Formalization of Forcing in Isabelle/ZF

Emmanuel Gunther* Miguel Pagano* Pedro Sánchez Terraf*†

September 3, 2020

Abstract

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

Contents

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

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

1 Introduction

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

Release notes

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

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

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

2 Forcing notions

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

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

2.1 Basic concepts

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

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

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

lemma *compat_inI* :
  ⟦ *d*∈*A* ; ⟨*d*,*p*⟩∈*r* ; ⟨*d*,*g*⟩∈*r* ⟧ ⟹ *compat_in*(*A*,*r*,*p*,*g*)
  ⟨*proof*⟩

lemma *refl_compat*:
  ⟦ *refl*(*A*,*r*) ; ⟨*p*,*q*⟩ ∈ *r* | *p*=*q* | ⟨*q*,*p*⟩ ∈ *r* ; *p*∈*A* ; *q*∈*A*⟧ ⟹ *compat_in*(*A*,*r*,*p*,*q*)
  ⟨*proof*⟩

lemma *chain_compat*:
  refl(*A*,*r*) ⟹ *linear*(*A*,*r*) ⟹ (∀ *p*∈*A*.∀ *q*∈*A*. *compat_in*(*A*,*r*,*p*,*q*))
  ⟨*proof*⟩

lemma *subset_fun_image*: *f*:*N*→*P* ⟹ *f*‘‘*N*⊆*P*
  ⟨*proof*⟩

lemma *refl_monot_domain*: *refl*(*B*,*r*) ⟹ *A*⊆*B* ⟹ *refl*(*A*,*r*)
  ⟨*proof*⟩

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

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

lemma *refl_leq*:
  r∈*P* ⟹ *r*≼*r*
  ⟨*proof*⟩

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

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

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

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

lemma *P_dense*: $dense(P)$
⟨*proof*⟩

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

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

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

lemma *leq_transD'*: $A \subseteq P \Longrightarrow a \preceq b \Longrightarrow b \preceq c \Longrightarrow a \in A \Longrightarrow b \in P \Longrightarrow c \in P \Longrightarrow a \preceq c$
⟨*proof*⟩

lemma *compatD*[*dest*!]: $compat(p,q) \Longrightarrow \exists d \in P.\ d \preceq p \wedge d \preceq q$
⟨*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 $\exists d \in D.\ d \in P \wedge d \preceq q$
⟨*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 $dense\_below(D,p)$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

lemma *filter_leqD* : $filter(G) \Longrightarrow x \in G \Longrightarrow y \in P \Longrightarrow x{\preceq}y \Longrightarrow y \in G$
⟨*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**
$upclosure(A) \equiv \{p{\in}P.\exists a{\in}A.\ a{\preceq}p\}$

lemma *upclosureI* [*intro*] : $p \in P \Longrightarrow a \in A \Longrightarrow a \preceq p \Longrightarrow p \in upclosure(A)$
⟨*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* :
assumes $A \subseteq P$
shows *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*:
assumes $A \subseteq P$ $(\forall\, p \in A.\forall\, q \in A.\ compat\_in(A,leq,p,q))$
shows *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*:
assumes $f \in nat \to P$ *preorder_on*(*P*,*leq*)
$\forall\, n \in nat.\ \langle f\ `\ succ(n),\ f\ `\ n\rangle \in leq$
$m \in nat$
shows $n \in nat \Longrightarrow n \le m \Longrightarrow \langle f\ `\ m,\ f\ `\ n\rangle \in leq$
⟨*proof*⟩

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

end

2.2 Towards Rasiowa-Sikorski Lemma (RSL)

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

begin

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

The next lemma identifies a sufficient condition for obtaining RSL.

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

end

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

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

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

context *countable_generic*
begin

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

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 $(\lambda n{\in}nat.\ RS\_seq(n,P,leq,p,enum,\mathcal{D})){:}\ nat \to P$
  ⟨*proof*⟩

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

end

end

3 A pointed version of DC

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

begin

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

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

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

lemma *witness_related* :
  assumes $a{\in}A$
    $(\forall X\ .\ X{\neq}0 \wedge X{\subseteq}A \longrightarrow s`X{\in}X)$

$\forall y \in A.\ \{x \in A.\ \langle y,x\rangle \in R\ \} \neq 0\ n \in nat$
shows $\langle dc\_witness(n,\ A,\ a,\ s,\ R), dc\_witness(succ(n),\ A,\ a,\ s,\ R)\rangle \in R$
$\langle proof\rangle$

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

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

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

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

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

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

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

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

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

⟨*proof*⟩

end

4 The general Rasiowa-Sikorski lemma

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

context *countable_generic*
begin

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

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

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

end

end

5 Auxiliary results on arithmetic

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *Un_leD2* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow Ord(k) \Longrightarrow i \cup j \le k \Longrightarrow j \le k$
⟨*proof*⟩

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

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

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

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

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

lemma *nat_union_abs2* :
$[\![Ord(i) ; Ord(j) ; i \le j]\!] \Longrightarrow j \cup i = j$
⟨*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* : $Ord(p) \Longrightarrow Ord(q) \Longrightarrow \neg\, p \le q \Longrightarrow q \le p$
⟨*proof*⟩

lemmas *nat_simp_union* = *nat_un_max nat_max_ty max_def*

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

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

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

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

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

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

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

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

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

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

5.1 Some results in ordinal arithmetic

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

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

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

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

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

6 Automatic synthesis of formulas

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

begin
⟨*ML*⟩

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

end

7 Aids to internalize formulas

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

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

lemma *nth_closed* :
  assumes $env \in list(A)$ $0 \in A$
  shows $nth(n,env) \in A$
  ⟨*proof*⟩

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

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

lemmas *nth_rules* = *nth_0 nth_ConsI nat_0I nat_succI*

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

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

lemmas *fm_defs* =
omega_fm_def limit_ordinal_fm_def empty_fm_def typed_function_fm_def
pair_fm_def upair_fm_def domain_fm_def function_fm_def succ_fm_def
cons_fm_def fun_apply_fm_def image_fm_def big_union_fm_def union_fm_def
relation_fm_def composition_fm_def field_fm_def ordinal_fm_def range_fm_def
transset_fm_def subset_fm_def Replace_fm_def

lemmas *formulas_def* = *fm_defs*
is_iterates_fm_def iterates_MH_fm_def is_wfrec_fm_def is_recfun_fm_def is_transrec_fm_def
is_nat_case_fm_def quasinat_fm_def number1_fm_def ordinal_fm_def finite_ordinal_fm_def
cartprod_fm_def sum_fm_def Inr_fm_def Inl_fm_def
formula_functor_fm_def
Memrel_fm_def transset_fm_def subset_fm_def pre_image_fm_def restriction_fm_def
list_functor_fm_def tl_fm_def quasilist_fm_def Cons_fm_def Nil_fm_def

⟨*ML*⟩

end

8 Some enhanced theorems on recursion

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

We prove results concerning definitions by well-founded recursion on some relation R and its transitive closure R ^∗

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

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

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

definition *tr_down* :: [i,i] ⇒ i
where *tr_down*(r,a) = (r^+)-‘‘$\{a\}$

lemma *tr_downD* : $x \in$ *tr_down*(r,a) $\Longrightarrow \langle x,a\rangle \in r$^+
⟨*proof*⟩

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

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

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

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

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

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

lemma *wfrec_trans_restr* : $relation(r) \Longrightarrow wf(r) \Longrightarrow trans(r) \Longrightarrow r\text{-}``\{a\} \subseteq A \Longrightarrow a \in A \Longrightarrow$
$wfrec(r, a, H) = wfrec[A](r, a, H)$
⟨*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 $restrict(f,Memrel(eclose(\{x\}))\text{-``}\{y\})`w$
$= restrict(f,(Memrel(eclose(\{x\}))\hat{}+)\text{-``}\{y\})`w$
⟨*proof*⟩

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

end

9 Relativization of the cumulative hierarchy

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

begin

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

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

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

$\Longrightarrow M(\bigcup x{\in}A.\ f(x))$
⟨*proof*⟩

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

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

lemma *is_powapply_closed*: $is\_powapply(M,f,y,z) \Longrightarrow M(z)$
⟨*proof*⟩

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

definition
is_Vfrom :: $[i{\Rightarrow}o,i,i,i] \Rightarrow o$ **where**
$is\_Vfrom(M,A,i,V) \equiv is\_transrec(M,is\_HVfrom(M,A),i,V)$

definition
is_Vset :: $[i{\Rightarrow}o,i,i] \Rightarrow o$ **where**
$is\_Vset(M,i,V) \equiv \exists z[M].\ empty(M,z) \wedge is\_Vfrom(M,z,i,V)$

9.1 Formula synthesis

schematic_goal *sats_is_powapply_fm_auto*:
assumes
$f{\in}nat\ y{\in}nat\ z{\in}nat\ env{\in}list(A)\ 0{\in}A$
shows
$is\_powapply(\#\#A,nth(f,\ env),nth(y,\ env),nth(z,\ env))$
$\longleftrightarrow sats(A,?ipa\_fm(f,y,z),env)$
⟨*proof*⟩

schematic_goal *is_powapply_iff_sats*:
assumes
$nth(f,env) = ff\ nth(y,env) = yy\ nth(z,env) = zz\ 0{\in}A$
$f \in nat\ \ y \in nat\ z \in nat\ env \in list(A)$
shows
$is\_powapply(\#\#A,ff,yy,zz) \longleftrightarrow sats(A,\ ?is\_one\_fm(a,r),\ env)$

⟨proof⟩

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

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

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

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

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

9.2 Absoluteness results

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

begin

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

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

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

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

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

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

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

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

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

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

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

10 Interface between set models and Constructibility

This theory provides an interface between Paulson's relativization results and set models of ZFC. In particular, it is used to prove that the locale *forcing_data* is a sublocale of all relevant locales in ZF-Constructibility (*M_trivial*, *M_basic*, *M_eclose*, etc).

theory *Interface*
imports
Nat_Miscellanea
Relative_Univ
begin

syntax

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

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

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

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

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

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

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

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

context *M_basic* **begin**

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

end

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

lemma *empty_intf* :
infinity_ax(*M*) ⟹
(∃ *z*[*M*]. *empty*(*M*,*z*))
⟨*proof*⟩

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

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

locale *M_ZF_trans* = *M_ZF* +
  assumes
    trans_M: *Transset*(M)
begin

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

10.1 Interface with *M_trivial*

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

end

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

sublocale *M_ZF_trans* $\subseteq$ *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
$(\exists p\in A.\ p\in r\ \&\ (\exists x\in A.\ x\in B\ \&\ pair(\#\#A,x,y,p))) \longleftrightarrow sats(A,?imfm(i,j,h),env)$
$\langle proof \rangle$

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

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

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

schematic_goal *rest_fm_auto*:
assumes
$nth(i,env) = z\ nth(j,env) = C$
$i \in nat\ j \in nat\ env \in list(A)$
shows
$(\exists x\in A.\ x\in C\ \&\ (\exists y\in A.\ pair(\#\#A,x,y,z)))$
$\longleftrightarrow sats(A,?rfm(i,j),env)$
$\langle proof \rangle$

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

schematic_goal *comp_fm_auto*:
  assumes
    $nth(i,env) = xz$ $nth(j,env) = S$ $nth(h,env) = R$
    $i \in nat$ $j \in nat$ $h \in nat$ $env \in list(A)$
  shows
    $(\exists x{\in}A.\ \exists y{\in}A.\ \exists z{\in}A.\ \exists xy{\in}A.\ \exists yz{\in}A.$
      $pair(\#\#A,x,z,xz)\ \&\ pair(\#\#A,x,y,xy)\ \&\ pair(\#\#A,y,z,yz)\ \&\ xy{\in}S$
$\&\ yz{\in}R)$
  $\longleftrightarrow sats(A,?cfm(i,j,h),env)$
  ⟨*proof*⟩

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

schematic_goal *pred_fm_auto*:
  assumes
    $nth(i,env) = y$ $nth(j,env) = R$ $nth(h,env) = X$
    $i \in nat$ $j \in nat$ $h \in nat$ $env \in list(A)$
  shows
    $(\exists p{\in}A.\ p{\in}R\ \&\ pair(\#\#A,y,X,p)) \longleftrightarrow sats(A,?pfm(i,j,h),env)$
  ⟨*proof*⟩

lemma *pred_sep_intf*:
  assumes
    $R{\in}M$
    and
    $X{\in}M$
  shows
    $separation(\#\#M,\ \lambda y.\ \exists p{\in}M.\ p{\in}R\ \&\ pair(\#\#M,y,X,p))$
⟨*proof*⟩

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

lemma *memrel_sep_intf*:
separation(##*M*, λ*z*. ∃ *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*,λ*x*. ∃ *xa*∈*M*. ∃ *xb*∈*M*.
pair(##*M*,*x*,*a*,*xa*) & *xa* ∈ *r* & *pair*(##*M*,*x*,*b*,*xb*) & *xb* ∈ *r* &
(∃ *fx*∈*M*. ∃ *gx*∈*M*. *fun_apply*(##*M*,*f*,*x*,*fx*) & *fun_apply*(##*M*,*g*,*x*,*gx*) &
fx ≠ *gx*))
⟨*proof*⟩

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

lemma *funspace_succ_rep_intf* :
assumes
n∈*M*
shows
strong_replacement(##*M*,

```
      λp z. ∃f∈M. ∃b∈M. ∃nb∈M. ∃cnbf∈M.
          pair(##M,f,b,p) & pair(##M,n,b,nb) & is_cons(##M,nb,f,cnbf)
&
          upair(##M,cnbf,cnbf,z))
⟨proof⟩
```

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

end

sublocale *M_ZF_trans* ⊆ *M_basic* ##*M*
⟨*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 trans_closure_fm_auto:
  assumes
```

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

⟨*ML*⟩

schematic_goal *wellfounded_trancl_fm_auto*:
assumes
nth(*i*,*env*) = *p* *nth*(*j*,*env*) = *r* *nth*(*k*,*env*) = *B*
i ∈ *nat* *j* ∈ *nat* *k* ∈ *nat* *env* ∈ *list*(*A*)
shows
wellfounded_trancl(##*A*,*B*,*r*,*p*) ⟷ *sats*(*A*,*?wtf*(*i*,*j*,*k*),*env*)
⟨*proof*⟩

context *M_ZF_trans*
begin

lemma *wftrancl_separation_intf*:
assumes
r∈*M* **and** *Z*∈*M*
shows
separation (##*M*, *wellfounded_trancl*(##*M*,*Z*,*r*))
⟨*proof*⟩

Proof that *nat* ∈ *M*

lemma *finite_sep_intf*: *separation*(##*M*, λ*x*. *x*∈*nat*)
⟨*proof*⟩

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

lemma *nat_subset_I*: ∃ *I*∈*M*. *nat* ⊆ *I*
⟨*proof*⟩

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

end

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*) *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
$A{\in}M$
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*⟩

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

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

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

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

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

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

declare *is_powapply_fm_def*[*fm_definitions add*]

lemma *sats_is_powapply_fm* :
  assumes
    $f \in nat$ $y \in nat$ $z \in nat$ $env \in list(A)$ $0 \in A$
  shows
    $is\_powapply(\#\#A, nth(f, env), nth(y, env), nth(z, env))$
    $\longleftrightarrow sats(A, is\_powapply\_fm(f,y,z), env)$
  ⟨*proof*⟩

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

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

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

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

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

definition
  $is\_Hrank\_fm :: [i,i,i] \Rightarrow i$ **where**

is_Hrank_fm(*x*,*f*,*hc*) ≡ *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*:
⟦ *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*⟩

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

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

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

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

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
$\varphi \in$ *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
Interface

begin

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

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

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

11.1 *Collects* in *M*

lemma *Collect_in_M_0p* :
assumes
Qfm : *Q_fm* ∈ *formula* **and**
Qarty : *arity*(*Q_fm*) = *1* **and**
Qsats : ⋀*x*. *x*∈*M* ⟹ *sats*(*M*,*Q_fm*,[*x*]) ⟷ *is_Q*(##*M*,*x*) **and**
Qabs : ⋀*x*. *x*∈*M* ⟹ *is_Q*(##*M*,*x*) ⟷ *Q*(*x*) **and**

$A{\in}M$
shows
$Collect(A,Q){\in}M$
⟨*proof*⟩

lemma *Collect_in_M_2p* :
assumes
Qfm : $Q\_fm \in formula$ **and**
Qarty : $arity(Q\_fm) = 3$ **and**
params_M : $y{\in}M$ $z{\in}M$ **and**
Qsats : $\bigwedge x.\ x{\in}M \Longrightarrow sats(M,Q\_fm,[x,y,z]) \longleftrightarrow is\_Q(\#\#M,x,y,z)$ **and**
Qabs : $\bigwedge x.\ x{\in}M \Longrightarrow is\_Q(\#\#M,x,y,z) \longleftrightarrow Q(x,y,z)$ **and**
$A{\in}M$
shows
$Collect(A,\lambda x.\ Q(x,y,z)){\in}M$
⟨*proof*⟩

lemma *Collect_in_M_4p* :
assumes
Qfm : $Q\_fm \in formula$ **and**
Qarty : $arity(Q\_fm) = 5$ **and**
params_M : $a1{\in}M$ $a2{\in}M$ $a3{\in}M$ $a4{\in}M$ **and**
Qsats : $\bigwedge x.\ x{\in}M \Longrightarrow sats(M,Q\_fm,[x,a1,a2,a3,a4]) \longleftrightarrow is\_Q(\#\#M,x,a1,a2,a3,a4)$
and
Qabs : $\bigwedge x.\ x{\in}M \Longrightarrow is\_Q(\#\#M,x,a1,a2,a3,a4) \longleftrightarrow Q(x,a1,a2,a3,a4)$ **and**
$A{\in}M$
shows
$Collect(A,\lambda x.\ Q(x,a1,a2,a3,a4)){\in}M$
⟨*proof*⟩

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

end

11.2 A forcing locale and generic filters

locale *forcing_data* = *forcing_notion* + *M_ctm* +
assumes *P_in_M*: $P \in M$
and *leq_in_M*: $leq \in M$

begin

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

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

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

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*]: $M\_generic(G) \Longrightarrow p{\in}G \Longrightarrow r{\in}G \Longrightarrow \exists\, q{\in}G.\ q{\preceq}p \land q{\preceq}r$
⟨*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*:
$p{\in}P \Longrightarrow \exists\, G.\ p{\in}G \land M\_generic(G)$
⟨*proof*⟩

end

lemma (**in** *M_trivial*) *compat_in_abs* :
assumes

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

context *forcing_data* **begin**

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

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

schematic_goal *ZF_power_auto*:
power_ax(##A) ⟷ (*A*, [] ⊨ *?zfpow*)
⟨*proof*⟩

⟨*ML*⟩

schematic_goal *ZF_pairing_auto*:
  upair_ax(##*A*) ⟷ (*A*, [] ⊨ *?zfpair*)
 ⟨*proof*⟩

⟨*ML*⟩

schematic_goal *ZF_foundation_auto*:
  foundation_ax(##*A*) ⟷ (*A*, [] ⊨ *?zfpow*)
 ⟨*proof*⟩

⟨*ML*⟩

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

⟨*ML*⟩

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

⟨*ML*⟩

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

⟨*ML*⟩

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**

$ZFC\_fin \equiv ZF\_fin \cup \{ZF\_choice\_fm\}$

lemma *ZFC_fin_type* : $ZFC\_fin \subseteq formula$
⟨*proof*⟩

12.1 The Axiom of Separation, internalized

lemma *iterates_Forall_type* [*TC*]:
$[\![n \in nat; p \in formula]\!] \Longrightarrow Forall\hat{\ }n(p) \in formula$
⟨*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**
$sep\_body\_fm(p) \equiv Forall(Exists(Forall($
$Iff(Member(0,1),And(Member(0,2),$
$incr\_bv1\hat{\ }2(p))))))$

lemma *sep_body_fm_type* [*TC*]: $p \in formula \Longrightarrow sep\_body\_fm(p) \in formula$
⟨*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,λx. M, [x] @ rest @ ms ⊨ φ)
⟨*proof*⟩

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

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

lemma *sats_ZF_separation_fm_iff*:
assumes
φ∈formula
shows
(*M*, [] ⊨ (*ZF_separation_fm(φ)*))
⟷
(∀ *env∈list(M)*. *arity(φ)* ≤ *1* #+ *length(env)* ⟶
separation(##M,λx. 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) ⟷ *A,env* ⊨ *?ufm(i)*
⟨*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
$A,env \models univalent\_fm(Q1\_fm,Q2\_fm,i) \longleftrightarrow univalent(\#\#A,B,Q)$
⟨*proof*⟩

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

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

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

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

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

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

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

definition
$rep\_body\_fm :: i \Rightarrow i$ **where**
$rep\_body\_fm(p) \equiv Forall(Implies($
$univalent\_fm(univalent\_Q1(incr\_bv(p)`2),univalent\_Q2(incr\_bv(p)`2),0),$
$Exists(Forall($
$Iff(Member(0,1),Exists(And(Member(0,3),incr\_bv(incr\_bv(p)`2)`2)))))))$

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

lemmas *ZF_replacement_simps* = *formula_add_params1*[*of* φ *2* _ *M* [_,_]]
sats_incr_bv_iff[*of* _ _ *M* _ []] — simplifies iterates of λ*x*. *incr_bv*(*x*) ‘ *0*
sats_incr_bv_iff[*of* _ _ *M* _ [_,_]]— simplifies λ*x*. *incr_bv*(*x*) ‘ *2*
sats_incr_bv1_iff[*of* _ _ *M*] *sats_swap_vars* **for** φ *M*

lemma *sats_rep_body_fm*:
assumes
φ ∈ *formula* *ms*∈*list*(*M*) *rest*∈*list*(*M*)
shows
M, *rest* @ *ms* ⊨ *rep_body_fm*(φ) ⟷
strong_replacement(##*M*,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*. *M*,[*x*,*y*] @ *env* ⊨ φ))
⟨*proof*⟩

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

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

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

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

definition
ZF :: *i* **where**

ZF ≡ *ZF_inf* ∪ *ZF_fin*

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

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

Satisfaction of a set of sentences

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

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

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

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

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

end

13 Renaming of variables in internalized formulas

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

lemma *app_nm* :
assumes *n*∈*nat* *m*∈*nat* *f*∈*n*→*m* *x* ∈ *nat*
shows *f*‘*x* ∈ *nat*
⟨*proof*⟩

13.1 Renaming of free variables

definition
union_fun :: [*i*,*i*,*i*,*i*] ⇒ *i* **where**
union_fun(*f*,*g*,*m*,*p*) ≡ λ*j* ∈ *m* ∪ *p* . *if j*∈*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**
  $sum(f,g,m,n,p) \equiv \lambda j \in m\#{+}p\ \ .\ if\ j{<}m\ then\ f`j\ else\ (g`(j\#{-}m))\#{+}n$

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

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

lemma *sum_action* :
  assumes $m \in nat$ $n{\in}nat$ $p{\in}nat$ $q{\in}nat$

$f \in m \to n$ $g \in p \to q$
$env \in list(M)$
$env' \in list(M)$
$env1 \in list(M)$
$env2 \in list(M)$
$length(env) = m$
$length(env1) = p$
$length(env') = n$
$\bigwedge i . i < m \Longrightarrow nth(i,env) = nth(f`i,env')$
$\bigwedge j. j < p \Longrightarrow nth(j,env1) = nth(g`j,env2)$
shows $\forall i . i < m\#+p \longrightarrow$
$nth(i,env@env1) = nth(sum(f,g,m,n,p)`i,env'@env2)$
$\langle proof \rangle$

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 $sum(f,g,m,n,p) \in (m\#+p) \to (n\#+q)$
$\langle proof \rangle$

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
$sum(f,id(length(env1)),length(env),length(env'),length(env1)) \in$
$(length(env)\#+length(env1)) \to (length(env')\#+length(env1))$
$\langle proof \rangle$

lemma *sum_type_id_aux2* :
assumes
$f \in m \to n$
$m \in nat$ $n \in nat$
$env1 \in list(M)$
shows
$sum(f,id(length(env1)),m,n,length(env1)) \in$
$(m\#+length(env1)) \to (n\#+length(env1))$
$\langle proof \rangle$

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**
$sum\_id(m,f) \equiv sum(\lambda x\in 1.x,f,1,1,m)$

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⟩

13.2 Renaming of formulas

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

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

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

ren(*Forall*(*p*)) =

  (λ *n* $\in$ *nat* . λ *m* $\in$ *nat*. $\lambda f \in n \to m$. *Forall* (*ren*(*p*)‘*succ*(*n*)‘*succ*(*m*)‘*sum_id*(*n*,*f*)))

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

lemma *nand_ar1D* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(Nand(p,q)) \leq n \Longrightarrow arity(p) \leq n$
  ⟨*proof*⟩
lemma *nand_ar2D* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(Nand(p,q)) \leq n \Longrightarrow arity(q) \leq n$
  ⟨*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* . $n \in nat \Longrightarrow m \in nat \Longrightarrow f \in n \to m \Longrightarrow arity(p) \leq n \Longrightarrow$ *arity*(*ren*(*p*)‘*n*‘*m*‘*f*)$\leq m$
  ⟨*proof*⟩

lemma *arity_forallE* : $p \in formula \Longrightarrow m \in nat \Longrightarrow arity(Forall(p)) \leq m \Longrightarrow arity(p) \leq succ(m)$
  ⟨*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 $\llbracket$ $n \in nat$; $m \in nat$; $\varrho \in list(M)$; $\varrho' \in list(M)$; $f \in n \to m$;
$arity(\varphi) \le n$;
$\bigwedge i . i < n \Longrightarrow nth(i,\varrho) = nth(f`i,\varrho')$ $\rrbracket \Longrightarrow$
$sats(M,\varphi,\varrho) \longleftrightarrow sats(M,ren(\varphi)`n`m`f,\varrho')$
⟨*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

14 Automatic relativization of terms.

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

reldb_add :: *thy_decl* % *ML*

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

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

M_trivial.nat_case_abs

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

M_basic.composition_abs
M_trans.restriction_abs
M_trans.Inter_abs
M_trivial.is_funspace_abs
M_trivial.bool_of_o_abs
M_trivial.not_abs
M_trivial.and_abs
M_trivial.or_abs
M_trivial.Nil_abs
M_trivial.Cons_abs

M_trivial.list_case_abs
M_trivial.hd_abs
M_trivial.tl_abs

lemmas *datatype_abs* =

M_datatypes.list_N_abs
M_datatypes.list_abs
M_datatypes.formula_N_abs
M_datatypes.formula_abs
M_eclose.is_eclose_n_abs
M_eclose.eclose_abs
M_datatypes.length_abs
M_datatypes.nth_abs
M_trivial.Member_abs
M_trivial.Equal_abs
M_trivial.Nand_abs
M_trivial.Forall_abs
M_datatypes.depth_abs
M_datatypes.formula_case_abs

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

⟨*ML*⟩

declare *relative_abs*[*Rel*]

declare *datatype_abs*[*Rel*]

end

15 Names and generic extensions

theory *Names*
imports
Forcing_Data
Interface
Recursion_Thms
Relativization
Synthetic_Definition
begin

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

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

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

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

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

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

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

15.1 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 $x \in A \lor (\exists\ B \in A\ .\ x \in eclose(B))$
⟨*proof*⟩

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

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

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

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

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

⟨*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*]: $x \in edrel(A) \Longrightarrow \exists\ a\in A.\ \exists\ b \in A.\ x = \langle a,b\rangle$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

definition
  GenExt :: [*i*,*i*,*i*]⇒*i*   (_^[_] [*71*,*1*])
  where $M^P[G] \equiv \{val(P,G,\tau). \tau \in M\}$

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

context *M_ctm*
begin

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

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

end

15.2 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*})) ^+

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* :
assumes *x* ∈ *M*
shows *field*(*Memrel*(*eclose*({*x*}))) ⊆ *M*
⟨*proof*⟩

lemma *aux_def_val*:
assumes *z* ∈ *domain*(*x*)

shows $wfrec(edrel(eclose(\{x\})),z,Hv(P,G)) = wfrec(edrel(eclose(\{z\})),z,Hv(P,G))$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *elem_of_val_pair'*:

assumes $\pi \in M$ $x \in val(P,G,\pi)$
shows $\exists \vartheta \in M.\ \exists p \in G.\ \langle \vartheta,p \rangle \in \pi \wedge val(P,G,\vartheta) = x$
$\langle proof \rangle$

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

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

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

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

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

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

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

lemma *one_in_M*: $one \in M$
$\langle proof \rangle$

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

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

declare *PHcheck_fm_def* [*fm_definitions*]

lemma *PHcheck_type* [*TC*]:
  ⟦ *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *u* ∈ *nat* ⟧ ⟹ *PHcheck_fm*(*x*,*y*,*z*,*u*) ∈ *formula*
  ⟨*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*)

declare *is_Hcheck_fm_def* [*fm_definitions*]

lemma *is_Hcheck_type* [*TC*]:
  ⟦ *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *u* ∈ *nat* ⟧ ⟹ *is_Hcheck_fm*(*x*,*y*,*z*,*u*) ∈ *formula*
  ⟨*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*∈*M* ; *f*∈*M* ; *x*∈*z*; *PHcheck*(*one*,*f*,*x*,*y*) ⟧ ⟹ (##*M*)(*y*)
⟨*proof*⟩

lemma *Hcheck_closed* :
∀ *y*∈*M*. ∀ *g*∈*M*. *function*(*g*) ⟶ *Hcheck*(*y*,*g*)∈*M*
⟨*proof*⟩

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

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

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

lemma *check_in_M* : *x*∈*M* ⟹ *check*(*x*) ∈ *M*
⟨*proof*⟩

end

definition
is_singleton :: [*i*⇒*o*,*i*,*i*] ⇒ *o* **where**
is_singleton(*A*,*x*,*z*) ≡ ∃ *c*[*A*]. *empty*(*A*,*c*) ∧ *is_cons*(*A*,*x*,*c*,*z*)

lemma (**in** *M_trivial*) *singleton_abs*[*simp*] : ⟦ *M*(*x*) ; *M*(*s*) ⟧ ⟹ *is_singleton*(*M*,*x*,*s*) ⟷ *s* = {*x*}
⟨*proof*⟩

definition
singleton_fm :: [*i*,*i*] ⇒ *i* **where**
singleton_fm(*i*,*j*) ≡ *Exists*(*And*(*empty_fm*(*0*), *cons_fm*(*succ*(*i*),*0*,*succ*(*j*))))

declare *singleton_fm_def*[*fm_definitions*]

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

lemma *is_singleton_iff_sats*:
⟦ *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*;
i ∈ *nat*; *j*∈*nat* ; *env* ∈ *list*(*A*)⟧
⟹ *is_singleton*(##*A*,*x*,*y*) ⟷ *sats*(*A*, *singleton_fm*(*i*,*j*), *env*)
⟨*proof*⟩

context *forcing_data* **begin**

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

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

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

⟨*ML*⟩

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

lemma *check_abs*[*Rel*] :
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**
[*fm_definitions*] :
$check\_fm(x,o,z) \equiv Exists(And(rcheck\_fm(1\#+x,0),$
$is\_wfrec\_fm(is\_Hcheck\_fm(6\#+o,2,1,0),0,1\#+x,1\#+z)))$

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

lemma *sats_check_fm* :
assumes
$nth(o,env) = one\ x\in nat\ z\in nat\ o\in nat\ env\in list(M)\ x < length(env)\ z < length(env)$

shows
  sats(*M*, *check_fm*(*x*,*o*,*z*), *env*) ⟷ *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*(*P*,*G*,*G_dot*) = *G*
⟨*proof*⟩

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

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

end

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

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

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

end
end

16 Well-founded relation on names

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

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

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

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

lemma (**in** *M_trivial*) *hcomp_abs*:
  assumes
    is_f_abs:$\bigwedge a\ z.\ M(a) \Longrightarrow M(z) \Longrightarrow is\_f(a,z) \longleftrightarrow z = f(a)$ **and**
    is_g_abs:$\bigwedge a\ z.\ M(a) \Longrightarrow M(z) \Longrightarrow is\_g(a,z) \longleftrightarrow z = g(a)$ **and**
    g_closed:$\bigwedge a.\ M(a) \Longrightarrow M(g(a))$
    $M(a)\ M(w)$
  shows
    $is\_hcomp(M,is\_f,is\_g,a,w) \longleftrightarrow w = f(g(a))$
  ⟨*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:$\bigwedge a\ b\ z.\ a \in nat \Longrightarrow b \in nat \Longrightarrow z \in M \Longrightarrow$
          $is\_f(nth(a,Cons(z,env)),nth(b,Cons(z,env))) \longleftrightarrow sats(M,pf(a,b),Cons(z,env))$
    and
    g_iff_sats:$\bigwedge a\ b\ z.\ a \in nat \Longrightarrow b \in nat \Longrightarrow z \in M \Longrightarrow$
          $is\_g(nth(a,Cons(z,env)),nth(b,Cons(z,env))) \longleftrightarrow sats(M,pg(a,b),Cons(z,env))$
    and
    $a \in nat\ w \in nat\ env \in list(M)$
  shows
    $sats(M,hcomp\_fm(pf,pg,a,w),env) \longleftrightarrow is\_hcomp(\#\#M,is\_f,is\_g,nth(a,env),nth(w,env))$
⟨*proof*⟩

definition
ftype :: $i \Rightarrow i$ **where**
ftype ≡ *fst*

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

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

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

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

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

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

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

lemmas *ecloseNI* = *ecloseNI1*

lemma *ecloseN_mono* :
assumes u ∈ *ecloseN*(x) *name1*(x) ∈ *ecloseN*(y) *name2*(x) ∈ *ecloseN*(y)

shows *u* ∈ *ecloseN*(*y*)
⟨*proof*⟩

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

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

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

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

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

lemma *is_ftype_iff_sats*:
assumes
nth(*a*,*env*) = *aa* *nth*(*b*,*env*) = *bb* *a*∈*nat* *b*∈*nat* *env* ∈ *list*(*A*)
shows
is_ftype(##*A*,*aa*,*bb*) ⟷ *sats*(*A*,*ftype_fm*(*a*,*b*), *env*)
⟨*proof*⟩

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

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

lemma *sats_snd_fm* :
⟦ *x* ∈ *nat*; *y* ∈ *nat*;*env* ∈ *list*(*A*) ⟧

⟹ *sats*(*A*, *snd_fm*(*x*,*y*), *env*) ⟷
    is_snd(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
⟨*proof*⟩

definition
is_name1 :: (*i*⇒*o*)⇒*i*⇒*i*⇒*o* **where**
is_name1(*M*,*x*,*t2*) ≡ *is_hcomp*(*M*,*is_fst*(*M*),*is_snd*(*M*),*x*,*t2*)

definition
name1_fm :: [*i*,*i*] ⇒ *i* **where**
name1_fm(*x*,*t*) ≡ *hcomp_fm*(*fst_fm*,*snd_fm*,*x*,*t*)

lemma *sats_name1_fm* :
⟦ *x* ∈ *nat*; *y* ∈ *nat*;*env* ∈ *list*(*A*) ⟧
⟹ *sats*(*A*, *name1_fm*(*x*,*y*), *env*) ⟷
    is_name1(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
⟨*proof*⟩

lemma *is_name1_iff_sats*:
assumes
nth(*a*,*env*) = *aa* *nth*(*b*,*env*) = *bb* *a*∈*nat* *b*∈*nat* *env* ∈ *list*(*A*)
shows
is_name1(##*A*,*aa*,*bb*) ⟷ *sats*(*A*,*name1_fm*(*a*,*b*), *env*)
⟨*proof*⟩

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

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

lemma *sats_snd2_fm* :
⟦ *x* ∈ *nat*; *y* ∈ *nat*;*env* ∈ *list*(*A*) ⟧
⟹ *sats*(*A*,*snd_snd_fm*(*x*,*y*), *env*) ⟷
    is_snd_snd(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
⟨*proof*⟩

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

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

lemma *sats_name2_fm* :
⟦ *x* ∈ *nat*; *y* ∈ *nat*;*env* ∈ *list*(*A*) ⟧

$\Longrightarrow$ *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*∈*nat* *b*∈*nat* *env* ∈ *list*(*A*)
shows
is_name2(##*A*,*aa*,*bb*) $\longleftrightarrow$ *sats*(*A*,*name2_fm*(*a*,*b*), *env*)
⟨*proof*⟩

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

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

lemma *sats_cond_of_fm* :
⟦ *x* ∈ *nat*; *y* ∈ *nat*;*env* ∈ *list*(*A*) ⟧
$\Longrightarrow$ *sats*(*A*,*cond_of_fm*(*x*,*y*), *env*) $\longleftrightarrow$
is_cond_of(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
⟨*proof*⟩

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

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

lemmas *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$(N,is\_name,en,t) \equiv$
  $\exists n1[N].\exists s1[N].\ is\_name(N,t,n1) \wedge is\_singleton(N,n1,s1) \wedge is\_eclose(N,s1,en)$

definition
  eclose_n1_fm :: $[i,i] \Rightarrow i$ **where**
  $eclose\_n1\_fm(m,t) \equiv Exists(Exists(And(And(name1\_fm(t\#+2,0),singleton\_fm(0,1)),$
  $is\_eclose\_fm(1,m\#+2))))$

definition
  eclose_n2_fm :: $[i,i] \Rightarrow i$ **where**
  $eclose\_n2\_fm(m,t) \equiv Exists(Exists(And(And(name2\_fm(t\#+2,0),singleton\_fm(0,1)),$
  $is\_eclose\_fm(1,m\#+2))))$

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

definition
  ecloseN_fm :: $[i,i] \Rightarrow i$ **where**
  $ecloseN\_fm(en,t) \equiv Exists(Exists(And(eclose\_n1\_fm(1,t\#+2),$
  $And(eclose\_n2\_fm(0,t\#+2),union\_fm(1,0,en\#+2)))))$
lemma *ecloseN_fm_type* $[TC]$:
  $[\![en \in nat ; t \in nat]\!] \Longrightarrow ecloseN\_fm(en,t) \in formula$
  ⟨*proof*⟩

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

definition
  frecR :: $i \Rightarrow i \Rightarrow o$ **where**
  $frecR(x,y) \equiv$
  $(ftype(x) = 1 \wedge ftype(y) = 0$
  $\wedge (name1(x) \in domain(name1(y)) \cup domain(name2(y)) \wedge (name2(x) = name1(y) \vee name2(x) = name2(y))))$
  $\vee (ftype(x) = 0 \wedge ftype(y) = 1 \wedge name1(x) = name1(y) \wedge name2(x) \in domain(name2(y)))$

lemma *frecR_ftypeD* :
  assumes $frecR(x,y)$
  shows $(ftype(x) = 0 \wedge ftype(y) = 1) \vee (ftype(x) = 1 \wedge ftype(y) = 0)$
  ⟨*proof*⟩

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

$n1,\ n2,\ q'\rangle)$
⟨proof⟩

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

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

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

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

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

lemma $frecR\_iff$:
$frecR(x,y) \longleftrightarrow$
$(ftype(x) = 1 \wedge ftype(y) = 0$
$\wedge (name1(x) \in domain(name1(y)) \cup domain(name2(y)) \wedge (name2(x) = name1(y) \vee name2(x) = name2(y))))$
$\vee (ftype(x) = 0 \wedge ftype(y) = 1 \wedge name1(x) = name1(y) \wedge name2(x) \in domain(name2(y)))$
⟨proof⟩

lemma $frecR\_D1$:
$frecR(x,y) \Longrightarrow ftype(y) = 0 \Longrightarrow ftype(x) = 1 \wedge$
$(name1(x) \in domain(name1(y)) \cup domain(name2(y)) \wedge (name2(x) = name1(y) \vee name2(x) = name2(y)))$
⟨proof⟩

lemma $frecR\_D2$:
$frecR(x,y) \Longrightarrow ftype(y) = 1 \Longrightarrow ftype(x) = 0 \wedge$
$ftype(x) = 0 \wedge ftype(y) = 1 \wedge name1(x) = name1(y) \wedge name2(x) \in domain(name2(y))$
⟨proof⟩

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

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

schematic_goal *sats_frecR_fm_auto*:
assumes
$i\in nat\ j\in nat\ env\in list(A)\ nth(i,env) = a\ nth(j,env) = b$
shows
is_frecR$(\#\#A,a,b) \longleftrightarrow sats(A,?fr\_fm(i,j),env)$
⟨*proof*⟩

⟨*ML*⟩

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

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

lemma *rank_names_types* [*TC*]:
shows *Ord*(*rank_names*(x))
⟨*proof*⟩

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

definition
type_form :: $i \Rightarrow i$ **where**
type_form$(x) \equiv$ *if* $ftype(x) = 0$ *then 1 else* *mtype_form*(x)

lemma *type_form_tc* [*TC*]:
shows *type_form*$(x) \in 3$
⟨*proof*⟩

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

shows $rank\_names(x) \le rank\_names(y)$
⟨*proof*⟩

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

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

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

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

lemma *frecrelI* :
assumes $x \in A$ $y \in A$ $frecR(x,y)$
shows $\langle x,y\rangle \in frecrel(A)$
⟨*proof*⟩

lemma *frecrelD* :
assumes $\langle x,y\rangle \in frecrel(A1 \times A2 \times A3 \times A4)$
shows $ftype(x) \in A1$ $ftype(x) \in A1$
$name1(x) \in A2$ $name1(y) \in A2$ $name2(x) \in A3$ $name2(x) \in A3$
$cond\_of(x) \in A4$ $cond\_of(y) \in A4$
$frecR(x,y)$
⟨*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*∈*A*×*A*. ∃ *x* *y*. *z* = ⟨*x*, *y*⟩ ∧ *frecR*(*x*,*y*)}
⟨*proof*⟩

lemma *frecrel_fst_snd*:
frecrel(*A*) = {*z* ∈ *A*×*A* .
ftype(*fst*(*z*)) = *1* ∧
ftype(*snd*(*z*)) = *0* ∧ *name1*(*fst*(*z*)) ∈ *domain*(*name1*(*snd*(*z*))) ∪ *domain*(*name2*(*snd*(*z*))) ∧
(*name2*(*fst*(*z*)) = *name1*(*snd*(*z*)) ∨ *name2*(*fst*(*z*)) = *name2*(*snd*(*z*)))
∨ (*ftype*(*fst*(*z*)) = *0* ∧
ftype(*snd*(*z*)) = *1* ∧ *name1*(*fst*(*z*)) = *name1*(*snd*(*z*)) ∧ *name2*(*fst*(*z*)) ∈ *domain*(*name2*(*snd*(*z*))))}
⟨*proof*⟩

end

17 Arities of internalized formulas

theory *Arities*
imports *FrecR*
begin

lemma *arity_upair_fm* : ⟦ *t1*∈*nat* ; *t2*∈*nat* ; *up*∈*nat* ⟧ ⟹
arity(*upair_fm*(*t1*,*t2*,*up*)) = ⋃ {*succ*(*t1*),*succ*(*t2*),*succ*(*up*)}
⟨*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* :
$[\![x\in nat ; y\in nat ; z\in nat]\!] \Longrightarrow arity(image\_fm(x,y,z)) = \bigcup \{succ(x), succ(y), succ(z)\}$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

∪ $succ(z)$
⟨*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

18 The definition of *forces*

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

This is the core of our development.

18.1 The relation *frecrel*

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

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

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

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

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

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

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

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

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

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

lemma *sats_frecrel_fm* :
  assumes
    a∈*nat* *r*∈*nat* *env*∈*list*(*A*)
  shows
    sats(*A*,*frecrel_fm*(*a*,*r*),*env*)
    ⟷ *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*∈ *nat* *r*∈ *nat* *env* ∈ *list*(*A*)
  shows
    is_frecrel(##*A*, *aa*,*rr*) ⟷ *sats*(*A*, *frecrel_fm*(*a*,*r*), *env*)
  ⟨*proof*⟩

definition
  names_below :: *i* ⇒ *i* ⇒ *i* **where**
  names_below(*P*,*x*) ≡ *2*×*ecloseN*(*x*)×*ecloseN*(*x*)×*P*

lemma *names_belowsD*:
  assumes *x* ∈ *names_below*(*P*,*z*)
  obtains *f* *n1* *n2* *p* **where**
    x = ⟨*f*,*n1*,*n2*,*p*⟩ *f*∈*2* *n1*∈*ecloseN*(*z*) *n2*∈*ecloseN*(*z*) *p*∈*P*
  ⟨*proof*⟩

definition
  is_names_below :: [*i*⇒*o*,*i*,*i*,*i*] ⇒ *o* **where**
  is_names_below(*M*,*P*,*x*,*nb*) ≡ ∃ *p1*[*M*]. ∃ *p0*[*M*]. ∃ *t*[*M*]. ∃ *ec*[*M*].
        is_ecloseN(*M*,*ec*,*x*) ∧ *number2*(*M*,*t*) ∧ *cartprod*(*M*,*ec*,*P*,*p0*) ∧ *cart-prod*(*M*,*ec*,*p0*,*p1*)
      ∧ *cartprod*(*M*,*t*,*p1*,*nb*)

definition
  number2_fm :: *i*⇒*i* **where**
  number2_fm(*a*) ≡ *Exists*(*And*(*number1_fm*(*0*), *succ_fm*(*0*,*succ*(*a*))))

lemma *number2_fm_type*[*TC*] :
  a∈*nat* ⟹ *number2_fm*(*a*) ∈ *formula*
  ⟨*proof*⟩

lemma *number2arity__fm* :
  a∈*nat* ⟹ *arity*(*number2_fm*(*a*)) = *succ*(*a*)
  ⟨*proof*⟩

lemma *sats_number2_fm* [*simp*]:
  ⟦ *x* ∈ *nat*; *env* ∈ *list*(*A*) ⟧
    ⟹ *sats*(*A*, *number2_fm*(*x*), *env*) ⟷ *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⟩

18.2 Definition of *forces* for equality and membership

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

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

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

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

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

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

⟨*ML*⟩

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

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

⟨*ML*⟩

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

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

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

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

declare *Hfrc_fm_def*[*fm_definitions*]

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

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*(*trans_closure_fm*(*x*,*f*)) = *succ*(*x*) ∪ *succ*(*f*)
⟨*proof*⟩

18.3 The well-founded relation *forcerel*

definition
forcerel :: *i* ⇒ *i* ⇒ *i* **where**

forcerel(*P*,*x*) ≡ *frecrel*(*names_below*(*P*,*x*)) ^+

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

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

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

18.4 *frc_at*, forcing for atomic formulas

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

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

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

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

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

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

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

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

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

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

definition
  $is\_forces\_eq' :: [i \Rightarrow o,i,i,i,i,i] \Rightarrow o$ **where**
  $is\_forces\_eq'(M,P,l,p,t1,t2) \equiv \exists o[M]. \exists z[M]. \exists t[M]. number1(M,o) \wedge empty(M,z)$
$\wedge$
                    $is\_tuple(M,z,t1,t2,p,t) \wedge is\_frc\_at(M,P,l,t,o)$

definition
  $is\_forces\_mem' :: [i \Rightarrow o,i,i,i,i,i] \Rightarrow o$ **where**
  $is\_forces\_mem'(M,P,l,p,t1,t2) \equiv \exists o[M]. \exists t[M]. number1(M,o) \wedge$
                    $is\_tuple(M,o,t1,t2,p,t) \wedge is\_frc\_at(M,P,l,t,o)$

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

definition
  $is\_forces\_nmem' :: [i \Rightarrow o,i,i,i,i,i] \Rightarrow o$ **where**
  $is\_forces\_nmem'(M,P,l,p,t1,t2) \equiv$

$\neg$ ($\exists$ *q*[*M*]. $\exists$ *qp*[*M*]. *q*∈*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*:
⟦*x*∈*M*; *y*∈*M* ⟧ ⟹ *is_ftype*(##*M*,*x*,*y*) ⟷ *y* = *ftype*(*x*) ⟨*proof*⟩

lemma *name1_abs*:
⟦*x*∈*M*; *y*∈*M* ⟧ ⟹ *is_name1*(##*M*,*x*,*y*) ⟷ *y* = *name1*(*x*)
⟨*proof*⟩

lemma *snd_snd_abs*:
⟦*x*∈*M*; *y*∈*M* ⟧ ⟹ *is_snd_snd*(##*M*,*x*,*y*) ⟷ *y* = *snd*(*snd*(*x*))

⟨*proof*⟩

lemma *name2_abs*:
  ⟦$x \in M$; $y \in M$ ⟧ ⟹ *is_name2*(##M,x,y) ⟷ y = *name2*(x)
  ⟨*proof*⟩

lemma *cond_of_abs*:
  ⟦$x \in M$; $y \in M$ ⟧ ⟹ *is_cond_of*(##M,x,y) ⟷ y = *cond_of*(x)
  ⟨*proof*⟩

lemma *tuple_abs*:
  ⟦$z \in M$;$t1 \in M$;$t2 \in M$;$p \in M$;$t \in M$⟧ ⟹
   is_tuple(##M,z,$t1$,$t2$,p,t) ⟷ t = ⟨z,$t1$,$t2$,p⟩
  ⟨*proof*⟩

lemmas *components_abs* = *ftype_abs name1_abs name2_abs cond_of_abs*
  tuple_abs

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) ⟷ *eq_case*($t1$,$t2$,p,P,*leq*,f)
⟨*proof*⟩

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

lemma *Hfrc_abs*:
  ⟦*fnnc*$\in M$; $f \in M$⟧ ⟹
   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 *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 ⟨x,y⟩ ∈ *frecrel*(*names_below*(P,z))
  shows *names_below*(P,x) ⊆ *names_below*(P,y)

⟨*proof*⟩

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

lemma *forcerel_mono2* :
  assumes ⟨*x*,*y*⟩ ∈ *frecrel*(*names_below*(*P*,*z*))
  shows *forcerel*(*P*,*x*) ⊆ *forcerel*(*P*,*y*)
  ⟨*proof*⟩

lemma *forcerel_mono_aux* :
  assumes ⟨*x*,*y*⟩ ∈ *frecrel*(*names_below*(*P*, *w*))^+
  shows *forcerel*(*P*,*x*) ⊆ *forcerel*(*P*,*y*)
  ⟨*proof*⟩

lemma *forcerel_mono* :
  assumes ⟨*x*,*y*⟩ ∈ *forcerel*(*P*,*z*)
  shows *forcerel*(*P*,*x*) ⊆ *forcerel*(*P*,*y*)
  ⟨*proof*⟩

lemma *aux*: *x* ∈ *names_below*(*P*, *w*) ⟹ ⟨*x*,*y*⟩ ∈ *forcerel*(*P*,*z*) ⟹
  (*y* ∈ *names_below*(*P*, *w*) ⟶ ⟨*x*,*y*⟩ ∈ *forcerel*(*P*,*w*))
  ⟨*proof*⟩

lemma *forcerel_eq* :
  assumes ⟨*z*,*x*⟩ ∈ *forcerel*(*P*,*x*)
  shows *forcerel*(*P*,*z*) = *forcerel*(*P*,*x*) ∩ *names_below*(*P*,*z*)×*names_below*(*P*,*z*)
  ⟨*proof*⟩

lemma *forcerel_below_aux* :
  assumes ⟨*z*,*x*⟩ ∈ *forcerel*(*P*,*x*) ⟨*u*,*z*⟩ ∈ *forcerel*(*P*,*x*)
  shows *u* ∈ *names_below*(*P*,*z*)
  ⟨*proof*⟩

lemma *forcerel_below* :
  assumes ⟨*z*,*x*⟩ ∈ *forcerel*(*P*,*x*)
  shows *forcerel*(*P*,*x*) -`` {*z*} ⊆ *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*⟩

18.5 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*⟩

18.6 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* :
assumes $Q \in M$ $x \in M$
$\bigwedge A1\ A2\ A3\ A4.\ A1 \in M \Longrightarrow A2 \in M \Longrightarrow A3 \in M \Longrightarrow A4 \in M \Longrightarrow R(A1 \times A2 \times A3 \times A4)$
shows 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*$\in M$ $z\in M$
shows *is_frc_at*(##M,P,*leq*,*fnnc*,z) $\longleftrightarrow$ z = *frc_at*(P,*leq*,*fnnc*)
⟨*proof*⟩

lemma *forces_eq'_abs* :
⟦$p \in M$; $t1 \in M$; $t2 \in M$⟧ $\Longrightarrow$ *is_forces_eq'*(##M,P,leq,p,t1,t2) $\longleftrightarrow$ *forces_eq'*(P,leq,p,t1,t2)
⟨*proof*⟩

lemma *forces_mem'_abs* :
⟦$p \in M$; $t1 \in M$; $t2 \in M$⟧ $\Longrightarrow$ *is_forces_mem'*(##M,P,leq,p,t1,t2) $\longleftrightarrow$ *forces_mem'*(P,leq,p,t1,t2)
⟨*proof*⟩

lemma *forces_neq'_abs* :
assumes
$p \in M$ $t1 \in M$ $t2 \in M$
shows
is_forces_neq'(##M,P,leq,p,t1,t2) $\longleftrightarrow$ *forces_neq'*(P,leq,p,t1,t2)
⟨*proof*⟩

lemma *forces_nmem'_abs* :
assumes
$p \in M$ $t1 \in M$ $t2 \in M$
shows
is_forces_nmem'(##M,P,leq,p,t1,t2) $\longleftrightarrow$ *forces_nmem'*(P,leq,p,t1,t2)
⟨*proof*⟩

end

18.7 Forcing for general formulas

definition
ren_forces_nand :: $i \Rightarrow i$ **where**
ren_forces_nand(φ) $\equiv$ *Exists*(*And*(*Equal*(0,1),*iterates*(λp. *incr_bv*(p)‘1 , 2, φ)))

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

lemma *arity_ren_forces_nand* :
assumes $\varphi \in$*formula*
shows *arity*(*ren_forces_nand*(φ)) $\leq$ *succ*(*arity*(φ))
⟨*proof*⟩

lemma *sats_ren_forces_nand*:
[q,P,leq,o,p] @ *env* $\in$ *list*(M) $\Longrightarrow$ $\varphi \in$*formula* $\Longrightarrow$
sats(M, *ren_forces_nand*(φ),[q,p,P,leq,o] @ *env*) $\longleftrightarrow$ *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*:
⟦ *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*) ⟷ *is_leq*(##*A*,*nth*(*leq*,*env*),*nth*(*q*,*env*),*nth*(*p*,*env*))
⟨*proof*⟩

18.7.1 The primitive recursion

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

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

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

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

context *forcing_data*
begin

18.8 Forcing for atomic formulas in context

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

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

definition
is_forces_eq :: [*i*,*i*,*i*] ⇒ *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*: $p{\in}P \Longrightarrow forces\_mem(p,t1,t2) \longleftrightarrow$
$(\forall v{\in}P.\ v \preceq p \longrightarrow$
$(\exists q.\ \exists s.\ \exists r.\ r{\in}P \wedge q{\in}P \wedge q \preceq v \wedge \langle s,r\rangle \in t2 \wedge q \preceq r \wedge forces\_eq(q,t1,s)))$
⟨*proof*⟩

lemma *forces_eq_abs* :
$[\![p{\in}M\ ;\ t1{\in}M\ ;\ t2{\in}M]\!] \Longrightarrow is\_forces\_eq(p,t1,t2) \longleftrightarrow forces\_eq(p,t1,t2)$
⟨*proof*⟩

lemma *forces_mem_abs* :
$[\![p{\in}M\ ;\ t1{\in}M\ ;\ t2{\in}M]\!] \Longrightarrow is\_forces\_mem(p,t1,t2) \longleftrightarrow forces\_mem(p,t1,t2)$
⟨*proof*⟩

lemma *sats_forces_eq_fm*:
assumes $p{\in}nat\ l{\in}nat\ q{\in}nat\ t1{\in}nat\ t2{\in}nat\ \ env{\in}list(M)$
$nth(p,env)=P\ nth(l,env)=leq$
shows $sats(M,forces\_eq\_fm(p,l,q,t1,t2),env) \longleftrightarrow$
$is\_forces\_eq(nth(q,env),nth(t1,env),nth(t2,env))$
⟨*proof*⟩

lemma *sats_forces_mem_fm*:
assumes $p{\in}nat\ l{\in}nat\ q{\in}nat\ t1{\in}nat\ t2{\in}nat\ \ env{\in}list(M)$
$nth(p,env)=P\ nth(l,env)=leq$
shows $sats(M,forces\_mem\_fm(p,l,q,t1,t2),env) \longleftrightarrow$
$is\_forces\_mem(nth(q,env),nth(t1,env),nth(t2,env))$
⟨*proof*⟩

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

definition
forces_nmem :: $[i,i,i] \Rightarrow o$ **where**
$forces\_nmem(p,t1,t2) \equiv \neg\ (\exists q{\in}P.\ q{\preceq}p \wedge forces\_mem(q,t1,t2))$

lemma *forces_neq* :
$forces\_neq(p,t1,t2) \longleftrightarrow forces\_neq'(P,leq,p,t1,t2)$
⟨*proof*⟩

lemma *forces_nmem* :
$forces\_nmem(p,t1,t2) \longleftrightarrow forces\_nmem'(P,leq,p,t1,t2)$
⟨*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*∈*M*
shows *sats*(*M*,*forces*(*Member*(*x*,*y*)),[*q*,*P*,*leq*,*one*]@*env*) ⟷
(*q*∈*P* ∧ *is_forces_mem*(*q*,*xx*,*yy*))
⟨*proof*⟩

lemma *sats_forces_Equal* :
assumes *x*∈*nat* *y*∈*nat* *env*∈*list*(*M*)
nth(*x*,*env*)=*xx* *nth*(*y*,*env*)=*yy* *q*∈*M*
shows *sats*(*M*,*forces*(*Equal*(*x*,*y*)),[*q*,*P*,*leq*,*one*]@*env*) ⟷
(*q*∈*P* ∧ *is_forces_eq*(*q*,*xx*,*yy*))
⟨*proof*⟩

lemma *sats_forces_Nand* :
assumes φ∈*formula* ψ∈*formula* *env*∈*list*(*M*) *p*∈*M*
shows *sats*(*M*,*forces*(*Nand*(φ,ψ)),[*p*,*P*,*leq*,*one*]@*env*) ⟷
(*p*∈*P* ∧ ¬(∃ *q*∈*M*. *q*∈*P* ∧ *is_leq*(##*M*,*leq*,*q*,*p*) ∧
(*sats*(*M*,*forces'*(φ),[*q*,*P*,*leq*,*one*]@*env*) ∧ *sats*(*M*,*forces'*(ψ),[*q*,*P*,*leq*,*one*]@*env*))))
⟨*proof*⟩

lemma *sats_forces_Neg* :
assumes φ∈*formula* *env*∈*list*(*M*) *p*∈*M*
shows *sats*(*M*,*forces*(*Neg*(φ)),[*p*,*P*,*leq*,*one*]@*env*) ⟷
(*p*∈*P* ∧ ¬(∃ *q*∈*M*. *q*∈*P* ∧ *is_leq*(##*M*,*leq*,*q*,*p*) ∧
(*sats*(*M*,*forces'*(φ),[*q*,*P*,*leq*,*one*]@*env*))))
⟨*proof*⟩

lemma *sats_forces_Forall* :
assumes φ∈*formula* *env*∈*list*(*M*) *p*∈*M*
shows *sats*(*M*,*forces*(*Forall*(φ)),[*p*,*P*,*leq*,*one*]@*env*) ⟷
p∈*P* ∧ (∀ *x*∈*M*. *sats*(*M*,*forces'*(φ),[*p*,*P*,*leq*,*one*,*x*]@*env*))
⟨*proof*⟩

end

18.9 The arity of *forces*

lemma *arity_forces_at*:
assumes *x* ∈ *nat* *y* ∈ *nat*
shows *arity*(*forces*(*Member*(*x*, *y*))) = (*succ*(*x*) ∪ *succ*(*y*)) #+ *4*
arity(*forces*(*Equal*(*x*, *y*))) = (*succ*(*x*) ∪ *succ*(*y*)) #+ *4*
⟨*proof*⟩

lemma *arity_forces'*:
assumes φ∈*formula*
shows *arity*(*forces'*(φ)) ≤ *arity*(φ) #+ *4*
⟨*proof*⟩

lemma *arity_forces* :
assumes φ∈*formula*

shows $arity(forces(\varphi)) \leq 4\#+arity(\varphi)$
⟨*proof*⟩

lemma *arity_forces_le* :
assumes $\varphi \in formula$ $n \in nat$ $arity(\varphi) \leq n$
shows $arity(forces(\varphi)) \leq 4\#+n$
⟨*proof*⟩

end

19 The Forcing Theorems

theory *Forcing_Theorems*
imports
Forces_Definition

begin

context *forcing_data*
begin

19.1 The forcing relation in context

abbreviation *Forces* :: $[i, i, i] \Rightarrow o$ (_ ⊩ _ _ [36,36,36] 60) **where**
$p \Vdash \varphi\ env \equiv M, ([p,P,leq,one] @ env) \models forces(\varphi)$

lemma *Collect_forces* :
assumes
fty: $\varphi \in formula$ **and**
far: $arity(\varphi) \leq length(env)$ **and**
envty: $env \in list(M)$
shows
$\{p \in P . p \Vdash \varphi\ env\} \in M$
⟨*proof*⟩

lemma *forces_mem_iff_dense_below*: $p \in P \Longrightarrow forces\_mem(p,t1,t2) \longleftrightarrow dense\_below($
$\{q \in P. \exists s. \exists r. r \in P \wedge \langle s,r \rangle \in t2 \wedge q \preceq r \wedge forces\_eq(q,t1,s)\}$
$,p)$
⟨*proof*⟩

19.2 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_eq*:
assumes $p \in P$ $r \in P$ $r \preceq p$ $forces\_eq(p,t1,t2)$
shows $forces\_eq(r,t1,t2)$
⟨*proof*⟩

19.3 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*⟩

19.4 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*⟩

19.5 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*⟩

19.6 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*⟩

19.7 The relation of forcing and connectives

lemma *Forces_Nand*:
  assumes
    $p \in P$ $env \in list(M)$ $\varphi \in formula$ $\psi \in formula$
  shows
    $(p \Vdash Nand(\varphi,\psi)\ env) \longleftrightarrow \neg(\exists q \in M.\ q \in P \wedge q \preceq p \wedge (q \Vdash \varphi\ env) \wedge (q \Vdash \psi\ env))$
  ⟨*proof*⟩

lemma *Forces_And_aux*:
  assumes
    $p \in P$ $env \in list(M)$ $\varphi \in formula$ $\psi \in formula$
  shows
    $p \Vdash And(\varphi,\psi)\ env \longleftrightarrow$
    $(\forall q \in M.\ q \in P \wedge q \preceq p \longrightarrow (\exists r \in M.\ r \in P \wedge r \preceq q \wedge (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)))$
  ⟨*proof*⟩

lemma *Forces_And_iff_dense_below*:
  assumes
    $p \in P$ $env \in list(M)$ $\varphi \in formula$ $\psi \in formula$
  shows
    $(p \Vdash And(\varphi,\psi)\ env) \longleftrightarrow dense\_below(\{r \in P.\ (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)\},p)$
  ⟨*proof*⟩

lemma *Forces_Forall*:
  assumes
    $p \in P$ $env \in list(M)$ $\varphi \in formula$
  shows
    $(p \Vdash Forall(\varphi)\ env) \longleftrightarrow (\forall x \in M.\ (p \Vdash \varphi\ ([x]\ @\ env)))$
  ⟨*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*: $\exists \vartheta.\ \exists p \in P.\ p \in G \wedge \langle \vartheta,p \rangle \in \pi \wedge val(P,G,\vartheta) = x \Longrightarrow x \in val(P,G,\pi)$
  ⟨*proof*⟩

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

lemma *left_in_M* : $tau \in M \Longrightarrow \langle a,b \rangle \in tau \Longrightarrow a \in M$
  ⟨*proof*⟩

19.8 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*⟩

19.9 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
$M\_generic(G)$ $p \in G$ $\pi \in M$ $\tau \in M$ $forces\_mem(p,\pi,\tau)$
$\bigwedge q\ \sigma.\ q \in P \Longrightarrow q \in G \Longrightarrow \sigma \in domain(\tau) \Longrightarrow forces\_eq(q,\pi,\sigma) \Longrightarrow$
$val(P,G,\pi) = val(P,G,\sigma)$
shows
$val(P,G,\pi) \in val(P,G,\tau)$
⟨*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: $\bigwedge q\ \sigma.\ q \in P \Longrightarrow q \in G \Longrightarrow \sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(forces\_mem(q,\sigma,\tau) \longrightarrow val(P,G,\sigma) \in val(P,G,\tau)) \wedge$
$(forces\_mem(q,\sigma,\vartheta) \longrightarrow val(P,G,\sigma) \in val(P,G,\vartheta))$
shows
$val(P,G,\tau) \subseteq val(P,G,\vartheta)$
⟨*proof*⟩

lemma *IV240a_eq_2nd_incl*:
assumes

$M\_generic(G)$ $p \in G$ $forces\_eq(p,\tau,\vartheta)$
and
$IH: \bigwedge q\ \sigma.\ q \in P \Longrightarrow q \in G \Longrightarrow \sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(forces\_mem(q,\sigma,\tau) \longrightarrow val(P,G,\sigma) \in val(P,G,\tau)) \wedge$
$(forces\_mem(q,\sigma,\vartheta) \longrightarrow val(P,G,\sigma) \in val(P,G,\vartheta))$
shows
$val(P,G,\vartheta) \subseteq val(P,G,\tau)$
$\langle proof \rangle$

lemma *IV240a_eq*:
assumes
$M\_generic(G)$ $p \in G$ $forces\_eq(p,\tau,\vartheta)$
and
$IH: \bigwedge q\ \sigma.\ q \in P \Longrightarrow q \in G \Longrightarrow \sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(forces\_mem(q,\sigma,\tau) \longrightarrow val(P,G,\sigma) \in val(P,G,\tau)) \wedge$
$(forces\_mem(q,\sigma,\vartheta) \longrightarrow val(P,G,\sigma) \in val(P,G,\vartheta))$
shows
$val(P,G,\tau) = val(P,G,\vartheta)$
$\langle proof \rangle$

19.10 Induction on names

lemma *core_induction*:
assumes
$\bigwedge \tau\ \vartheta\ p.\ p \in P \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![q \in P\ ;\ \sigma \in domain(\vartheta)]\!] \Longrightarrow Q(0,\tau,\sigma,q)]\!] \Longrightarrow Q(1,\tau,\vartheta,p)$
$\bigwedge \tau\ \vartheta\ p.\ p \in P \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![q \in P\ ;\ \sigma \in domain(\tau) \cup domain(\vartheta)]\!] \Longrightarrow Q(1,\sigma,\tau,q) \wedge Q(1,\sigma,\vartheta,q)]\!] \Longrightarrow Q(0,\tau,\vartheta,p)$
$ft \in 2$ $p \in P$
shows
$Q(ft,\tau,\vartheta,p)$
$\langle proof \rangle$

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*⟩

19.11 Lemma IV.2.40(a), in full

lemma *IV240a*:
assumes
$M\_generic(G)$
shows
$(\tau\in M \longrightarrow \vartheta\in M \longrightarrow (\forall p\in G.\ forces\_eq(p,\tau,\vartheta) \longrightarrow val(P,G,\tau) = val(P,G,\vartheta)))$
$\wedge$
$(\tau\in M \longrightarrow \vartheta\in M \longrightarrow (\forall p\in G.\ forces\_mem(p,\tau,\vartheta) \longrightarrow val(P,G,\tau) \in val(P,G,\vartheta)))$
(**is** $?Q(\tau,\vartheta) \wedge ?R(\tau,\vartheta)$)
⟨*proof*⟩

19.12 Lemma IV.2.40(b)

lemma *IV240b_mem*:
assumes
$M\_generic(G)\ val(P,G,\pi)\in val(P,G,\tau)\ \pi\in M\ \tau\in M$
and
$IH:\bigwedge\sigma.\ \sigma\in domain(\tau) \Longrightarrow val(P,G,\pi) = val(P,G,\sigma) \Longrightarrow$
$\exists\, p\in G.\ forces\_eq(p,\pi,\sigma)$
shows
$\exists\, p\in G.\ forces\_mem(p,\pi,\tau)$
⟨*proof*⟩

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
$M\_generic(G)\ val(P,G,\tau) = val(P,G,\vartheta)\ \tau\in M\ \vartheta\in M$
and
$IH:\bigwedge\sigma.\ \sigma\in domain(\tau)\cup domain(\vartheta) \Longrightarrow$
$(val(P,G,\sigma)\in val(P,G,\tau) \longrightarrow (\exists\, q\in G.\ forces\_mem(q,\sigma,\tau))) \wedge$
$(val(P,G,\sigma)\in val(P,G,\vartheta) \longrightarrow (\exists\, q\in G.\ forces\_mem(q,\sigma,\vartheta)))$

shows
$\exists\, p{\in}G.\ forces\_eq(p,\tau,\vartheta)$
$\langle proof \rangle$

lemma *IV240b*:
assumes
$M\_generic(G)$
shows
$(\tau{\in}M \longrightarrow \vartheta{\in}M \longrightarrow val(P,G,\tau) = val(P,G,\vartheta) \longrightarrow (\exists\, p{\in}G.\ forces\_eq(p,\tau,\vartheta))) \wedge$
$(\tau{\in}M \longrightarrow \vartheta{\in}M \longrightarrow val(P,G,\tau) \in val(P,G,\vartheta) \longrightarrow (\exists\, p{\in}G.\ forces\_mem(p,\tau,\vartheta)))$

(**is** $?Q(\tau,\vartheta) \wedge\ ?R(\tau,\vartheta)$)
$\langle proof \rangle$

lemma *map_val_in_MG*:
assumes
$env{\in}list(M)$
shows
$map(val(P,G),env){\in}list(M[G])$
$\langle proof \rangle$

lemma *truth_lemma_mem*:
assumes
$env{\in}list(M)$ $M\_generic(G)$
$n{\in}nat$ $m{\in}nat$ $n{<}length(env)$ $m{<}length(env)$
shows
$(\exists\, p{\in}G.\ p \Vdash Member(n,m)\ env) \longleftrightarrow M[G],\ map(val(P,G),env) \models Member(n,m)$
$\langle proof \rangle$

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

lemma *arities_at_aux*:
assumes
$n \in nat$ $m \in nat$ $env \in list(M)$ $succ(n) \cup succ(m) \le length(env)$
shows
$n < length(env)$ $m < length(env)$
$\langle proof \rangle$

19.13 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*⟩

19.14 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
$p \Vdash \varphi\ env \quad \longleftrightarrow \quad dense\_below(\{q{\in}P.\ (q \Vdash \varphi\ env)\},p)$
⟨*proof*⟩

19.15 The Truth Lemma

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

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: $(\exists\, p{\in}G.\ p \Vdash \varphi\ env) \longleftrightarrow M[G],\ map(val(P,G),env) \models \varphi$

shows
$(\exists\, p{\in}G.\ p \Vdash \mathit{Neg}(\varphi)\ \mathit{env}) \longleftrightarrow M[G],\ \mathit{map}(\mathit{val}(P,G),\mathit{env}) \models \mathit{Neg}(\varphi)$
$\langle \mathit{proof} \rangle$

lemma *truth_lemma_And*:
assumes
$\mathit{env}{\in}\mathit{list}(M)\ \varphi{\in}\mathit{formula}\ \psi{\in}\mathit{formula}$
$\mathit{arity}(\varphi){\le}\mathit{length}(\mathit{env})\ \mathit{arity}(\psi) \le \mathit{length}(\mathit{env})\ M\_\mathit{generic}(G)$
and
IH: $(\exists\, p{\in}G.\ p \Vdash \varphi\ \mathit{env}) \longleftrightarrow M[G],\ \mathit{map}(\mathit{val}(P,G),\mathit{env}) \models \varphi$
$(\exists\, p{\in}G.\ p \Vdash \psi\ \mathit{env}) \longleftrightarrow M[G],\ \mathit{map}(\mathit{val}(P,G),\mathit{env}) \models \psi$
shows
$(\exists\, p{\in}G.\ (p \Vdash \mathit{And}(\varphi,\psi)\ \mathit{env})) \longleftrightarrow M[G]\ ,\ \mathit{map}(\mathit{val}(P,G),\mathit{env}) \models \mathit{And}(\varphi,\psi)$
$\langle \mathit{proof} \rangle$

definition
ren_truth_lemma :: $i{\Rightarrow}i$ **where**
$\mathit{ren\_truth\_lemma}(\varphi) \equiv$
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*),
$\mathit{iterates}(\lambda p.\ \mathit{incr\_bv}(p)`5\ ,\ 6,\ \varphi))))))))))$

lemma *ren_truth_lemma_type*[*TC*] :
$\varphi{\in}\mathit{formula} \Longrightarrow \mathit{ren\_truth\_lemma}(\varphi)\ {\in}\mathit{formula}$
$\langle \mathit{proof} \rangle$

lemma *arity_ren_truth* :
assumes $\varphi{\in}\mathit{formula}$
shows $\mathit{arity}(\mathit{ren\_truth\_lemma}(\varphi)) \le 6 \cup \mathit{succ}(\mathit{arity}(\varphi))$
$\langle \mathit{proof} \rangle$

lemma *sats_ren_truth_lemma*:
$[q,b,d,a1,a2,a3]$ @ $\mathit{env} \in \mathit{list}(M) \Longrightarrow \varphi \in \mathit{formula} \Longrightarrow$
$(M,\ [q,b,d,a1,a2,a3]$ @ $\mathit{env} \models \mathit{ren\_truth\_lemma}(\varphi)\ ) \longleftrightarrow$
$(M,\ [q,a1,a2,a3,b]$ @ $\mathit{env} \models \varphi)$
$\langle \mathit{proof} \rangle$

lemma *truth_lemma'* :
assumes
$\varphi{\in}\mathit{formula}\ \mathit{env}{\in}\mathit{list}(M)\ \mathit{arity}(\varphi) \le \mathit{succ}(\mathit{length}(\mathit{env}))$
shows
$\mathit{separation}(\#\#M,\lambda d.\ \exists\, b{\in}M.\ \forall\, q{\in}P.\ q{\preceq}d \longrightarrow \neg(q \Vdash \varphi\ ([b]@\mathit{env})))$
$\langle \mathit{proof} \rangle$

lemma *truth_lemma*:
assumes
$\varphi{\in}\mathit{formula}\ M\_\mathit{generic}(G)$
shows

$\bigwedge env.\ env \in list(M) \Longrightarrow arity(\varphi) \le length(env) \Longrightarrow$
$(\exists\, p \in G.\ p \Vdash \varphi\ env) \quad \longleftrightarrow \quad M[G],\ map(val(P,G),env) \models \varphi$
⟨*proof*⟩

19.16 The "Definition of forcing"

lemma *definition_of_forcing*:
assumes
$p \in P\ \varphi \in formula\ env \in list(M)\ arity(\varphi) \le length(env)$
shows
$(p \Vdash \varphi\ env) \longleftrightarrow$
$(\forall\, G.\ M\_generic(G) \wedge p \in G \ \longrightarrow\ M[G],\ map(val(P,G),env) \models \varphi)$
⟨*proof*⟩

lemmas *definability* = *forces_type*
end

end

20 Auxiliary renamings for Separation

theory *Separation_Rename*
imports *Interface Renaming*
begin

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

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

lemma *nth_concat2* : $env \in list(A) \Longrightarrow nth(length(env),env @ [p,t]) = p$
⟨*proof*⟩

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

definition
sep_var :: $i \Rightarrow i$ **where**
$sep\_var(n) \equiv \{\langle 0,1\rangle,\langle 1,3\rangle,\langle 2,4\rangle,\langle 3,5\rangle,\langle 4,0\rangle,\langle 5\#+n,6\rangle,\langle 6\#+n,2\rangle\}$

definition
sep_env :: $i \Rightarrow i$ **where**
$sep\_env(n) \equiv \lambda\ i \in (5\#+n)\text{-}5\ .\ i\#+2$

definition *weak* :: $[i,\ i] \Rightarrow i$ **where**
$weak(n,m) \equiv \{i\#+m\ .\ i \in n\}$

lemma *weakD* :

assumes $n \in nat$ $k \in nat$ $x \in weak(n,k)$
shows $\exists\ i \in n\ .\ x = i\#{+}k$
⟨*proof*⟩

lemma *weak_equal* :
assumes $n \in nat$ $m \in nat$
shows $weak(n,m) = (m\#{+}n)$ - m
⟨*proof*⟩

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

lemma *weakening_diff* :
assumes $n \in nat$
shows $weak(n,7)$ - $weak(n,5) \subseteq \{5\#{+}n,\ 6\#{+}n\}$
⟨*proof*⟩

lemma *in_add_del* :
assumes $x \in j\#{+}n$ $n \in nat$ $j \in nat$
shows $x < j \vee x \in weak(n,j)$
⟨*proof*⟩

lemma *sep_env_action*:
assumes
$[t,p,u,P,leq,o,pi] \in list(M)$
$env \in list(M)$
shows $\forall\ i\ .\ i \in weak(length(env),5) \longrightarrow$
$nth(sep\_env(length(env))`i,[t,p,u,P,leq,o,pi]@env) = nth(i,[p,P,leq,o,t]\ @\ env$
$@\ [pi,u])$
⟨*proof*⟩

lemma *sep_env_type* :
assumes $n \in nat$
shows $sep\_env(n) : (5\#{+}n)\text{-}5 \rightarrow (7\#{+}n)\text{-}7$
⟨*proof*⟩

lemma *sep_var_fin_type* :
assumes $n \in nat$
shows $sep\_var(n) : 7\#{+}n$ -||> $7\#{+}n$
⟨*proof*⟩

lemma *sep_var_domain* :
assumes $n \in nat$
shows $domain(sep\_var(n)) = 7\#{+}n$ - $weak(n,5)$
⟨*proof*⟩

lemma *sep_var_type* :

assumes $n \in nat$
shows $sep\_var(n) : (7\#+n)\text{-}weak(n,5) \to 7\#+n$
⟨*proof*⟩

lemma *sep_var_action* :
assumes
$[t,p,u,P,leq,o,pi] \in list(M)$
$env \in list(M)$
shows $\forall\ i\ .\ i \in (7\#+length(env))\ \text{-}\ weak(length(env),5) \longrightarrow$
$nth(sep\_var(length(env))`i,[t,p,u,P,leq,o,pi]@env) = nth(i,[p,P,leq,o,t]\ @\ env\ @\ [pi,u])$
⟨*proof*⟩

definition
rensep :: $i \Rightarrow i$ **where**
$rensep(n) \equiv union\_fun(sep\_var(n),sep\_env(n),7\#+n\text{-}weak(n,5),weak(n,5))$

lemma *rensep_aux* :
assumes $n{\in}nat$
shows $(7\#+n\text{-}weak(n,5)) \cup weak(n,5) = 7\#+n\ 7\#+n \cup (\ 7\ \#+\ n\ \text{-}\ 7) = 7\#+n$
⟨*proof*⟩

lemma *rensep_type* :
assumes $n{\in}nat$
shows $rensep(n) \in 7\#+n \to 7\#+n$
⟨*proof*⟩

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

definition *sep_ren* :: $[i,i] \Rightarrow i$ **where**
$sep\_ren(n,\varphi) \equiv ren(\varphi)`(7\#+n)`(7\#+n)`rensep(n)$

lemma *arity_rensep*: **assumes** $\varphi{\in}formula\ env \in list(M)$
$arity(\varphi) \le 7\#+length(env)$
shows $arity(sep\_ren(length(env),\varphi)) \le 7\#+length(env)$
⟨*proof*⟩

lemma *type_rensep* $[TC]$:
assumes $\varphi{\in}formula\ env{\in}list(M)$
shows $sep\_ren(length(env),\varphi) \in formula$
⟨*proof*⟩

lemma *sepren_action*:
assumes $arity(\varphi) \le 7\ \#+\ length(env)$

```
    [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
```

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

```
theory Separation_Axiom
  imports Forcing_Theorems Separation_Rename
begin

context G_generic
begin

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

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

```
theory Pairing_Axiom imports Names begin

context forcing_data
```

begin

lemma *val_Upair* :
  $one \in G \Longrightarrow val(P,G,\{\langle\tau,one\rangle,\langle\varrho,one\rangle\}) = \{val(P,G,\tau),val(P,G,\varrho)\}$
  ⟨*proof*⟩

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

end
end

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

theory *Union_Axiom*
  imports *Names*
begin

context *forcing_data*
begin

definition *Union_name_body* :: $[i,i,i,i] \Rightarrow o$ **where**
  $Union\_name\_body(P',leq',\tau,\vartheta p) \equiv (\exists\ \sigma[\#\#M].$
    $\exists\ q[\#\#M].\ (q\in P' \wedge (\langle\sigma,q\rangle \in \tau \wedge$
    $(\exists\ r[\#\#M].r\in P' \wedge (\langle fst(\vartheta p),r\rangle \in \sigma \wedge \langle snd(\vartheta p),r\rangle \in leq' \wedge \langle snd(\vartheta p),q\rangle$
$\in leq')))))$

definition *Union_name_fm* :: *i* **where**
  Union_name_fm ≡
    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*

$\langle proof \rangle$

lemma *sats_Union_name_fm* :
$[\![\ env \in list(M); \ P' \in M \ ; \ p \in M \ ; \ \vartheta \in M \ ; \ \tau \in M \ ; \ leq' \in M \ ]\!] \Longrightarrow$
$sats(M,Union\_name\_fm,[\langle\vartheta,p\rangle,\tau,leq',P']@env) \longleftrightarrow$
$Union\_name\_body(P',leq',\tau,\langle\vartheta,p\rangle)$
$\langle proof \rangle$

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

lemma *Union_name_M* : **assumes** $\tau \in M$
shows $Union\_name(\tau) \in M$
$\langle proof \rangle$

lemma *Union_MG_Eq* :
assumes $a \in M[G]$ **and** $a = val(P,G,\tau)$ **and** $filter(G)$ **and** $\tau \in M$
shows $\bigcup \ a = val(P,G,Union\_name(\tau))$
$\langle proof \rangle$

lemma *union_in_MG* : **assumes** $filter(G)$
shows $Union\_ax(\#\#M[G])$
$\langle proof \rangle$

theorem *Union_MG* : $M\_generic(G) \Longrightarrow Union\_ax(\#\#M[G])$
$\langle proof \rangle$

end
end

24 The Powerset Axiom in $M[G]$

theory *Powerset_Axiom*
imports *Renaming_Auto Separation_Axiom Pairing_Axiom Union_Axiom*
begin

$\langle ML \rangle$

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

context *G_generic* **begin**

lemma *name_components_in_M*:

assumes $<\sigma,p>\in\vartheta$ $\vartheta \in M$
shows $\sigma\in M$ $p\in M$
⟨*proof*⟩

lemma *sats_fst_snd_in_M*:
assumes
$A\in M$ $B\in M$ $\varphi \in formula$ $p\in M$ $l\in M$ $o\in M$ $\chi\in M$
$arity(\varphi) \leq 6$
shows
$\{<s,q> \in A\times B$. $sats(M,\varphi,[q,p,l,o,s,\chi])\} \in M$
(**is** $?\vartheta \in M$)
⟨*proof*⟩

lemma *Pow_inter_MG*:
assumes
$a\in M[G]$
shows
$Pow(a) \cap M[G] \in M[G]$
⟨*proof*⟩
end

context *G_generic* **begin**

interpretation *mgtriv*: *M_trivial* $\#\#M[G]$
⟨*proof*⟩

theorem *power_in_MG* : $power\_ax(\#\#(M[G]))$
⟨*proof*⟩
end
end

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

theory *Extensionality_Axiom*
imports
Names
begin

context *forcing_data*
begin

lemma *extensionality_in_MG* : $extensionality(\#\#(M[G]))$
⟨*proof*⟩

end
end

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

theory *Foundation_Axiom*
imports
Names
begin

context *forcing_data*
begin

lemma *foundation_in_MG* : *foundation_ax*(##($M[G]$))
⟨*proof*⟩

lemma *foundation_ax*(##($M[G]$))
⟨*proof*⟩

end
end

27 The binder *Least*

theory *Least*
imports
Forcing_Data — only for a result to be moved below
Internalizations

begin

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

lemma *Least_Ord*: $(\mu\ \alpha.\ R(\alpha)) = (\mu\ \alpha.\ Ord(\alpha) \wedge R(\alpha))$
⟨*proof*⟩

lemma *Ord_Least_cong*:
assumes $\bigwedge y.\ Ord(y) \Longrightarrow R(y) \longleftrightarrow Q(y)$
shows $(\mu\ \alpha.\ R(\alpha)) = (\mu\ \alpha.\ Q(\alpha))$
⟨*proof*⟩

definition
least :: $[i \Rightarrow o, i \Rightarrow o, i] \Rightarrow o$ **where**
$least(M,Q,i) \equiv ordinal(M,i) \wedge ($
$(empty(M,i) \wedge (\forall b[M].\ ordinal(M,b) \longrightarrow \neg Q(b)))$
$\vee\ (Q(i) \wedge (\forall b[M].\ ordinal(M,b) \wedge b \in i \longrightarrow \neg Q(b))))$

definition
least_fm :: $[i,i] \Rightarrow i$ **where**
least_fm$(q,i) \equiv$ *And*(*ordinal_fm*(i),
Or(*And*(*empty_fm*(i),*Forall*(*Implies*(*ordinal_fm*(0),*Neg*(q))))),

And(*Exists*(*And*(*q*,*Equal*(*0*,*succ*(*i*)))),
Forall(*Implies*(*And*(*ordinal_fm*(*0*),*Member*(*0*,*succ*(*i*))),*Neg*(*q*))))))

lemma *least_fm_type*[*TC*] :*i* ∈ *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

27.1 Absoluteness and closure under *Least*

lemma *least_abs*:
assumes ⋀*x*. *Q*(*x*) ⟹ *Ord*(*x*) ⟹ ∃ *y*[*M*]. *Q*(*y*) ∧ *Ord*(*y*) *M*(*a*)
shows *least*(*M*,*Q*,*a*) ⟷ *a* = (μ *x*. *Q*(*x*))
⟨*proof*⟩

lemma *Least_closed*:
assumes ⋀*x*. *Q*(*x*) ⟹ *Ord*(*x*) ⟹ ∃ *y*[*M*]. *Q*(*y*) ∧ *Ord*(*y*)
shows *M*(μ *x*. *Q*(*x*))
⟨*proof*⟩

Older, easier to apply versions (with a simpler assumption on Q).

lemma *least_abs'*:

assumes $\bigwedge x.\ Q(x) \Longrightarrow M(x)\ M(a)$
shows $least(M,Q,a) \longleftrightarrow a = (\mu\ x.\ Q(x))$
⟨*proof*⟩

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

end

end

28 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* :
assumes *arity*(φ) ≤ *6* #+ *length*(*env*)
[*P*,*leq*,*o*,*p*,ϱ,τ] @ *env* ∈ *list*(*M*)
V ∈ *M* α ∈ *M*
φ∈*formula*
shows *sats*(*M*, φ, [*p*,*P*,*leq*,*o*,ϱ,τ] @ *env*) ⟷ *sats*(*M*, *renrep*(φ,*env*), [*V*,τ,ϱ,*p*,α,*P*,*leq*,*o*] @ *env*)
⟨*proof*⟩

⟨*ML*⟩

definition *renpbdy_fn* :: $i \Rightarrow i$ **where**
renpbdy_fn(*env*) ≡ *sum*(*renpbdy1_fn*,*id*(*length*(*env*)),*6*,*7*,*length*(*env*))

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

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

lemma
renbody_type [*TC*]: φ∈*formula* ⟹ *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*⟩

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

declare *is_eclose_fm_def*[*fm_definitions*]
        is_eclose_fm_def[*fm_definitions*]
        mem_eclose_fm_def[*fm_definitions*]
        eclose_n_fm_def[*fm_definitions*]

lemma *sats_prebody_fm*:
  assumes
   $[P,leq,one,p,\varrho]$ @ $nenv \in list(M)$ $\varphi \in formula$ $\alpha \in M$ $arity(\varphi) \leq 2 \ \#+ \ length(nenv)$
  shows
    $sats(M,prebody\_fm(\varphi,nenv),[\varrho,p,\alpha,P,leq,one]$ @ $nenv) \longleftrightarrow$
   $(\exists \tau \in M. \exists V \in M. is\_Vset(\#\#M,\alpha,V) \wedge \tau \in V \wedge sats(M,forces(\varphi),[p,P,leq,one,\varrho,\tau]$
@ $nenv))$
  ⟨*proof*⟩

lemma *arity_prebody_fm*:
  assumes
    $\varphi \in formula$ $\alpha \in M$ $env \in list(M)$ $arity(\varphi) \leq 2 \ \#+ \ length(env)$
  shows

$arity(prebody\_fm(\varphi,env))\leq 6\ \#+\ length(env)$
⟨proof⟩

definition
$body\_fm'$ :: $[i,i]\Rightarrow i$ **where**
$body\_fm'(\varphi,env) \equiv Exists(Exists(And(pair\_fm(0,1,2),renpbdy(prebody\_fm(\varphi,env),env))))$

lemma $body\_fm'\_type[TC]$: $\varphi\in formula \Longrightarrow env\in list(M) \Longrightarrow body\_fm'(\varphi,env)\in formula$
⟨proof⟩

lemma $arity\_body\_fm'$:
assumes
$\varphi\in formula\ \alpha\in M\ env\in list(M)\ arity(\varphi)\ \leq\ 2\ \#+\ length(env)$
shows
$arity(body\_fm'(\varphi,env))\leq 5\ \ \#+\ length(env)$
⟨proof⟩

lemma $sats\_body\_fm'$:
assumes
$\exists\, t\ p.\ x=\langle t,p\rangle\ x\in M\ [\alpha,P,leq,one,p,\varrho]\ @\ nenv\ \in list(M)\ \varphi\in formula\ arity(\varphi)\ \leq$
$2\ \#+\ length(nenv)$
shows
$sats(M,body\_fm'(\varphi,nenv),[x,\alpha,P,leq,one]\ @\ nenv) \longleftrightarrow$
$sats(M,renpbdy(prebody\_fm(\varphi,nenv),nenv),[fst(x),snd(x),x,\alpha,P,leq,one]\ @\ nenv)$
⟨proof⟩

definition
$body\_fm$ :: $[i,i]\Rightarrow i$ **where**
$body\_fm(\varphi,env)\ \equiv\ renbody(body\_fm'(\varphi,env),env)$

lemma $body\_fm\_type$ $[TC]$: $env\in list(M) \Longrightarrow \varphi\in formula \Longrightarrow\ body\_fm(\varphi,env)\in formula$
⟨proof⟩

lemma $sats\_body\_fm$:
assumes
$\exists\, t\ p.\ x=\langle t,p\rangle\ [\alpha,x,m,P,leq,one]\ @\ nenv\ \in list(M)$
$\varphi\in formula\ arity(\varphi)\ \leq\ 2\ \#+\ length(nenv)$
shows
$sats(M,body\_fm(\varphi,nenv),[\alpha,x,m,P,leq,one]\ @\ nenv) \longleftrightarrow$
$sats(M,renpbdy(prebody\_fm(\varphi,nenv),nenv),[fst(x),snd(x),x,\alpha,P,leq,one]\ @\ nenv)$
⟨proof⟩

lemma $sats\_renpbdy\_prebody\_fm$:
assumes
$\exists\, t\ p.\ x=\langle t,p\rangle\ x\in M\ [\alpha,m,P,leq,one]\ @\ nenv\ \in list(M)$
$\varphi\in formula\ arity(\varphi)\ \leq\ 2\ \#+\ length(nenv)$
shows
$sats(M,renpbdy(prebody\_fm(\varphi,nenv),nenv),[fst(x),snd(x),x,\alpha,P,leq,one]\ @\ nenv)$

$\longleftrightarrow$
  $sats(M, prebody\_fm(\varphi, nenv), [fst(x), snd(x), \alpha, P, leq, one] @ nenv)$
⟨*proof*⟩

lemma *body_lemma*:
assumes
$\exists\, t\ p.\ x=\langle t,p\rangle$ $x\in M$ $[x,\alpha,m,P,leq,one]$ @ $nenv \in list(M)$
$\varphi\in formula$ $arity(\varphi) \leq 2$ #+ $length(nenv)$
shows
$sats(M, body\_fm(\varphi, nenv), [\alpha, x, m, P, leq, one]$ @ $nenv) \longleftrightarrow$
$(\exists\, \tau\in M.\ \exists\, V\in M.\ is\_Vset(\lambda a.\ (\#\#M)(a), \alpha, V) \wedge \tau \in V \wedge (snd(x) \Vdash \varphi\ ([fst(x),\tau]@nenv)))$
⟨*proof*⟩

lemma *Replace_sats_in_MG*:
assumes
$c\in M[G]$ $env \in list(M[G])$
$\varphi \in formula$ $arity(\varphi) \leq 2$ #+ $length(env)$
$univalent(\#\#M[G],\ c,\ \lambda x\ v.\ (M[G]\ ,\ [x,v]@env \models \varphi)\ )$
shows
$\{v.\ x\in c,\ v\in M[G] \wedge (M[G]\ ,\ [x,v]@env \models \varphi)\} \in M[G]$
⟨*proof*⟩

theorem *strong_replacement_in_MG*:
assumes
$\varphi\in formula$ **and** $arity(\varphi) \leq 2$ #+ $length(env)$ $env \in list(M[G])$
shows
$strong\_replacement(\#\#M[G], \lambda x\ v.\ sats(M[G], \varphi, [x,v]$ @ $env))$
⟨*proof*⟩

end

end

29 The Axiom of Infinity in $M[G]$

theory *Infinity_Axiom*
imports *Pairing_Axiom Union_Axiom Separation_Axiom*
begin

context *G_generic* **begin**

interpretation *mg_triv*: *M_trivial*$\#\#M[G]$
⟨*proof*⟩

lemma *infinity_in_MG* : *infinity_ax*$(\#\#M[G])$
⟨*proof*⟩

end
end

30 The Axiom of Choice in $M[G]$

theory *Choice_Axiom*
imports *Powerset_Axiom Pairing_Axiom Union_Axiom Extensionality_Axiom Foundation_Axiom Powerset_Axiom Separation_Axiom Replacement_Axiom Interface Infinity_Axiom Relativization*
begin

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

lemma *domain_induced_surj*: *domain(induced_surj(f,a,e))* = *domain(f)*
⟨*proof*⟩

lemma *range_restrict_vimage*:
assumes *function(f)*
shows *range(restrict(f,f-``a))* ⊆ *a*
⟨*proof*⟩

lemma *induced_surj_type*:
assumes
function(f)
shows
induced_surj(f,a,e): *domain(f)* → {*e*} ∪ *a*
and
x ∈ *f-``a* ⟹ *induced_surj(f,a,e)`x* = *f`x*
⟨*proof*⟩

lemma *induced_surj_is_surj* :
assumes
e∈a function(f) domain(f) = *α* ⋀*y*. *y* ∈ *a* ⟹ ∃ *x*∈*α*. *f ` x* = *y*
shows
induced_surj(f,a,e) ∈ *surj(α,a)*
⟨*proof*⟩

context *G_generic*
begin

definition
upair_name :: *i* ⇒ *i* ⇒ *i* **where**
upair_name(τ,ϱ) ≡ *Upair(⟨τ,one⟩,⟨ϱ,one⟩)*

lemma *Upair_simp* : *Upair(a,b)* = {*a*,*b*}
⟨*proof*⟩

⟨*ML*⟩

lemma *upair_name_abs* :

assumes $x\in M$ $y\in M$ $z\in M$
shows $is\_upair\_name(\#\#M,x,y,z) \longleftrightarrow z = upair\_name(x,y)$
⟨proof⟩

definition
$opair\_name :: i \Rightarrow i \Rightarrow i$ **where**
$opair\_name(\tau,\varrho) \equiv upair\_name(upair\_name(\tau,\tau),upair\_name(\tau,\varrho))$

⟨ML⟩

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
$sats(M,upair\_name\_fm(x,y,o,z),env) \longleftrightarrow is\_upair\_name(\#\#M,nth(x,env),nth(y,env),nth(z,env))$
⟨proof⟩

lemma *opair_name_abs* :
assumes $x\in M$ $y\in M$ $z\in M$
shows $is\_opair\_name(\#\#M,x,y,z) \longleftrightarrow z = opair\_name(x,y)$
⟨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*] :
$[\![s\in nat;x\in nat;y\in nat;o\in nat]\!] \Longrightarrow opair\_name\_fm(s,x,y,o)\in formula$
⟨proof⟩

lemma *sats_opair_name_fm* :
assumes $x\in nat$ $y\in nat$ $z\in nat$ $o\in nat$ $env\in list(M)nth(o,env)=one$

shows
sats(*M*,*opair_name_fm*(*x*,*y*,*o*,*z*),*env*) ⟷ *is_opair_name*(##*M*,*nth*(*x*,*env*),*nth*(*y*,*env*),*nth*(*z*,*env*))
⟨*proof*⟩

lemma *val_upair_name* : *val*(*P*,*G*,*upair_name*(τ,ϱ)) = {*val*(*P*,*G*,τ),*val*(*P*,*G*,ϱ)}
⟨*proof*⟩

lemma *val_opair_name* : *val*(*P*,*G*,*opair_name*(τ,ϱ)) = ⟨*val*(*P*,*G*,τ),*val*(*P*,*G*,ϱ)⟩
⟨*proof*⟩

lemma *val_RepFun_one*: *val*(*P*,*G*,{⟨*f*(*x*),*one*⟩ . *x*∈*a*}) = {*val*(*P*,*G*,*f*(*x*)) . *x*∈*a*}
⟨*proof*⟩

30.1 $M[G]$ 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(##*M*,*chx*,*sx*,*y*)

definition
opname_check_fm :: [*i*,*i*,*i*,*i*] ⇒ *i* **where**
opname_check_fm(*s*,*x*,*y*,*o*) ≡ *Exists*(*Exists*(*And*(*check_fm*(*2*#+*x*,*2*#+*o*,*1*),
And(*fun_apply_fm*(*2*#+*s*,*2*#+*x*,*0*),*opair_name_fm*(*1*,*0*,*2*#+*o*,*2*#+*y*)))))

lemma *opname_check_fm_type*[*TC*] :
⟦ *s*∈*nat*;*x*∈*nat*;*y*∈*nat*;*o*∈*nat*⟧ ⟹ *opname_check_fm*(*s*,*x*,*y*,*o*)∈*formula*
⟨*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*∈*M* *x*∈*M* *y*∈*M*
shows *is_opname_check*(*s*,*x*,*y*) ⟷ *y* = *opair_name*(*check*(*x*),*s*‘*x*)
⟨*proof*⟩

lemma *repl_opname_check* :

assumes
A∈*M* *f*∈*M*
shows
{*opair_name*(*check*(*x*),*f*‘*x*). *x*∈*A*}∈*M*
⟨*proof*⟩

theorem *choice_in_MG*:
assumes *choice_ax*(##*M*)
shows *choice_ax*(##*M*[*G*])
⟨*proof*⟩

end

end

31 Ordinals in generic extensions

theory *Ordinals_In_MG*
imports
Forcing_Theorems Relative_Univ
begin

context *G_generic*
begin

lemma *rank_val*: *rank*(*val*(*P*,*G*,*x*)) ≤ *rank*(*x*) (**is** *?Q*(*x*))
⟨*proof*⟩

lemma *Ord_MG_iff*:
assumes *Ord*(α)
shows α ∈ *M* ⟷ α ∈ *M*[*G*]
⟨*proof*⟩

end

end

32 Separative notions and proper extensions

theory *Proper_Extension*
imports
Names

begin

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

locale *separative_notion* = *forcing_notion* +

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

33 A poset of successions

theory *Succession_Poset*
imports
Arities
Proper_Extension
Synthetic_Definition
Names
begin

33.1 The set of finite binary sequences

notation *nat* (‹ω›) — MOVE THIS to an appropriate place

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

definition
seqspace :: $[i,i] \Rightarrow i$ (‹_$^{<}$-› [*100*,*1*]*100*) **where**
$B^{<\alpha} \equiv \bigcup n{\in}\alpha.\ (n{\rightarrow}B)$

lemma *seqspaceI*[*intro*]: $n{\in}\alpha \Longrightarrow f{:}n{\rightarrow}B \Longrightarrow f{\in}B^{<\alpha}$
⟨*proof*⟩

lemma *seqspaceD*[*dest*]: $f{\in}B^{<\alpha} \Longrightarrow \exists\, n{\in}\alpha.\ f{:}n{\rightarrow}B$

⟨proof⟩

schematic_goal *seqspace_fm_auto*:
  assumes
    $nth(i,env) = n$ $nth(j,env) = z$ $nth(h,env) = B$
    $i \in nat$ $j \in nat$ $h \in nat$ $env \in list(A)$
  shows
    $(\exists om\in A.\ omega(\#\#A,om) \wedge n \in om \wedge is\_funspace(\#\#A,\ n,\ B,\ z)) \longleftrightarrow (A,$
$env \models (?sqsprp(i,j,h)))$
  ⟨proof⟩

⟨ML⟩

locale *M_seqspace* = *M_trancl* +
  assumes
   seqspace_replacement: $M(B) \Longrightarrow strong\_replacement(M,\lambda n\ z.\ n\in nat \wedge is\_funspace(M,n,B,z))$
begin

lemma *seqspace_closed*:
  $M(B) \Longrightarrow M(B^{<\omega})$
  ⟨proof⟩

end

sublocale $M\_ctm \subseteq M\_seqspace\ \#\#M$
⟨proof⟩

definition *seq_upd* :: $i \Rightarrow i \Rightarrow i$ **where**
  $seq\_upd(f,a) \equiv \lambda\ j \in succ(domain(f))\ .\ if\ j < domain(f)\ then\ f`j\ else\ a$

lemma *seq_upd_succ_type* :
  assumes $n\in nat$ $f\in n\to A$ $a\in A$
  shows $seq\_upd(f,a)\in succ(n) \to A$
⟨proof⟩

lemma *seq_upd_type* :
  assumes $f\in A^{<\omega}$ $a\in A$
  shows $seq\_upd(f,a) \in A^{<\omega}$
⟨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 $0 \in A^{<\omega}$
  ⟨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 $f\in 2^{<\omega}$ $a\in 2$
shows $\langle seq\_upd(f,a),f\rangle\in seqle$ (**is** $\langle ?f,\_\rangle\in\_$)
⟨*proof*⟩

lemma *preorder_on_seqle*: $preorder\_on(2^{<\omega},seqle)$
⟨*proof*⟩

lemma *zero_seqle_max*: $x\in 2^{<\omega} \Longrightarrow \langle x,0\rangle \in seqle$
⟨*proof*⟩

interpretation *sp*:*forcing_notion* $2^{<\omega}$ *seqle* *0*
⟨*proof*⟩

notation *sp.Leq* (**infixl** $\preceq s$ *50*)
notation *sp.Incompatible* (**infixl** $\perp s$ *50*)

lemma *seqspace_separative*:
assumes $f\in 2^{<\omega}$
shows $seq\_upd(f,0) \perp s\ seq\_upd(f,1)$ (**is** $?f \perp s\ ?g$)
⟨*proof*⟩

definition *is_seqleR* :: $[i\Rightarrow o,i,i] \Rightarrow o$ **where**
$is\_seqleR(Q,f,g) \equiv g \subseteq f$

definition *seqleR_fm* :: $i \Rightarrow i$ **where**
$seqleR\_fm(fg) \equiv Exists(Exists(And(pair\_fm(0,1,fg\#+2),subset\_fm(1,0))))$

lemma *type_seqleR_fm* :

$fg \in nat \Longrightarrow seqleR\_fm(fg) \in formula$
⟨*proof*⟩

lemma *arity_seqleR_fm* :
$fg \in nat \Longrightarrow arity(seqleR\_fm(fg)) = succ(fg)$
⟨*proof*⟩

lemma (**in** *M_basic*) *seqleR_abs*:
assumes $M(f)$ $M(g)$
shows $seqleR(f,g) \longleftrightarrow is\_seqleR(M,f,g)$
⟨*proof*⟩

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

lemma (**in** *M_ctm*) *seqleR_fm_sats* :
assumes $fg\in nat$ $env\in list(M)$
shows $sats(M,seqleR\_fm(fg),env) \longleftrightarrow relP(\#\#M,is\_seqleR,nth(fg, env))$
⟨*proof*⟩

lemma (**in** *M_basic*) *is_related_abs* :
assumes $\bigwedge f\ g\ .\ M(f) \Longrightarrow M(g) \Longrightarrow rel(f,g) \longleftrightarrow is\_rel(M,f,g)$
shows $\bigwedge z\ .\ M(z) \Longrightarrow relP(M,is\_rel,z) \longleftrightarrow (\exists x\ y.\ z = \langle x,y\rangle \wedge rel(x,y))$
⟨*proof*⟩

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

lemma (**in** *M_basic*) *is_Rrel_abs* :
assumes $M(A)$ $M(r)$
$\bigwedge f\ g\ .\ M(f) \Longrightarrow M(g) \Longrightarrow rel(f,g) \longleftrightarrow is\_rel(M,f,g)$
shows $is\_RRel(M,is\_rel,A,r) \longleftrightarrow r = Rrel(rel,A)$
⟨*proof*⟩

definition
$is\_seqlerel :: [i\Rightarrow o,i,i] \Rightarrow o$ **where**
$is\_seqlerel(M,A,r) \equiv is\_RRel(M,is\_seqleR,A,r)$

lemma (**in** *M_basic*) *seqlerel_abs* :
assumes $M(A)$ $M(r)$
shows $is\_seqlerel(M,A,r) \longleftrightarrow r = Rrel(seqleR,A)$
⟨*proof*⟩

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

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*⟩

33.2 Cohen extension is proper

interpretation *ctm_separative* $2^{<\omega}$ *seqle 0*
⟨*proof*⟩

lemma *cohen_extension_is_proper*: ∃ *G*. *M_generic*(*G*) ∧ *M* ≠ $M^{2^{<\omega}}[G]$
⟨*proof*⟩

end

end

34 The main theorem

theory *Forcing_Main*
imports
Internal_ZFC_Axioms
Choice_Axiom
Ordinals_In_MG
Succession_Poset

begin

34.1 The generic extension is countable

definition
minimum :: *i* ⇒ *i* ⇒ *i* **where**
minimum(*r*,*B*) ≡ *THE b*. *first*(*b*,*B*,*r*)

lemma *minimum_in*: ⟦ *well_ord*(*A*,*r*); *B*⊆*A*; *B*≠*0* ⟧ ⟹ *minimum*(*r*,*B*) ∈ *B*
⟨*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* :
  $\exists f. f \in surj(\omega,M[G])$
⟨*proof*⟩

lemma (**in** *G_generic*) *MG_eqpoll_nat*: $M[G] \approx \omega$
⟨*proof*⟩

34.2 The main result

theorem *extensions_of_ctms*:
  assumes
    $M \approx \omega$ $Transset(M)$ $M \models ZF$
  shows
    $\exists N.$
      $M \subseteq N \wedge N \approx \omega \wedge Transset(N) \wedge N \models ZF \wedge M{\neq}N \wedge$
      $(\forall \alpha. Ord(\alpha) \longrightarrow (\alpha \in M \longleftrightarrow \alpha \in N)) \wedge$
      $(M, [] \models AC \longrightarrow N \models ZFC)$
⟨*proof*⟩

end

35 Main definitions of the development

theory *Definitions_Main*
  imports *Forcing_Main*

begin

This theory gathers the main definitions of the Forcing session.

It might be considered as the bare minimum reading requisite to trust that our development indeed formalizes the theory of forcing. This should be mathematically clear since this is the only known method for obtaining proper extensions of ctms while preserving the ordinals.

The main theorem of this session and all of its relevant definitions appear in Section 35.3. The reader trusting all the libraries in which our development is based, might jump directly there. But in case one wants to dive deeper, the following sections treat some basic concepts in the ZF logic (Section 35.1) and in the ZF-Constructible library (Section 35.2) on which our definitions are built.

declare [[*show_question_marks=false*]]

35.1 ZF

For the basic logic ZF we restrict ourselves to just a few concepts.

thm *bij_def*[*unfolded inj_def surj_def*]

$bij(A, B) \equiv$
$\{f \in A \to B . \forall w{\in}A. \forall x{\in}A. f ` w = f ` x \longrightarrow w = x\} \cap$
$\{f \in A \to B . \forall y{\in}B. \exists x{\in}A. f ` x = y\}$

thm *eqpoll_def*

$A \approx B \equiv \exists f. f \in bij(A, B)$

thm *Transset_def*

$Transset(i) \equiv \forall x{\in}i. x \subseteq i$

thm *Ord_def*

$Ord(i) \equiv Transset(i) \wedge (\forall x{\in}i. Transset(x))$

thm *lt_def*

$i < j \equiv i \in j \wedge Ord(j)$

The set of natural numbers ω is defined as a fixpoint, but here we just write its characterization as the first limit ordinal.

thm *Limit_nat*[*unfolded Limit_def*] *nat_le_Limit*[*unfolded Limit_def*]

$Ord(\omega) \wedge 0 < \omega \wedge (\forall y. y < \omega \longrightarrow succ(y) < \omega)$
$Ord(i) \wedge 0 < i \wedge (\forall y. y < i \longrightarrow succ(y) < i) \Longrightarrow \omega \le i$

hide_const (**open**) *Order.pred*
thm *add_0_right add_succ_right pred_0 pred_succ_eq*

$m \#+ succ(n) = succ(m \#+ n)$
$m \in \omega \Longrightarrow m \#+ 0 = m$
$pred(0) = 0$
$pred(succ(y)) = y$

Lists

thm *Nil Cons list.induct*

$[] \in list(A)$
$[\![a \in A; l \in list(A)]\!] \Longrightarrow Cons(a, l) \in list(A)$
$[\![x \in list(A); P([]); \bigwedge a\ l.\ [\![a \in A; l \in list(A); P(l)]\!] \Longrightarrow P(Cons(a, l))]\!]$
$\Longrightarrow P(x)$

thm *length.simps app.simps nth_0 nth_Cons*

$length([]) = 0$
$length(Cons(a, l)) = succ(length(l))$
$[] @ ys = ys$
$Cons(a, l) @ ys = Cons(a, l @ ys)$
$nth(0, Cons(a, l)) = a$
$n \in \omega \Longrightarrow nth(succ(n), Cons(a, l)) = nth(n, l)$

Relative quantifications

lemma $\forall x[M].\ P(x) \equiv \forall x.\ M(x) \longrightarrow P(x)$
$\exists x[M].\ P(x) \equiv \exists x.\ M(x) \wedge P(x)$
⟨*proof*⟩

thm *setclass_iff*

$(\#\#A)(x) \longleftrightarrow x \in A$

35.2 ZF-Constructible

thm *big_union_def*

$big\_union(M, A, z) \equiv \forall x[M].\ x \in z \longleftrightarrow (\exists y[M].\ y \in A \wedge x \in y)$

thm *Union_ax_def*

$Union\_ax(M) \equiv \forall x[M].\ \exists z[M].\ big\_union(M, x, z)$

thm *power_ax_def[unfolded powerset_def subset_def]*

$power\_ax(M) \equiv \forall x[M].\ \exists z[M].\ \forall xa[M].\ xa \in z \longleftrightarrow (\forall xb[M].\ xb \in xa \longrightarrow xb \in x)$

thm *upair_def*

$upair(M, a, b, z) \equiv a \in z \wedge b \in z \wedge (\forall x[M]. x \in z \longrightarrow x = a \vee x = b)$

thm *pair_def*

$pair(M, a, b, z) \equiv$
$\exists x[M]. upair(M, a, a, x) \wedge (\exists y[M]. upair(M, a, b, y) \wedge upair(M, x, y, z))$

thm *successor_def[unfolded is_cons_def union_def]*

$successor(M, a, z) \equiv$
$\exists x[M]. upair(M, a, a, x) \wedge (\forall xa[M]. xa \in z \longleftrightarrow xa \in x \vee xa \in a)$

thm *upair_ax_def*

$upair\_ax(M) \equiv \forall x[M]. \forall y[M]. \exists z[M]. upair(M, x, y, z)$

thm *foundation_ax_def*

$foundation\_ax(M) \equiv$
$\forall x[M]. (\exists y[M]. y \in x) \longrightarrow (\exists y[M]. y \in x \wedge \neg (\exists z[M]. z \in x \wedge z \in y))$

thm *extensionality_def*

$extensionality(M) \equiv \forall x[M]. \forall y[M]. (\forall z[M]. z \in x \longleftrightarrow z \in y) \longrightarrow x = y$

thm *separation_def*

$separation(M, P) \equiv \forall z[M]. \exists y[M]. \forall x[M]. x \in y \longleftrightarrow x \in z \wedge P(x)$

thm *univalent_def*

$univalent(M, A, P) \equiv$
$\forall x[M]. x \in A \longrightarrow (\forall y[M]. \forall z[M]. P(x, y) \wedge P(x, z) \longrightarrow y = z)$

thm *strong_replacement_def*

$strong\_replacement(M, P) \equiv$
$\forall A[M].$
$univalent(M, A, P) \longrightarrow (\exists Y[M]. \forall b[M]. b \in Y \longleftrightarrow (\exists x[M]. x \in A \wedge P(x, b)))$

thm *empty_def*

$empty(M, z) \equiv \forall x[M]. \; x \notin z$

thm *transitive_set_def*[*unfolded subset_def*]

$transitive\_set(M, a) \equiv \forall x[M]. \; x \in a \longrightarrow (\forall xa[M]. \; xa \in x \longrightarrow xa \in a)$

thm *ordinal_def*

$ordinal(M, a) \equiv$
$transitive\_set(M, a) \wedge (\forall x[M]. \; x \in a \longrightarrow transitive\_set(M, x))$

thm *image_def*

$image(M, r, A, z) \equiv$
$\forall y[M]. \; y \in z \longleftrightarrow (\exists w[M]. \; w \in r \wedge (\exists x[M]. \; x \in A \wedge pair(M, x, y, w)))$

thm *fun_apply_def*

$fun\_apply(M, f, x, y) \equiv$
$\exists xs[M].$
$\quad \exists fxs[M]. \; upair(M, x, x, xs) \wedge image(M, f, xs, fxs) \wedge big\_union(M, fxs, y)$

thm *is_function_def*

$is\_function(M, r) \equiv$
$\forall x[M].$
$\quad \forall y[M].$
$\quad\quad \forall y'[M].$
$\quad\quad\quad \forall p[M].$
$\quad\quad\quad\quad \forall p'[M].$
$\quad\quad\quad\quad\quad pair(M, x, y, p) \longrightarrow$
$\quad\quad\quad\quad\quad pair(M, x, y', p') \longrightarrow p \in r \longrightarrow p' \in r \longrightarrow y = y'$

thm *is_relation_def*

$is\_relation(M, r) \equiv \forall z[M]. \; z \in r \longrightarrow (\exists x[M]. \; \exists y[M]. \; pair(M, x, y, z))$

thm *is_domain_def*

$is\_domain(M, r, z) \equiv$
$\forall x[M]. x \in z \longleftrightarrow (\exists w[M]. w \in r \wedge (\exists y[M]. pair(M, x, y, w)))$

thm *typed_function_def*

$typed\_function(M, A, B, r) \equiv$
$is\_function(M, r) \wedge$
$is\_relation(M, r) \wedge$
$is\_domain(M, r, A) \wedge$
$(\forall u[M]. u \in r \longrightarrow (\forall x[M]. \forall y[M]. pair(M, x, y, u) \longrightarrow y \in B))$

thm *surjection_def*

$surjection(M, A, B, f) \equiv$
$typed\_function(M, A, B, f) \wedge$
$(\forall y[M]. y \in B \longrightarrow (\exists x[M]. x \in A \wedge fun\_apply(M, f, x, y)))$

Internalized formulas

thm *Member Equal Nand Forall formula.induct*

$[\![x \in \omega; y \in \omega]\!] \Longrightarrow Member(x, y) \in formula$
$[\![x \in \omega; y \in \omega]\!] \Longrightarrow Equal(x, y) \in formula$
$[\![p \in formula; q \in formula]\!] \Longrightarrow Nand(p, q) \in formula$
$p \in formula \Longrightarrow Forall(p) \in formula$
$[\![x \in formula; \bigwedge x\ y. [\![x \in \omega; y \in \omega]\!] \Longrightarrow P(Member(x, y));$
$\bigwedge x\ y. [\![x \in \omega; y \in \omega]\!] \Longrightarrow P(Equal(x, y));$
$\bigwedge p\ q. [\![p \in formula; P(p); q \in formula; P(q)]\!] \Longrightarrow P(Nand(p, q));$
$\bigwedge p. [\![p \in formula; P(p)]\!] \Longrightarrow P(Forall(p))]\!]$
$\Longrightarrow P(x)$

thm *arity.simps*

$arity(Member(x, y)) = succ(x) \cup succ(y)$
$arity(Equal(x, y)) = succ(x) \cup succ(y)$
$arity(Nand(p, q)) = arity(p) \cup arity(q)$
$arity(Forall(p)) = pred(arity(p))$

thm *mem_iff_sats equal_iff_sats sats_Nand_iff sats_Forall_iff*

$[\![nth(i, env) = x; nth(j, env) = y; env \in list(A)]\!]$
$\Longrightarrow x \in y \longleftrightarrow A, env \models Member(i, j)$
$[\![nth(i, env) = x; nth(j, env) = y; env \in list(A)]\!]$
$\Longrightarrow x = y \longleftrightarrow A, env \models Equal(i, j)$
$env \in list(A) \Longrightarrow A, env \models Nand(p, q) \longleftrightarrow \neg (A, env \models p \wedge A, env \models q)$
$env \in list(A) \Longrightarrow A, env \models Forall(p) \longleftrightarrow (\forall x{\in}A. A, Cons(x, env) \models p)$

35.3 Forcing

thm *infinity_ax_def*

$infinity\_ax(M) \equiv$
$\exists I[M].$
$(\exists z[M].\ empty(M, z) \wedge z \in I) \wedge$
$(\forall y[M].\ y \in I \longrightarrow (\exists sy[M].\ successor(M, y, sy) \wedge sy \in I))$

thm *choice_ax_def*

$choice\_ax(M) \equiv \forall x[M].\ \exists a[M].\ \exists f[M].\ ordinal(M, a) \wedge surjection(M, a, x, f)$

thm *ZF_union_fm_iff_sats ZF_power_fm_iff_sats ZF_pairing_fm_iff_sats ZF_foundation_fm_iff_sats ZF_extensionality_fm_iff_sats ZF_infinity_fm_iff_sats sats_ZF_separation_fm_iff sats_ZF_replacement_fm_iff ZF_choice_fm_iff_sats*

$Union\_ax(\#\#A) \longleftrightarrow A, [] \models ZF\_union\_fm$
$power\_ax(\#\#A) \longleftrightarrow A, [] \models ZF\_power\_fm$
$upair\_ax(\#\#A) \longleftrightarrow A, [] \models ZF\_pairing\_fm$
$foundation\_ax(\#\#A) \longleftrightarrow A, [] \models ZF\_foundation\_fm$
$extensionality(\#\#A) \longleftrightarrow A, [] \models ZF\_extensionality\_fm$
$infinity\_ax(\#\#A) \longleftrightarrow A, [] \models ZF\_infinity\_fm$
$\varphi \in formula \Longrightarrow$
$M, [] \models ZF\_separation\_fm(\varphi) \longleftrightarrow$
$(\forall env{\in}list(M).$
$arity(\varphi) \leq 1 \#{+}\ length(env) \longrightarrow separation(\#\#M, \lambda x.\ M, [x] @ env \models \varphi))$
$\varphi \in formula \Longrightarrow$
$M, [] \models ZF\_replacement\_fm(\varphi) \longleftrightarrow$
$(\forall env{\in}list(M).$
$arity(\varphi) \leq 2 \#{+}\ length(env) \longrightarrow$
$strong\_replacement(\#\#M, \lambda x\ y.\ M, [x, y] @ env \models \varphi))$
$choice\_ax(\#\#A) \longleftrightarrow A, [] \models ZF\_choice\_fm$

thm *ZF_fin_def ZF_inf_def ZF_def ZFC_fin_def ZFC_def*

$ZF\_fin \equiv$
$\{ZF\_extensionality\_fm, ZF\_foundation\_fm, ZF\_pairing\_fm, ZF\_union\_fm,$
$ZF\_infinity\_fm, ZF\_power\_fm\}$
$ZF\_inf \equiv$
$\{ZF\_separation\_fm(p) .\ p \in formula\} \cup \{ZF\_replacement\_fm(p) .\ p \in formula\}$
$ZF \equiv ZF\_inf \cup ZF\_fin$
$ZFC\_fin \equiv ZF\_fin \cup \{ZF\_choice\_fm\}$
$ZFC \equiv ZF\_inf \cup ZFC\_fin$

thm *satT_def*

$A \models \Phi \equiv \forall \varphi{\in}\Phi.\ A,\ [] \models \varphi$

thm *extensions_of_ctms*

$[\![M \approx \omega;\ \mathit{Transset}(M);\ M \models \mathit{ZF}]\!]$
$\Longrightarrow \exists N.\ M \subseteq N \wedge$
$N \approx \omega \wedge$
$\mathit{Transset}(N) \wedge$
$N \models \mathit{ZF} \wedge$
$M \neq N \wedge$
$(\forall \alpha.\ \mathit{Ord}(\alpha) \longrightarrow \alpha \in M \longleftrightarrow \alpha \in N) \wedge (M,\ [] \models \mathit{ZF\_choice\_fm} \longrightarrow N$
$\models \mathit{ZFC})$

end

References

[1] E. Gunther, M. Pagano, P. Sánchez Terraf, First steps towards a formalization of forcing, in: Proceedings of the 13th Workshop on Logical and Semantic Frameworks with Applications, LSFA 2018, Fortaleza, Brazil, September 26-28, 2018, pp. 119–136 (2018).

[2] E. Gunther, M. Pagano, P. Sánchez Terraf, Mechanization of Separation in Generic Extensions, *arXiv e-prints* **1901.03313** (2019).

[3] E. Gunther, M. Pagano, P. Sánchez Terraf, Formalization of Forcing in Isabelle/ZF, *arXiv e-prints* **2001.09715** (2020).

[4] L.C. Paulson, K. Grabczewski, Mechanizing set theory, *J. Autom. Reasoning* **17**: 291–323 (1996).