows $\mathit{dense\_below}(D,p)$
 ⟨*proof*⟩

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

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

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

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

lemma *filterD* : $\mathit{filter}(G) \Longrightarrow x \in G \Longrightarrow x \in P$
 ⟨*proof*⟩

lemma *filter_leqD* : $\mathit{filter}(G) \Longrightarrow x \in G \Longrightarrow y \in P \Longrightarrow x{\preceq}y \Longrightarrow y \in G$
 ⟨*proof*⟩

lemma *filter_imp_compat*: $filter(G) \Longrightarrow p{\in}G \Longrightarrow q{\in}G \Longrightarrow compat(p,q)$
⟨*proof*⟩

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

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

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

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

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

lemma *upclosure_increasing* :
$A{\subseteq}P \Longrightarrow increasing(upclosure(A))$
⟨*proof*⟩

lemma *upclosure_in_P*: $A \subseteq P \Longrightarrow upclosure(A) \subseteq P$
⟨*proof*⟩

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

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

lemma *closure_compat_filter*:
$A{\subseteq}P \Longrightarrow (\forall\, p{\in}A.\forall\, q{\in}A.\ compat\_in(A,leq,p,q)) \Longrightarrow filter(upclosure(A))$
⟨*proof*⟩

lemma *aux_RS1*: $f \in N \to P \Longrightarrow n{\in}N \Longrightarrow f`n \in upclosure(f\ ``N)$
⟨*proof*⟩

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

⟨proof⟩

lemma *decr_seq_linear*: $refl(P,leq) \Longrightarrow f \in nat \to P \Longrightarrow$
$\forall n{\in}nat.\ \langle f \text{ ‘ } succ(n), f \text{ ‘ } n\rangle \in leq \Longrightarrow$
$trans[P](leq) \Longrightarrow linear(f \text{ ‘‘ } nat, leq)$
⟨proof⟩

end

1.2 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*: $\forall n \in nat.\ dense(\mathcal{D}\text{‘}n)$

begin

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

The next lemma identifies a sufficient condition for obtaining RSL.

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

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
$RS\_seq(0,P,leq,p,enum,\mathcal{D}) = p$
$RS\_seq(succ(n),P,leq,p,enum,\mathcal{D}) =$
$enum\text{‘}(\mu\ m.\ \langle enum\text{‘}m,\ RS\_seq(n,P,leq,p,enum,\mathcal{D})\rangle \in leq \wedge enum\text{‘}m \in \mathcal{D} \text{ ‘ } n)$

context *countable_generic*
begin

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

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

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

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

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

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

end

2 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*]: $a\in A \Longrightarrow n\in nat \Longrightarrow$
        $(\forall X\ .\ X\neq 0 \wedge X\subseteq A \longrightarrow s`X\in X) \Longrightarrow$

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

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

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

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

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

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

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

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

$\forall a\in A.\ (\exists f \in nat\to A.\ f`0 = a \wedge (\forall n \in nat.\ \langle f`n,f`succ(n)\rangle\in S`succ(n)))$
⟨proof⟩
end

3 The general Rasiowa-Sikorski lemma

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

context *countable_generic*
begin

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

4 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* : $m \in nat \Longrightarrow n \in m \Longrightarrow n \in nat$

$\langle proof \rangle$

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

lemma *ltI_neg* : $x \in nat \Longrightarrow j \le x \Longrightarrow j \neq x \Longrightarrow j < x$
$\langle proof \rangle$

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

lemma *succ_ltI* : $n \in nat \Longrightarrow succ(j) < n \Longrightarrow j < n$
$\langle proof \rangle$

lemma *succ_In* : $n \in nat \Longrightarrow succ(j) \in n \Longrightarrow j \in n$
$\langle proof \rangle$

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

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

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

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

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

lemma *succ_in* : $succ(x) \le y \Longrightarrow x \in y$
$\langle proof \rangle$

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

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

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

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

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* : $n \in nat \Longrightarrow i \in n \Longrightarrow i \neq 0 \Longrightarrow i \neq 1 \Longrightarrow 1 < i$

⟨*proof*⟩

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

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

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

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

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

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

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

lemma *le_not_lt_nat* : $Ord(p) \Longrightarrow Ord(q) \Longrightarrow \neg\ p \leq q \Longrightarrow q \leq p$
⟨*proof*⟩

lemmas *nat_simp_union* = *nat_un_max nat_max_ty max_def*

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

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

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

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

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

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

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

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

lemma *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*⟩

4.1 Some results in ordinal arithmetic

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

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

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

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

lemma *max_D1* :
  assumes $x = y$ $w < z$ $Ord(x)$ $Ord(w)$ $Ord(z)$ $max(x,w) = max(y,z)$
  shows $z{\leq}y$
⟨*proof*⟩

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

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

5 Renaming of variables in internalized formulas

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

lemma *app_nm* : $n\in nat \Longrightarrow m\in nat \Longrightarrow f\in n\to m \Longrightarrow x \in nat \Longrightarrow f`x \in nat$
  ⟨*proof*⟩

5.1 Renaming of free variables

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

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

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

lemma *id_fn_type* :
  assumes $n \in nat$

shows $id(n) \in n \to n$
⟨*proof*⟩

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

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

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 $m \in nat$ $n \in nat$ $p \in nat$
$g \in p \to q$ $m \leq x$ $x < m\#+p$
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
$f \in m \to n$
$m \in nat$ $n \in nat$
$env1 \in list(M)$
shows
$sum(f,id(length(env1)),m,n,length(env1)) \in$
$(m\#+length(env1)) \to (n\#+length(env1))$
⟨*proof*⟩

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

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

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

lemma *sum_id0* : $m \in nat \Longrightarrow sum\_id(m,f)`0 = 0$
⟨*proof*⟩

lemma *sum_idS* : $p \in nat \Longrightarrow q \in nat \Longrightarrow f \in p \to q \Longrightarrow x \in p \Longrightarrow sum\_id(p,f)`(succ(x)) = succ(f`x)$
⟨*proof*⟩

lemma *sum_id_tc_aux* :
$p \in nat \Longrightarrow q \in nat \Longrightarrow f \in p \to q \Longrightarrow sum\_id(p,f) \in 1\#+p \to 1\#+q$
⟨*proof*⟩

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

5.2 Renaming of formulas

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

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

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

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

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

lemma *nand_ar1D* : $p \in formula \Longrightarrow q{\in}formula \Longrightarrow arity(Nand(p,q)) \le n \Longrightarrow arity(p) \le n$
⟨*proof*⟩
lemma *nand_ar2D* : $p \in formula \Longrightarrow q{\in}formula \Longrightarrow arity(Nand(p,q)) \le n \Longrightarrow arity(q) \le n$
⟨*proof*⟩

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

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

6 Aids to internalize formulas

theory *Internalizations*
imports
ZF-Constructible-Trans.Formula
ZF-Constructible-Trans.L_axioms
ZF-Constructible-Trans.DPow_absolute
begin

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

lemma *nth_closed* :
assumes *0∈A env∈list*(*A*)
shows *nth*(*n*,*env*)∈*A*
⟨*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* ∈ *nat*|] ==> *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 ^∗

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

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

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

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

lemma *tr_downD* : $x \in tr\_down(r,a) \Longrightarrow$ <x,a> $\in r$^+
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

⟨*proof*⟩

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

lemma *restrict_trans_eq*:

assumes $w \in y$
shows *restrict*$(f$,*Memrel*(*eclose*($\{x\}$))-“$\{y\}$)‘w
= *restrict*$(f$,(*Memrel*(*eclose*($\{x\}$))ˆ+)-“$\{y\}$)‘w
⟨*proof*⟩

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

end

8 Relativization of the cumulative hierarchy

theory *Relative_Univ*
imports
ZF-Constructible-Trans.Rank
ZF-Constructible-Trans.Datatype_absolute
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*⇒*o*,*i*,*i*,*i*] ⇒ *o* **where**
is_powapply(*M*,*f*,*y*,*z*) ≡ *M*(*z*) ∧ (∃ *fy*[*M*]. *fun_apply*(*M*,*f*,*y*,*fy*) ∧ *powerset*(*M*,*fy*,*z*))

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

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

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

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

8.1 Formula synthesis

schematic_goal *sats_is_powapply_fm_auto*:
assumes
f∈*nat* *y*∈*nat* *z*∈*nat* *env*∈*list*(*A*) *0*∈*A*
shows
is_powapply(##*A*,*nth*(*f*, *env*),*nth*(*y*, *env*),*nth*(*z*, *env*))
⟷ *sats*(*A*,*?ipa_fm*(*f*,*y*,*z*),*env*)
⟨*proof*⟩

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

lemma *trivial_fm*:
assumes
A≠*0* *env*∈*list*(*A*)
shows
(∃ *P*. *P* ∈ *A*) ⟷ *sats*(*A*, *Equal*(*0*,*0*), *env*)
⟨*proof*⟩

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

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

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

definition
rrank :: *i* ⇒ *i* **where**
$rrank(a) == Memrel(eclose(\{a\}))\hat{}+$

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

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

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

lemma (**in** *M_eclose*) *rrank_in_M* : $M(x) \Longrightarrow M(rrank(x))$
⟨*proof*⟩

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*: $[\![M(f); M(y)]\!] \Longrightarrow is\_powapply(M,f,y,z) \longleftrightarrow M(z) \wedge z = \{x\in Pow(f`y).\ M(x)\}$
⟨*proof*⟩

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

$is\_HVfrom(M,A,x,f,h) \longleftrightarrow$
$(\exists R[M]. h = A \cup \bigcup R \wedge is\_Replace(M, x,\lambda x\ y.\ y = \{x \in Pow(f\ `\ x) . M(x)\}, R))$
⟨proof⟩

lemma *Replace_is_powapply*:
assumes
$M(R)\ M(A)\ M(f)$
shows
$is\_Replace(M, A, is\_powapply(M, f), R) \longleftrightarrow R = Replace(A,is\_powapply(M,f))$
⟨proof⟩

lemma *powapply_closed*:
$[\![M(y) ; M(f)]\!] \Longrightarrow M(\{x \in Pow(f\ `\ y) . M(x)\})$
⟨proof⟩

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

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

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

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

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

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

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

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

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

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

lemma *Hrank_trancl*:$Hrank(y,\ restrict(f,Memrel(eclose(\{x\}))\text{-``}\{y\}))$
$= Hrank(y,\ restrict(f,(Memrel(eclose(\{x\}))\hat{}+)\text{-``}\{y\}))$
⟨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 *ZF-Constructible-Trans.Formula*
keywords
synthesize :: *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 *ZF-Constructible-Trans.Relative*
    Renaming
    Renaming_Auto
    Relative_Univ
    Synthetic_Definition
begin

syntax
  sats :: $[i, i, i] \Rightarrow o$ ((, _ ⊨ _) [36,36,36] 60)
translations
  $(M,env \models \varphi) \rightleftharpoons$ *CONST sats*(M,φ,env)

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

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

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

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

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

definition
  infinity_ax :: $(i \Rightarrow o) \Rightarrow o$ **where**
  infinity_ax(M) ==
    $(\exists I[M]. (\exists z[M]. empty(M,z) \wedge z \in I) \wedge (\forall y[M]. y \in I \longrightarrow (\exists sy[M]. successor(M,y,sy) \wedge sy \in I)))$

definition
  choice_ax :: $(i \Rightarrow o) \Rightarrow o$ **where**
  choice_ax(M) == $\forall x[M]. \exists a[M]. \exists f[M]. ordinal(M,a) \wedge surjection(M,a,x,f)$

context *M_basic* **begin**

lemma *choice_ax_abs* :
  choice_ax(M) $\longleftrightarrow$ ($\forall x[M]$. $\exists a[M]$. $\exists f[M]$. $Ord(a) \wedge f \in surj(a,x)$)
  ⟨*proof*⟩

end

definition
  wellfounded_trancl :: [*i*=>*o*,*i*,*i*,*i*] => *o* **where**
  wellfounded_trancl(M,Z,r,p) ==
    $\exists w[M]$. $\exists wx[M]$. $\exists rp[M]$.
      $w \in Z$ & *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*,λx. *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*,λx y. *sats*(M,φ,[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∈*M*
shows
separation(##*M*,λ*z*. ∃ *p*∈*M*. *p*∈*R* & (∃ *x*∈*M*.∃ *y*∈*M*. *pair*(##*M*,*x*,*y*,*p*) & *pair*(##*M*,*y*,*x*,*z*)))
⟨*proof*⟩

schematic_goal *rest_fm_auto*:
assumes
nth(*i*,*env*) = *z* *nth*(*j*,*env*) = *C*
i ∈ *nat* *j* ∈ *nat* *env* ∈ *list*(*A*)
shows
(∃ *x*∈*A*. *x*∈*C* & (∃ *y*∈*A*. *pair*(##*A*,*x*,*y*,*z*)))
⟷ *sats*(*A*,*?rfm*(*i*,*j*),*env*)
⟨*proof*⟩

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

schematic_goal *comp_fm_auto*:
assumes
nth(*i*,*env*) = *xz* *nth*(*j*,*env*) = *S* *nth*(*h*,*env*) = *R*
i ∈ *nat* *j* ∈ *nat* *h* ∈ *nat* *env* ∈ *list*(*A*)
shows
(∃ *x*∈*A*. ∃ *y*∈*A*. ∃ *z*∈*A*. ∃ *xy*∈*A*. ∃ *yz*∈*A*.
pair(##*A*,*x*,*z*,*xz*) & *pair*(##*A*,*x*,*y*,*xy*) & *pair*(##*A*,*y*,*z*,*yz*) & *xy*∈*S* & *yz*∈*R*)
⟷ *sats*(*A*,*?cfm*(*i*,*j*,*h*),*env*)
⟨*proof*⟩

lemma *comp_sep_intf* :
assumes
R∈*M*
and
S∈*M*
shows
separation(##*M*,λ*xz*. ∃ *x*∈*M*. ∃ *y*∈*M*. ∃ *z*∈*M*. ∃ *xy*∈*M*. ∃ *yz*∈*M*.
pair(##*M*,*x*,*z*,*xz*) & *pair*(##*M*,*x*,*y*,*xy*) & *pair*(##*M*,*y*,*z*,*yz*) & *xy*∈*S* & *yz*∈*R*)
⟨*proof*⟩

schematic_goal *pred_fm_auto*:
assumes
nth(i,env) = y nth(j,env) = R nth(h,env) = X
i ∈ nat j ∈ nat h ∈ nat env ∈ list(A)
shows
(∃p∈A. p∈R & pair(##A,y,X,p)) ⟷ sats(A,?pfm(i,j,h),env)
⟨proof⟩

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

schematic_goal *mem_fm_auto*:
assumes
nth(i,env) = z i ∈ nat env ∈ list(A)
shows
(∃x∈A. ∃y∈A. pair(##A,x,y,z) & x ∈ y) ⟷ sats(A,?mfm(i),env)
⟨proof⟩

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

schematic_goal *recfun_fm_auto*:
assumes
nth(i1,env) = x nth(i2,env) = r nth(i3,env) = f nth(i4,env) = g nth(i5,env) = a
nth(i6,env) = b i1∈nat i2∈nat i3∈nat i4∈nat i5∈nat i6∈nat env ∈ list(A)
shows
(∃xa∈A. ∃xb∈A. pair(##A,x,a,xa) & xa ∈ r & pair(##A,x,b,xb) & xb ∈ r &
(∃fx∈A. ∃gx∈A. fun_apply(##A,f,x,fx) & fun_apply(##A,g,x,gx) & fx ≠ gx))
⟷ sats(A,?rffm(i1,i2,i3,i4,i5,i6),env)
⟨proof⟩

lemma *is_recfun_sep_intf* :
assumes
r∈M f∈M g∈M a∈M b∈M
shows
separation(##M,λx. ∃xa∈M. ∃xb∈M.
pair(##M,x,a,xa) & xa ∈ r & pair(##M,x,b,xb) & xb ∈ r &

($\exists$ *fx*∈*M*. $\exists$ *gx*∈*M*. *fun_apply*(##*M*,*f*,*x*,*fx*) & *fun_apply*(##*M*,*g*,*x*,*gx*)
&
fx ≠ *gx*))
⟨*proof*⟩

schematic_goal *funsp_fm_auto*:
assumes
nth(*i*,*env*) = *p* *nth*(*j*,*env*) = *z* *nth*(*h*,*env*) = *n*
i ∈ *nat* *j* ∈ *nat* *h* ∈ *nat* *env* ∈ *list*(*A*)
shows
($\exists$ *f*∈*A*. $\exists$ *b*∈*A*. $\exists$ *nb*∈*A*. $\exists$ *cnbf*∈*A*. *pair*(##*A*,*f*,*b*,*p*) & *pair*(##*A*,*n*,*b*,*nb*) & *is_cons*(##*A*,*nb*,*f*,*cnbf*) &
upair(##*A*,*cnbf*,*cnbf*,*z*)) ⟷ *sats*(*A*,*?fsfm*(*i*,*j*,*h*),*env*)
⟨*proof*⟩

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

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

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

end

sublocale *M_ZF_trans* ⊆ *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 ∈ *nat* *j* ∈ *nat* *k* ∈ *nat* *env* ∈ *list*(*A*)
shows
rtran_closure_mem(##*A*,*B*,*r*,*p*) ⟷ *sats*(*A*,*?rcfm*(*i*,*j*,*k*),*env*)
⟨*proof*⟩

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

schematic_goal *rtran_closure_fm_auto*:
assumes
nth(*i*,*env*) = *r* *nth*(*j*,*env*) = *rp*
i ∈ *nat* *j* ∈ *nat* *env* ∈ *list*(*A*)
shows
rtran_closure(##*A*,*r*,*rp*) ⟷ *sats*(*A*,*?rtc*(*i*,*j*),*env*)
⟨*proof*⟩

schematic_goal *tran_closure_fm_auto*:
assumes
nth(*i*,*env*) = *r* *nth*(*j*,*env*) = *rp*
i ∈ *nat* *j* ∈ *nat* *env* ∈ *list*(*A*)
shows
tran_closure(##*A*,*r*,*rp*) ⟷ *sats*(*A*,*?tc*(*i*,*j*),*env*)
⟨*proof*⟩

⟨*ML*⟩

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

lemma *tran_closure_iff_sats*:
assumes
nth(*i*,*env*) = *r* *nth*(*j*,*env*) = *rp*
i ∈ *nat* *j* ∈ *nat* *env* ∈ *list*(*A*)
shows
tran_closure(##*A*,*r*,*rp*) ⟷ *sats*(*A*,*tran_closure_fm*(*i*,*j*),*env*)
⟨*proof*⟩

lemma *sats_tran_closure_fm* :

assumes
  $i \in nat$ $j \in nat$ $env \in list(A)$
shows
  $sats(A,tran\_closure\_fm(i,j),env) \longleftrightarrow tran\_closure(\#\#A,nth(i,env),nth(j,env))$
  ⟨proof⟩

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

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

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

lemma (**in** *M_ZF_trans*) *mtrancl* : $M\_trancl(\#\#M)$
  ⟨proof⟩

sublocale *M_ZF_trans* ⊆ *M_trancl* $\#\#M$

⟨*proof*⟩

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
 $strong\_replacement(\#\#M,\lambda x\ z.\ sats(M,\varphi,Cons(x,Cons(z,env)))) \longleftrightarrow$
 $strong\_replacement(\#\#M,P)$
 ⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *nat_trans_M* :
 $n\in M$ **if** $n\in nat$ **for** n
 ⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *list_repl1_intf*:
 assumes
 $A\in M$
 shows
 $iterates\_replacement(\#\#M,\ is\_list\_functor(\#\#M,A),\ 0)$
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *iterates_repl_intf* :
 assumes
 $v\in M$ **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
 $iterates\_replacement(\#\#M,\lambda l'\ t.\ is\_tl(\#\#M,l',t),l)$
⟨*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*]:
[| $x \in nat$; $y \in nat$ |] ==> *powerset_fm*$(x,y) \in$ *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*] :
⟦$f \in nat$; $y \in nat$; $z \in nat$⟧ ⟹ *is_powapply_fm*$(f,y,z) \in$ *formula*
⟨*proof*⟩

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

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

definition
PHrank_fm :: [*i*,*i*,*i*] ⇒ *i* **where**
PHrank_fm(*f*,*y*,*z*) == *Exists*(*And*(*fun_apply_fm*(*succ*(*f*),*succ*(*y*),*0*)
,*succ_fm*(*0*,*succ*(*z*))))

lemma *PHrank_type* [*TC*]:
[| $x \in nat$; $y \in nat$; $z \in nat$ |] ==> *PHrank_fm*$(x,y,z) \in$ *formula*
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *sats_PHrank_fm* [*simp*]:
[| $x \in nat$; $y \in nat$; $z \in nat$; $env \in list(M)$|]

==> *sats*(*M*,*PHrank_fm*(*x*,*y*,*z*),*env*) ⟷
PHrank(##*M*,*nth*(*x*,*env*),*nth*(*y*,*env*),*nth*(*z*,*env*))
⟨*proof*⟩

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

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

lemma *is_Hrank_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *is_Hrank_fm*(*x*,*y*,*z*) ∈ *formula*
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *sats_is_Hrank_fm* [*simp*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*M*)|]
==> *sats*(*M*,*is_Hrank_fm*(*x*,*y*,*z*),*env*) ⟷
is_Hrank(##*M*,*nth*(*x*,*env*),*nth*(*y*,*env*),*nth*(*z*,*env*))
⟨*proof*⟩

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

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

lemma *is_HVfrom_type* [*TC*]:
[| *A*∈*nat*; *x* ∈ *nat*; *f* ∈ *nat*; *h* ∈ *nat* |] ==> *is_HVfrom_fm*(*A*,*x*,*f*,*h*) ∈ *formula*
⟨*proof*⟩

lemma *sats_is_HVfrom_fm* :
[| *a*∈*nat*; *x* ∈ *nat*; *f* ∈ *nat*; *h* ∈ *nat*; *env* ∈ *list*(*A*); *0*∈*A*|]
==> *sats*(*A*,*is_HVfrom_fm*(*a*,*x*,*f*,*h*),*env*) ⟷
is_HVfrom(##*A*,*nth*(*a*,*env*),*nth*(*x*,*env*),*nth*(*f*,*env*),*nth*(*h*,*env*))
⟨*proof*⟩

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

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

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

lemma (**in** *M_ZF_trans*) *memrel_eclose_sing* :
a∈*M* ⟹ ∃ *sa*∈*M*. ∃ *esa*∈*M*. ∃ *mesa*∈*M*.
upair(##*M*,*a*,*a*,*sa*) & *is_eclose*(##*M*,*sa*,*esa*) & *membership*(##*M*,*esa*,*mesa*)

⟨*proof*⟩

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

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

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

lemma (**in** *M_ZF_trans*) *repl_gen* :
assumes
f_abs: ⋀*x y*. ⟦ *x*∈*M*; *y*∈*M* ⟧ ⟹ *is_F*(##*M*,*x*,*y*) ⟷ *y* = *f*(*x*)
and
f_sats: ⋀*x y*. ⟦*x*∈*M* ; *y*∈*M* ⟧ ⟹
sats(*M*,*f_fm*,*Cons*(*x*,*Cons*(*y*,*env*))) ⟷ *is_F*(##*M*,*x*,*y*)
and
f_form: *f_fm* ∈ *formula*
and
f_arty: *arity*(*f_fm*) = *2*
and
env∈*list*(*M*)
shows
strong_replacement(##*M*, λ*x y*. *y* = *f*(*x*))
⟨*proof*⟩

lemma (**in** *M_ZF_trans*) *sep_in_M* :
assumes
φ ∈ *formula env*∈*list*(*M*)
arity(φ) ≤ *1* #+ *length*(*env*) *A*∈*M* **and**
satsQ: ⋀*x*. *x*∈*M* ⟹ *sats*(*M*,φ,[*x*]@*env*) ⟷ *Q*(*x*)
shows
{*y*∈*A* . *Q*(*y*)}∈*M*
⟨*proof*⟩

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
ZF-Constructible-Trans.Relative
ZF-Constructible-Trans.Formula
Interface

begin

lemma *Transset_M* :

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

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

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

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

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

lemma *zero_in_M*: *0* ∈ *M*
⟨*proof*⟩

lemma *tuples_in_M*: *A*∈*M* ⟹ *B*∈*M* ⟹ <*A*,*B*>∈*M*
⟨*proof*⟩

lemma *nat_in_M* : *nat* ∈ *M*
⟨*proof*⟩

lemma *n_in_M* : *n*∈*nat* ⟹ *n*∈*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* $\Longrightarrow$ *sats*(*M*,*Q_fm*,[*x*]) $\longleftrightarrow$ *is_Q*(##*M*,*x*) **and**
Qabs : $\bigwedge$*x*. *x*∈*M* $\Longrightarrow$ *is_Q*(##*M*,*x*) $\longleftrightarrow$ *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* $\Longrightarrow$ *sats*(*M*,*Q_fm*,[*x*,*y*,*z*]) $\longleftrightarrow$ *is_Q*(##*M*,*x*,*y*,*z*) **and**
Qabs : $\bigwedge$*x*. *x*∈*M* $\Longrightarrow$ *is_Q*(##*M*,*x*,*y*,*z*) $\longleftrightarrow$ *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* $\Longrightarrow$ *sats*(*M*,*Q_fm*,[*x*,*a1*,*a2*,*a3*,*a4*]) $\longleftrightarrow$ *is_Q*(##*M*,*x*,*a1*,*a2*,*a3*,*a4*)
and
Qabs : $\bigwedge$*x*. *x*∈*M* $\Longrightarrow$ *is_Q*(##*M*,*x*,*a1*,*a2*,*a3*,*a4*) $\longleftrightarrow$ *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* ∈ *formula* **and**
f_ar: *arity*(*f_fm*)≤ *2* #+ *length*(*env*) **and**
fsats: $\bigwedge$*x* *y*. *x*∈*M* $\Longrightarrow$ *y*∈*M* $\Longrightarrow$ *sats*(*M*,*f_fm*,[*x*,*y*]@*env*) $\longleftrightarrow$ *is_f*(*x*,*y*) **and**
fabs: $\bigwedge$*x* *y*. *x*∈*M* $\Longrightarrow$ *y*∈*M* $\Longrightarrow$ *is_f*(*x*,*y*) $\longleftrightarrow$ *y* = *f*(*x*) **and**
fclosed: $\bigwedge$*x*. *x*∈*A* $\Longrightarrow$ *f*(*x*) ∈ *M* **and**
A∈*M* *env*∈*list*(*M*)
shows {*f*(*x*). *x*∈*A*}∈*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**
  $M\_generic(G) == filter(G) \wedge (\forall D{\in}M.\ D{\subseteq}P \wedge dense(D){\longrightarrow}D{\cap}G{\neq}0)$

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∈M r∈M p∈M q∈M
shows
is_compat_in(##M,A,r,p,q) ⟷ *compat_in(A,r,p,q)*
⟨*proof*⟩

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

lemma *compat_in_fm_type*[*TC*] :
⟦ *A∈nat;r∈nat;p∈nat;q∈nat*⟧ ⟹ *compat_in_fm(A,r,p,q)∈formula*
⟨*proof*⟩

lemma *sats_compat_in_fm*:
assumes
A∈nat r∈nat p∈nat q∈nat env∈list(M)
shows
sats(M,compat_in_fm(A,r,p,q),env) ⟷
is_compat_in(##M,nth(A, env),nth(r, env),nth(p, env),nth(q, env))
⟨*proof*⟩

end

end

12 The ZFC axioms, internalized

theory *Internal_ZFC_Axioms*
imports
Forcing_Data

begin

schematic_goal *ZF_union_auto*:
Union_ax(##A) ⟷ (*A*, [] ⊨ *?zfunion*)
⟨*proof*⟩

⟨*ML*⟩

lemma *ZF_union_fm_ty*[*TC*] :
ZF_union_fm∈formula

⟨*proof*⟩

lemma *sats_ZF_union_fm* :
$(A, [] \models ZF\_union\_fm) \longleftrightarrow Union\_ax(\#\#A)$
⟨*proof*⟩

lemma *Union_ax_iff_sats* :
$Union\_ax(\#\#A) \longleftrightarrow (A, [] \models ZF\_union\_fm)$
⟨*proof*⟩

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

⟨*ML*⟩

lemma *ZF_power_fm_ty*[*TC*] :
$ZF\_power\_fm \in formula$
⟨*proof*⟩

lemma *sats_ZF_power_fm* :
$(A, [] \models ZF\_power\_fm) \longleftrightarrow power\_ax(\#\#A)$
⟨*proof*⟩

lemma *power_ax_iff_sats* :
$power\_ax(\#\#A) \longleftrightarrow (A, [] \models ZF\_power\_fm)$
⟨*proof*⟩

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

⟨*ML*⟩

lemma *ZF_pairing_fm_ty*[*TC*] :
$ZF\_pairing\_fm \in formula$
⟨*proof*⟩

lemma *sats_ZF_pairing_fm* :
$(A, [] \models ZF\_pairing\_fm) \longleftrightarrow upair\_ax(\#\#A)$
⟨*proof*⟩

lemma *upair_ax_iff_sats* :
$upair\_ax(\#\#A) \longleftrightarrow (A, [] \models ZF\_pairing\_fm)$
⟨*proof*⟩

schematic_goal *ZF_foundation_auto*:
$foundation\_ax(\#\#A) \longleftrightarrow (A, [] \models ?zfpow)$

⟨*proof*⟩

⟨*ML*⟩

lemma *ZF_foundation_fm_ty*[*TC*] :
ZF_foundation_fm∈*formula*
⟨*proof*⟩

lemma *sats_ZF_foundation_fm* :
$(A, [] \models$ *ZF_foundation_fm*) ⟷ *foundation_ax*(##*A*)
⟨*proof*⟩

lemma *foundation_ax_iff_sats* :
foundation_ax(##*A*) ⟷ $(A, [] \models$ *ZF_foundation_fm*)
⟨*proof*⟩

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

⟨*ML*⟩

lemma *ZF_extensionality_fm_ty*[*TC*] :
ZF_extensionality_fm∈*formula*
⟨*proof*⟩

lemma *sats_ZF_extensionality_fm* :
$(A, [] \models$ *ZF_extensionality_fm*) ⟷ *extensionality*(##*A*)
⟨*proof*⟩

lemma *extensionality_iff_sats* :
extensionality(##*A*) ⟷ $(A, [] \models$ *ZF_extensionality_fm*)
⟨*proof*⟩

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

⟨*ML*⟩

lemma *ZF_infinity_fm_ty*[*TC*] :
ZF_infinity_fm∈*formula*
⟨*proof*⟩

lemma *sats_ZF_infinity_fm* :
$(A, [] \models$ *ZF_infinity_fm*) ⟷ *infinity_ax*(##*A*)
⟨*proof*⟩

lemma *infinity_iff_sats* :

infinity_ax(##*A*) ⟷ (*A*, [] ⊨ *ZF_infinity_fm*)
⟨*proof*⟩

schematic_goal *ZF_choice_auto*:
choice_ax(##*A*) ⟷ (*A*, [] ⊨ (*?φ*(*i*,*j*,*h*)))
⟨*proof*⟩

⟨*ML*⟩

lemma *ZF_choice_fm_ty*[*TC*] :
ZF_choice_fm∈*formula*
⟨*proof*⟩

lemma *sats_ZF_choice_fm* :
(*A*, [] ⊨ *ZF_choice_fm*) ⟷ *choice_ax*(##*A*)
⟨*proof*⟩

lemma *choice_iff_sats* :
choice_ax(##*A*) ⟷ (*A*, [] ⊨ *ZF_choice_fm*)
⟨*proof*⟩

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

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

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

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

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

lemma *ZFC_fin_type* : *ZFC_fin* ⊆ *formula*
⟨*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 $l \in list(A)$ $length(l) = succ(n)$
  shows $\exists\ a{\in}A.\ \exists\, l'{\in}list(A).\ l = l'@[a]$
⟨*proof*⟩

lemma *take_drop_eq* :
  assumes $l{\in}list(M)$
  shows $\bigwedge\ n\ .\ n < succ(length(l)) \Longrightarrow l = take(n,l)\ @\ drop(n,l)$
  ⟨*proof*⟩

lemma *list_split* :
assumes $n \le succ(length(rest))$ $rest \in list(M)$
shows $\exists\, re{\in}list(M).\ \exists\, st{\in}list(M).\ rest = re\ @\ st \wedge length(re) = pred(n)$
⟨*proof*⟩

lemma *sats_nForall*:
  assumes
    $\varphi \in formula$
  shows
    $n{\in}nat \Longrightarrow ms \in list(M) \Longrightarrow$
      $M,\ ms \models (Forall\hat{\ }n(\varphi)) \longleftrightarrow$
      $(\forall\, rest \in list(M).\ length(rest) = n \longrightarrow M,\ rest\ @\ ms \models \varphi)$
⟨*proof*⟩

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

lemma *sep_body_fm_type* [*TC*]: $p \in formula \Longrightarrow sep\_body\_fm(p) \in formula$
  ⟨*proof*⟩

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

definition
  ZF_separation_fm :: $i \Rightarrow i$ **where**
  $ZF\_separation\_fm(p) == Forall\hat{\ }(pred(arity(p)))(sep\_body\_fm(p))$

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

lemma *sats_ZF_separation_fm_iff*:
  assumes
    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*) ⟷ *sats*(*A*,*?ufm*(*i*),*env*)
  ⟨*proof*⟩

⟨*ML*⟩

lemma *univalent_fm_type* [*TC*]: *q1*∈ *formula* ⟹ *q2*∈*formula* ⟹ *i*∈*nat* ⟹
  univalent_fm(*q2*,*q1*,*i*) ∈*formula*
  ⟨*proof*⟩

lemma *sats_univalent_fm* :
  assumes
    Q_iff_sats:⋀*x y z*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *z*∈*A* ⟹
        Q(*x*,*z*) ⟷ (*A*,*Cons*(*z*,*Cons*(*y*,*Cons*(*x*,*env*))) ⊨ *Q1_fm*)
      ⋀*x y z*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *z*∈*A* ⟹
        Q(*x*,*y*) ⟷ (*A*,*Cons*(*z*,*Cons*(*y*,*Cons*(*x*,*env*))) ⊨ *Q2_fm*)
    and
    asms: *nth*(*i*,*env*) = *B* *i* ∈ *nat* *env* ∈ *list*(*A*)
  shows
    sats(*A*,*univalent_fm*(*Q1_fm*,*Q2_fm*,*i*),*env*) ⟷ *univalent*(##*A*,*B*,*Q*)
  ⟨*proof*⟩

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

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

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

definition
univalent_Q1 :: $i \Rightarrow i$ **where**
univalent_Q1(φ) $\equiv$ *incr_bv1*(*swap_vars*(φ))

definition
univalent_Q2 :: $i \Rightarrow i$ **where**
univalent_Q2(φ) $\equiv$ *incr_bv*(*swap_vars*(φ))‘*0*

lemma *univalent_Qs_type* [*TC*]:
assumes $\varphi\in$*formula*
shows *univalent_Q1*(φ) $\in$ *formula* *univalent_Q2*(φ) $\in$ *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*(φ)))
(A, ([x,y] @ *env*) $\models \varphi$) $\longleftrightarrow$ (A, *Cons*(z,*Cons*(y,*Cons*(x,*env*))) $\models$ (*univalent_Q2*(φ)))
⟨*proof*⟩

definition
rep_body_fm :: $i \Rightarrow i$ **where**
rep_body_fm(p) == *Forall*(*Implies*(
univalent_fm(*univalent_Q1*(*incr_bv*(p)‘*2*),*univalent_Q2*(*incr_bv*(p)‘*2*),*0*),
Exists(*Forall*(
Iff(*Member*(*0*,*1*),*Exists*(*And*(*Member*(*0*,*3*),*incr_bv*(*incr_bv*(p)‘*2*)‘*2*)))))))

lemma *rep_body_fm_type* [*TC*]: $p \in$ *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* ⊨ *rep_body_fm*(φ) ⟷
  strong_replacement(##*M*,λ*x y*. *M*, [*x*,*y*] @ *rest* @ *ms* ⊨ φ)
⟨*proof*⟩

definition
ZF_replacement_fm :: *i* ⇒ *i* **where**
ZF_replacement_fm(*p*) ≡ *Forall*^(*pred*(*pred*(*arity*(*p*))))(*rep_body_fm*(*p*))

lemma *ZF_replacement_fm_type* [*TC*]: *p* ∈ *formula* ⟹ *ZF_replacement_fm*(*p*) ∈ *formula*
⟨*proof*⟩

lemma *sats_ZF_replacement_fm_iff*:
assumes
  φ∈*formula*
shows
(*M*, [] ⊨ (*ZF_replacement_fm*(φ)))
⟷
(∀ *env*∈*list*(*M*). *arity*(φ) ≤ *2* #+ *length*(*env*) ⟶
  strong_replacement(##*M*,λ*x y*. *sats*(*M*,φ,[*x*,*y*] @ *env*)))
⟨*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 :: $[i,i] \Rightarrow o$ (_ $\models$ _ $[36,36]$ 60) **where**
$A \models \Phi \equiv \forall\varphi\in\Phi.\ (A,[] \models \varphi)$

lemma *satTI* [*intro*!]:
assumes $\bigwedge\varphi.\ \varphi\in\Phi \Longrightarrow A,[] \models \varphi$
shows $A \models \Phi$
⟨*proof*⟩

lemma *satTD* [*dest*] :$A \models \Phi \Longrightarrow \varphi\in\Phi \Longrightarrow A,[] \models \varphi$
⟨*proof*⟩

lemma *sats_ZFC_iff_sats_ZF_AC*:
$(N \models ZFC) \longleftrightarrow (N \models ZF) \wedge (N, [] \models AC)$
⟨*proof*⟩

lemma *M_ZF_iff_M_satT*: $M\_ZF(M) \longleftrightarrow (M \models ZF)$
⟨*proof*⟩

end

13 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**
$SepReplace(A,b,Q) == \{y\ .\ x\in A,\ y=b(x) \wedge Q(x)\}$

syntax
SepReplace :: [*i*, *pttrn*, *i*, *o*] => *i* ((*1*{ ../ _ ∈ _, _}))
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*⟩

13.1 The well-founded relation *ed*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *restrict_edrel_eq* :
  assumes $z \in domain(x)$

shows *edrel*(*eclose*({*x*}))∩ *eclose*({*z*})∗*eclose*({*z*}) = *edrel*(*eclose*({*z*}))
⟨*proof*⟩

lemma *tr_edrel_subset* :
assumes *z* ∈ *domain*(*x*)
shows *tr_down*(*edrel*(*eclose*({*x*})),*z*) ⊆ *eclose*({*z*})
⟨*proof*⟩

context *M_ctm*
begin

lemma *upairM* : *x* ∈ *M* ⟹ *y* ∈ *M* ⟹ {*x*,*y*} ∈ *M*
⟨*proof*⟩

lemma *singletonM* : *a* ∈ *M* ⟹ {*a*} ∈ *M*
⟨*proof*⟩

lemma *pairM* : *x* ∈ *M* ⟹ *y* ∈ *M* ⟹ <*x*,*y*> ∈ *M*
⟨*proof*⟩

lemma *Rep_simp* : *Replace*(*u*,λ *y z* . *z* = *f*(*y*)) = { *f*(*y*) . *y* ∈ *u*}
⟨*proof*⟩

end

13.2 Values and check-names

context *forcing_data*
begin

definition
Hcheck :: [*i*,*i*] ⇒ *i* **where**
Hcheck(*z*,*f*) == { <*f*‘*y*,*one*> . *y* ∈ *z*}

definition
check :: *i* ⇒ *i* **where**
check(*x*) == *transrec*(*x* , *Hcheck*)

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

definition
rcheck :: *i* ⇒ *i* **where**
rcheck(*x*) == *Memrel*(*eclose*({*x*}))ˆ+

lemma *Hcheck_trancl*:*Hcheck*(*y*, *restrict*(*f*,*Memrel*(*eclose*({*x*}))-‘‘{*y*}))

= *Hcheck*(*y*, *restrict*(*f*,(*Memrel*(*eclose*({*x*}))^+)-``{*y*}))
⟨*proof*⟩

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

lemma *rcheck_in_M* :
x ∈ *M* ⟹ *rcheck*(*x*) ∈ *M*
⟨*proof*⟩

lemma *aux_def_check*: *x* ∈ *y* ⟹
wfrec(*Memrel*(*eclose*({*y*})), *x*, *Hcheck*) =
wfrec(*Memrel*(*eclose*({*x*})), *x*, *Hcheck*)
⟨*proof*⟩

lemma *def_check* : *check*(*y*) = { <*check*(*w*),*one*> . *w* ∈ *y*}
⟨*proof*⟩

lemma *def_checkS* :
fixes *n*
assumes *n* ∈ *nat*
shows *check*(*succ*(*n*)) = *check*(*n*) ∪ {<*check*(*n*),*one*>}
⟨*proof*⟩

lemma *field_Memrel2* : *x* ∈ *M* ⟹ *field*(*Memrel*(*eclose*({*x*}))) ⊆ *M*
⟨*proof*⟩

definition
Hv :: *i*⇒*i*⇒*i*⇒*i* **where**
Hv(*G*,*x*,*f*) == { *f*`*y* .. *y*∈ *domain*(*x*), ∃ *p*∈*P*. <*y*,*p*> ∈ *x* ∧ *p* ∈ *G* }

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

definition
val :: *i*⇒*i*⇒*i* **where**
val(*G*,τ) == *wfrec*(*edrel*(*eclose*({τ})), τ ,*Hv*(*G*))

lemma *aux_def_val*:
assumes *z* ∈ *domain*(*x*)
shows *wfrec*(*edrel*(*eclose*({*x*})),*z*,*Hv*(*G*)) = *wfrec*(*edrel*(*eclose*({*z*})),*z*,*Hv*(*G*))
⟨*proof*⟩

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

lemma *def_val*: *val*(*G*,*x*) = {*val*(*G*,*t*) .. *t*∈*domain*(*x*) , ∃ *p*∈*P* . <*t*,*p*>∈*x* ∧ *p* ∈ *G* }

⟨*proof*⟩

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

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

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

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

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

lemma *val_only_names*: $val(F,\tau) = val(F,\{x \in \tau.\ \exists t \in domain(\tau).\ \exists p \in P.\ x=<t,p>\})$

(**is** _ = $val(F,?name)$)
⟨*proof*⟩

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

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

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

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

lemma *val_of_elem*: $<\vartheta,p> \in \pi \Longrightarrow p \in G \Longrightarrow p \in P \Longrightarrow val(G,\vartheta) \in val(G,\pi)$
⟨*proof*⟩

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

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

lemma *elem_of_val_pair'*:
assumes $\pi \in M$ $x \in val(G,\pi)$
shows $\exists \vartheta \in M.\ \exists p \in G.\ <\vartheta,p> \in \pi \wedge val(G,\vartheta) = x$

⟨*proof*⟩

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

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

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

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

lemma *check_n_M* :
  fixes *n*
  assumes $n \in nat$
  shows $check(n) \in M$
  ⟨*proof*⟩

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

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

lemma *one_in_M*: $one \in M$
  ⟨*proof*⟩

lemma *def_PHcheck*:
  assumes
    $z\in M$ $f\in M$
  shows
    $Hcheck(z,f) = Replace(z,PHcheck(one,f))$
⟨*proof*⟩

definition
  PHcheck_fm :: [*i*,*i*,*i*,*i*] ⇒ *i* **where**
  $PHcheck\_fm(o,f,y,p) == Exists(And(fun\_apply\_fm(succ(f),succ(y),0)$
  $,pair\_fm(0,succ(o),succ(p))))$

lemma *PHcheck_type* [*TC*]:
[| x ∈ nat; y ∈ nat; z ∈ nat; u ∈ nat |] ==> PHcheck_fm(x,y,z,u) ∈ formula
⟨proof⟩

lemma *sats_PHcheck_fm* [*simp*]:
[| x ∈ nat; y ∈ nat; z ∈ nat; u ∈ nat ; env ∈ list(M)|]
==> sats(M,PHcheck_fm(x,y,z,u),env) ⟷
PHcheck(nth(x,env),nth(y,env),nth(z,env),nth(u,env))
⟨proof⟩

definition
is_Hcheck_fm :: [i,i,i,i] ⇒ i **where**
is_Hcheck_fm(o,z,f,hc) == Replace_fm(z,PHcheck_fm(succ(succ(o)),succ(succ(f)),0,1),hc)

lemma *is_Hcheck_type* [*TC*]:
[| x ∈ nat; y ∈ nat; z ∈ nat; u ∈ nat |] ==> is_Hcheck_fm(x,y,z,u) ∈ formula
⟨proof⟩

lemma *sats_is_Hcheck_fm* [*simp*]:
[| x ∈ nat; y ∈ nat; z ∈ nat; u ∈ nat ; env ∈ list(M)|]
==> sats(M,is_Hcheck_fm(x,y,z,u),env) ⟷
is_Hcheck(nth(x,env),nth(y,env),nth(z,env),nth(u,env))
⟨proof⟩

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

lemma *repl_PHcheck* :
assumes
f∈M
shows
strong_replacement(##M,PHcheck(one,f))
⟨proof⟩

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

lemma *relation2_Hcheck* :
relation2(##M,is_Hcheck(one),Hcheck)
⟨proof⟩

lemma *PHcheck_closed* :
⟦$z \in M$; $f \in M$; $x \in z$; $PHcheck(one,f,x,y)$ ⟧ $\Longrightarrow$ $(\#\#M)(y)$
⟨*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**
$is\_singleton(A,x,z) == \exists c[A].\ empty(A,c) \wedge is\_cons(A,x,c,z)$

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**
$singleton\_fm(i,j) == Exists(And(empty\_fm(0),\ cons\_fm(succ(i),0,succ(j))))$

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

lemma *sats_singleton_fm*:
⟦ $i \in nat$; $j \in nat$; $env \in list(A)$ ⟧
$\Longrightarrow sats(A,singleton\_fm(i,j),env) \longleftrightarrow is\_singleton(\#\#A,nth(i,env),nth(j,env))$
⟨*proof*⟩

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

context *forcing_data* **begin**

definition
is_rcheck :: $[i,i] \Rightarrow o$ **where**
is_rcheck$(x,z) == \exists r\in M.$ *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
nth$(i,env) = x$ *nth*$(j,env) = z$
$i \in nat\ j \in nat\ env \in list(M)$
shows
is_rcheck$(x,z) \longleftrightarrow$ *sats*$(M,?rch(i,j),env)$
⟨*proof*⟩

⟨*ML*⟩

lemma *sats_rcheck_fm* :
assumes
$i \in nat\ j \in nat\ i<length(env)\ j<length(env)\ env \in list(M)$
shows
sats$(M,$*rcheck_fm*$(i,j),env) \longleftrightarrow$ *is_rcheck*$(nth(i,env),nth(j,env))$
⟨*proof*⟩

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

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

lemma *check_abs* :
assumes
$x\in M\ z\in M$
shows
is_check$(x,z) \longleftrightarrow z =$ *check*(x)
⟨*proof*⟩

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*∈*nat*;*o*∈*nat*;*z*∈*nat*⟧ ⟹ *check_fm*(*x*,*o*,*z*)∈*formula*
⟨*proof*⟩

lemma *sats_check_fm* :
assumes
nth(*o*,*env*) = *one* *x*∈*nat* *z*∈*nat* *o*∈*nat* *env*∈*list*(*M*) *x* < *length*(*env*) *z* < *length*(*env*)
shows
sats(*M*, *check_fm*(*x*,*o*,*z*), *env*) ⟷ *is_check*(*nth*(*x*,*env*),*nth*(*z*,*env*))
⟨*proof*⟩

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

lemma *pair_check* : ⟦ *p*∈*M* ; *y*∈*M* ⟧ ⟹ (∃ *c*∈*M*. *is_check*(*p*,*c*) ∧ *pair*(*##M*,*c*,*p*,*y*)) ⟷ *y* = <*check*(*p*),*p*>
⟨*proof*⟩

lemma *M_subset_MG* : *one* ∈ *G* ⟹ *M* ⊆ *M*[*G*]
⟨*proof*⟩

The name for the generic filter

definition
G_dot :: *i* **where**
G_dot == {<*check*(*p*),*p*> . *p*∈*P*}

lemma *G_dot_in_M* :
G_dot ∈ *M*
⟨*proof*⟩

lemma *val_G_dot* :
assumes *G* ⊆ *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*: $p{\in}M \Longrightarrow fst(p) \in M \wedge snd(p){\in} M$
⟨*proof*⟩

end

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

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

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

end
end

14 Well-founded relation on names

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

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

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

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

lemma (**in** *M_trivial*) *hcomp_abs*:
assumes
is_f_abs:$\bigwedge a\ z.\ M(a) \Longrightarrow M(z) \Longrightarrow is\_f(a,z) \longleftrightarrow z = f(a)$ **and**
is_g_abs:$\bigwedge a\ z.\ M(a) \Longrightarrow M(z) \Longrightarrow is\_g(a,z) \longleftrightarrow z = g(a)$ **and**
g_closed:$\bigwedge a.\ M(a) \Longrightarrow M(g(a))$
$M(a)\ M(w)$
shows
is_hcomp$(M,is\_f,is\_g,a,w) \longleftrightarrow w = f(g(a))$
⟨*proof*⟩

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:⋀*a b z*. *a*∈*nat* ⟹ *b*∈*nat* ⟹ *z*∈*M* ⟹
is_f(*nth*(*a*,*Cons*(*z*,*env*)),*nth*(*b*,*Cons*(*z*,*env*))) ⟷ *sats*(*M*,*pf*(*a*,*b*),*Cons*(*z*,*env*))
and
g_iff_sats:⋀*a b z*. *a*∈*nat* ⟹ *b*∈*nat* ⟹ *z*∈*M* ⟹
is_g(*nth*(*a*,*Cons*(*z*,*env*)),*nth*(*b*,*Cons*(*z*,*env*))) ⟷ *sats*(*M*,*pg*(*a*,*b*),*Cons*(*z*,*env*))
and
a∈*nat* *w*∈*nat* *env*∈*list*(*M*)
shows
sats(*M*,*hcomp_fm*(*pf*,*pg*,*a*,*w*),*env*) ⟷ *is_hcomp*(##*M*,*is_f*,*is_g*,*nth*(*a*,*env*),*nth*(*w*,*env*))

⟨*proof*⟩

definition
ftype :: *i*⇒*i* **where**
ftype == *fst*

definition
name1 :: *i*⇒*i* **where**
name1(*x*) == *fst*(*snd*(*x*))

definition
name2 :: *i*⇒*i* **where**
name2(*x*) == *fst*(*snd*(*snd*(*x*)))

definition
cond_of :: *i*⇒*i* **where**
cond_of(*x*) == *snd*(*snd*(*snd*((*x*))))

lemma *components_simp*:
ftype(<*f*,*n1*,*n2*,*c*>) = *f*
name1(<*f*,*n1*,*n2*,*c*>) = *n1*
name2(<*f*,*n1*,*n2*,*c*>) = *n2*
cond_of(<*f*,*n1*,*n2*,*c*>) = *c*
⟨*proof*⟩

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

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

lemma *components_in_eclose* :
n1 ∈ *ecloseN*(<*f*,*n1*,*n2*,*c*>)
n2 ∈ *ecloseN*(<*f*,*n1*,*n2*,*c*>)

⟨*proof*⟩

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

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

lemma *ecloseNI2* :
  assumes *y* ∈ *eclose*(*n2*)
  shows *y* ∈ *ecloseN*(<*f*,*n1*,*n2*,*c*>)
⟨*proof*⟩

lemmas *ecloseNI* = *ecloseNI1 ecloseNI2*

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*] ⇒ *i* **where**
  ftype_fm ≡ *fst_fm*

lemma *sats_ftype_fm* :

⟦ $x \in nat$; $y \in nat$;$env \in list(A)$ ⟧
⟹ $sats(A, ftype\_fm(x,y), env) \longleftrightarrow$
$is\_ftype(\#\#A, nth(x,env), nth(y,env))$
⟨*proof*⟩

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

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

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* :
⟦ $x \in nat$; $y \in nat$;$env \in list(A)$ ⟧
⟹ $sats(A, snd\_fm(x,y), env) \longleftrightarrow$
$is\_snd(\#\#A, nth(x,env), nth(y,env))$
⟨*proof*⟩

definition
is_name1 :: $(i \Rightarrow o) \Rightarrow i \Rightarrow i \Rightarrow o$ **where**
$is\_name1(M,x,t2) == is\_hcomp(M,is\_fst(M),is\_snd(M),x,t2)$

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* :
⟦ $x \in nat$; $y \in nat$;$env \in list(A)$ ⟧
⟹ $sats(A, name1\_fm(x,y), env) \longleftrightarrow$
$is\_name1(\#\#A, nth(x,env), nth(y,env))$
⟨*proof*⟩

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**
$is\_snd\_snd(M,x,t) == is\_hcomp(M,is\_snd(M),is\_snd(M),x,t)$

definition
$snd\_snd\_fm :: [i,i]{\Rightarrow}i$ **where**
$snd\_snd\_fm(x,t) == hcomp\_fm(snd\_fm,snd\_fm,x,t)$

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**
$is\_name2(M,x,t3) == is\_hcomp(M,is\_fst(M),is\_snd\_snd(M),x,t3)$

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**
$is\_cond\_of(M,x,t4) == is\_hcomp(M,is\_snd(M),is\_snd\_snd(M),x,t4)$

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*) ⟷ *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* = *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*) ∧ *is_eclose_n*(*N*,*is_name2*,*en2*,*t*)∧
union(*N*,*en1*,*en2*,*en*)

definition
ecloseN_fm :: [*i*,*i*] ⇒ *i* **where**
ecloseN_fm(*en*,*t*) == *Exists*(*Exists*(*And*(*eclose_n1_fm*(*1*,*t*#+*2*),

And(eclose_n2_fm(0,t#+2),union_fm(1,0,en#+2)))))

lemma *ecloseN_fm_type* [*TC*] :
⟦ *en* ∈ *nat* ; *t* ∈ *nat* ⟧ ⟹ *ecloseN_fm(en,t)* ∈ *formula*
⟨*proof*⟩

lemma *sats_ecloseN_fm* [*simp*]:
[| *en* ∈ *nat*; *t* ∈ *nat* ; *env* ∈ *list(A)*|]
==> *sats(A, ecloseN_fm(en,t), env)* ⟷ *is_ecloseN(##A,nth(en,env),nth(t,env))*
⟨*proof*⟩

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

lemma *frecR_ftypeD* :
assumes *frecR(x,y)*
shows (*ftype(x)* = *0* ∧ *ftype(y)* = *1*) ∨ (*ftype(x)* = *1* ∧ *ftype(y)* = *0*)
⟨*proof*⟩

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

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

lemma *frecRI2*: *s* ∈ *domain(n1)* ∨ *s* ∈ *domain(n2)* ⟹ *frecR*(⟨*1*, *s*, *n2*, *q*⟩, ⟨*0*, *n1*, *n2*, *q'*⟩)
⟨*proof*⟩

lemma *frecRI2'*: *s* ∈ *domain(n1)* ∪ *domain(n2)* ⟹ *frecR*(⟨*1*, *s*, *n2*, *q*⟩, ⟨*0*, *n1*, *n2*, *q'*⟩)
⟨*proof*⟩

lemma *frecRI3*: ⟨*s*, *r*⟩ ∈ *n2* ⟹ *frecR*(⟨*0*, *n1*, *s*, *q*⟩, ⟨*1*, *n1*, *n2*, *q'*⟩)
⟨*proof*⟩

lemma *frecRI3'*: *s* ∈ *domain(n2)* ⟹ *frecR*(⟨*0*, *n1*, *s*, *q*⟩, ⟨*1*, *n1*, *n2*, *q'*⟩)
⟨*proof*⟩

lemma *frecR_iff* :

frecR(x,y) ⟷
(ftype(x) = 1 ∧ ftype(y) = 0
∧ (name1(x) ∈ domain(name1(y)) ∪ domain(name2(y)) ∧ (name2(x) = name1(y) ∨ name2(x) = name2(y))))
∨ (ftype(x) = 0 ∧ ftype(y) = 1 ∧ name1(x) = name1(y) ∧ name2(x) ∈ domain(name2(y)))
⟨proof⟩

lemma frecR_D1 :
frecR(x,y) ⟹ ftype(y) = 0 ⟹ ftype(x) = 1 ∧
(name1(x) ∈ domain(name1(y)) ∪ domain(name2(y)) ∧ (name2(x) = name1(y) ∨ name2(x) = name2(y)))
⟨proof⟩

lemma frecR_D2 :
frecR(x,y) ⟹ ftype(y) = 1 ⟹ ftype(x) = 0 ∧
ftype(x) = 0 ∧ ftype(y) = 1 ∧ name1(x) = name1(y) ∧ name2(x) ∈ domain(name2(y))
⟨proof⟩

lemma frecR_DI :
assumes frecR(<a,b,c,d>,<ftype(y),name1(y),name2(y),cond_of(y)>)
shows frecR(<a,b,c,d>,y)
⟨proof⟩

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

schematic_goal sats_frecR_fm_auto:
assumes
a∈nat b∈nat env∈list(A)
shows
is_frecR(##A,nth(a,env),nth(b,env)) ⟷ sats(A,?fr_fm(a),env)
⟨proof⟩

⟨ML⟩

lemma frecR_fm_type[TC] :
⟦a∈nat;b∈nat⟧ ⟹ frecR_fm(a,b)∈formula
⟨proof⟩

lemma *sats_frecR_fm* :
assumes $a \in nat$ $b \in nat$ $env \in list(A)$
shows $sats(A, frecR\_fm(a,b), env) \longleftrightarrow is\_frecR(\#\#A, nth(a,env), nth(b,env))$
⟨*proof*⟩

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

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

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

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

definition
$mtype\_form :: i \Rightarrow i$ **where**
$mtype\_form(x) == if\ rank(name1(x)) < rank(name2(x))\ then\ 0\ else\ 2$

definition
$type\_form :: i \Rightarrow i$ **where**
$type\_form(x) == if\ ftype(x) = 0\ then\ 1\ else\ mtype\_form(x)$

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 $<x,y> \in frecrel(A)$
⟨*proof*⟩

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

lemma *wf_frecrel* :
shows $wf(frecrel(A))$
⟨*proof*⟩

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

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

lemma *frecrel_fst_snd*:
$frecrel(A) = \{z \in A \times A\ .$

ftype(*fst*(*z*)) = *1* ∧
ftype(*snd*(*z*)) = *0* ∧ *name1*(*fst*(*z*)) ∈ *domain*(*name1*(*snd*(*z*))) ∪ *domain*(*name2*(*snd*(*z*))) ∧
(*name2*(*fst*(*z*)) = *name1*(*snd*(*z*)) ∨ *name2*(*fst*(*z*)) = *name2*(*snd*(*z*)))
∨ (*ftype*(*fst*(*z*)) = *0* ∧
ftype(*snd*(*z*)) = *1* ∧ *name1*(*fst*(*z*)) = *name1*(*snd*(*z*)) ∧ *name2*(*fst*(*z*)) ∈ *domain*(*name2*(*snd*(*z*))))}
⟨*proof*⟩

end

15 Arities of internalized formulas

theory *Arities*
imports *FrecR*
ZF-Constructible-Trans.Formula
ZF-Constructible-Trans.L_axioms
begin

lemma *arity_upair_fm* : ⟦ *t1*∈*nat* ; *t2*∈*nat* ; *up*∈*nat* ⟧ ⟹
arity(*upair_fm*(*t1*,*t2*,*up*)) = ⋃ {*succ*(*t1*),*succ*(*t2*),*succ*(*up*)}
⟨*proof*⟩

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

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

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

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

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

lemma *arity_image_fm* :
⟦ *x*∈*nat* ; *y*∈*nat* ; *z*∈*nat* ⟧ ⟹ *arity*(*image_fm*(*x*,*y*,*z*)) = ⋃ {*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* :
  $[\![x \in nat]\!] \Longrightarrow arity(ordinal\_fm(x)) = succ(x)$
  ⟨*proof*⟩

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

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

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

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

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

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

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

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

⟨proof⟩

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

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

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

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

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

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

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

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

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

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

lemma *arity_is_iterates_fm* :

assumes *p*∈*formula* *v*∈*nat* *n*∈*nat* *Z*∈*nat* *i*∈*nat*
arity(*p*) = *i*
shows *arity*(*is_iterates_fm*(*p*,*v*,*n*,*Z*)) = *succ*(*v*) ∪ *succ*(*n*) ∪ *succ*(*Z*) ∪
pred(*pred*(*pred*(*pred*(*pred*(*pred*(*pred*(*pred*(*pred*(*pred*(*pred*(*i*)))))))))))
⟨*proof*⟩

lemma *arity_eclose_n_fm* :
assumes *A*∈*nat* *x*∈*nat* *t*∈*nat*
shows *arity*(*eclose_n_fm*(*A*,*x*,*t*)) = *succ*(*A*) ∪ *succ*(*x*) ∪ *succ*(*t*)
⟨*proof*⟩

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

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

lemma *eclose_n1arity__fm* :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*eclose_n1_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
⟨*proof*⟩

lemma *eclose_n2arity__fm* :
⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*eclose_n2_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
⟨*proof*⟩

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

lemma *arity_frecR_fm* :
⟦*a*∈*nat*;*b*∈*nat*⟧ ⟹ *arity*(*frecR_fm*(*a*,*b*)) = *succ*(*a*) ∪ *succ*(*b*)
⟨*proof*⟩

lemma *arity_Collect_fm* :
assumes *x* ∈ *nat* *y* ∈ *nat* *p*∈*formula*
shows *arity*(*Collect_fm*(*x*,*p*,*y*)) = *succ*(*x*) ∪ *succ*(*y*) ∪ *pred*(*arity*(*p*))
⟨*proof*⟩

end

16 The definition of *forces*

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

This is the core of our development.

16.1 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**
$frecrelP\_fm(a) == Exists(Exists(And(pair\_fm(1,0,a\#+2),frecR\_fm(1,0))))$

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**
$x = <f,n1,n2,p>$ $f \in 2$ $n1 \in ecloseN(z)$ $n2 \in ecloseN(z)$ $p \in P$
⟨*proof*⟩

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

definition
number2_fm :: $i \Rightarrow i$ **where**
$number2\_fm(a) == Exists(And(number1\_fm(0), succ\_fm(0,succ(a))))$

lemma *number2_fm_type*[*TC*] :
$a \in nat \Longrightarrow number2\_fm(a) \in formula$
⟨*proof*⟩

lemma *number2arity__fm* :
$a \in nat \Longrightarrow arity(number2\_fm(a)) = succ(a)$
⟨*proof*⟩

lemma *sats_number2_fm* [*simp*]:
$[| x \in nat; env \in list(A)|]$
$==> sats(A, number2\_fm(x), env) \longleftrightarrow number2(\#\#A, nth(x,env))$
⟨*proof*⟩

definition
is_names_below_fm :: [*i*,*i*,*i*] ⇒ *i* **where**
is_names_below_fm(*P*,*x*,*nb*) == *Exists*(*Exists*(*Exists*(*Exists*(
And(*ecloseN_fm*(*0*,*x* #+ *4*),*And*(*number2_fm*(*1*),
And(*cartprod_fm*(*0*,*P* #+ *4*,*2*),*And*(*cartprod_fm*(*0*,*2*,*3*),*cartprod_fm*(*1*,*3*,*nb*#+*4*)))))))))

lemma *arity_is_names_below_fm* :
⟦*P*∈*nat*;*x*∈*nat*;*nb*∈*nat*⟧ ⟹ *arity*(*is_names_below_fm*(*P*,*x*,*nb*)) = *succ*(*P*) ∪ *succ*(*x*)
∪ *succ*(*nb*)
⟨*proof*⟩

lemma *is_names_below_fm_type*[*TC*]:
⟦*P*∈*nat*;*x*∈*nat*;*nb*∈*nat*⟧ ⟹ *is_names_below_fm*(*P*,*x*,*nb*)∈*formula*
⟨*proof*⟩

lemma *sats_is_names_below_fm* :
assumes
P∈*nat* *x*∈*nat* *nb*∈*nat* *env*∈*list*(*A*)
shows
sats(*A*,*is_names_below_fm*(*P*,*x*,*nb*),*env*)
⟷ *is_names_below*(##*A*,*nth*(*P*, *env*),*nth*(*x*, *env*),*nth*(*nb*, *env*))
⟨*proof*⟩

definition
is_tuple :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_tuple(*M*,*z*,*t1*,*t2*,*p*,*t*) == ∃ *t1t2p*[*M*]. ∃ *t2p*[*M*]. *pair*(*M*,*t2*,*p*,*t2p*) ∧ *pair*(*M*,*t1*,*t2p*,*t1t2p*)
∧
pair(*M*,*z*,*t1t2p*,*t*)

definition
is_tuple_fm :: [*i*,*i*,*i*,*i*,*i*] ⇒ *i* **where**
is_tuple_fm(*z*,*t1*,*t2*,*p*,*tup*) = *Exists*(*Exists*(*And*(*pair_fm*(*t2* #+ *2*,*p* #+ *2*,*0*),
And(*pair_fm*(*t1* #+ *2*,*0*,*1*),*pair_fm*(*z* #+ *2*,*1*,*tup* #+ *2*)))))

lemma *arity_is_tuple_fm* : ⟦ *z*∈*nat* ; *t1*∈*nat* ; *t2*∈*nat* ; *p*∈*nat* ; *tup*∈*nat* ⟧ ⟹
arity(*is_tuple_fm*(*z*,*t1*,*t2*,*p*,*tup*)) = ⋃ {*succ*(*z*),*succ*(*t1*),*succ*(*t2*),*succ*(*p*),*succ*(*tup*)}
⟨*proof*⟩

lemma *is_tuple_fm_type*[*TC*] :
z∈*nat* ⟹ *t1*∈*nat* ⟹ *t2*∈*nat* ⟹ *p*∈*nat* ⟹ *tup*∈*nat* ⟹ *is_tuple_fm*(*z*,*t1*,*t2*,*p*,*tup*)∈*formula*

⟨*proof*⟩

lemma *sats_is_tuple_fm* :
assumes

z∈nat t1∈nat t2∈nat p∈nat tup∈nat env∈list(A)
shows
sats(A,is_tuple_fm(z,t1,t2,p,tup),env)
⟷ is_tuple(##A,nth(z, env),nth(t1, env),nth(t2, env),nth(p, env),nth(tup, env))
⟨*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∈nat b∈nat c∈nat d∈nat e∈nat env ∈ list(A)
shows
is_tuple(##A,aa,bb,cc,dd,ee) ⟷ sats(A, is_tuple_fm(a,b,c,d,e), env)
⟨*proof*⟩

16.2 Definition of *forces* for equality and membership

definition
eq_case :: [*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
eq_case(t1,t2,p,P,leq,f) ≡ ∀ s. s∈domain(t1) ∪ domain(t2) ⟶
(∀ q. q∈P ∧ <q,p>∈leq ⟶ (f‘<1,s,t1,q>=1 ⟷ f‘<1,s,t2,q> =1))

definition
is_eq_case :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_eq_case(M,t1,t2,p,P,leq,f) ≡
∀ s[M]. (∃ d[M]. is_domain(M,t1,d) ∧ s∈d) ∨ (∃ d[M]. is_domain(M,t2,d) ∧ s∈d)
⟶ (∀ q[M]. q∈P ∧ (∃ qp[M]. pair(M,q,p,qp) ∧ qp∈leq) ⟶
(∃ ost1q[M]. ∃ ost2q[M]. ∃ o[M]. ∃ vf1[M]. ∃ vf2[M].
is_tuple(M,o,s,t1,q,ost1q) ∧
is_tuple(M,o,s,t2,q,ost2q) ∧ number1(M,o) ∧
fun_apply(M,f,ost1q,vf1) ∧ fun_apply(M,f,ost2q,vf2) ∧
(vf1 = o ⟷ vf2 = o)))

definition
mem_case :: [*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
mem_case(t1,t2,p,P,leq,f) ≡ ∀ v∈P. <v,p>∈leq ⟶
(∃ q. ∃ s. ∃ r. r∈P ∧ q∈P ∧ <q,v>∈leq ∧ <s,r> ∈ t2 ∧ <q,r>∈leq ∧ f‘<0,t1,s,q> = 1)

definition
is_mem_case :: [*i*⇒*o*,*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
is_mem_case(M,t1,t2,p,P,leq,f) ≡ ∀ v[M]. ∀ vp[M]. v∈P ∧ pair(M,v,p,vp) ∧ vp∈leq ⟶
(∃ q[M]. ∃ s[M]. ∃ r[M]. ∃ qv[M]. ∃ sr[M]. ∃ qr[M]. ∃ z[M]. ∃ zt1sq[M]. ∃ o[M].
r∈ P ∧ q∈P ∧ pair(M,q,v,qv) ∧ pair(M,s,r,sr) ∧ pair(M,q,r,qr) ∧

empty(M,z) ∧ is_tuple(M,z,t1,s,q,zt1sq) ∧
number1(M,o) ∧ qv∈leq ∧ sr∈t2 ∧ qr∈leq ∧ fun_apply(M,f,zt1sq,o))

schematic_goal *sats_is_mem_case_fm_auto*:
assumes
n1∈nat n2∈nat p∈nat P∈nat leq∈nat f∈nat env∈list(A)
shows
is_mem_case(##A, nth(n1, env),nth(n2, env),nth(p, env),nth(P, env), nth(leq, env),nth(f,env))
⟷ sats(A,?imc_fm(n1,n2,p,P,leq,f),env)
⟨*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*⟩

lemma *mem_case_fm_type[TC]* :
⟦*n1∈nat;n2∈nat;p∈nat;P∈nat;leq∈nat;f∈nat*⟧ ⟹ *mem_case_fm(n1,n2,p,P,leq,f)∈formula*
⟨*proof*⟩

lemma *eq_case_fm_type[TC]* :
⟦*n1∈nat;n2∈nat;p∈nat;P∈nat;leq∈nat;f∈nat*⟧ ⟹ *eq_case_fm(n1,n2,p,P,leq,f)∈formula*

⟨proof⟩

lemma *sats_eq_case_fm* :
assumes
n1∈nat n2∈nat p∈nat P∈nat leq∈nat f∈nat env∈list(A)
shows
sats(A,eq_case_fm(n1,n2,p,P,leq,f),env) ⟷
is_eq_case(##A, nth(n1, env),nth(n2, env),nth(p, env),nth(P, env), nth(leq, env),nth(f,env))
⟨proof⟩

lemma *sats_mem_case_fm* :
assumes
n1∈nat n2∈nat p∈nat P∈nat leq∈nat f∈nat env∈list(A)
shows
sats(A,mem_case_fm(n1,n2,p,P,leq,f),env) ⟷
is_mem_case(##A, nth(n1, env),nth(n2, env),nth(p, env),nth(P, env), nth(leq, env),nth(f,env))
⟨proof⟩

lemma *mem_case_iff_sats*:
assumes
n1∈nat n2∈nat p∈nat P∈nat leq∈nat f∈nat env∈list(A)
nth(n1,env) = nn1 nth(n2,env) = nn2 nth(p,env) = pp nth(P,env) = PP
nth(leq,env) = lleq nth(f,env) = ff
shows
is_mem_case(##A, nn1,nn2,pp,PP, lleq,ff)
⟷ sats(A,mem_case_fm(n1,n2,p,P,leq,f),env)
⟨proof⟩

lemma *eq_case_iff_sats* :
assumes
n1∈nat n2∈nat p∈nat P∈nat leq∈nat f∈nat env∈list(A)
nth(n1,env) = nn1 nth(n2,env) = nn2 nth(p,env) = pp nth(P,env) = PP
nth(leq,env) = lleq nth(f,env) = ff
shows
is_eq_case(##A, nn1,nn2,pp,PP, lleq,ff)
⟷ sats(A,eq_case_fm(n1,n2,p,P,leq,f),env)
⟨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) ≡

$\exists ft[M]. \exists n1[M]. \exists n2[M]. \exists co[M].$
$co \in P \wedge is\_tuple(M,ft,n1,n2,co,fnnc) \wedge$
$((empty(M,ft) \wedge is\_eq\_case(M,n1,n2,co,P,leq,f))$
$\vee (number1(M,ft) \wedge is\_mem\_case(M,n1,n2,co,P,leq,f)))$

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*∈*nat*;*leq*∈*nat*;*fnnc*∈*nat*;*f*∈*nat*;*z*∈*nat*⟧ ⟹ *Hfrc_at_fm*(*P*,*leq*,*fnnc*,*f*,*z*)∈*formula*
⟨*proof*⟩

lemma *sats_Hfrc_at_fm*:
assumes
P∈*nat* *leq*∈*nat* *fnnc*∈*nat* *f*∈*nat* *z*∈*nat* *env*∈*list*(*A*)
shows
sats(*A*,*Hfrc_at_fm*(*P*,*leq*,*fnnc*,*f*,*z*),*env*)
⟷ *is_Hfrc_at*(##*A*,*nth*(*P*, *env*), *nth*(*leq*, *env*), *nth*(*fnnc*, *env*),*nth*(*f*, *env*),*nth*(*z*, *env*))
⟨*proof*⟩

lemma *is_Hfrc_at_iff_sats*:
assumes
P∈*nat* *leq*∈*nat* *fnnc*∈*nat* *f*∈*nat* *z*∈*nat* *env*∈*list*(*A*)
nth(*P*,*env*) = *PP* *nth*(*leq*,*env*) = *lleq* *nth*(*fnnc*,*env*) = *ffnnc*
nth(*f*,*env*) = *ff* *nth*(*z*,*env*) = *zz*
shows
is_Hfrc_at(##*A*, *PP*, *lleq*,*ffnnc*,*ff*,*zz*)
⟷ *sats*(*A*,*Hfrc_at_fm*(*P*,*leq*,*fnnc*,*f*,*z*),*env*)
⟨*proof*⟩

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

16.3 The well-founded relation *forcerel*

definition
forcerel :: *i* ⇒ *i* ⇒ *i* **where**
forcerel(*P*,*x*) ≡ *frecrel*(*names_below*(*P*,*x*)) ^+

definition
is_forcerel :: [*i*⇒*o*,*i*,*i*,*i*] ⇒ *o* **where**
is_forcerel(*M*,*P*,*x*,*z*) == ∃ *r*[*M*]. ∃ *nb*[*M*]. *tran_closure*(*M*,*r*,*z*) ∧

(is_names_below(M,P,x,nb) ∧ is_frecrel(M,nb,r))

definition
forcerel_fm :: i⇒ i ⇒ i ⇒ i **where**
forcerel_fm(p,x,z) == Exists(Exists(And(tran_closure_fm(1, z#+2),
And(is_names_below_fm(p#+2,x#+2,0),frecrel_fm(0,1)))))

lemma arity_forcerel_fm:
⟦p∈nat;x∈nat;z∈nat⟧ ⟹ arity(forcerel_fm(p,x,z)) = succ(p) ∪ succ(x) ∪ succ(z)

⟨proof⟩

lemma forcerel_fm_type[TC]:
⟦p∈nat;x∈nat;z∈nat⟧ ⟹ forcerel_fm(p,x,z)∈formula
⟨proof⟩

lemma sats_forcerel_fm:
assumes
p∈nat x∈nat z∈nat env∈list(A)
shows
sats(A,forcerel_fm(p,x,z),env) ⟷ is_forcerel(##A,nth(p,env),nth(x, env),nth(z, env))
⟨proof⟩

16.4 *frc_at*, forcing for atomic formulas

definition
frc_at :: [i,i,i] ⇒ i **where**
frc_at(P,leq,fnnc) ≡ wfrec(frecrel(names_below(P,fnnc)),fnnc,
λx f. bool_of_o(Hfrc(P,leq,x,f)))

definition
is_frc_at :: [i⇒o,i,i,i,i] ⇒ o **where**
is_frc_at(M,P,leq,x,z) ≡ ∃ r[M]. is_forcerel(M,P,x,r) ∧
is_wfrec(M,is_Hfrc_at(M,P,leq),r,x,z)

definition
frc_at_fm :: [i,i,i,i] ⇒ i **where**
frc_at_fm(p,l,x,z) == Exists(And(forcerel_fm(succ(p),succ(x),0),
is_wfrec_fm(Hfrc_at_fm(6#+p,6#+l,2,1,0),0,succ(x),succ(z))))

lemma frc_at_fm_type [TC] :
⟦p∈nat;l∈nat;x∈nat;z∈nat⟧ ⟹ frc_at_fm(p,l,x,z)∈formula
⟨proof⟩

lemma arity_frc_at_fm :

assumes *p∈nat l∈nat x∈nat z∈nat*
shows *arity(frc_at_fm(p,l,x,z)) = succ(p) ∪ succ(l) ∪ succ(x) ∪ succ(z)*
⟨*proof*⟩

lemma *sats_frc_at_fm* :
assumes
p∈nat l∈nat i∈nat j∈nat env∈list(A) i < length(env) j < length(env)
shows
sats(A,frc_at_fm(p,l,i,j),env) ⟷
is_frc_at(##A,nth(p,env),nth(l,env),nth(i,env),nth(j,env))
⟨*proof*⟩

definition
forces_eq′ :: [*i,i,i,i,i*] ⇒ *o* **where**
forces_eq′(P,l,p,t1,t2) ≡ frc_at(P,l,<0,t1,t2,p>) = 1

definition
forces_mem′ :: [*i,i,i,i,i*] ⇒ *o* **where**
forces_mem′(P,l,p,t1,t2) ≡ frc_at(P,l,<1,t1,t2,p>) = 1

definition
forces_neq′ :: [*i,i,i,i,i*] ⇒ *o* **where**
forces_neq′(P,l,p,t1,t2) ≡ ¬ (∃ q∈P. <q,p>∈l ∧ forces_eq′(P,l,q,t1,t2))

definition
forces_nmem′ :: [*i,i,i,i,i*] ⇒ *o* **where**
forces_nmem′(P,l,p,t1,t2) ≡ ¬ (∃ q∈P. <q,p>∈l ∧ forces_mem′(P,l,q,t1,t2))

definition
is_forces_eq′ :: [*i⇒o,i,i,i,i,i*] ⇒ *o* **where**
is_forces_eq′(M,P,l,p,t1,t2) == ∃ o[M]. ∃ z[M]. ∃ t[M]. number1(M,o) ∧ empty(M,z)
∧
is_tuple(M,z,t1,t2,p,t) ∧ is_frc_at(M,P,l,t,o)

definition
is_forces_mem′ :: [*i⇒o,i,i,i,i,i*] ⇒ *o* **where**
is_forces_mem′(M,P,l,p,t1,t2) == ∃ o[M]. ∃ t[M]. number1(M,o) ∧
is_tuple(M,o,t1,t2,p,t) ∧ is_frc_at(M,P,l,t,o)

definition
is_forces_neq′ :: [*i⇒o,i,i,i,i,i*] ⇒ *o* **where**
is_forces_neq′(M,P,l,p,t1,t2) ≡
¬ (∃ q[M]. q∈P ∧ (∃ qp[M]. pair(M,q,p,qp) ∧ qp∈l ∧ is_forces_eq′(M,P,l,q,t1,t2)))

definition
is_forces_nmem' :: [i⇒o,i,i,i,i,i] ⇒ o **where**
is_forces_nmem'(M,P,l,p,t1,t2) ≡
¬ (∃ q[M]. ∃ qp[M]. q∈P ∧ *pair*(M,q,p,qp) ∧ qp∈l ∧ *is_forces_mem'*(M,P,l,q,t1,t2))

definition
forces_eq_fm :: [i,i,i,i,i] ⇒ i **where**
forces_eq_fm(p,l,q,t1,t2) ≡
Exists(*Exists*(*Exists*(*And*(*number1_fm*(2),*And*(*empty_fm*(1),
And(*is_tuple_fm*(1,t1#+3,t2#+3,q#+3,0),*frc_at_fm*(p#+3,l#+3,0,2)
))))))

definition
forces_mem_fm :: [i,i,i,i,i] ⇒ i **where**
forces_mem_fm(p,l,q,t1,t2) ≡ *Exists*(*Exists*(*And*(*number1_fm*(1),
And(*is_tuple_fm*(1,t1#+2,t2#+2,q#+2,0),*frc_at_fm*(p#+2,l#+2,0,1)))))

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

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

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

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

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

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

lemma *arity_forces_eq_fm* :
p∈nat ⟹ l∈nat ⟹ q∈nat ⟹ t1 ∈ nat ⟹ t2 ∈ nat ⟹

arity(*forces_eq_fm*(*p*,*l*,*q*,*t1*,*t2*)) = *succ*(*t1*) ∪ *succ*(*t2*) ∪ *succ*(*q*) ∪ *succ*(*p*) ∪ *succ*(*l*)
⟨*proof*⟩

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

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

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

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

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

context *forcing_data*
begin

lemma *fst_abs* [*simp*]:
⟦*x*∈*M*; *y*∈*M* ⟧ ⟹ *is_fst*(##*M*,*x*,*y*) ⟷ *y* = *fst*(*x*)
⟨*proof*⟩

lemma *snd_abs* [*simp*]:
⟦*x*∈*M*; *y*∈*M* ⟧ ⟹ *is_snd*(##*M*,*x*,*y*) ⟷ *y* = *snd*(*x*)
⟨*proof*⟩

lemma *ftype_abs*[*simp*] :
⟦*x*∈*M*; *y*∈*M* ⟧ ⟹ *is_ftype*(##*M*,*x*,*y*) ⟷ *y* = *ftype*(*x*) ⟨*proof*⟩

lemma *name1_abs*[*simp*] :
  $[\![x\in M;\ y\in M\ ]\!] \Longrightarrow$ *is_name1*(##*M*,*x*,*y*) $\longleftrightarrow$ *y* = *name1*(*x*)
  ⟨*proof*⟩

lemma *snd_snd_abs*:
  $[\![x\in M;\ y\in M\ ]\!] \Longrightarrow$ *is_snd_snd*(##*M*,*x*,*y*) $\longleftrightarrow$ *y* = *snd*(*snd*(*x*))
  ⟨*proof*⟩

lemma *name2_abs*[*simp*]:
  $[\![x\in M;\ y\in M\ ]\!] \Longrightarrow$ *is_name2*(##*M*,*x*,*y*) $\longleftrightarrow$ *y* = *name2*(*x*)
  ⟨*proof*⟩

lemma *cond_of_abs*[*simp*]:
  $[\![x\in M;\ y\in M\ ]\!] \Longrightarrow$ *is_cond_of*(##*M*,*x*,*y*) $\longleftrightarrow$ *y* = *cond_of*(*x*)
  ⟨*proof*⟩

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* = <*z*,*t1*,*t2*,*p*>
  ⟨*proof*⟩

lemma *oneN_in_M*[*simp*]: $1\in M$
  ⟨*proof*⟩

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

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

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

lemma *mem_case_abs* [*simp*]:
  assumes
    $t1\in M$ $t2\in M$ $p\in M$ $f\in M$
  shows
    is_mem_case(##*M*,*t1*,*t2*,*p*,*P*,*leq*,*f*) $\longleftrightarrow$ *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∈M ec∈M
shows is_eclose_n(##M,is_name1,ec,x) ⟷ ec = eclose_n(name1,x)
is_eclose_n(##M,is_name2,ec,x) ⟷ ec = eclose_n(name2,x)
⟨proof⟩

lemma is_ecloseN_abs :
⟦x∈M;ec∈M⟧ ⟹ is_ecloseN(##M,ec,x) ⟷ ec = ecloseN(x)
⟨proof⟩

lemma frecR_abs :
x∈M ⟹ y∈M ⟹ frecR(x,y) ⟷ is_frecR(##M,x,y)
⟨proof⟩

lemma frecrelP_abs :
z∈M ⟹ frecrelP(##M,z) ⟷ (∃ x y. z = <x,y> ∧ frecR(x,y))
⟨proof⟩

lemma frecrel_abs:
assumes
A∈M r∈M
shows
is_frecrel(##M,A,r) ⟷ r = frecrel(A)
⟨proof⟩

lemma frecrel_closed:

assumes
x∈*M*
shows
frecrel(*x*)∈*M*
⟨*proof*⟩

lemma *field_frecrel* : *field*(*frecrel*(*names_below*(*P*,*x*))) ⊆ *names_below*(*P*,*x*)
⟨*proof*⟩

lemma *forcerelD* : *uv* ∈ *forcerel*(*P*,*x*) ⟹ *uv*∈ *names_below*(*P*,*x*) × *names_below*(*P*,*x*)
⟨*proof*⟩

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

lemma *restrict_trancl_forcerel*:
assumes *frecR*(*w*,*y*)
shows *restrict*(*f*,*frecrel*(*names_below*(*P*,*x*))-‘‘{*y*})‘*w*
= *restrict*(*f*,*forcerel*(*P*,*x*)-‘‘{*y*})‘*w*
⟨*proof*⟩

lemma *names_belowI* :
assumes *frecR*(<*ft*,*n1*,*n2*,*p*>,<*a*,*b*,*c*,*d*>) *p*∈*P*
shows <*ft*,*n1*,*n2*,*p*> ∈ *names_below*(*P*,<*a*,*b*,*c*,*d*>) (**is** *?x* ∈ *names_below*(_,*?y*))
⟨*proof*⟩

lemma *names_below_tr* :
assumes *x*∈ *names_below*(*P*,*y*)
y∈ *names_below*(*P*,*z*)
shows *x*∈ *names_below*(*P*,*z*)
⟨*proof*⟩

lemma *arg_into_names_below2* :
assumes <*x*,*y*> ∈ *frecrel*(*names_below*(*P*,*z*))
shows *x* ∈ *names_below*(*P*,*y*)
⟨*proof*⟩

lemma *arg_into_names_below* :
assumes <*x*,*y*> ∈ *frecrel*(*names_below*(*P*,*z*))
shows *x* ∈ *names_below*(*P*,*x*)
⟨*proof*⟩

lemma *forcerel_arg_into_names_below* :
assumes <*x*,*y*> ∈ *forcerel*(*P*,*z*)
shows *x* ∈ *names_below*(*P*,*x*)
⟨*proof*⟩

lemma *names_below_mono* :

assumes $\langle x,y\rangle \in$ *frecrel*(*names_below*(P,z))
shows *names_below*(P,x) $\subseteq$ *names_below*(P,y)
⟨*proof*⟩

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

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

lemma *forcerel_mono_aux* :
assumes $\langle x,y\rangle \in$ *frecrel*(*names_below*(P, w))^+
shows *forcerel*(P,x) $\subseteq$ *forcerel*(P,y)
⟨*proof*⟩

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

lemma *aux*: $x \in$ *names_below*(P, w) $\Longrightarrow$ <x,y> $\in$ *forcerel*(P,z) $\Longrightarrow$
($y \in$ *names_below*(P, w) $\longrightarrow$ <x,y> $\in$ *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 <z,x> $\in$ *forcerel*(P,x) <u,z> $\in$ *forcerel*(P,x)
shows $u \in$ *names_below*(P,z)
⟨*proof*⟩

lemma *forcerel_below* :
assumes <z,x> $\in$ *forcerel*(P,x)
shows *forcerel*(P,x) -`` $\{z\}$ $\subseteq$ *names_below*(P,z)
⟨*proof*⟩

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

lemma *Hfrc_restrict_trancl*: *bool_of_o*(*Hfrc*(P, *leq*, y, *restrict*(f,*frecrel*(*names_below*(P,x))-``$\{y\}$)))

= *bool_of_o*(*Hfrc*(*P*, *leq*, *y*, *restrict*(*f*,(*frecrel*(*names_below*(*P*,*x*)) ^+)-‘‘{*y*})))
⟨*proof*⟩

lemma *frc_at_trancl*: *frc_at*(*P*,*leq*,*z*) = *wfrec*(*forcerel*(*P*,*z*),*z*,λ*x f*. *bool_of_o*(*Hfrc*(*P*,*leq*,*x*,*f*)))
⟨*proof*⟩

lemma *forcerelI1* :
assumes *n1* ∈ *domain*(*b*) ∨ *n1* ∈ *domain*(*c*) *p*∈*P* *d*∈*P*
shows ⟨⟨*1*, *n1*, *b*, *p*⟩, ⟨*0*,*b*,*c*,*d*⟩⟩∈ *forcerel*(*P*,⟨*0*,*b*,*c*,*d*⟩)
⟨*proof*⟩

lemma *forcerelI2* :
assumes *n1* ∈ *domain*(*b*) ∨ *n1* ∈ *domain*(*c*) *p*∈*P* *d*∈*P*
shows ⟨⟨*1*, *n1*, *c*, *p*⟩, ⟨*0*,*b*,*c*,*d*⟩⟩∈ *forcerel*(*P*,⟨*0*,*b*,*c*,*d*⟩)
⟨*proof*⟩

lemma *forcerelI3* :
assumes ⟨*n2*, *r*⟩ ∈ *c* *p*∈*P* *d*∈*P* *r* ∈ *P*
shows ⟨⟨*0*, *b*, *n2*, *p*⟩,⟨*1*, *b*, *c*, *d*⟩⟩ ∈ *forcerel*(*P*,<*1*,*b*,*c*,*d*>)
⟨*proof*⟩

lemmas *forcerelI* = *forcerelI1*[*THEN vimage_singleton_iff*[*THEN iffD2*]]
forcerelI2[*THEN vimage_singleton_iff*[*THEN iffD2*]]
forcerelI3[*THEN vimage_singleton_iff*[*THEN iffD2*]]

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

16.5 Recursive expression of *frc_at*

lemma *def_frc_at* :
assumes *p*∈*P*
shows
frc_at(*P*,*leq*,<*ft*,*n1*,*n2*,*p*>) =
bool_of_o(*p* ∈*P* ∧
(*ft* = *0* ∧ (∀ *s*. *s*∈*domain*(*n1*) ∪ *domain*(*n2*) ⟶
(∀ *q*. *q*∈*P* ∧ *q* ⪯ *p* ⟶ (*frc_at*(*P*,*leq*,<*1*,*s*,*n1*,*q*>) =*1* ⟷ *frc_at*(*P*,*leq*,<*1*,*s*,*n2*,*q*>)
=*1*)))
∨ *ft* = *1* ∧ (∀ *v*∈*P*. *v* ⪯ *p* ⟶
(∃ *q*. ∃ *s*. ∃ *r*. *r*∈*P* ∧ *q*∈*P* ∧ *q* ⪯ *v* ∧ <*s*,*r*> ∈ *n2* ∧ *q* ⪯ *r* ∧ *frc_at*(*P*,*leq*,<*0*,*n1*,*s*,*q*>)
= *1*))))
⟨*proof*⟩

16.6 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* :
$[\![Q \in M; x \in M; nb \in M]\!] \Longrightarrow$ *is_names_below*(##M,Q,x,*nb*) $\longleftrightarrow$ *nb* = *names_below*(Q,x)

⟨*proof*⟩

lemma *names_below_closed*:
$[\![Q \in M; x \in M]\!] \Longrightarrow$ *names_below*(Q,x) $\in M$
⟨*proof*⟩

lemma *names_below_productE* :
$Q \in M \Longrightarrow$
$x \in M \Longrightarrow (\bigwedge A1\ A2\ A3\ A4.\ A1 \in M \Longrightarrow A2 \in M \Longrightarrow A3 \in M \Longrightarrow A4 \in M$
$\Longrightarrow R(A1 \times A2 \times A3 \times A4))$
$\Longrightarrow R($*names_below*$(Q,x))$
⟨*proof*⟩

lemma *forcerel_abs* :
$[\![x \in M; z \in M]\!] \Longrightarrow$ *is_forcerel*(##M,P,x,z) $\longleftrightarrow$ 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

16.7 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⇒o,i,i,i] ⇒ o **where**
is_leq(A,l,q,p) ≡ ∃ qp[A]. (pair(A,q,p,qp) ∧ qp∈l)

lemma (**in** forcing_data) leq_abs[simp]:
⟦ l∈M ; q∈M ; p∈M ⟧ ⟹ is_leq(##M,l,q,p) ⟷ <q,p>∈l
⟨proof⟩

definition
leq_fm :: [i,i,i] ⇒ i **where**
leq_fm(leq,q,p) ≡ Exists(And(pair_fm(q#+1,p#+1,0),Member(0,leq#+1)))

lemma arity_leq_fm :
⟦leq∈nat;q∈nat;p∈nat⟧ ⟹ arity(leq_fm(leq,q,p)) = succ(q) ∪ succ(p) ∪ succ(leq)
⟨proof⟩

lemma leq_fm_type[TC] :
⟦leq∈nat;q∈nat;p∈nat⟧ ⟹ leq_fm(leq,q,p)∈formula
⟨proof⟩

lemma sats_leq_fm :
⟦ leq∈nat;q∈nat;p∈nat;env∈list(A) ⟧ ⟹

$sats(A,leq\_fm(leq,q,p),env) \longleftrightarrow is\_leq(\#\#A,nth(leq,env),nth(q,env),nth(p,env))$

⟨proof⟩

16.7.1 The primitive recursion

consts *forces'* :: *i⇒i*
primrec
forces'(Member(x,y)) = forces_mem_fm(1,2,0,x#+4,y#+4)
forces'(Equal(x,y)) = forces_eq_fm(1,2,0,x#+4,y#+4)
forces'(Nand(p,q)) =
Neg(Exists(And(Member(0,2),And(leq_fm(3,0,1),And(ren_forces_nand(forces'(p)),
ren_forces_nand(forces'(q)))))))
forces'(Forall(p)) = Forall(ren_forces_forall(forces'(p)))

definition
forces :: *i⇒i* **where**
forces(φ) ≡ And(Member(0,1),forces'(φ))

lemma *forces'_type* [*TC*]: *φ∈formula ⟹ forces'(φ) ∈ formula*
⟨proof⟩

lemma *forces_type*[*TC*] : *φ∈formula ⟹ forces(φ) ∈ formula*
⟨proof⟩

context *forcing_data*
begin

16.8 Forcing for atomic formulas in context

definition
forces_eq :: [*i,i,i*] ⇒ *o* **where**
forces_eq ≡ forces_eq'(P,leq)

definition
forces_mem :: [*i,i,i*] ⇒ *o* **where**
forces_mem ≡ forces_mem'(P,leq)

definition
is_forces_eq :: [*i,i,i*] ⇒ *o* **where**
is_forces_eq ≡ is_forces_eq'(##M,P,leq)

definition
is_forces_mem :: [*i,i,i*] ⇒ *o* **where**
is_forces_mem ≡ is_forces_mem'(##M,P,leq)

lemma *def_forces_eq*: p∈P ⟹ forces_eq(p,t1,t2) ⟷
(∀ s∈domain(t1) ∪ domain(t2). ∀ q. q∈P ∧ q ≼ p ⟶
(forces_mem(q,s,t1) ⟷ forces_mem(q,s,t2)))
⟨proof⟩

lemma *def_forces_mem*: p∈P ⟹ forces_mem(p,t1,t2) ⟷
(∀ v∈P. v ≼ p ⟶
(∃ q. ∃ s. ∃ r. r∈P ∧ q∈P ∧ q ≼ v ∧ <s,r> ∈ t2 ∧ q ≼ r ∧ forces_eq(q,t1,s)))
⟨proof⟩

lemma *forces_eq_abs* :
⟦p∈M ; t1∈M ; t2∈M⟧ ⟹ is_forces_eq(p,t1,t2) ⟷ forces_eq(p,t1,t2)
⟨proof⟩

lemma *forces_mem_abs* :
⟦p∈M ; t1∈M ; t2∈M⟧ ⟹ is_forces_mem(p,t1,t2) ⟷ forces_mem(p,t1,t2)
⟨proof⟩

lemma *sats_forces_eq_fm*:
assumes p∈nat l∈nat q∈nat t1∈nat t2∈nat env∈list(M)
nth(p,env)=P nth(l,env)=leq
shows sats(M,forces_eq_fm(p,l,q,t1,t2),env) ⟷
is_forces_eq(nth(q,env),nth(t1,env),nth(t2,env))
⟨proof⟩

lemma *sats_forces_mem_fm*:
assumes p∈nat l∈nat q∈nat t1∈nat t2∈nat env∈list(M)
nth(p,env)=P nth(l,env)=leq
shows sats(M,forces_mem_fm(p,l,q,t1,t2),env) ⟷
is_forces_mem(nth(q,env),nth(t1,env),nth(t2,env))
⟨proof⟩

definition
forces_neq :: [i,i,i] ⇒ o **where**
forces_neq(p,t1,t2) ≡ ¬ (∃ q∈P. q≼p ∧ forces_eq(q,t1,t2))

definition
forces_nmem :: [i,i,i] ⇒ o **where**
forces_nmem(p,t1,t2) ≡ ¬ (∃ q∈P. q≼p ∧ forces_mem(q,t1,t2))

lemma *forces_neq* :
forces_neq(p,t1,t2) ⟷ forces_neq'(P,leq,p,t1,t2)
⟨proof⟩

lemma *forces_nmem* :
forces_nmem(p,t1,t2) ⟷ forces_nmem'(P,leq,p,t1,t2)
⟨proof⟩

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 $sats(M,forces(Member(x,y)),[q,P,leq,one]@env) \longleftrightarrow$
$(q{\in}P \wedge is\_forces\_mem(q,xx,yy))$
⟨*proof*⟩

lemma *sats_forces_Equal* :
assumes $x{\in}nat$ $y{\in}nat$ $env{\in}list(M)$
$nth(x,env)=xx$ $nth(y,env)=yy$ $q{\in}M$
shows $sats(M,forces(Equal(x,y)),[q,P,leq,one]@env) \longleftrightarrow$
$(q{\in}P \wedge is\_forces\_eq(q,xx,yy))$
⟨*proof*⟩

lemma *sats_forces_Nand* :
assumes $\varphi{\in}formula$ $\psi{\in}formula$ $env{\in}list(M)$ $p{\in}M$
shows $sats(M,forces(Nand(\varphi,\psi)),[p,P,leq,one]@env) \longleftrightarrow$
$(p{\in}P \wedge \neg(\exists\, q{\in}M.\ q{\in}P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
$(sats(M,forces'(\varphi),[q,P,leq,one]@env) \wedge sats(M,forces'(\psi),[q,P,leq,one]@env))))$
⟨*proof*⟩

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

16.9 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 φ∈*formula*
  shows *arity*(*forces*(φ)) ≤ 4#+*arity*(φ)
  ⟨proof⟩

lemma *arity_forces_le* :
  assumes φ∈*formula* *n*∈*nat* *arity*(φ) ≤ *n*
  shows *arity*(*forces*(φ)) ≤ 4#+*n*
  ⟨proof⟩

end

17 The Forcing Theorems

theory *Forcing_Theorems*
  imports
    Forces_Definition

begin

context *forcing_data*
begin

17.1 The forcing relation in context

abbreviation *Forces* :: [*i*, *i*, *i*] ⇒ *o* (_ ⊩ _ _ [36,36,36] 60) **where**
  p ⊩ φ *env* ≡ *M*, ([*p*,*P*,*leq*,*one*] @ *env*) ⊨ *forces*(φ)

lemma *Collect_forces* :
  assumes
    fty: φ∈*formula* **and**
    far: *arity*(φ)≤*length*(*env*) **and**
    envty: *env*∈*list*(*M*)
  shows
    {*p*∈*P* . *p* ⊩ φ *env*} ∈ *M*
⟨proof⟩

lemma *forces_mem_iff_dense_below*: *p*∈*P* ⟹ *forces_mem*(*p*,*t1*,*t2*) ⟷ *dense_below*(
    {*q*∈*P*. ∃ *s*. ∃ *r*. *r*∈*P* ∧ <*s*,*r*> ∈ *t2* ∧ *q*≼*r* ∧ *forces_eq*(*q*,*t1*,*s*)}
    ,*p*)
  ⟨proof⟩

17.2 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_eq*:
  assumes *p*∈*P* *r*∈*P* *r*≼*p* *forces_eq*(*p*,*t1*,*t2*)
  shows *forces_eq*(*r*,*t1*,*t2*)

⟨proof⟩

17.3 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 $forces\_mem(r,t1,t2)$
  ⟨proof⟩

17.4 Kunen 2013, Lemma IV.2.37(b)

lemma *density_mem*:
  assumes $p \in P$
  shows $forces\_mem(p,t1,t2) \longleftrightarrow dense\_below(\{q \in P.\ forces\_mem(q,t1,t2)\},p)$
⟨proof⟩

lemma *aux_density_eq*:
  assumes
    $dense\_below($
    $\{q' \in P.\ \forall q.\ q \in P \wedge q \preceq q' \longrightarrow forces\_mem(q,s,t1) \longleftrightarrow forces\_mem(q,s,t2)\}$
    $,p)$
    $forces\_mem(q,s,t1)$ $q \in P$ $p \in P$ $q \preceq p$
  shows
    $dense\_below(\{r \in P.\ forces\_mem(r,s,t2)\},q)$
⟨proof⟩

lemma *density_eq*:
  assumes $p \in P$
  shows $forces\_eq(p,t1,t2) \longleftrightarrow dense\_below(\{q \in P.\ forces\_eq(q,t1,t2)\},p)$
⟨proof⟩

17.5 Kunen 2013, Lemma IV.2.38

lemma *not_forces_neq*:
  assumes $p \in P$
  shows $forces\_eq(p,t1,t2) \longleftrightarrow \neg (\exists q \in P.\ q \preceq p \wedge forces\_neq(q,t1,t2))$
  ⟨proof⟩

lemma *not_forces_nmem*:
  assumes $p \in P$
  shows $forces\_mem(p,t1,t2) \longleftrightarrow \neg (\exists q \in P.\ q \preceq p \wedge forces\_nmem(q,t1,t2))$
  ⟨proof⟩

lemma *sats_forces_Nand'*:

assumes
$p{\in}P$ $\varphi{\in}formula$ $\psi{\in}formula$ $env \in list(M)$
shows
$M,\ [p,P,leq,one]\ @\ env \models forces(Nand(\varphi,\psi)) \longleftrightarrow$
$\neg(\exists\, q{\in}M.\ q{\in}P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
$M,\ [q,P,leq,one]\ @\ env \models forces(\varphi) \wedge$
$M,\ [q,P,leq,one]\ @\ env \models forces(\psi))$
⟨*proof*⟩

lemma *sats_forces_Neg'*:
assumes
$p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$
shows
$M,\ [p,P,leq,one]\ @\ env \models forces(Neg(\varphi)) \quad \longleftrightarrow$
$\neg(\exists\, q{\in}M.\ q{\in}P \wedge is\_leq(\#\#M,leq,q,p) \wedge$
$M,\ [q,P,leq,one]@env \models forces(\varphi))$
⟨*proof*⟩

lemma *sats_forces_Forall'*:
assumes
$p{\in}P$ $env \in list(M)$ $\varphi{\in}formula$
shows
$M,[p,P,leq,one]\ @\ env \models forces(Forall(\varphi)) \longleftrightarrow$
$(\forall\, x{\in}M.\quad M,\ [p,P,leq,one,x]\ @\ env \models forces(\varphi))$
⟨*proof*⟩

17.6 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⟩

17.7 The relation of forcing and connectives

lemma *Forces_Nand*:
assumes
p∈P env ∈ list(M) φ∈formula ψ∈formula
shows
(p ⊩ Nand(φ,ψ) env) ⟷ ¬(∃ q∈M. q∈P ∧ q⪯p ∧ (q ⊩ φ env) ∧ (q ⊩ ψ env))
⟨proof⟩

lemma *Forces_And_aux*:
assumes
p∈P env ∈ list(M) φ∈formula ψ∈formula
shows
p ⊩ And(φ,ψ) env ⟷
(∀ q∈M. q∈P ∧ q⪯p ⟶ (∃ r∈M. r∈P ∧ r⪯q ∧ (r ⊩ φ env) ∧ (r ⊩ ψ env)))
⟨proof⟩

lemma *Forces_And_iff_dense_below*:
assumes
p∈P env ∈ list(M) φ∈formula ψ∈formula
shows
(p ⊩ And(φ,ψ) env) ⟷ dense_below({r∈P. (r ⊩ φ env) ∧ (r ⊩ ψ env) },p)
⟨proof⟩

lemma *Forces_Forall*:
assumes
p∈P env ∈ list(M) φ∈formula
shows
(p ⊩ Forall(φ) env) ⟷ (∀ x∈M. (p ⊩ φ ([x] @ env)))
⟨proof⟩

bundle *some_rules* = *elem_of_val_pair* [*dest*] *SepReplace_iff* [*simp del*] *SepReplace_iff* [*iff*]

context
includes *some_rules*
begin

lemma *elem_of_valI*: *∃ ϑ. ∃ p∈P. p∈G ∧ <ϑ,p>∈π ∧ val(G,ϑ) = x ⟹ x∈val(G,π)*
⟨proof⟩

lemma *GenExtD*: *x∈M[G] ⟷ (∃ τ∈M. x = val(G,τ))*
⟨proof⟩

lemma *left_in_M* : *tau∈M ⟹ <a,b>∈tau ⟹ a∈M*
⟨proof⟩

17.8 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*⟩

17.9 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
$val(G,\vartheta) \subseteq val(G,\tau)$
$\langle proof \rangle$

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
$val(G,\tau) = val(G,\vartheta)$
$\langle proof \rangle$

17.10 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

$Q(\tau,\vartheta) \wedge R(\tau,\vartheta)$
⟨proof⟩

17.11 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** $?Q(\tau,\vartheta) \wedge ?R(\tau,\vartheta)$)
⟨proof⟩

17.12 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
$\exists p\in G.\ forces\_mem(p,\pi,\tau)$
⟨proof⟩

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
$(\exists\, p{\in}G.\ p \Vdash Equal(n,m)\ env) \longleftrightarrow M[G],\ map(val(G),env) \models Equal(n,m)$
⟨*proof*⟩

lemma *arities_at_aux*:
assumes
$n \in nat\ m \in nat\ env \in list(M)\ succ(n) \cup succ(m) \leq length(env)$
shows
$n < length(env)\ m < length(env)$
⟨*proof*⟩

17.13 The Strenghtening Lemma

lemma *strengthening_lemma*:
assumes
$p{\in}P\ \varphi{\in}formula\ r{\in}P\ r{\preceq}p$
shows

$\bigwedge env.\ env{\in}list(M) \Longrightarrow arity(\varphi){\le}length(env) \Longrightarrow p \Vdash \varphi\ env \Longrightarrow r \Vdash \varphi\ env$
⟨proof⟩

17.14 The Density Lemma

lemma *arity_Nand_le*:
assumes $\varphi \in formula\ \psi \in formula\ arity(Nand(\varphi, \psi)) \le length(env)\ env{\in}list(A)$
shows $arity(\varphi) \le length(env)\ arity(\psi) \le length(env)$
⟨proof⟩

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

lemma *density_lemma*:
assumes
$p{\in}P\ \varphi{\in}formula\ env{\in}list(M)\ arity(\varphi){\le}length(env)$
shows
$p \Vdash \varphi\ env \longleftrightarrow dense\_below(\{q{\in}P.\ (q \Vdash \varphi\ env)\},p)$
⟨proof⟩

17.15 The Truth Lemma

lemma *Forces_And*:
assumes
$p{\in}P\ env \in list(M)\ \varphi{\in}formula\ \psi{\in}formula$
$arity(\varphi) \le length(env)\ arity(\psi) \le length(env)$
shows
$p \Vdash And(\varphi,\psi)\ env \longleftrightarrow (p \Vdash \varphi\ env) \wedge (p \Vdash \psi\ env)$
⟨proof⟩

lemma *Forces_Nand_alt*:
assumes
$p{\in}P\ env \in list(M)\ \varphi{\in}formula\ \psi{\in}formula$
$arity(\varphi) \le length(env)\ arity(\psi) \le length(env)$
shows
$(p \Vdash Nand(\varphi,\psi)\ env) \longleftrightarrow (p \Vdash Neg(And(\varphi,\psi))\ env)$
⟨proof⟩

lemma *truth_lemma_Neg*:
assumes
$\varphi{\in}formula\ M\_generic(G)\ env{\in}list(M)\ arity(\varphi){\le}length(env)$ **and**
IH: $(\exists p{\in}G.\ p \Vdash \varphi\ env) \longleftrightarrow M[G],\ map(val(G),env) \models \varphi$
shows
$(\exists p{\in}G.\ p \Vdash Neg(\varphi)\ env) \longleftrightarrow M[G],\ map(val(G),env) \models Neg(\varphi)$
⟨proof⟩

lemma *truth_lemma_And*:
  assumes
    $env{\in}list(M)$ $\varphi{\in}formula$ $\psi{\in}formula$
    $arity(\varphi){\le}length(env)$ $arity(\psi) \le length(env)$ $M\_generic(G)$
    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 $arity(ren\_truth\_lemma(\varphi)) \le 6 \cup succ(arity(\varphi))$
⟨*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
    $\varphi{\in}formula$ $env{\in}list(M)$ $arity(\varphi) \le succ(length(env))$
  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
    $\bigwedge env.\ env{\in}list(M) \Longrightarrow arity(\varphi){\le}length(env) \Longrightarrow$
     $(\exists\, p{\in}G.\ p \Vdash \varphi\ env) \longleftrightarrow M[G],\ map(val(G),env) \models \varphi$
  ⟨*proof*⟩

17.16 The "Definition of forcing"

lemma *definition_of_forcing*:
assumes
p∈*P* *φ*∈*formula* *env*∈*list*(*M*) *arity*(*φ*)≤*length*(*env*)
shows
(*p* ⊩ *φ* *env*) ⟷
(∀ *G*. *M_generic*(*G*) ∧ *p*∈*G* ⟶ *M*[*G*], *map*(*val*(*G*),*env*) ⊨ *φ*)
⟨*proof*⟩

lemmas *definability* = *forces_type*
end

end

18 Auxiliary renamings for Separation

theory *Separation_Rename*
imports *Interface*
begin

lemma *apply_fun*: *f* ∈ *Pi*(*A*,*B*) ==> <*a*,*b*>: *f* ⟹ *f*‘*a* = *b*
⟨*proof*⟩

lemma *nth_concat* : [*p*,*t*] ∈ *list*(*A*) ⟹ *env*∈ *list*(*A*) ⟹ *nth*(*1* #+ *length*(*env*),[*p*]@ *env* @ [*t*]) = *t*
⟨*proof*⟩

lemma *nth_concat2* : *env*∈ *list*(*A*) ⟹ *nth*(*length*(*env*),*env* @ [*p*,*t*]) = *p*
⟨*proof*⟩

lemma *nth_concat3* : *env*∈ *list*(*A*) ⟹ *u* = *nth*(*succ*(*length*(*env*)), *env* @ [*pi*, *u*])
⟨*proof*⟩

definition
sep_var :: *i* ⇒ *i* **where**
sep_var(*n*) == {<*0*,*1*>,<*1*,*3*>,<*2*,*4*>,<*3*,*5*>,<*4*,*0*>,<*5*#+*n*,*6*>,<*6*#+*n*,*2*>}

definition
sep_env :: *i* ⇒ *i* **where**
sep_env(*n*) == λ *i* ∈ (*5*#+*n*)-*5* . *i*#+*2*

definition *weak* :: [*i*, *i*] ⇒ *i* **where**
weak(*n*,*m*) == {*i*#+*m* . *i* ∈ *n*}

lemma *weakD* :
assumes *n* ∈ *nat* *k*∈*nat* *x* ∈ *weak*(*n*,*k*)
shows ∃ *i* ∈ *n* . *x* = *i*#+*k*
⟨*proof*⟩

lemma *weak_equal* :
  assumes *n*∈*nat* *m*∈*nat*
  shows *weak*(*n*,*m*) = (*m*#+*n*) - *m*
⟨*proof*⟩

lemma *weak_zero*:
  shows *weak*(*0*,*n*) = *0*
  ⟨*proof*⟩

lemma *weakening_diff* :
  assumes *n* ∈ *nat*
  shows *weak*(*n*,*7*) - *weak*(*n*,*5*) ⊆ {*5*#+*n*, *6*#+*n*}
  ⟨*proof*⟩

lemma *in_add_del* :
  assumes *x*∈*j*#+*n* *n*∈*nat* *j*∈*nat*
  shows *x* < *j* ∨ *x* ∈ *weak*(*n*,*j*)
⟨*proof*⟩

lemma *sep_env_action*:
  assumes
    [*t*,*p*,*u*,*P*,*leq*,*o*,*pi*] ∈ *list*(*M*)
    env ∈ *list*(*M*)
  shows ∀ *i* . *i* ∈ *weak*(*length*(*env*),*5*) ⟶
     nth(*sep_env*(*length*(*env*))‘*i*,[*t*,*p*,*u*,*P*,*leq*,*o*,*pi*]@*env*) = *nth*(*i*,[*p*,*P*,*leq*,*o*,*t*] @ *env* @ [*pi*,*u*])
⟨*proof*⟩

lemma *sep_env_type* :
  assumes *n* ∈ *nat*
  shows *sep_env*(*n*) : (*5*#+*n*)-*5* → (*7*#+*n*)-*7*
⟨*proof*⟩

lemma *sep_var_fin_type* :
  assumes *n* ∈ *nat*
  shows *sep_var*(*n*) : *7*#+*n* -||> *7*#+*n*
  ⟨*proof*⟩

lemma *sep_var_domain* :
  assumes *n* ∈ *nat*
  shows *domain*(*sep_var*(*n*)) = *7*#+*n* - *weak*(*n*,*5*)
⟨*proof*⟩

lemma *sep_var_type* :
  assumes *n* ∈ *nat*
  shows *sep_var*(*n*) : (*7*#+*n*)-*weak*(*n*,*5*) → *7*#+*n*
  ⟨*proof*⟩

lemma *sep_var_action* :
  assumes
    [t,p,u,P,leq,o,pi] ∈ list(M)
    env ∈ list(M)
  shows ∀ i . i ∈ (7#+length(env)) - weak(length(env),5) ⟶
    nth(sep_var(length(env))`i,[t,p,u,P,leq,o,pi]@env) = nth(i,[p,P,leq,o,t] @ env @ [pi,u])
  ⟨proof⟩

definition
  rensep :: i ⇒ i **where**
  rensep(n) == union_fun(sep_var(n),sep_env(n),7#+n-weak(n,5),weak(n,5))

lemma *rensep_aux* :
  assumes n∈nat
  shows (7#+n-weak(n,5)) ∪ weak(n,5) = 7#+n 7#+n ∪ (7 #+ n - 7) = 7#+n
⟨proof⟩

lemma *rensep_type* :
  assumes n∈nat
  shows rensep(n) ∈ 7#+n → 7#+n
⟨proof⟩

lemma *rensep_action* :
  assumes [t,p,u,P,leq,o,pi] @ env ∈ list(M)
  shows ∀ i . i < 7#+length(env) ⟶ nth(rensep(length(env))`i,[t,p,u,P,leq,o,pi]@env) = nth(i,[p,P,leq,o,t] @ env @ [pi,u])
⟨proof⟩

definition *sep_ren* :: [i,i] ⇒ i **where**
  sep_ren(n,φ) == ren(φ)`(7#+n)`(7#+n)`rensep(n)

lemma *arity_rensep*: **assumes** φ∈formula env ∈ list(M)
  arity(φ) ≤ 7#+length(env)
shows arity(sep_ren(length(env),φ)) ≤ 7#+length(env)
  ⟨proof⟩

lemma *type_rensep* [TC]:
  assumes φ∈formula env∈list(M)
  shows sep_ren(length(env),φ) ∈ formula
  ⟨proof⟩

lemma *sepren_action*:
  assumes arity(φ) ≤ 7 #+ length(env)
    [t,p,u,P,leq,o,pi] ∈ list(M)
    env∈list(M)
    φ∈formula

shows *sats*(*M*, *sep_ren*(*length*(*env*),*φ*),[*t*,*p*,*u*,*P*,*leq*,*o*,*pi*] @ *env*) ⟷ *sats*(*M*, *φ*,[*p*,*P*,*leq*,*o*,*t*] @ *env* @ [*pi*,*u*])
⟨*proof*⟩

end

19 The Axiom of Separation in $M[G]$

theory *Separation_Axiom*
imports *Forcing_Theorems Separation_Rename*
begin

context *G_generic*
begin

lemma *map_val* :
assumes *env*∈*list*(*M*[*G*])
shows ∃ *nenv*∈*list*(*M*). *env* = *map*(*val*(*G*),*nenv*)
⟨*proof*⟩

lemma *Collect_sats_in_MG* :
assumes
c∈*M*[*G*]
φ ∈ *formula* *env*∈*list*(*M*[*G*]) *arity*(*φ*) ≤ *1* #+ *length*(*env*)
shows
{*x*∈*c*. (*M*[*G*], [*x*] @ *env* ⊨ *φ*)}∈ *M*[*G*]
⟨*proof*⟩

theorem *separation_in_MG*:
assumes
φ∈*formula* **and** *arity*(*φ*) ≤ *1* #+ *length*(*env*) **and** *env*∈*list*(*M*[*G*])
shows
separation(##*M*[*G*],*λx*. (*M*[*G*], [*x*] @ *env* ⊨ *φ*))
⟨*proof*⟩

end

end

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

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

context *forcing_data*
begin

lemma *val_Upair* :

one ∈ *G* ⟹ *val*(*G*,{⟨τ,*one*⟩,⟨ϱ,*one*⟩}) = {*val*(*G*,τ),*val*(*G*,ϱ)}
⟨*proof*⟩

lemma *pairing_in_MG* :
assumes *M_generic*(*G*)
shows *upair_ax*(##*M*[*G*])
⟨*proof*⟩

end
end

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

theory *Union_Axiom*
imports *Names*
begin

context *forcing_data*
begin

definition *Union_name_body* :: [*i*,*i*,*i*,*i*] ⇒ *o* **where**
Union_name_body(*P'*,*leq'*,τ,ϑ*p*) == (∃ σ[##*M*].
∃ *q*[##*M*]. (*q*∈ *P'* ∧ (<σ,*q*> ∈ τ ∧
(∃ *r*[##*M*].*r*∈*P'* ∧ (<*fst*(ϑ*p*),*r*> ∈ σ ∧ <*snd*(ϑ*p*),*r*> ∈ *leq'* ∧
<*snd*(ϑ*p*),*q*> ∈ *leq'*)))))

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

lemma *Union_name_fm_type* [*TC*]:
Union_name_fm ∈*formula*
⟨*proof*⟩

lemma *arity_Union_name_fm* :
arity(*Union_name_fm*) = *4*
⟨*proof*⟩

lemma *sats_Union_name_fm* :

$[\![\ a \in M;\ b \in M\ ;\ P' \in M\ ;\ p \in M\ ;\ \vartheta \in M\ ;\ \tau \in M\ ;\ leq' \in M\ ]\!] \Longrightarrow$
$sats(M, Union\_name\_fm, [<\vartheta,p>,\tau,leq',P']@[a,b]) \longleftrightarrow$
$Union\_name\_body(P',leq',\tau,<\vartheta,p>)$
⟨*proof*⟩

lemma *domD* :
assumes $\tau \in M$ $\sigma \in domain(\tau)$
shows $\sigma \in M$
⟨*proof*⟩

definition *Union_name* :: $i \Rightarrow i$ **where**
$Union\_name(\tau) ==$
$\{u \in domain(\bigcup(domain(\tau))) \times P\ .\ Union\_name\_body(P,leq,\tau,u)\}$

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

22 The Powerset Axiom in $M[G]$

theory *Powerset_Axiom*
imports *Separation_Axiom Pairing_Axiom Union_Axiom*
begin

⟨*ML*⟩

lemma *Collect_inter_Transset*:
assumes
$Transset(M)$ $b \in M$
shows

{x∈b . P(x)} = {x∈b . P(x)} ∩ M
⟨proof⟩

context *G_generic* **begin**

lemma *name_components_in_M*:
assumes <σ,p>∈ϑ ϑ ∈ M
shows σ∈M p∈M
⟨proof⟩

lemma *sats_fst_snd_in_M*:
assumes
A∈M B∈M φ ∈ formula p∈M l∈M o∈M χ∈M
arity(φ) ≤ 6
shows
{sq ∈A×B . sats(M,φ,[snd(sq),p,l,o,fst(sq),χ])} ∈ M
(**is** ?ϑ ∈ M)
⟨proof⟩

lemma *Pow_inter_MG*:
assumes
a∈M[G]
shows
Pow(a) ∩ M[G] ∈ 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

23 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

24 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

25 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*] ⇒ *i* **where**
least_fm(*q*,*i*) ≡ *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* ∈ *nat* ⟹ *q*∈*formula* ⟹ *least_fm*(*q*,*i*) ∈ *formula*
⟨*proof*⟩

lemmas *basic_fm_simps* = *sats_subset_fm′* *sats_transset_fm′* *sats_ordinal_fm′*

lemma *sats_least_fm* :
assumes *p_iff_sats*:
⋀*a*. *a* ∈ *A* ⟹ *P*(*a*) ⟷ *sats*(*A*, *p*, *Cons*(*a*, *env*))
shows
⟦*y* ∈ *nat*; *env* ∈ *list*(*A*) ; *0*∈*A*⟧
⟹ *sats*(*A*, *least_fm*(*p*,*y*), *env*) ⟷
least(##*A*, *P*, *nth*(*y*,*env*))
⟨*proof*⟩

lemma *least_iff_sats*:
assumes *is_Q_iff_sats*:
⋀*a*. *a* ∈ *A* ⟹ *is_Q*(*a*) ⟷ *sats*(*A*, *q*, *Cons*(*a*,*env*))
shows
⟦*nth*(*j*,*env*) = *y*; *j* ∈ *nat*; *env* ∈ *list*(*A*); *0*∈*A*⟧
⟹ *least*(##*A*, *is_Q*, *y*) ⟷ *sats*(*A*, *least_fm*(*q*,*j*), *env*)
⟨*proof*⟩

lemma *least_conj*: *a*∈*M* ⟹ *least*(##*M*, λ*x*. *x*∈*M* ∧ *Q*(*x*),*a*) ⟷ *least*(##*M*,*Q*,*a*)
⟨*proof*⟩

lemma (**in** *M_ctm*) *unique_least*: *a*∈*M* ⟹ *b*∈*M* ⟹ *least*(##*M*,*Q*,*a*) ⟹ *least*(##*M*,*Q*,*b*)
⟹ *a*=*b*
⟨*proof*⟩

context *M_trivial*
begin

25.1 Absoluteness and closure under *Least*

lemma *least_abs*:
assumes ⋀*x*. *Q*(*x*) ⟹ *M*(*x*) *M*(*a*)
shows *least*(*M*,*Q*,*a*) ⟷ *a* = (μ *x*. *Q*(*x*))

⟨*proof*⟩

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

end

end

26 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*) == *sum*(*renrep1_fn*,*id*(*length*(*env*)),*6*,*8*,*length*(*env*))

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

lemma *renrep_type* [*TC*]:
  assumes φ∈*formula* *env* ∈ *list*(*M*)
    shows *renrep*(φ,*env*) ∈ *formula*
  ⟨*proof*⟩

lemma *arity_renrep*:
  assumes φ∈*formula* *arity*(φ)≤ *6*#+*length*(*env*) *env* ∈ *list*(*M*)
    shows *arity*(*renrep*(φ,*env*)) ≤ *8*#+*length*(*env*)
 ⟨*proof*⟩

lemma *renrep_sats* :
    arity(φ) ≤ *6* #+ *length*(*env*) ⟹
    [*P*,*leq*,*o*,*p*,ϱ,τ] @ *env* ∈ *list*(*M*) ⟹
    V ∈ *M* ⟹ α ∈ *M* ⟹
    φ∈*formula* ⟹
  sats(*M*, φ, [*p*,*P*,*leq*,*o*,ϱ,τ] @ *env*) ⟷ *sats*(*M*, *renrep*(φ,*env*), [*V*,τ,ϱ,*p*,α,*P*,*leq*,*o*] @ *env*)
  ⟨*proof*⟩

⟨*ML*⟩

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

renpbdy_fn(*env*) == *sum*(*renpbdy1_fn*,*id*(*length*(*env*)),*6*,*7*,*length*(*env*))

definition
renpbdy :: [*i*,*i*] ⇒ *i* **where**
renpbdy(*φ*,*env*) = *ren*(*φ*)‘(*6*#+*length*(*env*))‘(*7*#+*length*(*env*))‘*renpbdy_fn*(*env*)

lemma
renpbdy_type [*TC*]: *φ*∈*formula* ⟹ *env*∈*list*(*M*) ⟹ *renpbdy*(*φ*,*env*) ∈ *formula*
⟨*proof*⟩

lemma *arity_renpbdy*: *φ*∈*formula* ⟹ *arity*(*φ*) ≤ *6* #+ *length*(*env*) ⟹ *env*∈*list*(*M*) ⟹ *arity*(*renpbdy*(*φ*,*env*)) ≤ *7* #+ *length*(*env*)
⟨*proof*⟩

lemma
sats_renpbdy: *arity*(*φ*) ≤ *6* #+ *length*(*nenv*) ⟹ [*ϱ*,*p*,*x*,*α*,*P*,*leq*,*o*,*π*] @ *nenv* ∈ *list*(*M*) ⟹ *φ*∈*formula* ⟹
sats(*M*, *φ*, [*ϱ*,*p*,*α*,*P*,*leq*,*o*] @ *nenv*) ⟷ *sats*(*M*, *renpbdy*(*φ*,*nenv*), [*ϱ*,*p*,*x*,*α*,*P*,*leq*,*o*] @ *nenv*)
⟨*proof*⟩

⟨*ML*⟩

definition *renbody_fn* :: *i* ⇒ *i* **where**
renbody_fn(*env*) == *sum*(*renbody1_fn*,*id*(*length*(*env*)),*5*,*6*,*length*(*env*))

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

lemma
renbody_type [*TC*]: *φ*∈*formula* ⟹ *env*∈*list*(*M*) ⟹ *renbody*(*φ*,*env*) ∈ *formula*
⟨*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
    $x\in M$ $y\in M$
  shows
    $powerset(\#\#M,x,y) \longleftrightarrow y = Pow(x) \cap M$
  ⟨*proof*⟩

schematic_goal *sats_prebody_fm_auto*:
  assumes
    $\varphi\in formula$ $[P,leq,one,p,\varrho,\pi]$ @ $nenv \in list(M)$ $\alpha\in M$ $arity(\varphi) \leq 2 \#+ length(nenv)$

  shows
    $(\exists\,\tau\in M.\ \exists\,V\in M.\ is\_Vset(\#\#M,\alpha,V) \wedge \tau\in V \wedge sats(M,forces(\varphi),[p,P,leq,one,\varrho,\tau]$
@ $nenv))$
    $\longleftrightarrow sats(M,?prebody\_fm,[\varrho,p,\alpha,P,leq,one]$ @ $nenv)$
  ⟨*proof*⟩

⟨*ML*⟩

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 *prebody_fm_type* [*TC*]:
  assumes $\varphi\in formula$
    $env \in list(M)$
  shows $prebody\_fm(\varphi,env)\in formula$
⟨*proof*⟩

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(φ,env))≤6 #+ length(env)
⟨*proof*⟩

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

lemma *body_fm′_type[TC]*: *φ∈formula ⟹ env∈list(M) ⟹ body_fm′(φ,env)∈formula*
⟨*proof*⟩

lemma *arity_body_fm′*:
assumes
φ∈formula α∈M env∈list(M) arity(φ) ≤ 2 #+ length(env)
shows
arity(body_fm′(φ,env))≤5 #+ length(env)
⟨*proof*⟩

lemma *sats_body_fm′*:
assumes
∃t p. x=<t,p> x∈M [α,P,leq,one,p,ϱ] @ nenv ∈list(M) φ∈formula arity(φ) ≤ 2 #+ length(nenv)
shows
sats(M,body_fm′(φ,nenv),[x,α,P,leq,one] @ nenv) ⟷
sats(M,renpbdy(prebody_fm(φ,nenv),nenv),[fst(x),snd(x),x,α,P,leq,one] @ nenv)
⟨*proof*⟩

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

lemma *body_fm_type [TC]*: *env∈list(M) ⟹ φ∈formula ⟹ body_fm(φ,env)∈formula*
⟨*proof*⟩

lemma *sats_body_fm*:
assumes
∃t p. x=<t,p> [α,x,m,P,leq,one] @ nenv ∈list(M)
φ∈formula arity(φ) ≤ 2 #+ length(nenv)
shows
sats(M,body_fm(φ,nenv),[α,x,m,P,leq,one] @ nenv) ⟷
sats(M,renpbdy(prebody_fm(φ,nenv),nenv),[fst(x),snd(x),x,α,P,leq,one] @ nenv)
⟨*proof*⟩

lemma *sats_renpbdy_prebody_fm*:
assumes
∃t p. x=<t,p> x∈M [α,m,P,leq,one] @ nenv ∈list(M)
φ∈formula arity(φ) ≤ 2 #+ length(nenv)
shows
sats(M,renpbdy(prebody_fm(φ,nenv),nenv),[fst(x),snd(x),x,α,P,leq,one] @ nenv)

⟷
sats(M,prebody_fm(φ,nenv),[fst(x),snd(x),α,P,leq,one] @ nenv)
⟨proof⟩

lemma *body_lemma*:
assumes
∃t p. x=<t,p> x∈M [x,α,m,P,leq,one] @ nenv ∈list(M)
φ∈formula arity(φ) ≤ 2 #+ length(nenv)
shows
sats(M,body_fm(φ,nenv),[α,x,m,P,leq,one] @ nenv) ⟷
(∃τ∈M. ∃V∈M. is_Vset(λa. (##M)(a),α,V) ∧ τ ∈ V ∧ (snd(x) ⊩ φ ([fst(x),τ]@nenv)))
⟨proof⟩

lemma *Replace_sats_in_MG*:
assumes
c∈M[G] env ∈ list(M[G])
φ ∈ formula arity(φ) ≤ 2 #+ length(env)
univalent(##M[G], c, λx v. (M[G] , [x,v]@env ⊨ φ))
shows
{v. x∈c, v∈M[G] ∧ (M[G] , [x,v]@env ⊨ φ)} ∈ M[G]
⟨proof⟩

theorem *strong_replacement_in_MG*:
assumes
φ∈formula **and** arity(φ) ≤ 2 #+ length(env) env ∈ list(M[G])
shows
strong_replacement(##M[G],λx v. sats(M[G],φ,[x,v] @ env))
⟨proof⟩

end

end

27 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

28 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 :: $i \Rightarrow i \Rightarrow i \Rightarrow i$ **where**
$induced\_surj(f,a,e) == f\text{-}``(range(f)\text{-}a) \times \{e\} \cup restrict(f,f\text{-}``a)$

lemma *domain_induced_surj*: $domain(induced\_surj(f,a,e)) = domain(f)$
⟨*proof*⟩

lemma *range_restrict_vimage*:
assumes $function(f)$
shows $range(restrict(f,f\text{-}``a)) \subseteq a$
⟨*proof*⟩

lemma *induced_surj_type*:
assumes
$function(f)$
shows
$induced\_surj(f,a,e)$: $domain(f) \to \{e\} \cup a$
and
$x \in f\text{-}``a \Longrightarrow induced\_surj(f,a,e)`x = f`x$
⟨*proof*⟩

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
$induced\_surj(f,a,e) \in surj(\alpha,a)$
⟨*proof*⟩

context *G_generic*
begin

definition
upair_name :: $i \Rightarrow i \Rightarrow i$ **where**
$upair\_name(\tau,\varrho) == \{\langle\tau,one\rangle,\langle\varrho,one\rangle\}$

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 $is\_upair\_name(x,y,z) \longleftrightarrow z = upair\_name(x,y)$
⟨*proof*⟩

lemma *upair_name_closed* :
$[\![x \in M; y \in M]\!] \Longrightarrow upair\_name(x,y) \in M$
⟨*proof*⟩

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*] :
$[\![s \in nat;x \in nat;y \in nat;o \in nat]\!] \Longrightarrow upair\_name\_fm(s,x,y,o) \in formula$
⟨*proof*⟩

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
$sats(M,upair\_name\_fm(x,y,o,z),env) \longleftrightarrow is\_upair\_name(nth(x,env),nth(y,env),nth(z,env))$
⟨*proof*⟩

definition
$opair\_name :: i \Rightarrow i \Rightarrow i$ **where**
$opair\_name(\tau,\varrho) == upair\_name(upair\_name(\tau,\tau),upair\_name(\tau,\varrho))$

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 $is\_opair\_name(x,y,z) \longleftrightarrow z = opair\_name(x,y)$
⟨*proof*⟩

lemma *opair_name_closed* :
$[\![x \in M; y \in M]\!] \Longrightarrow opair\_name(x,y) \in M$
⟨*proof*⟩

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*∈*nat*;*x*∈*nat*;*y*∈*nat*;*o*∈*nat*⟧ ⟹ *opair_name_fm*(*s*,*x*,*y*,*o*)∈*formula*
⟨*proof*⟩

lemma *sats_opair_name_fm* :
assumes *x*∈*nat* *y*∈*nat* *z*∈*nat* *o*∈*nat* *env*∈*list*(*M*) *nth*(*o*,*env*)=*one*
shows
sats(*M*,*opair_name_fm*(*x*,*y*,*o*,*z*),*env*) ⟷ *is_opair_name*(*nth*(*x*,*env*),*nth*(*y*,*env*),*nth*(*z*,*env*))
⟨*proof*⟩

lemma *val_upair_name* : *val*(*G*,*upair_name*(τ,ϱ)) = {*val*(*G*,τ),*val*(*G*,ϱ)}
⟨*proof*⟩

lemma *val_opair_name* : *val*(*G*,*opair_name*(τ,ϱ)) = <*val*(*G*,τ),*val*(*G*,ϱ)>
⟨*proof*⟩

lemma *val_RepFun_one*: *val*(*G*,{⟨*f*(*x*),*one*⟩ . *x*∈*a*}) = {*val*(*G*,*f*(*x*)) . *x*∈*a*}
⟨*proof*⟩

28.1 $M[G]$ is a transitive model of ZF

interpretation *mgzf*: *M_ZF_trans* *M*[*G*]
⟨*proof*⟩

definition
is_opname_check :: [*i*,*i*,*i*] ⇒ *o* **where**
is_opname_check(*s*,*x*,*y*) ≡ ∃ *chx*∈*M*. ∃ *sx*∈*M*. *is_check*(*x*,*chx*) ∧ *fun_apply*(##*M*,*s*,*x*,*sx*)
∧
is_opair_name(*chx*,*sx*,*y*)

definition
opname_check_fm :: [*i*,*i*,*i*,*i*] ⇒ *i* **where**
opname_check_fm(*s*,*x*,*y*,*o*) ≡ *Exists*(*Exists*(*And*(*check_fm*(*2*#+*x*,*2*#+*o*,*1*),
And(*fun_apply_fm*(*2*#+*s*,*2*#+*x*,*0*),*opair_name_fm*(*1*,*0*,*2*#+*o*,*2*#+*y*)))))

lemma *opname_check_fm_type*[*TC*] :
⟦ *s*∈*nat*;*x*∈*nat*;*y*∈*nat*;*o*∈*nat*⟧ ⟹ *opname_check_fm*(*s*,*x*,*y*,*o*)∈*formula*
⟨*proof*⟩

lemma *sats_opname_check_fm*:
assumes *x*∈*nat* *y*∈*nat* *z*∈*nat* *o*∈*nat* *env*∈*list*(*M*) *nth*(*o*,*env*)=*one*
y<*length*(*env*)
shows
sats(*M*,*opname_check_fm*(*x*,*y*,*z*,*o*),*env*) ⟷ *is_opname_check*(*nth*(*x*,*env*),*nth*(*y*,*env*),*nth*(*z*,*env*))
⟨*proof*⟩

lemma *opname_check_abs* :
  assumes $s{\in}M$ $x{\in}M$ $y{\in}M$
  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

29 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

30 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

31 A poset of successions

theory *Succession_Poset*
imports
Arities Proper_Extension Synthetic_Definition
Names
begin

31.1 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* ⇒ *i* (_ˆ<ω [100]100) **where**
seqspace(*B*) ≡ ⋃ *n*∈*nat*. (*n*→*B*)

lemma *seqspaceI*[*intro*]: *n*∈*nat* ⟹ *f*:*n*→*B* ⟹ *f*∈*seqspace*(*B*)
⟨*proof*⟩

lemma *seqspaceD*[*dest*]: *f*∈*seqspace*(*B*) ⟹ ∃ *n*∈*nat*. *f*:*n*→*B*
⟨*proof*⟩

lemma *seqspace_type*:
f ∈ *B*ˆ<ω ⟹ ∃ *n*∈*nat*. *f*:*n*→*B*
⟨*proof*⟩

schematic_goal *seqspace_fm_auto*:
assumes
nth(*i*,*env*) = *n* *nth*(*j*,*env*) = *z* *nth*(*h*,*env*) = *B*
i ∈ *nat* *j* ∈ *nat* *h*∈*nat* *env* ∈ *list*(*A*)
shows
(∃ *om*∈*A*. *omega*(##*A*,*om*) ∧ *n* ∈ *om* ∧ *is_funspace*(##*A*, *n*, *B*, *z*)) ⟷ (*A*, *env* ⊨ (*?sqsprp*(*i*,*j*,*h*)))
⟨*proof*⟩

⟨*ML*⟩

locale *M_seqspace* = *M_trancl* +
assumes
seqspace_replacement: *M*(*B*) ⟹ *strong_replacement*(*M*,λ*n z*. *n*∈*nat* ∧ *is_funspace*(*M*,*n*,*B*,*z*))
begin

lemma *seqspace_closed*:
M(*B*) ⟹ *M*(*B*ˆ<ω)
⟨*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*ˆ<ω *a*∈*A*

shows *seq_upd*$(f,a) \in A$ ^<ω
⟨*proof*⟩

lemma *seq_upd_apply_domain* [*simp*]:
assumes f:$n\to A$ $n\in nat$
shows *seq_upd*(f,a)‘$n = a$
⟨*proof*⟩

lemma *zero_in_seqspace* :
shows $0 \in A$ ^<ω
⟨*proof*⟩

definition
seqleR :: $i \Rightarrow i \Rightarrow o$ **where**
seqleR$(f,g) \equiv g \subseteq f$

definition
seqlerel :: $i \Rightarrow i$ **where**
seqlerel$(A) \equiv$ *Rrel*$(\lambda x\ y.\ y \subseteq x, A$ ^<$\omega)$

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

lemma *seqleI*[*intro*!]:
$\langle f,g\rangle \in 2$^<$\omega\times 2$^<$\omega \Longrightarrow g \subseteq f \Longrightarrow \langle f,g\rangle \in$ *seqle*
⟨*proof*⟩

lemma *seqleD*[*dest*!]:
$z \in$ *seqle* $\Longrightarrow \exists x\ y.\ \langle x,y\rangle \in 2$^<$\omega\times 2$^<$\omega \wedge y \subseteq x \wedge z = \langle x,y\rangle$
⟨*proof*⟩

lemma *upd_leI* :
assumes $f\in 2$^<ω $a\in 2$
shows <*seq_upd*(f,a),f>$\in$*seqle* (**is** <?f,_>$\in$_)
⟨*proof*⟩

lemma *preorder_on_seqle*: *preorder_on*$(2$^<ω,*seqle*$)$
⟨*proof*⟩

lemma *zero_seqle_max*: $x\in 2$^<$\omega \Longrightarrow \langle x,0\rangle \in$ *seqle*
⟨*proof*⟩

interpretation *forcing_notion* 2^<ω *seqle* 0
⟨*proof*⟩

abbreviation *SEQle* :: $[i, i] \Rightarrow o$ (**infixl** $\preceq s$ *50*)
where $x \preceq s\ y \equiv$ *Leq*(x,y)

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

lemma *seqspace_separative*:
assumes *f*∈*2*^<ω
shows *seq_upd*(*f*,*0*) ⊥*s* *seq_upd*(*f*,*1*) (**is** *?f* ⊥*s* *?g*)
⟨*proof*⟩

definition *is_seqleR* :: [*i*⇒*o*,*i*,*i*] ⇒ *o* **where**
is_seqleR(*Q*,*f*,*g*) ≡ *g* ⊆ *f*

definition *seqleR_fm* :: *i* ⇒ *i* **where**
seqleR_fm(*fg*) ≡ *Exists*(*Exists*(*And*(*pair_fm*(*0*,*1*,*fg*#+*2*),*subset_fm*(*1*,*0*))))

lemma *type_seqleR_fm* :
fg ∈ *nat* ⟹ *seqleR_fm*(*fg*) ∈ *formula*
⟨*proof*⟩

lemma *arity_seqleR_fm* :
fg ∈ *nat* ⟹ *arity*(*seqleR_fm*(*fg*)) = *succ*(*fg*)
⟨*proof*⟩

lemma (**in** *M_basic*) *seqleR_abs*:
assumes *M*(*f*) *M*(*g*)
shows *seqleR*(*f*,*g*) ⟷ *is_seqleR*(*M*,*f*,*g*)
⟨*proof*⟩

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

lemma (**in** *M_ctm*) *seqleR_fm_sats* :
assumes *fg*∈*nat* *env*∈*list*(*M*)
shows *sats*(*M*,*seqleR_fm*(*fg*),*env*) ⟷ *relP*(##*M*,*is_seqleR*,*nth*(*fg*, *env*))
⟨*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⟩

31.2 Cohen extension is proper

interpretation ctm_separative 2^<ω seqle 0
⟨proof⟩

lemma cohen_extension_is_proper: ∃ G. M_generic(G) ∧ M ≠ GenExt(G)
⟨proof⟩

end

end

32 The main theorem

theory Forcing_Main
imports

Internal_ZFC_Axioms
Choice_Axiom
Ordinals_In_MG
Succession_Poset

begin

32.1 The generic extension is countable

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

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*⟩

32.2 The main result

theorem *extensions_of_ctms*:
assumes
$M \approx nat$ $Transset(M)$ $M \models ZF$
shows
$\exists N.$
$M \subseteq N \wedge N \approx nat \wedge Transset(N) \wedge N \models ZF \wedge M{\neq}N \wedge$
$(\forall\, \alpha.\ Ord(\alpha) \longrightarrow (\alpha \in M \longleftrightarrow \alpha \in N)) \wedge$
$(M,\ [] {\models}\ AC \longrightarrow N \models ZFC)$
⟨*proof*⟩

end