

Formalization of Forcing in Isabelle/ZF

Emmanuel Gunther* Miguel Pagano* Pedro Sánchez Terraf*†

April 10, 2026

Abstract

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

Contents

1	Introduction	4
2	Forcing notions	4
2.1	Basic concepts	4
2.2	Towards Rasiowa-Sikorski Lemma (RSL)	9
3	A pointed version of DC	10
4	The general Rasiowa-Sikorski lemma	12
5	Auxiliary results on arithmetic	12
5.1	Some results in ordinal arithmetic	15
6	Aids to internalize formulas	16
7	Some enhanced theorems on recursion	17
8	Relativization of the cumulative hierarchy	19
8.1	Formula synthesis	20
8.2	Absoluteness results	21
9	Automatic synthesis of formulas	24

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

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

10	Interface between set models and Constructibility	24
10.1	Interface with $M_trivial$	26
10.2	Interface with M_basic	27
10.3	Interface with M_trancl	31
10.4	Interface with M_eclose	33
11	Transitive set models of ZF	38
11.1	$Collects$ in M	41
11.2	A forcing locale and generic filters	42
12	The ZFC axioms, internalized	43
12.1	The Axiom of Separation, internalized	45
12.2	The Axiom of Replacement, internalized	46
13	Renaming of variables in internalized formulas	49
13.1	Renaming of free variables	50
13.2	Renaming of formulas	53
14	Names and generic extensions	55
14.1	The well-founded relation ed	55
14.2	Values and check-names	58
15	Well-founded relation on names	65
16	Arities of internalized formulas	74
17	The definition of $forces$	79
17.1	The relation $frecrel$	79
17.2	Definition of $forces$ for equality and membership	82
17.3	The well-founded relation $forcerec$	86
17.4	frc_at , forcing for atomic formulas	87
17.5	Recursive expression of frc_at	95
17.6	Absoluteness of frc_at	95
17.7	Forcing for general formulas	97
17.7.1	The primitive recursion	98
17.8	Forcing for atomic formulas in context	99
17.9	The arity of $forces$	101
18	The Forcing Theorems	101
18.1	The forcing relation in context	101
18.2	Kunen 2013, Lemma IV.2.37(a)	102
18.3	Kunen 2013, Lemma IV.2.37(a)	102
18.4	Kunen 2013, Lemma IV.2.37(b)	102
18.5	Kunen 2013, Lemma IV.2.38	103
18.6	The relation of forcing and atomic formulas	103

18.7	The relation of forcing and connectives	104
18.8	Kunen 2013, Lemma IV.2.29	105
18.9	Auxiliary results for Lemma IV.2.40(a)	105
18.10	Induction on names	106
18.11	Lemma IV.2.40(a), in full	107
18.12	Lemma IV.2.40(b)	107
18.13	The Strenghtening Lemma	109
18.14	The Density Lemma	109
18.15	The Truth Lemma	109
18.16	The “Definition of forcing”	111
19	Auxiliary renamings for Separation	111
20	The Axiom of Separation in $M[G]$	114
21	The Axiom of Pairing in $M[G]$	115
22	The Axiom of Unions in $M[G]$	115
23	The Powerset Axiom in $M[G]$	117
24	The Axiom of Extensionality in $M[G]$	118
25	The Axiom of Foundation in $M[G]$	118
26	The binder <i>Least</i>	118
26.1	Absoluteness and closure under <i>Least</i>	120
27	The Axiom of Replacement in $M[G]$	120
28	The Axiom of Infinity in $M[G]$	125
29	The Axiom of Choice in $M[G]$	125
29.1	$M[G]$ is a transitive model of ZF	127
30	Ordinals in generic extensions	128
31	Separative notions and proper extensions	129
32	A poset of successions	130
32.1	The set of finite binary sequences	130
32.2	Cohen extension is proper	133
33	The main theorem	134
33.1	The generic extension is countable	134
33.2	The main result	134

1 Introduction

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

Release notes

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

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

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

2 Forcing notions

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

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

2.1 Basic concepts

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

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

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

lemma *compat_inI* :

$\llbracket d \in A ; \langle d, p \rangle \in r ; \langle d, g \rangle \in r \rrbracket \implies \text{compat_in}(A, r, p, g)$
<proof>

lemma *refl_compat*:

$\llbracket \text{refl}(A, r) ; \langle p, q \rangle \in r \mid p = q \mid \langle q, p \rangle \in r ; p \in A ; q \in A \rrbracket \implies \text{compat_in}(A, r, p, q)$
<proof>

lemma *chain_compat*:

$\text{refl}(A, r) \implies \text{linear}(A, r) \implies (\forall p \in A. \forall q \in A. \text{compat_in}(A, r, p, q))$
<proof>

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

<proof>

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

<proof>

definition

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

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

definition

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

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

locale *forcing_notion* =

fixes P *leq one*

assumes *one_in_P*: $one \in P$

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

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

begin

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

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

lemma *refl_leq*:

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

<proof>

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

definition

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

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

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

definition

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

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

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

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

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

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

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

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

lemma *dense_belowD [dest]*:
assumes $\text{dense_below}(D, p)$ $q \in P$ $q \preceq p$
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 $\text{dense_below}(D, p)$

<proof>

lemma *dense_below_cong*: $p \in P \implies D = D' \implies \text{dense_below}(D, p) \longleftrightarrow \text{dense_below}(D', p)$
<proof>

lemma *dense_below_cong'*: $p \in P \implies [\![\bigwedge x. x \in P \implies Q(x) \longleftrightarrow Q'(x)]\!] \implies$
 $\text{dense_below}(\{q \in P. Q(q)\}, p) \longleftrightarrow \text{dense_below}(\{q \in P. Q'(q)\}, p)$
<proof>

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

lemma *dense_below_under*:
assumes $\text{dense_below}(D, p)$ $p \in P$ $q \in P$ $q \preceq p$
shows $\text{dense_below}(D, q)$
<proof>

lemma *ideal_dense_below*:
assumes $\bigwedge q. q \in P \implies q \preceq p \implies q \in D$
shows $\text{dense_below}(D, p)$
<proof>

lemma *dense_below_dense_below*:
assumes $\text{dense_below}(\{q \in P. \text{dense_below}(D, q)\}, p)$ $p \in P$
shows $\text{dense_below}(D, p)$
<proof>

definition

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

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

definition

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

lemma *filterD* : $\text{filter}(G) \implies x \in G \implies x \in P$
<proof>

lemma *filter_leqD* : $\text{filter}(G) \implies x \in G \implies y \in P \implies x \preceq y \implies y \in G$
<proof>

lemma *filter_imp_compat*: $\text{filter}(G) \implies p \in G \implies q \in G \implies \text{compat}(p, q)$
<proof>

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

shows $\exists r \in G. r \preceq p \wedge r \preceq q$
 ⟨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] : $p \in P \Rightarrow a \in A \Rightarrow a \preceq p \Rightarrow p \in upclosure(A)$
 ⟨proof⟩

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

lemma $upclosureD$ [dest] :
 $p \in upclosure(A) \Rightarrow \exists a \in A. (a \preceq p) \wedge p \in P$
 ⟨proof⟩

lemma $upclosure_increasing$:
assumes $A \subseteq P$
shows $increasing(upclosure(A))$
 ⟨proof⟩

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

lemma $A_sub_upclosure$: $A \subseteq P \Rightarrow A \subseteq upclosure(A)$
 ⟨proof⟩

lemma $elem_upclosure$: $A \subseteq P \Rightarrow x \in A \Rightarrow x \in upclosure(A)$
 ⟨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 : $f \in N \rightarrow P \Rightarrow n \in N \Rightarrow f^n \in upclosure(f^{\text{``}N})$
 ⟨proof⟩

lemma $decr_succ_decr$:
assumes $f \in nat \rightarrow P$ $preorder_on(P, leq)$
 $\forall n \in nat. \langle f^{\text{'}} succ(n), f^{\text{'}} n \rangle \in leq$
 $m \in nat$
shows $n \in nat \Rightarrow n \leq m \Rightarrow \langle f^{\text{'}} m, f^{\text{'}} n \rangle \in leq$
 ⟨proof⟩

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

end

2.2 Towards Rasiowa-Sikorski Lemma (RSL)

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

begin

definition

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

The next lemma identifies a sufficient condition for obtaining RSL.

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

end

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

RS_sequence_imp_rasiowa_sikorski

consts *RS_seq* :: $[i, i, i, i, i, i] \Rightarrow i$

primrec

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

context *countable_generic*

begin

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

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

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 $n \in nat$ $p \in P$ $P \subseteq range(enum)$ $enum: nat \rightarrow M$
shows
 $f\ '0 = p$ $f\ 'succ(n) \preceq f\ 'n \wedge f\ 'succ(n) \in \mathcal{D} \ 'n$ $f\ 'succ(n) \in P$
 $\langle proof \rangle$

lemma *RS_seq_type*:
assumes $n \in nat$ $p \in P$ $P \subseteq range(enum)$ $enum: nat \rightarrow M$
shows $RS_seq(n, P, leq, p, enum, \mathcal{D}) \in P$
 $\langle proof \rangle$

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

lemmas *countable_rasiowa_sikorski* =
 $RS_sequence_imp_rasiowa_sikorski[OF_RS_seq_funtype\ countable_RS_sequence(1,2)]$
end

end

3 A pointed version of DC

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

begin

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

consts $dc_witness :: i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow i$

primrec

$wit0 : dc_witness(0, A, a, s, R) = a$

$witrec : dc_witness(succ(n), A, a, s, R) = s\{x \in A. \langle dc_witness(n, A, a, s, R), x \rangle \in R\}$

lemma *witness_into_A* [TC]:

assumes $a \in A$

$(\forall X . X \neq 0 \wedge X \subseteq A \longrightarrow s'X \in X)$
 $\forall y \in A. \{x \in A. \langle y, x \rangle \in R\} \neq 0 \ n \in \text{nat}$
shows $dc_witness(n, A, a, s, R) \in A$
 $\langle \text{proof} \rangle$

lemma *witness_related* :

assumes $a \in A$
 $(\forall X . X \neq 0 \wedge X \subseteq A \longrightarrow s'X \in X)$
 $\forall y \in A. \{x \in A. \langle y, x \rangle \in R\} \neq 0 \ n \in \text{nat}$
shows $\langle dc_witness(n, A, a, s, R), dc_witness(\text{succ}(n), A, a, s, R) \rangle \in R$
 $\langle \text{proof} \rangle$

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 $(\lambda n \in \text{nat}. dc_witness(n, A, a, s, R)) \in \text{nat} \rightarrow A$ (**is** $?f \in _ \rightarrow _$)
 $\langle \text{proof} \rangle$

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 $\exists f \in \text{nat} \rightarrow A. \forall n \in \text{nat}. f'n = dc_witness(n, A, a, s, R)$
 $\langle \text{proof} \rangle$

theorem *pointed_DC* :

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

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

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

corollary *DC_on_A_x_nat* :

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

lemma *aux_sequence_DC* :

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

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

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

end

4 The general Rasiowa-Sikorski lemma

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

context *countable_generic*
begin

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

lemma *DC_imp_RS_sequence*:
assumes $p \in P$
shows $\exists f. f: \text{nat} \rightarrow P \wedge f'0 = p \wedge$
 $(\forall n \in \text{nat}. f'\text{succ}(n) \preceq f'n \wedge f'\text{succ}(n) \in \mathcal{D}'n)$
 $\langle \text{proof} \rangle$

theorem *rasiowa_sikorski*:
 $p \in P \implies \exists G. p \in G \wedge D_generic(G)$
 $\langle \text{proof} \rangle$

end

end

5 Auxiliary results on arithmetic

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

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

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

lemma *nat_succD* : $m \in \text{nat} \implies \text{succ}(n) \in \text{succ}(m) \implies n \in m$
 $\langle \text{proof} \rangle$

lemmas $zero_in = ltD [OF nat_0_le]$

lemma $in_n_in_nat : m \in nat \implies n \in m \implies n \in nat$
(proof)

lemma $in_succ_in_nat : m \in nat \implies n \in succ(m) \implies n \in nat$
(proof)

lemma $ltI_neg : x \in nat \implies j \leq x \implies j \neq x \implies j < x$
(proof)

lemma $succ_pred_eq : m \in nat \implies m \neq 0 \implies succ(pred(m)) = m$
(proof)

lemma $succ_ltI : succ(j) < n \implies j < n$
(proof)

lemma $succ_In : n \in nat \implies succ(j) \in n \implies j \in n$
(proof)

lemmas $succ_leD = succ_leE [OF leI]$

lemma $succpred_leI : n \in nat \implies n \leq succ(pred(n))$
(proof)

lemma $succpred_n0 : succ(n) \in p \implies p \neq 0$
(proof)

lemma $funcI : f \in A \rightarrow B \implies a \in A \implies b = f \ ' \ a \implies \langle a, b \rangle \in f$
(proof)

lemmas $natEin = natE [OF lt_nat_in_nat]$

lemma $succ_in : succ(x) \leq y \implies x \in y$
(proof)

lemmas $Un_least_lt_iffn = Un_least_lt_iff [OF nat_into_Ord nat_into_Ord]$

lemma $pred_le2 : n \in nat \implies m \in nat \implies pred(n) \leq m \implies n \leq succ(m)$
(proof)

lemma $pred_le : n \in nat \implies m \in nat \implies n \leq succ(m) \implies pred(n) \leq m$
(proof)

lemma $Un_leD1 : Ord(i) \implies Ord(j) \implies Ord(k) \implies i \cup j \leq k \implies i \leq k$
(proof)

lemma $Un_leD2 : Ord(i) \implies Ord(j) \implies Ord(k) \implies i \cup j \leq k \implies j \leq k$

<proof>

lemma *gt1* : $n \in \text{nat} \implies i \in n \implies i \neq 0 \implies i \neq 1 \implies 1 < i$
<proof>

lemma *pred_mono* : $m \in \text{nat} \implies n \leq m \implies \text{pred}(n) \leq \text{pred}(m)$
<proof>

lemma *succ_mono* : $m \in \text{nat} \implies n \leq m \implies \text{succ}(n) \leq \text{succ}(m)$
<proof>

lemma *pred2_Un*:
 assumes $j \in \text{nat} \ m \leq j \ n \leq j$
 shows $\text{pred}(\text{pred}(m \cup n)) \leq \text{pred}(\text{pred}(j))$
<proof>

lemma *nat_union_abs1* :
 $\llbracket \text{Ord}(i) ; \text{Ord}(j) ; i \leq j \rrbracket \implies i \cup j = j$
<proof>

lemma *nat_union_abs2* :
 $\llbracket \text{Ord}(i) ; \text{Ord}(j) ; i \leq j \rrbracket \implies j \cup i = j$
<proof>

lemma *nat_un_max* : $\text{Ord}(i) \implies \text{Ord}(j) \implies i \cup j = \text{max}(i,j)$
<proof>

lemma *nat_max_ty* : $\text{Ord}(i) \implies \text{Ord}(j) \implies \text{Ord}(\text{max}(i,j))$
<proof>

lemma *le_not_lt_nat* : $\text{Ord}(p) \implies \text{Ord}(q) \implies \neg p \leq q \implies q \leq p$
<proof>

lemmas *nat_simp_union* = *nat_un_max nat_max_ty max_def*

lemma *le_succ* : $x \in \text{nat} \implies x \leq \text{succ}(x)$ *<proof>*

lemma *le_pred* : $x \in \text{nat} \implies \text{pred}(x) \leq x$
<proof>

lemma *Un_le_compat* : $o \leq p \implies q \leq r \implies \text{Ord}(o) \implies \text{Ord}(p) \implies \text{Ord}(q) \implies \text{Ord}(r) \implies o \cup q \leq p \cup r$
<proof>

lemma *Un_le* : $p \leq r \implies q \leq r \implies \text{Ord}(p) \implies \text{Ord}(q) \implies \text{Ord}(r) \implies p \cup q \leq r$
<proof>

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

$Ord(o) \implies Ord(p) \implies Ord(q) \implies Ord(r) \implies$
 $o \cup p \cup q \leq r$

<proof>

lemma *diff_mono* :
assumes $m \in nat$ $n \in nat$ $p \in nat$ $m < n$ $p \leq m$
shows $m \# - p < n \# - p$
<proof>

lemma *pred_Un*:
 $x \in nat \implies y \in nat \implies Arith.pred(succ(x) \cup y) = x \cup Arith.pred(y)$
 $x \in nat \implies y \in nat \implies Arith.pred(x \cup succ(y)) = Arith.pred(x) \cup y$
<proof>

lemma *le_natI* : $j \leq n \implies n \in nat \implies j \in nat$
<proof>

lemma *le_natE* : $n \in nat \implies j < n \implies j \in n$
<proof>

lemma *diff_cancel* :
assumes $m \in nat$ $n \in nat$ $m < n$
shows $m \# - n = 0$
<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 *freqR*

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 $w = y \vee w = z \ x < y \ \text{Ord}(x) \ \text{Ord}(w) \ \text{Ord}(y) \ \text{Ord}(z) \ \text{max}(x,w) = \text{max}(y,z)$

shows $x < w$
 ⟨proof⟩

lemma *oadd_lt_mono2* :

assumes $\text{Ord}(n) \ \text{Ord}(\alpha) \ \text{Ord}(\beta) \ \alpha < \beta \ x < n \ y < n \ 0 < n$

shows $n ** \alpha ++ x < n ** \beta ++ y$
 ⟨proof⟩

end

6 Aids to internalize formulas

theory *Internalizations*

imports

ZF-Constructible.DPow_absolute

begin

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

lemma *nth_closed* :

assumes $0 \in A \ \text{env} \in \text{list}(A)$

shows $\text{nth}(n, \text{env}) \in A$

⟨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*: $\llbracket \text{nth}(n, l) = x; n \in \text{nat} \rrbracket \implies \text{nth}(\text{succ}(n), \text{Cons}(a, l)) = x$

⟨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 (*termsep_rules*) used in formula synthesis

lemmas *fm_defs = omega_fm_def limit_ordinal_fm_def empty_fm_def typed_function_fm_def*

pair_fm_def upair_fm_def domain_fm_def function_fm_def

succ_fm_def

cons_fm_def fun_apply_fm_def image_fm_def big_union_fm_def

union_fm_def

$relation_fm_def$ $composition_fm_def$ $field_fm_def$ $ordinal_fm_def$
 $range_fm_def$
 $transset_fm_def$ $subset_fm_def$ $Replace_fm_def$

end

7 Some enhanced theorems on recursion

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

We prove results concerning definitions by well-founded recursion on some relation R and its transitive closure $R^{\hat{+}}$

lemma *fld_restrict_eq* : $a \in A \implies (r \cap A \times A)^{\hat{-}}\{a\} = (r^{\hat{-}}\{a\} \cap A)$
 $\langle proof \rangle$

lemma *fld_restrict_mono* : $relation(r) \implies A \subseteq B \implies r \cap A \times A \subseteq r \cap B \times B$
 $\langle proof \rangle$

lemma *fld_restrict_dom* :
assumes $relation(r)$ $domain(r) \subseteq A$ $range(r) \subseteq A$
shows $r \cap A \times A = r$
 $\langle proof \rangle$

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

lemma *tr_downD* : $x \in tr_down(r,a) \implies \langle x,a \rangle \in r^{\hat{+}}$
 $\langle proof \rangle$

lemma *pred_down* : $relation(r) \implies r^{\hat{-}}\{a\} \subseteq tr_down(r,a)$
 $\langle proof \rangle$

lemma *tr_down_mono* : $relation(r) \implies x \in r^{\hat{-}}\{a\} \implies tr_down(r,x) \subseteq tr_down(r,a)$
 $\langle proof \rangle$

lemma *rest_eq* :
assumes $relation(r)$ **and** $r^{\hat{-}}\{a\} \subseteq B$ **and** $a \in B$
shows $r^{\hat{-}}\{a\} = (r \cap B \times B)^{\hat{-}}\{a\}$
 $\langle proof \rangle$

lemma *wfrec_restr_eq* : $r' = r \cap A \times A \implies wfrec[A](r,a,H) = wfrec(r',a,H)$
 $\langle proof \rangle$

lemma *wfrec_restr* :
assumes $rr: relation(r)$ **and** $wfr: wf(r)$
shows $a \in A \implies tr_down(r,a) \subseteq A \implies wfrec(r,a,H) = wfrec[A](r,a,H)$
 $\langle proof \rangle$

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

lemma *wfrec_trans_restr* : *relation*(*r*) \implies *wf*(*r*) \implies *trans*(*r*) \implies *r*-“{*a*} \subseteq *A* \implies
a \in *A* \implies
wfrec(*r*, *a*, *H*) = *wfrec*[*A*](*r*, *a*, *H*)
 ⟨*proof*⟩

lemma *field_trancl* : *field*(*r*⁺) = *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* = ⟨*x*, *y*⟩ \wedge *R*(*x*,*y*)}

lemma *RrelI* : *x* \in *A* \implies *y* \in *A* \implies *R*(*x*,*y*) \implies ⟨*x*,*y*⟩ \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* \implies *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*)-“{*y*}”)’*w*
 = *restrict*(*f*,(*Rrel*(*R*,*d*)⁺)-“{*y*}”)’*w*

⟨*proof*⟩

lemma *restrict_trans_eq*:

assumes *w* \in *y*

shows *restrict*(*f*,*Memrel*(*eclose*({*x*})-“{*y*}”)’*w*
 = *restrict*(*f*,(*Memrel*(*eclose*({*x*})⁺)-“{*y*}”)’*w*

⟨*proof*⟩

lemma *wf_eq_trancl*:
assumes $\bigwedge f y . H(y, \text{restrict}(f, R \cdot \{\!-\!\} y)) = H(y, \text{restrict}(f, R^{\wedge+} \cdot \{\!-\!\} y))$
shows $\text{wfrec}(R, x, H) = \text{wfrec}(R^{\wedge+}, x, H)$ (**is** $\text{wfrec}(?r, _, _) = \text{wfrec}(?r', _, _)$)
 $\langle \text{proof} \rangle$

end

8 Relativization of the cumulative hierarchy

theory *Relative_Univ*

imports
ZF-Constructible.Rank
Internalizations
Recursion_Thms

begin

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

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

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

definition

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

definition

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

lemma *is_powapply_closed*: $is_powapply(M, f, y, z) \implies M(z)$
 ⟨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) $\equiv \exists z[M]. empty(M, z) \wedge is_Vfrom(M, z, i, V)$

8.1 Formula synthesis

schematic_goal *sats_is_powapply_fm_auto*:

assumes

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

shows

$is_powapply(\#\#A, nth(f, env), nth(y, env), nth(z, env))$
 $\longleftrightarrow sats(A, ?ipa_fm(f, y, z), env)$

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

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

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

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

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

8.2 Absoluteness results

locale $M_eclose_pow = M_eclose +$
assumes

$power_ax : power_ax(M)$ **and**
 $powapply_replacement : M(f) \Longrightarrow strong_replacement(M, is_powapply(M, f))$

and

$HVfrom_replacement : \llbracket M(i) ; M(A) \rrbracket \Longrightarrow$
 $transrec_replacement(M, is_HVfrom(M, A), i)$ **and**
 $PHrank_replacement : M(f) \Longrightarrow strong_replacement(M, PHrank(M, f))$ **and**
 $is_Hrank_replacement : M(x) \Longrightarrow wfrec_replacement(M, is_Hrank(M), rrank(x))$

begin

lemma $is_powapply_abs : \llbracket M(f) ; M(y) \rrbracket \Longrightarrow is_powapply(M, f, y, z) \longleftrightarrow M(z) \wedge$
 $z = \{x \in Pow(f'y). M(x)\}$
 $\langle proof \rangle$

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

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 $powapply_closed :$

$\llbracket M(y) ; M(f) \rrbracket \implies M(\{x \in \text{Pow}(f' y) . M(x)\})$
 $\langle \text{proof} \rangle$

lemma *RepFun_is_powapply*:

assumes

$M(R) \ M(A) \ M(f)$

shows

$\text{Replace}(A, \text{is_powapply}(M, f)) = \text{RepFun}(A, \lambda y. \{x \in \text{Pow}(f' y) . M(x)\})$
 $\langle \text{proof} \rangle$

lemma *RepFun_powapply_closed*:

assumes

$M(f) \ M(A)$

shows

$M(\text{Replace}(A, \text{is_powapply}(M, f)))$
 $\langle \text{proof} \rangle$

lemma *Union_powapply_closed*:

assumes

$M(x) \ M(f)$

shows

$M(\bigcup y \in x. \{a \in \text{Pow}(f' y) . M(a)\})$
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *HVfrom_closed* :

$M(A) \implies \forall x[M]. \forall g[M]. \text{function}(g) \longrightarrow M(\text{HVfrom}(M, A, x, g))$

$\langle \text{proof} \rangle$

lemma *transrec_HVfrom*:

assumes $M(A)$

shows $\text{Ord}(i) \implies \{x \in \text{Vfrom}(A, i) . M(x)\} = \text{transrec}(i, \text{HVfrom}(M, A))$

$\langle \text{proof} \rangle$

lemma *Vfrom_abs*: $\llbracket M(A) ; M(i) ; M(V) ; \text{Ord}(i) \rrbracket \implies \text{is_Vfrom}(M, A, i, V) \longleftrightarrow$

$V = \{x \in \text{Vfrom}(A, i) . M(x)\}$

$\langle \text{proof} \rangle$

lemma *Vfrom_closed*: $\llbracket M(A) ; M(i) ; \text{Ord}(i) \rrbracket \implies M(\{x \in \text{Vfrom}(A, i) . M(x)\})$

$\langle \text{proof} \rangle$

lemma *Vset_abs*: $\llbracket M(i) ; M(V) ; \text{Ord}(i) \rrbracket \implies \text{is_Vset}(M, i, V) \longleftrightarrow V = \{x \in \text{Vset}(i) .$

$M(x)\}$

$\langle \text{proof} \rangle$

lemma *Vset_closed*: $\llbracket M(i) ; \text{Ord}(i) \rrbracket \implies M(\{x \in \text{Vset}(i) . M(x)\})$

$\langle \text{proof} \rangle$

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

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

lemma *univ_PHrank* : $\llbracket M(z) ; M(f) \rrbracket \implies univalent(M, z, PHrank(M, f))$
 ⟨*proof*⟩

lemma *PHrank_abs* :
 $\llbracket M(f) ; M(y) \rrbracket \implies PHrank(M, f, y, z) \longleftrightarrow M(z) \wedge z = succ(f'y)$
 ⟨*proof*⟩

lemma *PHrank_closed* : $PHrank(M, f, y, z) \implies M(z)$
 ⟨*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

$M(\bigcup y \in x. \text{succ}(f'y))$
 $\langle \text{proof} \rangle$

lemma *is_Hrank_closed* :
 $M(A) \implies \forall x[M]. \forall g[M]. \text{function}(g) \longrightarrow M(\text{Hrank}(x,g))$
 $\langle \text{proof} \rangle$

lemma *rank_closed*: $M(a) \implies M(\text{rank}(a))$
 $\langle \text{proof} \rangle$

lemma *M_into_Vset*:
assumes $M(a)$
shows $\exists i[M]. \exists V[M]. \text{ordinal}(M,i) \wedge \text{is_Vfrom}(M,\theta,i,V) \wedge a \in V$
 $\langle \text{proof} \rangle$

end
end

9 Automatic synthesis of formulas

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

$\langle \text{ML} \rangle$

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

end

10 Interface between set models and Constructibility

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

theory *Interface*
imports
Nat_Miscellanea
Relative_Univ
Synthetic_Definition

begin

syntax

$_sats :: [i, i, i] \Rightarrow o \ (\langle _ , _ \models _ \rangle \ [36,36,36] \ 60)$

syntax_consts

$_sats \equiv sats$

translations

$(M, env \models \varphi) \equiv CONST \ sats(M, \varphi, env)$

abbreviation

$dec10 :: i \ (\langle 10 \rangle) \ \mathbf{where} \ 10 \equiv succ(9)$

abbreviation

$dec11 :: i \ (\langle 11 \rangle) \ \mathbf{where} \ 11 \equiv succ(10)$

abbreviation

$dec12 :: i \ (\langle 12 \rangle) \ \mathbf{where} \ 12 \equiv succ(11)$

abbreviation

$dec13 :: i \ (\langle 13 \rangle) \ \mathbf{where} \ 13 \equiv succ(12)$

abbreviation

$dec14 :: i \ (\langle 14 \rangle) \ \mathbf{where} \ 14 \equiv succ(13)$

definition

$infinity_ax :: (i \Rightarrow o) \Rightarrow o \ \mathbf{where}$

$infinity_ax(M) \equiv$

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

definition

$choice_ax :: (i \Rightarrow o) \Rightarrow o \ \mathbf{where}$

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

context M_basic **begin**

lemma $choice_ax_abs :$

$choice_ax(M) \longleftrightarrow (\forall x[M]. \exists a[M]. \exists f[M]. Ord(a) \wedge f \in surj(a, x))$

$\langle proof \rangle$

end

definition

$wellfounded_trancl :: [i=>o, i, i, i] \Rightarrow o \ \mathbf{where}$

$wellfounded_trancl(M, Z, r, p) \equiv$

$\exists w[M]. \exists wx[M]. \exists rp[M].$

$w \in Z \ \& \ pair(M, w, p, wx) \ \& \ tran_closure(M, r, rp) \ \& \ wx \in rp$

lemma *empty_intf* :
 $\text{infinity_ax}(M) \implies$
 $(\exists z[M]. \text{empty}(M,z))$
 $\langle \text{proof} \rangle$

lemma *Transset_intf* :
 $\text{Transset}(M) \implies y \in x \implies x \in M \implies y \in M$
 $\langle \text{proof} \rangle$

locale *M_ZF_trans* =
fixes *M*
assumes

upair_ax: $\text{upair_ax}(\#\#M)$
and *Union_ax*: $\text{Union_ax}(\#\#M)$
and *power_ax*: $\text{power_ax}(\#\#M)$
and *extensionality*: $\text{extensionality}(\#\#M)$
and *foundation_ax*: $\text{foundation_ax}(\#\#M)$
and *infinity_ax*: $\text{infinity_ax}(\#\#M)$
and *separation_ax*: $\varphi \in \text{formula} \implies \text{env} \in \text{list}(M) \implies \text{arity}(\varphi) \leq 1 \ \#\+$
 $\text{length}(\text{env}) \implies$
 $\text{separation}(\#\#M, \lambda x. \text{sats}(M, \varphi, [x] \ @ \ \text{env}))$
and *replacement_ax*: $\varphi \in \text{formula} \implies \text{env} \in \text{list}(M) \implies \text{arity}(\varphi) \leq 2 \ \#\+$
 $\text{length}(\text{env}) \implies$
 $\text{strong_replacement}(\#\#M, \lambda x y. \text{sats}(M, \varphi, [x,y] \ @ \ \text{env}))$
and *trans_M*: $\text{Transset}(M)$
begin

lemma *TranssetI* :
 $(\bigwedge y x. y \in x \implies x \in M \implies y \in M) \implies \text{Transset}(M)$
 $\langle \text{proof} \rangle$

lemma *zero_in_M*: $0 \in M$
 $\langle \text{proof} \rangle$

10.1 Interface with *M_trivial*

lemma *mtrans* :
 $M_trans(\#\#M)$
 $\langle \text{proof} \rangle$

lemma *mtriv* :
 $M_trivial(\#\#M)$
 $\langle \text{proof} \rangle$

end

sublocale *M_ZF_trans* \subseteq *M_trivial* $\#\#M$

$\langle proof \rangle$

context M_ZF_trans
begin

10.2 Interface with M_basic

schematic_goal $inter_fm_auto$:

assumes

$nth(i,env) = x \quad nth(j,env) = B$

$i \in nat \quad j \in nat \quad env \in list(A)$

shows

$(\forall y \in A . y \in B \longrightarrow x \in y) \longleftrightarrow sats(A, ?ifm(i,j), env)$

$\langle proof \rangle$

lemma $inter_sep_intf$:

assumes

$A \in M$

shows

$separation(\#\#M, \lambda x . \forall y \in M . y \in A \longrightarrow x \in y)$

$\langle proof \rangle$

schematic_goal $diff_fm_auto$:

assumes

$nth(i,env) = x \quad nth(j,env) = B$

$i \in nat \quad j \in nat \quad env \in list(A)$

shows

$x \notin B \longleftrightarrow sats(A, ?dfm(i,j), env)$

$\langle proof \rangle$

lemma $diff_sep_intf$:

assumes

$B \in M$

shows

$separation(\#\#M, \lambda x . x \notin B)$

$\langle proof \rangle$

schematic_goal $cprod_fm_auto$:

assumes

$nth(i,env) = z \quad nth(j,env) = B \quad nth(h,env) = C$

$i \in nat \quad j \in nat \quad h \in nat \quad env \in list(A)$

shows

$(\exists x \in A . x \in B \wedge (\exists y \in A . y \in C \wedge pair(\#\#A, x, y, z))) \longleftrightarrow sats(A, ?cpfm(i,j,h), env)$

$\langle proof \rangle$

lemma $cartprod_sep_intf$:

assumes
 $A \in M$
and
 $B \in M$
shows
 $\text{separation}(\#\#M, \lambda z. \exists x \in M. x \in A \wedge (\exists y \in M. y \in B \wedge \text{pair}(\#\#M, x, y, z)))$
 $\langle \text{proof} \rangle$

schematic_goal *im_fm_auto*:

assumes
 $\text{nth}(i, \text{env}) = y \text{ nth}(j, \text{env}) = r \text{ nth}(h, \text{env}) = B$
 $i \in \text{nat } j \in \text{nat } h \in \text{nat } \text{env} \in \text{list}(A)$
shows
 $(\exists p \in A. p \in r \ \& \ (\exists x \in A. x \in B \ \& \ \text{pair}(\#\#A, x, y, p))) \longleftrightarrow \text{sats}(A, ?\text{imfm}(i, j, h), \text{env})$
 $\langle \text{proof} \rangle$

lemma *image_sep_intf* :

assumes
 $A \in M$
and
 $r \in M$
shows
 $\text{separation}(\#\#M, \lambda y. \exists p \in M. p \in r \ \& \ (\exists x \in M. x \in A \ \& \ \text{pair}(\#\#M, x, y, p)))$
 $\langle \text{proof} \rangle$

schematic_goal *con_fm_auto*:

assumes
 $\text{nth}(i, \text{env}) = z \text{ nth}(j, \text{env}) = R$
 $i \in \text{nat } j \in \text{nat } \text{env} \in \text{list}(A)$
shows
 $(\exists p \in A. p \in R \ \& \ (\exists x \in A. \exists y \in A. \text{pair}(\#\#A, x, y, p) \ \& \ \text{pair}(\#\#A, y, x, z)))$
 $\longleftrightarrow \text{sats}(A, ?\text{cfm}(i, j), \text{env})$
 $\langle \text{proof} \rangle$

lemma *converse_sep_intf* :

assumes
 $R \in M$
shows
 $\text{separation}(\#\#M, \lambda z. \exists p \in M. p \in R \ \& \ (\exists x \in M. \exists y \in M. \text{pair}(\#\#M, x, y, p) \ \& \ \text{pair}(\#\#M, y, x, z)))$
 $\langle \text{proof} \rangle$

schematic_goal *rest_fm_auto*:

assumes
 $\text{nth}(i, \text{env}) = z \text{ nth}(j, \text{env}) = C$
 $i \in \text{nat } j \in \text{nat } \text{env} \in \text{list}(A)$
shows

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

lemma *restrict_sep_intf* :

assumes

$A \in M$

shows

$\text{separation}(\#\#M, \lambda z. \exists x \in M. x \in A \ \& \ (\exists y \in M. \text{pair}(\#\#M, x, y, z)))$

$\langle \text{proof} \rangle$

schematic_goal *comp_fm_auto*:

assumes

$\text{nth}(i, env) = xz \ \text{nth}(j, env) = S \ \text{nth}(h, env) = R$

$i \in \text{nat} \ j \in \text{nat} \ h \in \text{nat} \ env \in \text{list}(A)$

shows

$(\exists x \in A. \exists y \in A. \exists z \in A. \exists xy \in A. \exists yz \in A.$

$\text{pair}(\#\#A, x, z, xz) \ \& \ \text{pair}(\#\#A, x, y, xy) \ \& \ \text{pair}(\#\#A, y, z, yz) \ \& \ xy \in S \ \&$

$yz \in R)$

$\longleftrightarrow \text{sats}(A, ?cfm(i, j, h), env)$

$\langle \text{proof} \rangle$

lemma *comp_sep_intf* :

assumes

$R \in M$

and

$S \in M$

shows

$\text{separation}(\#\#M, \lambda xz. \exists x \in M. \exists y \in M. \exists z \in M. \exists xy \in M. \exists yz \in M.$

$\text{pair}(\#\#M, x, z, xz) \ \& \ \text{pair}(\#\#M, x, y, xy) \ \& \ \text{pair}(\#\#M, y, z, yz) \ \& \ xy \in S$

$\ \& \ yz \in R)$

$\langle \text{proof} \rangle$

schematic_goal *pred_fm_auto*:

assumes

$\text{nth}(i, env) = y \ \text{nth}(j, env) = R \ \text{nth}(h, env) = X$

$i \in \text{nat} \ j \in \text{nat} \ h \in \text{nat} \ env \in \text{list}(A)$

shows

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

$\langle \text{proof} \rangle$

lemma *pred_sep_intf*:

assumes

$R \in M$

and

$X \in M$
shows
 $\text{separation}(\#\#M, \lambda y. \exists p \in M. p \in R \ \& \ \text{pair}(\#\#M, y, X, p))$
 $\langle \text{proof} \rangle$

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

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

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

lemma *is_recfun_sep_intf* :
assumes
 $r \in M \ f \in M \ g \in M \ a \in M \ b \in M$
shows
 $\text{separation}(\#\#M, \lambda x. \exists xa \in M. \exists xb \in M. \text{pair}(\#\#M, x, a, xa) \ \& \ xa \in r \ \& \ \text{pair}(\#\#M, x, b, xb) \ \& \ xb \in r \ \& \ (\exists fx \in M. \exists gx \in M. \text{fun_apply}(\#\#M, f, x, fx) \ \& \ \text{fun_apply}(\#\#M, g, x, gx) \ \& \ fx \neq gx))$
 $\langle \text{proof} \rangle$

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

$(\exists f \in A. \exists b \in A. \exists nb \in A. \exists cnbf \in A. \text{pair}(\#\#A, f, b, p) \ \& \ \text{pair}(\#\#A, n, b, nb) \ \& \ \text{is_cons}(\#\#A, nb, f, cnbf) \ \& \ \text{upair}(\#\#A, cnbf, cnbf, z)) \longleftrightarrow \text{sats}(A, ?\text{fsfm}(i, j, h), \text{env})$
 <proof>

lemma *funspace_succ_rep_intf* :

assumes

$n \in M$

shows

strong_replacement($\#\#M$,

$\lambda p z. \exists f \in M. \exists b \in M. \exists nb \in M. \exists cnbf \in M.$

$\text{pair}(\#\#M, f, b, p) \ \& \ \text{pair}(\#\#M, n, b, nb) \ \& \ \text{is_cons}(\#\#M, nb, f, cnbf)$

&

$\text{upair}(\#\#M, cnbf, cnbf, z)$)

<proof>

lemmas *M_basic_sep_instances* =

inter_sep_intf *diff_sep_intf* *cartprod_sep_intf*

image_sep_intf *converse_sep_intf* *restrict_sep_intf*

pred_sep_intf *memrel_sep_intf* *comp_sep_intf* *is_recfun_sep_intf*

lemma *mbasic* : $M_basic(\#\#M)$

<proof>

end

sublocale $M_ZF_trans \subseteq M_basic \ \#\#M$

<proof>

10.3 Interface with *M_trancl*

schematic_goal *rtran_closure_mem_auto*:

assumes

$\text{nth}(i, \text{env}) = p \ \text{nth}(j, \text{env}) = r \ \text{nth}(k, \text{env}) = B$

$i \in \text{nat} \ j \in \text{nat} \ k \in \text{nat} \ \text{env} \in \text{list}(A)$

shows

$\text{rtran_closure_mem}(\#\#A, B, r, p) \longleftrightarrow \text{sats}(A, ?\text{rcfm}(i, j, k), \text{env})$

<proof>

lemma (in M_ZF_trans) *rtrancl_separation_intf*:

assumes

$r \in M$

and

$A \in M$

shows
separation ($\#\#M, rtran_closure_mem(\#\#M, A, r)$)
 $\langle proof \rangle$

schematic_goal *rtran_closure_fm_auto*:

assumes

$nth(i, env) = r \ nth(j, env) = rp$

$i \in nat \ j \in nat \ env \in list(A)$

shows

$rtran_closure(\#\#A, r, rp) \longleftrightarrow sats(A, ?rtc(i, j), env)$

$\langle proof \rangle$

schematic_goal *trans_closure_fm_auto*:

assumes

$nth(i, env) = r \ nth(j, env) = rp$

$i \in nat \ j \in nat \ env \in list(A)$

shows

$trans_closure(\#\#A, r, rp) \longleftrightarrow sats(A, ?tc(i, j), env)$

$\langle proof \rangle$

$\langle ML \rangle$

schematic_goal *wellfounded_trancl_fm_auto*:

assumes

$nth(i, env) = p \ nth(j, env) = r \ nth(k, env) = B$

$i \in nat \ j \in nat \ k \in nat \ env \in list(A)$

shows

$wellfounded_trancl(\#\#A, B, r, p) \longleftrightarrow sats(A, ?wtf(i, j, k), env)$

$\langle proof \rangle$

lemma (**in** *M_ZF_trans*) *wftrancl_separation_intf*:

assumes

$r \in M$

and

$Z \in M$

shows

$separation(\#\#M, wellfounded_trancl(\#\#M, Z, r))$

$\langle proof \rangle$

lemma (**in** *M_ZF_trans*) *finite_sep_intf*:

$separation(\#\#M, \lambda x. x \in nat)$

$\langle proof \rangle$

lemma (**in** *M_ZF_trans*) *nat_subset_I'*:

$\llbracket I \in M ; 0 \in I ; \bigwedge x. x \in I \implies succ(x) \in I \rrbracket \implies nat \subseteq I$

$\langle proof \rangle$

lemma (in M_ZF_trans) nat_subset_I :
 $\exists I \in M. nat \subseteq I$
 <proof>

lemma (in M_ZF_trans) nat_in_M :
 $nat \in M$
 <proof>

lemma (in M_ZF_trans) $mtrancl$: $M_trancl(\#\#M)$
 <proof>

sublocale $M_ZF_trans \subseteq M_trancl \#\#M$
 <proof>

10.4 Interface with M_eclose

lemma $repl_sats$:
assumes
 $sat: \bigwedge x z. x \in M \implies z \in M \implies sats(M, \varphi, Cons(x, Cons(z, env))) \longleftrightarrow P(x, z)$
shows
 $strong_replacement(\#\#M, \lambda x z. sats(M, \varphi, Cons(x, Cons(z, env)))) \longleftrightarrow$
 $strong_replacement(\#\#M, P)$
 <proof>

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

lemma (in M_ZF_trans) $list_repl1_intf$:
assumes
 $A \in M$
shows
 $iterates_replacement(\#\#M, is_list_functor(\#\#M, A), 0)$
 <proof>

lemma (in M_ZF_trans) $iterates_repl_intf$:
assumes
 $v \in M$ **and**
 $isfm: is_F_fm \in formula$ **and**
 $arty: arity(is_F_fm) = 2$ **and**
 $satsf: \bigwedge a b env'. \llbracket a \in M ; b \in M ; env' \in list(M) \rrbracket$
 $\implies is_F(a, b) \longleftrightarrow sats(M, is_F_fm, [b, a]@env')$

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
 $A \in M$
 shows
 $strong_replacement(\#\#M, \lambda n y. n \in nat \ \& \ is_iterates(\#\#M, is_list_functor(\#\#M, A),$
 $0, n, y))$
(proof)

lemma (in M_ZF_trans) $formula_repl2_intf$:
 $strong_replacement(\#\#M, \lambda n y. n \in nat \ \& \ is_iterates(\#\#M, is_formula_functor(\#\#M),$
 $0, n, y))$
(proof)

lemma (in M_ZF_trans) $eclose_repl2_intf$:
 assumes
 $A \in M$
 shows
 $strong_replacement(\#\#M, \lambda n y. n \in nat \ \& \ is_iterates(\#\#M, big_union(\#\#M),$
 $A, n, y))$
(proof)

lemma (in M_ZF_trans) $mdatatypes : M_datatypes(\#\#M)$

<proof>

sublocale $M_ZF_trans \subseteq M_datatypes \#\#M$
<proof>

lemma (in M_ZF_trans) $meclose : M_eclose(\#\#M)$
<proof>

sublocale $M_ZF_trans \subseteq M_eclose \#\#M$
<proof>

definition

$powerset_fm :: [i,i] \Rightarrow i$ **where**
 $powerset_fm(A,z) \equiv Forall(Iff(Member(0,succ(z)),subset_fm(0,succ(A))))$

lemma $powerset_type [TC]$:
 $\llbracket x \in nat; y \in nat \rrbracket \Longrightarrow powerset_fm(x,y) \in formula$
<proof>

definition

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

lemma $is_powapply_type [TC]$:
 $\llbracket f \in nat ; y \in nat; z \in nat \rrbracket \Longrightarrow is_powapply_fm(f,y,z) \in formula$
<proof>

lemma $sats_is_powapply_fm :$

assumes

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

shows

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

<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**
 $PHrank_fm(f,y,z) \equiv \text{Exists}(\text{And}(\text{fun_apply_fm}(\text{succ}(f),\text{succ}(y),0)$
 $\quad, \text{succ_fm}(0,\text{succ}(z))))$

lemma $PHrank_type$ [TC]:

$\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \Longrightarrow PHrank_fm(x,y,z) \in \text{formula}$
 $\langle \text{proof} \rangle$

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

$\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; env \in \text{list}(M) \rrbracket$
 $\Longrightarrow \text{sats}(M, PHrank_fm(x,y,z), env) \longleftrightarrow$
 $PHrank(\#\#M, nth(x, env), nth(y, env), nth(z, env))$
 $\langle \text{proof} \rangle$

lemma (in M_ZF_trans) $phrank_repl$:**assumes** $f \in M$ **shows** $\text{strong_replacement}(\#\#M, PHrank(\#\#M, f))$ $\langle \text{proof} \rangle$ **definition**

$is_Hrank_fm :: [i,i,i] \Rightarrow i$ **where**
 $is_Hrank_fm(x,f,hc) \equiv \text{Exists}(\text{And}(\text{big_union_fm}(0,\text{succ}(hc)),$
 $\quad \text{Replace_fm}(\text{succ}(x), PHrank_fm(\text{succ}(\text{succ}(\text{succ}(f))), 0, 1), 0)))$

lemma is_Hrank_type [TC]:

$\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \Longrightarrow is_Hrank_fm(x,y,z) \in \text{formula}$
 $\langle \text{proof} \rangle$

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

$\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; env \in \text{list}(M) \rrbracket$
 $\Longrightarrow \text{sats}(M, is_Hrank_fm(x,y,z), env) \longleftrightarrow$
 $is_Hrank(\#\#M, nth(x, env), nth(y, env), nth(z, env))$
 $\langle \text{proof} \rangle$

lemma (in M_ZF_trans) $wfrec_rank$:**assumes** $X \in M$ **shows** $\text{wfrec_replacement}(\#\#M, is_Hrank(\#\#M), \text{rrank}(X))$

<proof>

definition

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

lemma is_HVfrom_type [TC]:

$\llbracket A \in nat; x \in nat; f \in nat; h \in nat \rrbracket \Longrightarrow is_HVfrom_fm(A,x,f,h) \in formula$
<proof>

lemma $sats_is_HVfrom_fm$:

$\llbracket a \in nat; x \in nat; f \in nat; h \in nat; env \in list(A); 0 \in A \rrbracket$
 $\Longrightarrow sats(A, is_HVfrom_fm(a,x,f,h), env) \longleftrightarrow$
 $is_HVfrom(\#\#A, nth(a, env), nth(x, env), nth(f, env), nth(h, env))$
<proof>

lemma $is_HVfrom_iff_sats$:

assumes

$nth(a, env) = aa \ nth(x, env) = xx \ nth(f, env) = ff \ nth(h, env) = hh$
 $a \in nat \ x \in nat \ f \in nat \ h \in nat \ env \in list(A) \ 0 \in A$

shows

$is_HVfrom(\#\#A, aa, xx, ff, hh) \longleftrightarrow sats(A, is_HVfrom_fm(a,x,f,h), env)$
<proof>

schematic_goal $sats_is_Vset_fm_auto$:

assumes

$i \in nat \ v \in nat \ env \in list(A) \ 0 \in A$
 $i < length(env) \ v < length(env)$

shows

$is_Vset(\#\#A, nth(i, env), nth(v, env))$
 $\longleftrightarrow sats(A, ?ivs_fm(i,v), env)$
<proof>

schematic_goal $is_Vset_iff_sats$:

assumes

$nth(i, env) = ii \ nth(v, env) = vv$
 $i \in nat \ v \in nat \ env \in list(A) \ 0 \in A$
 $i < length(env) \ v < length(env)$

shows

$is_Vset(\#\#A, ii, vv) \longleftrightarrow sats(A, ?ivs_fm(i,v), env)$
<proof>

lemma (in M_ZF_trans) $memrel_eclose_sing$:

$a \in M \Longrightarrow \exists sa \in M. \exists esa \in M. \exists mesa \in M.$

$upair(\#\#M, a, a, sa) \ \& \ is_eclose(\#\#M, sa, esa) \ \& \ membership(\#\#M, esa, mesa)$
 <proof>

lemma (in M_ZF_trans) $trans_repl_HVFrom$:
assumes
 $A \in M \ i \in M$
shows
 $transrec_replacement(\#\#M, is_HVfrom(\#\#M, A), i)$
 <proof>

lemma (in M_ZF_trans) $meclase_pow : M_eclose_pow(\#\#M)$
 <proof>

sublocale $M_ZF_trans \subseteq M_eclose_pow \ \#\#M$
 <proof>

lemma (in M_ZF_trans) $repl_gen$:
assumes
 $f_abs: \bigwedge x y. \llbracket x \in M; y \in M \rrbracket \implies is_F(\#\#M, x, y) \longleftrightarrow y = f(x)$
and
 $f_sats: \bigwedge x y. \llbracket x \in M; y \in M \rrbracket \implies$
 $sats(M, f_fm, Cons(x, Cons(y, env))) \longleftrightarrow is_F(\#\#M, x, y)$
and
 $f_form: f_fm \in formula$
and
 $f_arty: arity(f_fm) = 2$
and
 $env \in list(M)$
shows
 $strong_replacement(\#\#M, \lambda x y. y = f(x))$
 <proof>

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

end

11 Transitive set models of ZF

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

theory *Forcing_Data*

imports

Forcing_Notions

Interface

begin

lemma *Transset_M* :

$Transset(M) \implies y \in x \implies x \in M \implies y \in M$
 $\langle proof \rangle$

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 $separation_ax: \varphi \in formula \implies env \in list(M) \implies arity(\varphi) \leq 1 \#+$
 $length(env) \implies$

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

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

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

locale $M_ctm = M_ZF +$

fixes $enum$

assumes $M_countable: enum \in bij(nat, M)$

and $trans_M: Transset(M)$

begin

interpretation $intf: M_ZF_trans M$

$\langle proof \rangle$

lemmas $transitivity = Transset_intf[OF trans_M]$

lemma $zero_in_M: 0 \in M$

$\langle proof \rangle$

lemma $tuples_in_M: A \in M \implies B \in M \implies \langle A, B \rangle \in M$

$\langle proof \rangle$

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

lemma *n_in_M* : $n \in \text{nat} \implies n \in M$
 ⟨*proof*⟩

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

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

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

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

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

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

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

end

sublocale $M_ctm \subseteq M_trivial \#\#M$
 ⟨*proof*⟩

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

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

sublocale $M_ctm \subseteq M_trancl \#\#M$
 ⟨*proof*⟩

sublocale $M_ctm \subseteq M_datatypes \#\#M$
 ⟨*proof*⟩

sublocale $M_ctm \subseteq M_eclose \#\#M$

<proof>

sublocale $M_ctm \subseteq M_eclose_pow \#\#M$
<proof>

context M_ctm
begin

11.1 Collects in M

lemma $Collect_in_M_0p$:

assumes

$Q_fm : Q_fm \in \text{formula}$ **and**

$Q_arty : \text{arity}(Q_fm) = 1$ **and**

$Qsats : \bigwedge x. x \in M \implies \text{sats}(M, Q_fm, [x]) \longleftrightarrow \text{is_}Q(\#\#M, x)$ **and**

$Qabs : \bigwedge x. x \in M \implies \text{is_}Q(\#\#M, x) \longleftrightarrow Q(x)$ **and**

$A \in M$

shows

$Collect(A, Q) \in M$

<proof>

lemma $Collect_in_M_2p$:

assumes

$Q_fm : Q_fm \in \text{formula}$ **and**

$Q_arty : \text{arity}(Q_fm) = 3$ **and**

$params_M : y \in M \ z \in M$ **and**

$Qsats : \bigwedge x. x \in M \implies \text{sats}(M, Q_fm, [x, y, z]) \longleftrightarrow \text{is_}Q(\#\#M, x, y, z)$ **and**

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

$A \in M$

shows

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

<proof>

lemma $Collect_in_M_4p$:

assumes

$Q_fm : Q_fm \in \text{formula}$ **and**

$Q_arty : \text{arity}(Q_fm) = 5$ **and**

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

$Qsats : \bigwedge x. x \in M \implies \text{sats}(M, Q_fm, [x, a1, a2, a3, a4]) \longleftrightarrow \text{is_}Q(\#\#M, x, a1, a2, a3, a4)$

and

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

$A \in M$

shows

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

<proof>

lemma $Repl_in_M$:

assumes
 $f_fm: f_fm \in \text{formula}$ **and**
 $f_ar: \text{arity}(f_fm) \leq 2 \ \#\ + \ \text{length}(env)$ **and**
 $fsats: \bigwedge x y. x \in M \implies y \in M \implies \text{sats}(M, f_fm, [x, y] @ env) \longleftrightarrow \text{is}_f(x, y)$ **and**
 $fabs: \bigwedge x y. x \in M \implies y \in M \implies \text{is}_f(x, y) \longleftrightarrow y = f(x)$ **and**
 $fclosed: \bigwedge x. x \in A \implies f(x) \in M$ **and**
 $A \in M \ \text{env} \in \text{list}(M)$
shows $\{f(x). x \in A\} \in M$
 $\langle \text{proof} \rangle$
end

11.2 A forcing locale and generic filters

locale $\text{forcing_data} = \text{forcing_notion} + M_ctm +$

assumes $P_in_M: P \in M$
and $leq_in_M: leq \in M$

begin

lemma $\text{transD} : \text{Transset}(M) \implies y \in M \implies y \subseteq M$
 $\langle \text{proof} \rangle$

lemmas $P_sub_M = \text{transD}[OF \ \text{trans}_M \ P_in_M]$

definition

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

lemma $M_genericD \ [dest]: M_generic(G) \implies x \in G \implies x \in P$
 $\langle \text{proof} \rangle$

lemma $M_generic_leqD \ [dest]: M_generic(G) \implies p \in G \implies q \in P \implies p \preceq q \implies q \in G$
 $\langle \text{proof} \rangle$

lemma $M_generic_compatD \ [dest]: M_generic(G) \implies p \in G \implies r \in G \implies \exists q \in G. q \preceq p \wedge q \preceq r$
 $\langle \text{proof} \rangle$

lemma $M_generic_denseD \ [dest]: M_generic(G) \implies \text{dense}(D) \implies D \subseteq P \implies D \in M \implies \exists q \in G. q \in D$
 $\langle \text{proof} \rangle$

lemma $G_nonempty: M_generic(G) \implies G \neq 0$
 $\langle \text{proof} \rangle$

lemma $\text{one_in_}G :$

assumes $M_generic(G)$
shows $one \in G$
 $\langle proof \rangle$

lemma $G_subset_M: M_generic(G) \implies G \subseteq M$
 $\langle proof \rangle$

declare iff_trans [$trans$]

lemma $generic_filter_existence:$
 $p \in P \implies \exists G. p \in G \wedge M_generic(G)$
 $\langle proof \rangle$

lemma $compat_in_abs :$
assumes
 $A \in M \ r \in M \ p \in M \ q \in M$
shows
 $is_compat_in(\#\#M, A, r, p, q) \longleftrightarrow compat_in(A, r, p, q)$
 $\langle proof \rangle$

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

lemma $compat_in_fm_type[TC] :$
 $\llbracket A \in nat; r \in nat; p \in nat; q \in nat \rrbracket \implies compat_in_fm(A, r, p, q) \in formula$
 $\langle proof \rangle$

lemma $sats_compat_in_fm:$
assumes
 $A \in nat \ r \in nat \ p \in nat \ q \in nat \ env \in list(M)$
shows
 $sats(M, compat_in_fm(A, r, p, q), env) \longleftrightarrow$
 $is_compat_in(\#\#M, nth(A, env), nth(r, env), nth(p, env), nth(q, env))$
 $\langle proof \rangle$

end

end

12 The ZFC axioms, internalized

theory $Internal_ZFC_Axioms$
imports

Forcing_Data

begin

schematic_goal *ZF_union_auto*:

$Union_ax(\#\#A) \longleftrightarrow (A, [] \models ?zfunion)$
<proof>

<ML>

schematic_goal *ZF_power_auto*:

$power_ax(\#\#A) \longleftrightarrow (A, [] \models ?zfpow)$
<proof>

<ML>

schematic_goal *ZF_pairing_auto*:

$upair_ax(\#\#A) \longleftrightarrow (A, [] \models ?zfpair)$
<proof>

<ML>

schematic_goal *ZF_foundation_auto*:

$foundation_ax(\#\#A) \longleftrightarrow (A, [] \models ?zfpow)$
<proof>

<ML>

schematic_goal *ZF_extensionality_auto*:

$extensionality(\#\#A) \longleftrightarrow (A, [] \models ?zfpow)$
<proof>

<ML>

schematic_goal *ZF_infinity_auto*:

$infinity_ax(\#\#A) \longleftrightarrow (A, [] \models (? \varphi(i,j,h)))$
<proof>

<ML>

schematic_goal *ZF_choice_auto*:

$choice_ax(\#\#A) \longleftrightarrow (A, [] \models (? \varphi(i,j,h)))$
<proof>

<ML>

syntax

_choice :: *i* (*<AC>*)

syntax_consts

$_choice \equiv ZF_choice_fm$

translations

$AC \mapsto CONST\ ZF_choice_fm$

lemmas $ZFC_fm_defs = ZF_extensionality_fm_def\ ZF_foundation_fm_def\ ZF_pairing_fm_def$
 $ZF_union_fm_def\ ZF_infinity_fm_def\ ZF_power_fm_def\ ZF_choice_fm_def$

lemmas $ZFC_fm_sats = ZF_extensionality_auto\ ZF_foundation_auto\ ZF_pairing_auto$
 $ZF_union_auto\ ZF_infinity_auto\ ZF_power_auto\ ZF_choice_auto$

definition

$ZF_fin :: i$ **where**

$ZF_fin \equiv \{ ZF_extensionality_fm, ZF_foundation_fm, ZF_pairing_fm,$
 $ZF_union_fm, ZF_infinity_fm, ZF_power_fm \}$

definition

$ZFC_fin :: i$ **where**

$ZFC_fin \equiv ZF_fin \cup \{ZF_choice_fm\}$

lemma $ZFC_fin_type : ZFC_fin \subseteq formula$

$\langle proof \rangle$

12.1 The Axiom of Separation, internalized

lemma $iterates_Forall_type [TC]:$

$\llbracket n \in nat; p \in formula \rrbracket \implies Forall^n(p) \in formula$

$\langle proof \rangle$

lemma $last_init_eq :$

assumes $l \in list(A)\ length(l) = succ(n)$

shows $\exists a \in A. \exists l' \in list(A). l = l' @ [a]$

$\langle proof \rangle$

lemma $take_drop_eq :$

assumes $l \in list(M)$

shows $\bigwedge n. n < succ(length(l)) \implies l = take(n,l) @ drop(n,l)$

$\langle proof \rangle$

lemma $list_split :$

assumes $n \leq succ(length(rest))\ rest \in list(M)$

shows $\exists re \in list(M). \exists st \in list(M). rest = re @ st \wedge length(re) = pred(n)$

$\langle proof \rangle$

lemma $sats_nForall:$

assumes

$\varphi \in formula$

shows

$n \in nat \implies ms \in list(M) \implies$

$M, ms \models (Forall^n(\varphi)) \longleftrightarrow$

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

definition

$sep_body_fm :: i \Rightarrow i$ **where**
 $sep_body_fm(p) \equiv Forall(Exists(Forall(Iff(Member(0,1),And(Member(0,2),incr_bv1^2(p))))))$

lemma $sep_body_fm_type$ [TC]: $p \in formula \Longrightarrow sep_body_fm(p) \in formula$
 ⟨proof⟩

lemma $sats_sep_body_fm$:

assumes
 $\varphi \in formula$ $ms \in list(M)$ $rest \in list(M)$
shows
 $M, rest @ ms \models sep_body_fm(\varphi) \longleftrightarrow$
 $separation(\#\#M, \lambda x. M, [x] @ rest @ ms \models \varphi)$
 ⟨proof⟩

definition

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

lemma $ZF_separation_fm_type$ [TC]: $p \in formula \Longrightarrow ZF_separation_fm(p) \in formula$
 ⟨proof⟩

lemma $sats_ZF_separation_fm_iff$:

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

12.2 The Axiom of Replacement, internalized

schematic_goal $sats_univalent_fm_auto$:

assumes
 $Q_iff_sats: \bigwedge x y z. x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow$
 $Q(x,z) \longleftrightarrow (A, Cons(z, Cons(y, Cons(x, env))) \models Q1_fm)$
 $\bigwedge x y z. x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow$
 $Q(x,y) \longleftrightarrow (A, Cons(z, Cons(y, Cons(x, env))) \models Q2_fm)$
and
 $asms: nth(i, env) = B \ i \in nat \ env \in list(A)$

shows

$univalent(\#\#A,B,Q) \longleftrightarrow A,env \models ?ufm(i)$
 $\langle proof \rangle$

$\langle ML \rangle$

lemma *univalent_fm_type* [TC]: $q1 \in formula \implies q2 \in formula \implies i \in nat \implies$
 $univalent_fm(q2,q1,i) \in formula$
 $\langle proof \rangle$

lemma *sats_univalent_fm* :

assumes

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

and

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

shows

$A, env \models univalent_fm(Q1_fm, Q2_fm, i) \longleftrightarrow univalent(\#\#A,B,Q)$
 $\langle proof \rangle$

definition

swap_vars :: $i \Rightarrow i$ **where**
swap_vars(φ) \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] :

$\varphi \in formula \implies swap_vars(\varphi) \in formula$
 $\langle proof \rangle$

lemma *sats_swap_vars* :

$[x,y] @ env \in list(M) \implies \varphi \in formula \implies$
 $M, [x,y] @ env \models swap_vars(\varphi) \longleftrightarrow M, [y,x] @ env \models \varphi$
 $\langle proof \rangle$

definition

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

definition

univalent_Q2 :: $i \Rightarrow i$ **where**
univalent_Q2(φ) $\equiv incr_bv(swap_vars(\varphi))'0$

lemma *univalent_Qs_type* [TC]:

assumes $\varphi \in formula$

shows $univalent_Q1(\varphi) \in formula \ univalent_Q2(\varphi) \in formula$
 $\langle proof \rangle$

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 $\varphi \ 2 \ M \ [_,_]$]

sats_incr_bv_iff [of $_ _ M \ []$] — simplifies iterates of $\lambda x. incr_bv(x)^2$

sats_incr_bv_iff [of $_ _ M \ [_,_]$] — simplifies $\lambda x. incr_bv(x)^2$

sats_incr_bv1_iff [of $_ _ M$] *sats_swap_vars* **for** $\varphi \ M$

lemma *sats_rep_body_fm*:

assumes

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

shows

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

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

<proof>

definition

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

$ZF_replacement_fm(p) \equiv Forall^{\wedge}(pred(pred(arity(p))))(rep_body_fm(p))$

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

$\in formula$

<proof>

lemma *sats_ZF_replacement_fm_iff*:

assumes

$\varphi \in formula$

shows

$(M, [] \models (ZF_replacement_fm(\varphi)))$

\longleftrightarrow

$(\forall env \in list(M). arity(\varphi) \leq 2 \ \#\ + \ length(env) \longrightarrow$

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

<proof>

definition

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

lemma $Un_subset_formula: A \subseteq formula \wedge B \subseteq formula \implies A \cup B \subseteq formula$
 $\langle proof \rangle$

lemma $ZF_inf_subset_formula : ZF_inf \subseteq formula$
 $\langle proof \rangle$

definition

$ZFC :: i$ **where**
 $ZFC \equiv ZF_inf \cup ZFC_fn$

definition

$ZF :: i$ **where**
 $ZF \equiv ZF_inf \cup ZF_fn$

definition

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

lemma $ZFC_subset_formula: ZFC \subseteq formula$
 $\langle proof \rangle$

Satisfaction of a set of sentences

definition

$satT :: [i,i] \Rightarrow o$ ($\langle _ \models _ \rangle$ [36,36] 60) **where**
 $A \models \Phi \equiv \forall \varphi \in \Phi. (A, [] \models \varphi)$

lemma $satTI$ [intro]:

assumes $\bigwedge \varphi. \varphi \in \Phi \implies A, [] \models \varphi$
shows $A \models \Phi$
 $\langle proof \rangle$

lemma $satTD$ [dest] : $A \models \Phi \implies \varphi \in \Phi \implies A, [] \models \varphi$
 $\langle proof \rangle$

lemma $sats_ZFC_iff_sats_ZF_AC:$

$(N \models ZFC) \longleftrightarrow (N \models ZF) \wedge (N, [] \models AC)$
 $\langle proof \rangle$

lemma $M_ZF_iff_M_satT: M_ZF(M) \longleftrightarrow (M \models ZF)$
 $\langle proof \rangle$

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) ≡ λj ∈ m ∪ p . if j ∈ m then f'j else g'j

```

```

lemma union_fun_type:
  assumes f ∈ m → n
    g ∈ p → q
  shows union_fun(f,g,m,p) ∈ m ∪ p → n ∪ q
⟨proof⟩

```

```

lemma union_fun_action :
  assumes
    env ∈ list(M)
    env' ∈ list(M)
    length(env) = m ∪ p
    ∀ i . i ∈ m → nth(f'i,env') = nth(i,env)
    ∀ j . j ∈ p → nth(g'j,env') = nth(j,env)
  shows ∀ i . i ∈ m ∪ p →
    nth(i,env) = nth(union_fun(f,g,m,p)'i,env')
⟨proof⟩

```

```

lemma id_fn_type :
  assumes n ∈ nat
  shows id(n) ∈ n → n
⟨proof⟩

```

```

lemma id_fn_action:
  assumes n ∈ nat env ∈ list(M)
  shows ∧ j . j < n ⇒ nth(j,env) = nth(id(n)'j,env)
⟨proof⟩

```

```

definition
  sum :: [i,i,i,i,i] ⇒ i where

```

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

lemma *sum_inl*:

assumes $m \in nat \ n \in nat$

$f \in m \rightarrow n \ x \in m$

shows $sum(f,g,m,n,p)'x = f'x$
 <proof>

lemma *sum_inr*:

assumes $m \in nat \ n \in nat \ p \in nat$

$g \in p \rightarrow q \ m \leq x \ x < m\#+p$

shows $sum(f,g,m,n,p)'x = g'(x\#-m)\#+n$
 <proof>

lemma *sum_action* :

assumes $m \in nat \ n \in nat \ p \in nat \ q \in nat$

$f \in m \rightarrow n \ g \in p \rightarrow q$

$env \in list(M)$

$env' \in list(M)$

$env1 \in list(M)$

$env2 \in list(M)$

$length(env) = m$

$length(env1) = p$

$length(env') = n$

$\bigwedge i . i < m \implies nth(i,env) = nth(f'i,env')$

$\bigwedge j . j < p \implies nth(j,env1) = nth(g'j,env2)$

shows $\forall i . i < m\#+p \longrightarrow$
 $nth(i,env@env1) = nth(sum(f,g,m,n,p)'i,env'@env2)$
 <proof>

lemma *sum_type* :

assumes $m \in nat \ n \in nat \ p \in nat \ q \in nat$

$f \in m \rightarrow n \ g \in p \rightarrow q$

shows $sum(f,g,m,n,p) \in (m\#+p) \rightarrow (n\#+q)$
 <proof>

lemma *sum_type_id* :

assumes

$f \in length(env) \rightarrow length(env')$

$env \in list(M)$

$env' \in list(M)$

$env1 \in list(M)$

shows

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

<proof>

lemma *sum_type_id_aux2* :

assumes

$f \in m \rightarrow n$
 $m \in \text{nat } n \in \text{nat}$
 $\text{env1} \in \text{list}(M)$

shows

$\text{sum}(f, \text{id}(\text{length}(\text{env1})), m, n, \text{length}(\text{env1})) \in$
 $(m \# + \text{length}(\text{env1})) \rightarrow (n \# + \text{length}(\text{env1}))$
 $\langle \text{proof} \rangle$

lemma *sum_action_id* :

assumes

$\text{env} \in \text{list}(M)$
 $\text{env}' \in \text{list}(M)$
 $f \in \text{length}(\text{env}) \rightarrow \text{length}(\text{env}')$
 $\text{env1} \in \text{list}(M)$
 $\bigwedge i . i < \text{length}(\text{env}) \implies \text{nth}(i, \text{env}) = \text{nth}(f'i, \text{env}')$

shows $\bigwedge i . i < \text{length}(\text{env}) \# + \text{length}(\text{env1}) \implies$

$\text{nth}(i, \text{env} @ \text{env1}) = \text{nth}(\text{sum}(f, \text{id}(\text{length}(\text{env1})), \text{length}(\text{env}), \text{length}(\text{env}'), \text{length}(\text{env1})))'i, \text{env}' @ \text{env1}$
 $\langle \text{proof} \rangle$

lemma *sum_action_id_aux* :

assumes

$f \in m \rightarrow n$
 $\text{env} \in \text{list}(M)$
 $\text{env}' \in \text{list}(M)$
 $\text{env1} \in \text{list}(M)$
 $\text{length}(\text{env}) = m$
 $\text{length}(\text{env}') = n$
 $\text{length}(\text{env1}) = p$
 $\bigwedge i . i < m \implies \text{nth}(i, \text{env}) = \text{nth}(f'i, \text{env}')$

shows $\bigwedge i . i < m \# + \text{length}(\text{env1}) \implies$

$\text{nth}(i, \text{env} @ \text{env1}) = \text{nth}(\text{sum}(f, \text{id}(\text{length}(\text{env1})), m, n, \text{length}(\text{env1})))'i, \text{env}' @ \text{env1}$
 $\langle \text{proof} \rangle$

definition

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

lemma *sum_id0* : $m \in \text{nat} \implies \text{sum_id}(m, f)'0 = 0$

$\langle \text{proof} \rangle$

lemma *sum_idS* : $p \in \text{nat} \implies q \in \text{nat} \implies f \in p \rightarrow q \implies x \in p \implies \text{sum_id}(p, f)'(\text{succ}(x))$

$= \text{succ}(f'x)$

$\langle \text{proof} \rangle$

lemma *sum_id_tc_aux* :

$p \in \text{nat} \implies q \in \text{nat} \implies f \in p \rightarrow q \implies \text{sum_id}(p, f) \in 1 \# + p \rightarrow 1 \# + q$

$\langle \text{proof} \rangle$

lemma *sum_id_tc* :

$n \in \text{nat} \implies m \in \text{nat} \implies f \in n \rightarrow m \implies \text{sum_id}(n,f) \in \text{succ}(n) \rightarrow \text{succ}(m)$
<proof>

13.2 Renaming of formulas

consts *ren* :: $i \Rightarrow i$

primrec

$\text{ren}(\text{Member}(x,y)) =$
 $(\lambda n \in \text{nat} . \lambda m \in \text{nat} . \lambda f \in n \rightarrow m . \text{Member}(f'x, f'y))$

$\text{ren}(\text{Equal}(x,y)) =$
 $(\lambda n \in \text{nat} . \lambda m \in \text{nat} . \lambda f \in n \rightarrow m . \text{Equal}(f'x, f'y))$

$\text{ren}(\text{Nand}(p,q)) =$
 $(\lambda n \in \text{nat} . \lambda m \in \text{nat} . \lambda f \in n \rightarrow m . \text{Nand}(\text{ren}(p)'n'm'f, \text{ren}(q)'n'm'f))$

$\text{ren}(\text{Forall}(p)) =$
 $(\lambda n \in \text{nat} . \lambda m \in \text{nat} . \lambda f \in n \rightarrow m . \text{Forall}(\text{ren}(p)'succ(n)'succ(m)'sum_id(n,f)))$

lemma *arity_meml* : $l \in \text{nat} \implies \text{Member}(x,y) \in \text{formula} \implies \text{arity}(\text{Member}(x,y)) \leq l \implies x \in l$
<proof>

lemma *arity_memr* : $l \in \text{nat} \implies \text{Member}(x,y) \in \text{formula} \implies \text{arity}(\text{Member}(x,y)) \leq l \implies y \in l$
<proof>

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

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

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

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

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

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

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

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

```

lemma arity_ren :
  fixes p
  assumes p ∈ formula
  shows  $\bigwedge n m f . n \in \text{nat} \implies m \in \text{nat} \implies f \in n \rightarrow m \implies \text{arity}(p) \leq n \implies$ 
 $\text{arity}(\text{ren}(p) \text{ 'n'm'f}) \leq m$ 
  ⟨proof⟩

lemma arity_forallE : p ∈ formula  $\implies m \in \text{nat} \implies \text{arity}(\text{Forall}(p)) \leq m \implies$ 
 $\text{arity}(p) \leq \text{succ}(m)$ 
  ⟨proof⟩

lemma env_coincidence_sum_id :
  assumes m ∈ nat n ∈ nat
   $\varrho \in \text{list}(A)$   $\varrho' \in \text{list}(A)$ 
   $f \in n \rightarrow m$ 
   $\bigwedge i . i < n \implies \text{nth}(i, \varrho) = \text{nth}(f^i, \varrho')$ 
   $a \in A$   $j \in \text{succ}(n)$ 
  shows  $\text{nth}(j, \text{Cons}(a, \varrho)) = \text{nth}(\text{sum\_id}(n, f) ^j, \text{Cons}(a, \varrho'))$ 
  ⟨proof⟩

lemma sats_iff_sats_ren :
  fixes  $\varphi$ 
  assumes  $\varphi \in \text{formula}$ 
  shows  $\llbracket n \in \text{nat} ; m \in \text{nat} ; \varrho \in \text{list}(M) ; \varrho' \in \text{list}(M) ; f \in n \rightarrow m ;$ 
 $\text{arity}(\varphi) \leq n ;$ 
 $\bigwedge i . i < n \implies \text{nth}(i, \varrho) = \text{nth}(f^i, \varrho') \rrbracket \implies$ 
 $\text{sats}(M, \varphi, \varrho) \longleftrightarrow \text{sats}(M, \text{ren}(\varphi) \text{ 'n'm'f}, \varrho')$ 
  ⟨proof⟩

end
theory Renaming_Auto
  imports
    Renaming
    Utils
    ZF.Finite
    ZF.List
  keywords rename :: thy_decl % ML
  and simple_rename :: thy_decl % ML
  and src
  and tgt
  abbrevs simple_rename =
begin

lemmas app_fun = apply_iff[THEN iffD1]
lemmas nat_succI = nat_succ_iff[THEN iffD2]

  ⟨ML⟩

```

end

14 Names and generic extensions

theory *Names*

imports

Forcing_Data

Interface

Recursion_Thms

Synthetic_Definition

begin

definition

SepReplace :: [*i*, *i*⇒*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{** .. / *_* ∈ *_*, *_*)⟩)⟩

syntax_consts

_SepReplace == *SepReplace*

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*⊆*A'*⇒ {*b* .. *x*∈*A*, *Q*}⊆{*b* .. *x*∈*A'*, *Q*}

⟨*proof*⟩

lemma *SepReplace_iff [simp]*: *y*∈{*b*(*x*) .. *x*∈*A*, *P*(*x*)} ↔ (∃*x*∈*A*. *y*=*b*(*x*) & *P*(*x*))

⟨*proof*⟩

lemma *SepReplace_dom implies* :

(∧ *x* . *x* ∈ *A* ⇒ *b*(*x*) = *b'*(*x*))⇒ {*b*(*x*) .. *x*∈*A*, *Q*(*x*)}={*b'*(*x*) .. *x*∈*A*, *Q*(*x*)}

⟨*proof*⟩

lemma *SepReplace_pred implies* :

∀*x*. *Q*(*x*)→ *b*(*x*) = *b'*(*x*)⇒ {*b*(*x*) .. *x*∈*A*, *Q*(*x*)}={*b'*(*x*) .. *x*∈*A*, *Q*(*x*)}

⟨*proof*⟩

14.1 The well-founded relation *ed*

lemma *eclose_sing* : *x* ∈ *eclose*(*a*) ⇒ *x* ∈ *eclose*({*a*})

⟨*proof*⟩

lemma *ecloseE* :

assumes *x* ∈ *eclose*(*A*)

shows *x* ∈ *A* ∨ (∃ *B* ∈ *A* . *x* ∈ *eclose*(*B*))

$\langle proof \rangle$

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

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

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

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

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

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

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

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

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

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

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

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

lemma *edrel_trans*: $\text{Transset}(A) \implies y \in A \implies x \in \text{domain}(y) \implies \langle x, y \rangle \in \text{edrel}(A)$
 $\langle proof \rangle$

lemma *domain_trans*: $\text{Transset}(A) \implies y \in A \implies x \in \text{domain}(y) \implies x \in A$
<proof>

lemma *relation_edrel* : $\text{relation}(\text{edrel}(A))$
<proof>

lemma *field_edrel* : $\text{field}(\text{edrel}(A)) \subseteq A$
<proof>

lemma *edrel_sub_memrel*: $\text{edrel}(A) \subseteq \text{trancl}(\text{Memrel}(\text{eclose}(A)))$
<proof>

lemma *wf_edrel* : $\text{wf}(\text{edrel}(A))$
<proof>

lemma *ed_induction*:
 assumes $\bigwedge x. [\bigwedge y. \text{ed}(y,x) \implies Q(y)] \implies Q(x)$
 shows $Q(a)$
<proof>

lemma *dom_under_edrel_eclose*: $\text{edrel}(\text{eclose}(\{x\})) \text{ -'' } \{x\} = \text{domain}(x)$
<proof>

lemma *ed_eclose* : $\langle y,z \rangle \in \text{edrel}(A) \implies y \in \text{eclose}(z)$
<proof>

lemma *tr_edrel_eclose* : $\langle y,z \rangle \in \text{edrel}(\text{eclose}(\{x\}))^{\wedge+} \implies y \in \text{eclose}(z)$
<proof>

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

lemma *tr_edrel_subset* :
 assumes $z \in \text{domain}(x)$
 shows $\text{tr_down}(\text{edrel}(\text{eclose}(\{x\})), z) \subseteq \text{eclose}(\{z\})$
<proof>

context *M_ctm*
begin

lemma *upairM* : $x \in M \implies y \in M \implies \{x,y\} \in M$
<proof>

lemma *singletonM* : $a \in M \implies \{a\} \in M$
<proof>

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

end

14.2 Values and check-names

context *forcing_data*
begin

definition

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

definition

$check :: i \Rightarrow i$ **where**
 $check(x) \equiv \text{transrec}(x, Hcheck)$

lemma *checkD*:

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

definition

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

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

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

lemma *rcheck_in_M* :

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

lemma *aux_def_check*: $x \in y \Longrightarrow$

$\text{wfrec}(\text{Memrel}(\text{eclose}(\{y\})), x, Hcheck) =$
 $\text{wfrec}(\text{Memrel}(\text{eclose}(\{x\})), x, Hcheck)$
 ⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

definition

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

lemma *val_of_elem*: $\langle \vartheta, p \rangle \in \pi \Rightarrow p \in G \Rightarrow p \in P \Rightarrow val(G, \vartheta) \in val(G, \pi)$
<proof>

lemma *elem_of_val*: $x \in val(G, \pi) \Rightarrow \exists \vartheta \in domain(\pi). val(G, \vartheta) = x$
<proof>

lemma *elem_of_val_pair*: $x \in val(G, \pi) \Rightarrow \exists \vartheta. \exists p \in G. \langle \vartheta, p \rangle \in \pi \wedge val(G, \vartheta) = x$
<proof>

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

lemma *GenExtD*:
 $x \in M[G] \Rightarrow \exists \tau \in M. x = val(G, \tau)$
<proof>

lemma *GenExtI*:
 $x \in M \Rightarrow val(G, x) \in M[G]$
<proof>

lemma *Transset_MG* : $Transset(M[G])$
<proof>

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

lemma *check_n_M* :
fixes n

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

definition

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

definition

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

lemma $\text{one_in_}M$: $\text{one} \in M$
 $\langle \text{proof} \rangle$

lemma def_PHcheck :

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

definition

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

lemma PHcheck_type [TC]:

$\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; u \in \text{nat} \rrbracket \Longrightarrow \text{PHcheck_fm}(x, y, z, u) \in \text{formula}$
 $\langle \text{proof} \rangle$

lemma sats_PHcheck_fm [simp]:

$\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; u \in \text{nat}; \text{env} \in \text{list}(M) \rrbracket$
 $\Longrightarrow \text{sats}(M, \text{PHcheck_fm}(x, y, z, u), \text{env}) \longleftrightarrow$
 $\text{PHcheck}(\text{nth}(x, \text{env}), \text{nth}(y, \text{env}), \text{nth}(z, \text{env}), \text{nth}(u, \text{env}))$
 $\langle \text{proof} \rangle$

definition

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

lemma is_Hcheck_type [TC]:

$\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; u \in \text{nat} \rrbracket \Longrightarrow \text{is_Hcheck_fm}(x, y, z, u) \in \text{formula}$
 $\langle \text{proof} \rangle$

lemma sats_is_Hcheck_fm [simp]:

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

$$\implies \text{sats}(M, \text{is_Hcheck_fm}(x, y, z, u), \text{env}) \longleftrightarrow$$

$$\text{is_Hcheck}(\text{nth}(x, \text{env}), \text{nth}(y, \text{env}), \text{nth}(z, \text{env}), \text{nth}(u, \text{env}))$$

$$\langle \text{proof} \rangle$$

lemma *wfrec_Hcheck* :

assumes

$X \in M$

shows

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

$\langle \text{proof} \rangle$

lemma *repl_PHcheck* :

assumes

$f \in M$

shows

$\text{strong_replacement}(\#\#M, \text{PHcheck}(\text{one}, f))$

$\langle \text{proof} \rangle$

lemma *univ_PHcheck* : $\llbracket z \in M ; f \in M \rrbracket \implies \text{univalent}(\#\#M, z, \text{PHcheck}(\text{one}, f))$

$\langle \text{proof} \rangle$

lemma *relation2_Hcheck* :

$\text{relation2}(\#\#M, \text{is_Hcheck}(\text{one}), \text{Hcheck})$

$\langle \text{proof} \rangle$

lemma *PHcheck_closed* :

$\llbracket z \in M ; f \in M ; x \in z; \text{PHcheck}(\text{one}, f, x, y) \rrbracket \implies (\#\#M)(y)$

$\langle \text{proof} \rangle$

lemma *Hcheck_closed* :

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

$\langle \text{proof} \rangle$

lemma *wf_rcheck* : $x \in M \implies \text{wf}(\text{rcheck}(x))$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

end

definition

$is_singleton :: [i \Rightarrow o, i, i] \Rightarrow o$ **where**
 $is_singleton(A, x, z) \equiv \exists c[A]. empty(A, c) \wedge is_cons(A, x, c, z)$

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

definition

$singleton_fm :: [i, i] \Rightarrow i$ **where**
 $singleton_fm(i, j) \equiv Exists(And(empty_fm(0), cons_fm(succ(i), 0, succ(j))))$

lemma $singleton_type[TC] : \llbracket x \in nat ; y \in nat \rrbracket \Longrightarrow singleton_fm(x, y) \in formula$
 $\langle proof \rangle$

lemma $is_singleton_iff_sats:$

$\llbracket nth(i, env) = x ; nth(j, env) = y ;$
 $i \in nat ; j \in nat ; env \in list(A) \rrbracket$
 $\Longrightarrow is_singleton(\#\#A, x, y) \longleftrightarrow sats(A, singleton_fm(i, j), env)$
 $\langle proof \rangle$

context $forcing_data$ **begin**

definition

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

lemma $rcheck_abs :$

$\llbracket x \in M ; r \in M \rrbracket \Longrightarrow is_rcheck(x, r) \longleftrightarrow r = rcheck(x)$
 $\langle proof \rangle$

schematic_goal $rcheck_fm_auto:$

assumes

$i \in nat \ j \in nat \ env \in list(M)$

shows

$is_rcheck(nth(i, env), nth(j, env)) \longleftrightarrow sats(M, ?rch(i, j), env)$

$\langle proof \rangle$

$\langle ML \rangle$

definition

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

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

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

lemma *check_fm_type*[TC] :
 $\llbracket x \in nat; o \in nat; z \in nat \rrbracket \Longrightarrow check_fm(x,o,z) \in formula$
 $\langle proof \rangle$

lemma *sats_check_fm* :
assumes
 $nth(o,env) = one \ x \in nat \ z \in nat \ o \in nat \ env \in list(M) \ x < length(env) \ z < length(env)$
shows
 $sats(M, check_fm(x,o,z), env) \longleftrightarrow is_check(nth(x,env),nth(z,env))$
 $\langle proof \rangle$

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

lemma *pair_check* : $\llbracket p \in M ; y \in M \rrbracket \Longrightarrow (\exists c \in M. is_check(p,c) \wedge pair(\#\#M,c,p,y))$
 $\longleftrightarrow y = \langle check(p),p \rangle$
 $\langle proof \rangle$

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

The name for the generic filter

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

lemma *G_dot_in_M* :
 $G_dot \in M$
 $\langle proof \rangle$

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

$one \in G$
shows $val(G, G_dot) = G$
 $\langle proof \rangle$

lemma $G_in_Gen_Ext$:
assumes $G \subseteq P$ **and** $one \in G$
shows $G \in M[G]$
 $\langle proof \rangle$

lemma fst_snd_closed : $p \in M \implies fst(p) \in M \wedge snd(p) \in M$
 $\langle proof \rangle$

end

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

lemma $zero_in_MG$:
 $0 \in M[G]$
 $\langle proof \rangle$

lemma $G_nonempty$: $G \neq 0$
 $\langle proof \rangle$

end

end

15 Well-founded relation on names

theory $freqR$ **imports** $Names$ $Synthetic_Definition$ **begin**

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

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

definition

$is_hcomp :: [i \Rightarrow o, i \Rightarrow i \Rightarrow o, i \Rightarrow i \Rightarrow o, i, i] \Rightarrow o$ **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

$is_f_abs: \bigwedge a z. M(a) \implies M(z) \implies is_f(a, z) \longleftrightarrow z = f(a)$ **and**

$is_g_abs: \bigwedge a z. M(a) \implies M(z) \implies is_g(a, z) \longleftrightarrow z = g(a)$ **and**

$g_closed: \bigwedge a. M(a) \implies M(g(a))$

$M(a) M(w)$
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

$f_type :: i \Rightarrow i$ **where**
 $f_type \equiv fst$

definition

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

definition

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

definition

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

lemma components_simp:

$f_type(\langle f, n1, n2, c \rangle) = f$
 $name1(\langle f, n1, n2, c \rangle) = n1$
 $name2(\langle f, n1, n2, c \rangle) = n2$
 $cond_of(\langle f, n1, n2, c \rangle) = c$
 ⟨proof⟩

definition eclose_n :: [i ⇒ i, i] ⇒ i where

$eclose_n(name,x) = eclose(\{name(x)\})$

definition

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

lemma *components_in_eclose* :

$n1 \in ecloseN(\langle f,n1,n2,c \rangle)$
 $n2 \in ecloseN(\langle f,n1,n2,c \rangle)$
<proof>

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

lemma *ecloseNI1* :

assumes $x \in eclose(n1) \vee x \in eclose(n2)$
shows $x \in ecloseN(\langle f,n1,n2,c \rangle)$
<proof>

lemmas *ecloseNI* = *ecloseNI1*

lemma *ecloseN_mono* :

assumes $u \in ecloseN(x)$ $name1(x) \in ecloseN(y)$ $name2(x) \in ecloseN(y)$
shows $u \in ecloseN(y)$
<proof>

definition

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

definition

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

lemma *sats_fst_fm* :

$\llbracket x \in nat; y \in nat; env \in list(A) \rrbracket$
 $\implies sats(A, fst_fm(x,y), env) \longleftrightarrow$
 $is_fst(\#\#A, nth(x,env), nth(y,env))$
<proof>

definition

$is_ftype :: (i \Rightarrow o) \Rightarrow i \Rightarrow i \Rightarrow o$ **where**
 $is_ftype \equiv is_fst$

definition

$f_{type_fm} :: [i,i] \Rightarrow i$ **where**
 $f_{type_fm} \equiv f_{st_fm}$

lemma $sats_f_{type_fm}$:
 $\llbracket x \in nat; y \in nat; env \in list(A) \rrbracket$
 $\implies sats(A, f_{type_fm}(x,y), env) \longleftrightarrow$
 $is_f_{type}(\#\#A, nth(x,env), nth(y,env))$
 $\langle proof \rangle$

lemma $is_f_{type_iff_sats}$:
assumes
 $nth(a,env) = aa \ nth(b,env) = bb \ a \in nat \ b \in nat \ env \in list(A)$
shows
 $is_f_{type}(\#\#A, aa, bb) \longleftrightarrow sats(A, f_{type_fm}(a,b), env)$
 $\langle proof \rangle$

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

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

lemma $sats_snd_fm$:
 $\llbracket x \in nat; y \in nat; env \in list(A) \rrbracket$
 $\implies sats(A, snd_fm(x,y), env) \longleftrightarrow$
 $is_snd(\#\#A, nth(x,env), nth(y,env))$
 $\langle proof \rangle$

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

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

lemma $sats_name1_fm$:
 $\llbracket x \in nat; y \in nat; env \in list(A) \rrbracket$
 $\implies sats(A, name1_fm(x,y), env) \longleftrightarrow$
 $is_name1(\#\#A, nth(x,env), nth(y,env))$
 $\langle proof \rangle$

lemma $is_name1_iff_sats$:
assumes
 $nth(a,env) = aa \ nth(b,env) = bb \ a \in nat \ b \in nat \ env \in list(A)$

shows

$is_name1(\#\#A,aa,bb) \longleftrightarrow sats(A,name1_fm(a,b), env)$
(proof)

definition

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

definition

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

lemma sats_snd2_fm :

$\llbracket x \in nat; y \in nat; env \in list(A) \rrbracket$
 $\implies sats(A,snd_snd_fm(x,y), env) \longleftrightarrow$
 $is_snd_snd(\#\#A, nth(x,env), nth(y,env))$
(proof)

definition

$is_name2 :: (i \Rightarrow o) \Rightarrow i \Rightarrow i \Rightarrow o$ **where**
 $is_name2(M,x,t3) \equiv is_hcomp(M,is_fst(M),is_snd_snd(M),x,t3)$

definition

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

lemma sats_name2_fm :

$\llbracket x \in nat; y \in nat; env \in list(A) \rrbracket$
 $\implies sats(A,name2_fm(x,y), env) \longleftrightarrow$
 $is_name2(\#\#A, nth(x,env), nth(y,env))$
(proof)

lemma is_name2_iff_sats:

assumes

$nth(a,env) = aa \quad nth(b,env) = bb \quad a \in nat \quad b \in nat \quad env \in list(A)$

shows

$is_name2(\#\#A,aa,bb) \longleftrightarrow sats(A,name2_fm(a,b), env)$
(proof)

definition

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

definition

$cond_of_fm :: [i,i] \Rightarrow i$ **where**
 $cond_of_fm(x,t4) \equiv hcomp_fm(snd_fm,snd_snd_fm,x,t4)$

lemma sats_cond_of_fm :

$\llbracket x \in nat; y \in nat; env \in list(A) \rrbracket$

$\implies \text{sats}(A, \text{cond_of_fm}(x, y), \text{env}) \longleftrightarrow$
 $\text{is_cond_of}(\#\#A, \text{nth}(x, \text{env}), \text{nth}(y, \text{env}))$
 ⟨proof⟩

lemma *is_cond_of_iff_sats*:

assumes

$\text{nth}(a, \text{env}) = aa \ \text{nth}(b, \text{env}) = bb \ a \in \text{nat} \ b \in \text{nat} \ \text{env} \in \text{list}(A)$

shows

$\text{is_cond_of}(\#\#A, aa, bb) \longleftrightarrow \text{sats}(A, \text{cond_of_fm}(a, b), \text{env})$

⟨proof⟩

lemma *components_type*[TC] :

assumes $a \in \text{nat} \ b \in \text{nat}$

shows

$\text{ftype_fm}(a, b) \in \text{formula}$

$\text{name1_fm}(a, b) \in \text{formula}$

$\text{name2_fm}(a, b) \in \text{formula}$

$\text{cond_of_fm}(a, b) \in \text{formula}$

⟨proof⟩

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

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

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

definition

is_eclose_n :: $[i \Rightarrow o, [i \Rightarrow o, i, i] \Rightarrow o, i, i] \Rightarrow o$ **where**

$\text{is_eclose_n}(N, \text{is_name}, \text{en}, t) \equiv$

$\exists n1[N]. \exists s1[N]. \text{is_name}(N, t, n1) \wedge \text{is_singleton}(N, n1, s1) \wedge \text{is_eclose}(N, s1, \text{en})$

definition

eclose_n1_fm :: $[i, i] \Rightarrow i$ **where**

$\text{eclose_n1_fm}(m, t) \equiv \text{Exists}(\text{Exists}(\text{And}(\text{And}(\text{name1_fm}(t\#\#2, 0), \text{singleton_fm}(0, 1)),$
 $\text{is_eclose_fm}(1, m\#\#2))))$

definition

eclose_n2_fm :: $[i, i] \Rightarrow i$ **where**

$\text{eclose_n2_fm}(m, t) \equiv \text{Exists}(\text{Exists}(\text{And}(\text{And}(\text{name2_fm}(t\#\#2, 0), \text{singleton_fm}(0, 1)),$
 $\text{is_eclose_fm}(1, m\#\#2))))$

definition

is_ecloseN :: $[i \Rightarrow o, i, i] \Rightarrow o$ **where**

$\text{is_ecloseN}(N, \text{en}, t) \equiv \exists \text{en1}[N]. \exists \text{en2}[N].$

$$is_eclose_n(N, is_name1, en1, t) \wedge is_eclose_n(N, is_name2, en2, t) \wedge union(N, en1, en2, en)$$

definition

$ecloseN_fm :: [i, i] \Rightarrow i$ **where**
 $ecloseN_fm(en, t) \equiv Exists(Exists(And(eclose_n1_fm(1, t\#+2),$
 $And(eclose_n2_fm(0, t\#+2), union_fm(1, 0, en\#+2))))))$

lemma $ecloseN_fm_type$ [TC] :

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

lemma $sats_ecloseN_fm$ [simp]:

$\llbracket en \in nat ; t \in nat ; env \in list(A) \rrbracket$
 $\Longrightarrow sats(A, ecloseN_fm(en, t), env) \longleftrightarrow is_ecloseN(\#\#A, nth(en, env), nth(t, env))$
 $\langle proof \rangle$

definition

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

lemma $frecR_ftypeD$:

assumes $frecR(x, y)$
shows $(ftype(x) = 0 \wedge ftype(y) = 1) \vee (ftype(x) = 1 \wedge ftype(y) = 0)$
 $\langle proof \rangle$

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

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

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

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

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

<proof>

lemma *frecRI3'*: $s \in \text{domain}(n2) \implies \text{frecR}(\langle 0, n1, s, q \rangle, \langle 1, n1, n2, q \rangle)$
<proof>

lemma *frecR_iff* :

$\text{frecR}(x,y) \longleftrightarrow$
 $(\text{ftype}(x) = 1 \wedge \text{ftype}(y) = 0$
 $\wedge (\text{name1}(x) \in \text{domain}(\text{name1}(y)) \cup \text{domain}(\text{name2}(y)) \wedge (\text{name2}(x) =$
 $\text{name1}(y) \vee \text{name2}(x) = \text{name2}(y))))$
 $\vee (\text{ftype}(x) = 0 \wedge \text{ftype}(y) = 1 \wedge \text{name1}(x) = \text{name1}(y) \wedge \text{name2}(x) \in$
 $\text{domain}(\text{name2}(y)))$
<proof>

lemma *frecR_D1* :

$\text{frecR}(x,y) \implies \text{ftype}(y) = 0 \implies \text{ftype}(x) = 1 \wedge$
 $(\text{name1}(x) \in \text{domain}(\text{name1}(y)) \cup \text{domain}(\text{name2}(y)) \wedge (\text{name2}(x) =$
 $\text{name1}(y) \vee \text{name2}(x) = \text{name2}(y)))$
<proof>

lemma *frecR_D2* :

$\text{frecR}(x,y) \implies \text{ftype}(y) = 1 \implies \text{ftype}(x) = 0 \wedge$
 $\text{ftype}(x) = 0 \wedge \text{ftype}(y) = 1 \wedge \text{name1}(x) = \text{name1}(y) \wedge \text{name2}(x) \in$
 $\text{domain}(\text{name2}(y))$
<proof>

lemma *frecR_DI* :

assumes $\text{frecR}(\langle a,b,c,d \rangle, \langle \text{ftype}(y), \text{name1}(y), \text{name2}(y), \text{cond_of}(y) \rangle)$
shows $\text{frecR}(\langle a,b,c,d \rangle, y)$
<proof>

definition

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

schematic_goal *sats_frecR_fm_auto*:

assumes
 $i \in \text{nat } j \in \text{nat } \text{env} \in \text{list}(A) \text{ nth}(i, \text{env}) = a \text{ nth}(j, \text{env}) = b$
shows
 $\text{is_frecR}(\#\# A, a, b) \longleftrightarrow \text{sats}(A, ?\text{fr_fm}(i, j), \text{env})$
<proof>

$\langle ML \rangle$

lemma *eq_ftypep_not_freclR*:
 assumes $f_{type}(x) = f_{type}(y)$
 shows $\neg freclR(x,y)$
 $\langle proof \rangle$

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

lemma *rank_names_types* [TC]:
 shows $Ord(rank_names(x))$
 $\langle proof \rangle$

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**
 $type_form(x) \equiv if f_{type}(x) = 0 then 1 else mtype_form(x)$

lemma *type_form_tc* [TC]:
 shows $type_form(x) \in \mathcal{I}$
 $\langle proof \rangle$

lemma *freclR_le_rank_names* :
 assumes $freclR(x,y)$
 shows $rank_names(x) \leq rank_names(y)$
 $\langle proof \rangle$

definition
 $\Gamma :: i \Rightarrow i$ **where**
 $\Gamma(x) = \mathcal{I} ** rank_names(x) ++ type_form(x)$

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

lemma *Γ_mono* :
 assumes $freclR(x,y)$
 shows $\Gamma(x) < \Gamma(y)$
 $\langle proof \rangle$

definition

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

lemma $frecrelI$:

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

lemma $frecrelD$:

assumes $\langle x,y \rangle \in frecrel(A1 \times A2 \times A3 \times A4)$
shows $f_{type}(x) \in A1 \ f_{type}(y) \in A1$
 $name1(x) \in A2 \ name1(y) \in A2 \ name2(x) \in A3 \ name2(y) \in A3$
 $cond_of(x) \in A4 \ cond_of(y) \in A4$
 $frecR(x,y)$
 $\langle proof \rangle$

lemma $wf_frecrel$:

shows $wf(frecrel(A))$
 $\langle proof \rangle$

lemma $core_induction_aux$:

fixes $A1 \ A2 :: i$
assumes
 $Transset(A1)$
 $\bigwedge \tau \ \vartheta \ p. \ p \in A2 \Longrightarrow \llbracket \bigwedge q \ \sigma. \llbracket q \in A2 ; \sigma \in domain(\vartheta) \rrbracket \Longrightarrow Q(0,\tau,\sigma,q) \rrbracket \Longrightarrow$
 $Q(1,\tau,\vartheta,p)$
 $\bigwedge \tau \ \vartheta \ p. \ p \in A2 \Longrightarrow \llbracket \bigwedge q \ \sigma. \llbracket q \in A2 ; \sigma \in domain(\tau) \cup domain(\vartheta) \rrbracket \Longrightarrow$
 $Q(1,\sigma,\tau,q) \wedge Q(1,\sigma,\vartheta,q) \rrbracket \Longrightarrow Q(0,\tau,\vartheta,p)$
shows $a \in 2 \times A1 \times A1 \times A2 \Longrightarrow Q(f_{type}(a),name1(a),name2(a),cond_of(a))$
 $\langle proof \rangle$

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

lemma $frecrel_fst_snd$:

$frecrel(A) = \{z \in A \times A .$
 $f_{type}(fst(z)) = 1 \wedge$
 $f_{type}(snd(z)) = 0 \wedge name1(fst(z)) \in domain(name1(snd(z))) \cup do-$
 $main(name2(snd(z))) \wedge$
 $(name2(fst(z)) = name1(snd(z)) \vee name2(fst(z)) = name2(snd(z)))$
 $\vee (f_{type}(fst(z)) = 0 \wedge$
 $f_{type}(snd(z)) = 1 \wedge name1(fst(z)) = name1(snd(z)) \wedge name2(fst(z)) \in$
 $domain(name2(snd(z))))\}$
 $\langle proof \rangle$

end

16 Arities of internalized formulas

theory *Arities*
 imports *FrecR*
begin

lemma *arity_upair_fm* : $\llbracket t1 \in nat ; t2 \in nat ; up \in nat \rrbracket \implies$
 $arity(upair_fm(t1, t2, up)) = \bigcup \{succ(t1), succ(t2), succ(up)\}$
 $\langle proof \rangle$

lemma *arity_pair_fm* : $\llbracket t1 \in nat ; t2 \in nat ; p \in nat \rrbracket \implies$
 $arity(pair_fm(t1, t2, p)) = \bigcup \{succ(t1), succ(t2), succ(p)\}$
 $\langle proof \rangle$

lemma *arity_composition_fm* :
 $\llbracket r \in nat ; s \in nat ; t \in nat \rrbracket \implies arity(composition_fm(r, s, t)) = \bigcup \{succ(r),$
 $succ(s), succ(t)\}$
 $\langle proof \rangle$

lemma *arity_domain_fm* :
 $\llbracket r \in nat ; z \in nat \rrbracket \implies arity(domain_fm(r, z)) = succ(r) \cup succ(z)$
 $\langle proof \rangle$

lemma *arity_range_fm* :
 $\llbracket r \in nat ; z \in nat \rrbracket \implies arity(range_fm(r, z)) = succ(r) \cup succ(z)$
 $\langle proof \rangle$

lemma *arity_union_fm* :
 $\llbracket x \in nat ; y \in nat ; z \in nat \rrbracket \implies arity(union_fm(x, y, z)) = \bigcup \{succ(x), succ(y),$
 $succ(z)\}$
 $\langle proof \rangle$

lemma *arity_image_fm* :
 $\llbracket x \in nat ; y \in nat ; z \in nat \rrbracket \implies arity(image_fm(x, y, z)) = \bigcup \{succ(x), succ(y),$
 $succ(z)\}$
 $\langle proof \rangle$

lemma *arity_pre_image_fm* :
 $\llbracket x \in nat ; y \in nat ; z \in nat \rrbracket \implies arity(pre_image_fm(x, y, z)) = \bigcup \{succ(x),$
 $succ(y), succ(z)\}$
 $\langle proof \rangle$

lemma *arity_big_union_fm* :
 $\llbracket x \in nat ; y \in nat \rrbracket \implies arity(big_union_fm(x, y)) = succ(x) \cup succ(y)$
 $\langle proof \rangle$

lemma *arity_fun_apply_fm* :

$\llbracket x \in \text{nat} ; y \in \text{nat} ; f \in \text{nat} \rrbracket \implies$
 $\text{arity}(\text{fun_apply_fm}(f,x,y)) = \text{succ}(f) \cup \text{succ}(x) \cup \text{succ}(y)$
 $\langle \text{proof} \rangle$

lemma *arity_field_fm* :
 $\llbracket r \in \text{nat} ; z \in \text{nat} \rrbracket \implies \text{arity}(\text{field_fm}(r,z)) = \text{succ}(r) \cup \text{succ}(z)$
 $\langle \text{proof} \rangle$

lemma *arity_empty_fm* :
 $\llbracket r \in \text{nat} \rrbracket \implies \text{arity}(\text{empty_fm}(r)) = \text{succ}(r)$
 $\langle \text{proof} \rangle$

lemma *arity_succ_fm* :
 $\llbracket x \in \text{nat} ; y \in \text{nat} \rrbracket \implies \text{arity}(\text{succ_fm}(x,y)) = \text{succ}(x) \cup \text{succ}(y)$
 $\langle \text{proof} \rangle$

lemma *number1arity_fm* :
 $\llbracket r \in \text{nat} \rrbracket \implies \text{arity}(\text{number1_fm}(r)) = \text{succ}(r)$
 $\langle \text{proof} \rangle$

lemma *arity_function_fm* :
 $\llbracket r \in \text{nat} \rrbracket \implies \text{arity}(\text{function_fm}(r)) = \text{succ}(r)$
 $\langle \text{proof} \rangle$

lemma *arity_relation_fm* :
 $\llbracket r \in \text{nat} \rrbracket \implies \text{arity}(\text{relation_fm}(r)) = \text{succ}(r)$
 $\langle \text{proof} \rangle$

lemma *arity_restriction_fm* :
 $\llbracket r \in \text{nat} ; z \in \text{nat} ; A \in \text{nat} \rrbracket \implies \text{arity}(\text{restriction_fm}(A,z,r)) = \text{succ}(A) \cup \text{succ}(r)$
 $\cup \text{succ}(z)$
 $\langle \text{proof} \rangle$

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

lemma *arity_subset_fm* :
 $\llbracket x \in \text{nat} ; y \in \text{nat} \rrbracket \implies \text{arity}(\text{subset_fm}(x,y)) = \text{succ}(x) \cup \text{succ}(y)$
 $\langle \text{proof} \rangle$

lemma *arity_transset_fm* :
 $\llbracket x \in \text{nat} \rrbracket \implies \text{arity}(\text{transset_fm}(x)) = \text{succ}(x)$
 $\langle \text{proof} \rangle$

lemma *arity_ordinal_fm* :
 $\llbracket x \in \text{nat} \rrbracket \implies \text{arity}(\text{ordinal_fm}(x)) = \text{succ}(x)$
<proof>

lemma *arity_limit_ordinal_fm* :
 $\llbracket x \in \text{nat} \rrbracket \implies \text{arity}(\text{limit_ordinal_fm}(x)) = \text{succ}(x)$
<proof>

lemma *arity_finite_ordinal_fm* :
 $\llbracket x \in \text{nat} \rrbracket \implies \text{arity}(\text{finite_ordinal_fm}(x)) = \text{succ}(x)$
<proof>

lemma *arity_omega_fm* :
 $\llbracket x \in \text{nat} \rrbracket \implies \text{arity}(\text{omega_fm}(x)) = \text{succ}(x)$
<proof>

lemma *arity_cartprod_fm* :
 $\llbracket A \in \text{nat} ; B \in \text{nat} ; z \in \text{nat} \rrbracket \implies \text{arity}(\text{cartprod_fm}(A, B, z)) = \text{succ}(A) \cup \text{succ}(B) \cup \text{succ}(z)$
<proof>

lemma *arity_fst_fm* :
 $\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{fst_fm}(x, t)) = \text{succ}(x) \cup \text{succ}(t)$
<proof>

lemma *arity_snd_fm* :
 $\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{snd_fm}(x, t)) = \text{succ}(x) \cup \text{succ}(t)$
<proof>

lemma *arity_snd_snd_fm* :
 $\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{snd_snd_fm}(x, t)) = \text{succ}(x) \cup \text{succ}(t)$
<proof>

lemma *arity_fstype_fm* :
 $\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{fstype_fm}(x, t)) = \text{succ}(x) \cup \text{succ}(t)$
<proof>

lemma *name1arity_fm* :
 $\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{name1_fm}(x, t)) = \text{succ}(x) \cup \text{succ}(t)$
<proof>

lemma *name2arity_fm* :
 $\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{name2_fm}(x, t)) = \text{succ}(x) \cup \text{succ}(t)$
<proof>

lemma *arity_cond_of_fm* :
 $\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{cond_of_fm}(x, t)) = \text{succ}(x) \cup \text{succ}(t)$
<proof>

lemma *arity_singleton_fm* :

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

lemma *arity_Memrel_fm* :

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

lemma *arity_quasinat_fm* :

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

lemma *arity_is_recfun_fm* :

$\llbracket p \in \text{formula} ; v \in \text{nat} ; n \in \text{nat} ; Z \in \text{nat} ; i \in \text{nat} \rrbracket \implies \text{arity}(p) = i \implies$
 $\text{arity}(\text{is_recfun_fm}(p,v,n,Z)) = \text{succ}(v) \cup \text{succ}(n) \cup \text{succ}(Z) \cup \text{pred}(\text{pred}(\text{pred}(\text{pred}(i))))$
<proof>

lemma *arity_is_wfrec_fm* :

$\llbracket p \in \text{formula} ; v \in \text{nat} ; n \in \text{nat} ; Z \in \text{nat} ; i \in \text{nat} \rrbracket \implies \text{arity}(p) = i \implies$
 $\text{arity}(\text{is_wfrec_fm}(p,v,n,Z)) = \text{succ}(v) \cup \text{succ}(n) \cup \text{succ}(Z) \cup \text{pred}(\text{pred}(\text{pred}(\text{pred}(\text{pred}(i)))))$
<proof>

lemma *arity_is_nat_case_fm* :

$\llbracket p \in \text{formula} ; v \in \text{nat} ; n \in \text{nat} ; Z \in \text{nat} ; i \in \text{nat} \rrbracket \implies \text{arity}(p) = i \implies$
 $\text{arity}(\text{is_nat_case_fm}(v,p,n,Z)) = \text{succ}(v) \cup \text{succ}(n) \cup \text{succ}(Z) \cup \text{pred}(\text{pred}(i))$
<proof>

lemma *arity_iterates_MH_fm* :

assumes $isF \in \text{formula} \ v \in \text{nat} \ n \in \text{nat} \ g \in \text{nat} \ z \in \text{nat} \ i \in \text{nat}$
 $\text{arity}(isF) = i$
shows $\text{arity}(\text{iterates_MH_fm}(isF,v,n,g,z)) =$
 $\text{succ}(v) \cup \text{succ}(n) \cup \text{succ}(g) \cup \text{succ}(z) \cup \text{pred}(\text{pred}(\text{pred}(\text{pred}(i))))$
<proof>

lemma *arity_is_iterates_fm* :

assumes $p \in \text{formula} \ v \in \text{nat} \ n \in \text{nat} \ Z \in \text{nat} \ i \in \text{nat}$
 $\text{arity}(p) = i$
shows $\text{arity}(\text{is_iterates_fm}(p,v,n,Z)) = \text{succ}(v) \cup \text{succ}(n) \cup \text{succ}(Z) \cup$
 $\text{pred}(\text{pred}(\text{pred}(\text{pred}(\text{pred}(\text{pred}(\text{pred}(\text{pred}(\text{pred}(\text{pred}(i))))))))))$
<proof>

lemma *arity_eclose_n_fm* :

assumes $A \in \text{nat} \ x \in \text{nat} \ t \in \text{nat}$
shows $\text{arity}(\text{eclose_n_fm}(A,x,t)) = \text{succ}(A) \cup \text{succ}(x) \cup \text{succ}(t)$
<proof>

lemma *arity_mem_eclose_fm* :

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

$\langle proof \rangle$

lemma *arity_is_eclose_fm* :

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

lemma *eclose_n1arity_fm* :

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

lemma *eclose_n2arity_fm* :

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

lemma *arity_ecloseN_fm* :

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

lemma *arity_frecl_fm* :

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

lemma *arity_Collect_fm* :

assumes $x \in nat$ $y \in nat$ $p \in \text{formula}$
shows $\text{arity}(\text{Collect_fm}(x,p,y)) = \text{succ}(x) \cup \text{succ}(y) \cup \text{pred}(\text{arity}(p))$
 $\langle proof \rangle$

end

17 The definition of forces

theory *Forces_Definition* **imports** *Arities FreclR Synthetic_Definition* **begin**

This is the core of our development.

17.1 The relation *frecl*

definition

freclP :: $[i \Rightarrow o, i] \Rightarrow o$ **where**
 $\text{freclP}(M,xy) \equiv (\exists x[M]. \exists y[M]. \text{pair}(M,x,y,xy) \wedge \text{is_freclR}(M,x,y))$

definition

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

lemma *arity_freclP_fm* :

$a \in nat \implies \text{arity}(\text{freclP_fm}(a)) = \text{succ}(a)$
 $\langle proof \rangle$

lemma *frecrelP_fm_type*[TC] :
 $a \in \text{nat} \implies \text{frecrelP_fm}(a) \in \text{formula}$
 ⟨proof⟩

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

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

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

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

lemma *frecrel_fm_type*[TC] :
 $\llbracket a \in \text{nat}; b \in \text{nat} \rrbracket \implies \text{frecrel_fm}(a, b) \in \text{formula}$
 ⟨proof⟩

lemma *arity_frecrel_fm* :
assumes $a \in \text{nat} \ b \in \text{nat}$
shows $\text{arity}(\text{frecrel_fm}(a, b)) = \text{succ}(a) \cup \text{succ}(b)$
 ⟨proof⟩

lemma *sats_frecrel_fm* :
assumes
 $a \in \text{nat} \ r \in \text{nat} \ \text{env} \in \text{list}(A)$
shows
 $\text{sats}(A, \text{frecrel_fm}(a, r), \text{env})$
 $\longleftrightarrow \text{is_frecrel}(\#\#A, \text{nth}(a, \text{env}), \text{nth}(r, \text{env}))$
 ⟨proof⟩

lemma *is_frecrel_iff_sats*:
assumes
 $\text{nth}(a, \text{env}) = aa \ \text{nth}(r, \text{env}) = rr \ a \in \text{nat} \ r \in \text{nat} \ \text{env} \in \text{list}(A)$
shows
 $\text{is_frecrel}(\#\#A, aa, rr) \longleftrightarrow \text{sats}(A, \text{frecrel_fm}(a, r), \text{env})$
 ⟨proof⟩

definition

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

lemma names_belowsD:

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

definition

$is_names_below :: [i \Rightarrow o, i, i, i] \Rightarrow o$ **where**
 $is_names_below(M,P,x,nb) \equiv \exists p1[M]. \exists p0[M]. \exists t[M]. \exists ec[M].$
 $is_ecloseN(M,ec,x) \wedge number2(M,t) \wedge cartprod(M,ec,P,p0) \wedge cart-$
 $prod(M,ec,p0,p1)$
 $\wedge cartprod(M,t,p1,nb)$

definition

$number2_fm :: i \Rightarrow i$ **where**
 $number2_fm(a) \equiv Exists(And(number1_fm(0), succ_fm(0, succ(a))))$

lemma number2_fm_type[TC] :

$a \in nat \Longrightarrow number2_fm(a) \in formula$
 $\langle proof \rangle$

lemma number2arity_fm :

$a \in nat \Longrightarrow arity(number2_fm(a)) = succ(a)$
 $\langle proof \rangle$

lemma sats_number2_fm [simp]:

$\llbracket x \in nat; env \in list(A) \rrbracket$
 $\Longrightarrow sats(A, number2_fm(x), env) \longleftrightarrow number2(\#\#A, nth(x,env))$
 $\langle proof \rangle$

definition

$is_names_below_fm :: [i, i, i] \Rightarrow i$ **where**
 $is_names_below_fm(P,x,nb) \equiv Exists(Exists(Exists(Exists($
 $And(ecloseN_fm(0,x \#\# 4), And(number2_fm(1),$
 $And(cartprod_fm(0,P \#\# 4,2), And(cartprod_fm(0,2,3), cartprod_fm(1,3,nb \#\# 4))))))))$

lemma arity_is_names_below_fm :

$\llbracket P \in nat; x \in nat; nb \in nat \rrbracket \Longrightarrow arity(is_names_below_fm(P,x,nb)) = succ(P) \cup$
 $succ(x) \cup succ(nb)$
 $\langle proof \rangle$

lemma is_names_below_fm_type[TC]:

$\llbracket P \in nat; x \in nat; nb \in nat \rrbracket \Longrightarrow is_names_below_fm(P,x,nb) \in formula$

<proof>

lemma *sats_is_names_below_fm* :

assumes

$P \in \text{nat } x \in \text{nat } nb \in \text{nat } env \in \text{list}(A)$

shows

$\text{sats}(A, \text{is_names_below_fm}(P, x, nb), env)$

$\longleftrightarrow \text{is_names_below}(\#\#A, \text{nth}(P, env), \text{nth}(x, env), \text{nth}(nb, env))$

<proof>

definition

$\text{is_tuple} :: [i \Rightarrow o, i, i, i, i] \Rightarrow o$ **where**

$\text{is_tuple}(M, z, t1, t2, p, t) \equiv \exists t1t2p[M]. \exists t2p[M]. \text{pair}(M, t2, p, t2p) \wedge \text{pair}(M, t1, t2p, t1t2p)$

\wedge

$\text{pair}(M, z, t1t2p, t)$

definition

$\text{is_tuple_fm} :: [i, i, i, i, i] \Rightarrow i$ **where**

$\text{is_tuple_fm}(z, t1, t2, p, tup) = \text{Exists}(\text{Exists}(\text{And}(\text{pair_fm}(t2 \#+ 2, p \#+ 2, 0), \text{And}(\text{pair_fm}(t1 \#+ 2, 0, 1), \text{pair_fm}(z \#+ 2, 1, tup \#+ 2))))))$

lemma *arity_is_tuple_fm* : $\llbracket z \in \text{nat} ; t1 \in \text{nat} ; t2 \in \text{nat} ; p \in \text{nat} ; tup \in \text{nat} \rrbracket \Longrightarrow$

$\text{arity}(\text{is_tuple_fm}(z, t1, t2, p, tup)) = \bigcup \{ \text{succ}(z), \text{succ}(t1), \text{succ}(t2), \text{succ}(p), \text{succ}(tup) \}$

<proof>

lemma *is_tuple_fm_type[TC]* :

$z \in \text{nat} \Longrightarrow t1 \in \text{nat} \Longrightarrow t2 \in \text{nat} \Longrightarrow p \in \text{nat} \Longrightarrow tup \in \text{nat} \Longrightarrow \text{is_tuple_fm}(z, t1, t2, p, tup) \in \text{formula}$

<proof>

lemma *sats_is_tuple_fm* :

assumes

$z \in \text{nat } t1 \in \text{nat } t2 \in \text{nat } p \in \text{nat } tup \in \text{nat } env \in \text{list}(A)$

shows

$\text{sats}(A, \text{is_tuple_fm}(z, t1, t2, p, tup), env)$

$\longleftrightarrow \text{is_tuple}(\#\#A, \text{nth}(z, env), \text{nth}(t1, env), \text{nth}(t2, env), \text{nth}(p, env), \text{nth}(tup, env))$

<proof>

lemma *is_tuple_iff_sats*:

assumes

$\text{nth}(a, env) = aa \text{ nth}(b, env) = bb \text{ nth}(c, env) = cc \text{ nth}(d, env) = dd \text{ nth}(e, env) = ee$

$a \in \text{nat } b \in \text{nat } c \in \text{nat } d \in \text{nat } e \in \text{nat } env \in \text{list}(A)$

shows

$\text{is_tuple}(\#\#A, aa, bb, cc, dd, ee) \longleftrightarrow \text{sats}(A, \text{is_tuple_fm}(a, b, c, d, e), env)$

<proof>

17.2 Definition of forces for equality and membership

definition

$$\begin{aligned}
 eq_case &:: [i,i,i,i,i,i] \Rightarrow o \text{ where} \\
 eq_case(t1,t2,p,P,leq,f) &\equiv \forall s. s \in domain(t1) \cup domain(t2) \longrightarrow \\
 &(\forall q. q \in P \wedge \langle q,p \rangle \in leq \longrightarrow (f' \langle 1,s,t1,q \rangle = 1 \iff f' \langle 1,s,t2,q \rangle = 1))
 \end{aligned}$$

definition

$$\begin{aligned}
 is_eq_case &:: [i \Rightarrow o, i, i, i, i, i, i] \Rightarrow o \text{ where} \\
 is_eq_case(M,t1,t2,p,P,leq,f) &\equiv \\
 \forall s[M]. (\exists d[M]. is_domain(M,t1,d) \wedge s \in d) \vee (\exists d[M]. is_domain(M,t2,d) \wedge \\
 s \in d) \\
 \longrightarrow (\forall q[M]. q \in P \wedge (\exists qp[M]. pair(M,q,p,qp) \wedge qp \in leq) \longrightarrow \\
 (\exists ost1q[M]. \exists ost2q[M]. \exists o[M]. \exists vf1[M]. \exists vf2[M]. \\
 is_tuple(M,o,s,t1,q,ost1q) \wedge \\
 is_tuple(M,o,s,t2,q,ost2q) \wedge number1(M,o) \wedge \\
 fun_apply(M,f,ost1q,vf1) \wedge fun_apply(M,f,ost2q,vf2) \wedge \\
 (vf1 = o \iff vf2 = o)))
 \end{aligned}$$

definition

$$\begin{aligned}
 mem_case &:: [i,i,i,i,i,i] \Rightarrow o \text{ where} \\
 mem_case(t1,t2,p,P,leq,f) &\equiv \forall v \in P. \langle v,p \rangle \in leq \longrightarrow \\
 (\exists q. \exists s. \exists r. r \in P \wedge q \in P \wedge \langle q,v \rangle \in leq \wedge \langle s,r \rangle \in t2 \wedge \langle q,r \rangle \in leq \wedge f' \langle 0,t1,s,q \rangle \\
 = 1)
 \end{aligned}$$

definition

$$\begin{aligned}
 is_mem_case &:: [i \Rightarrow o, i, i, i, i, i, i] \Rightarrow o \text{ where} \\
 is_mem_case(M,t1,t2,p,P,leq,f) &\equiv \forall v[M]. \forall vp[M]. v \in P \wedge pair(M,v,p,vp) \wedge \\
 vp \in leq \longrightarrow \\
 (\exists q[M]. \exists s[M]. \exists r[M]. \exists qv[M]. \exists sr[M]. \exists qr[M]. \exists z[M]. \exists zt1sq[M]. \exists o[M]. \\
 r \in P \wedge q \in P \wedge pair(M,q,v,qv) \wedge pair(M,s,r,sr) \wedge pair(M,q,r,qr) \wedge \\
 empty(M,z) \wedge is_tuple(M,z,t1,s,q,zt1sq) \wedge \\
 number1(M,o) \wedge qv \in leq \wedge sr \in t2 \wedge qr \in leq \wedge fun_apply(M,f,zt1sq,o))
 \end{aligned}$$

schematic_goal sats_is_mem_case_fm_auto:

assumes

$$n1 \in nat \ n2 \in nat \ p \in nat \ P \in nat \ leq \in nat \ f \in nat \ env \in list(A)$$

shows

$$\begin{aligned}
 is_mem_case(\#\#A, nth(n1, env), nth(n2, env), nth(p, env), nth(P, env), nth(leq, \\
 env), nth(f, env)) \\
 \iff sats(A, ?imc_fm(n1, n2, p, P, leq, f), env) \\
 \langle proof \rangle
 \end{aligned}$$

$\langle ML \rangle$

lemma arity_mem_case_fm :

lemma *Hfrc_fm_type*[TC] :
 $\llbracket P \in \text{nat}; \text{leq} \in \text{nat}; \text{fnnc} \in \text{nat}; f \in \text{nat} \rrbracket \implies \text{Hfrc_fm}(P, \text{leq}, \text{fnnc}, f) \in \text{formula}$
 ⟨proof⟩

lemma *arity_Hfrc_fm* :
assumes
 $P \in \text{nat} \ \text{leq} \in \text{nat} \ \text{fnnc} \in \text{nat} \ f \in \text{nat}$
shows
 $\text{arity}(\text{Hfrc_fm}(P, \text{leq}, \text{fnnc}, f)) = \text{succ}(P) \cup \text{succ}(\text{leq}) \cup \text{succ}(\text{fnnc}) \cup \text{succ}(f)$
 ⟨proof⟩

lemma *sats_Hfrc_fm*:
assumes
 $P \in \text{nat} \ \text{leq} \in \text{nat} \ \text{fnnc} \in \text{nat} \ f \in \text{nat} \ \text{env} \in \text{list}(A)$
shows
 $\text{sats}(A, \text{Hfrc_fm}(P, \text{leq}, \text{fnnc}, f), \text{env})$
 $\longleftrightarrow \text{is_Hfrc}(\#\#A, \text{nth}(P, \text{env}), \text{nth}(\text{leq}, \text{env}), \text{nth}(\text{fnnc}, \text{env}), \text{nth}(f, \text{env}))$
 ⟨proof⟩

lemma *Hfrc_iff_sats*:
assumes
 $P \in \text{nat} \ \text{leq} \in \text{nat} \ \text{fnnc} \in \text{nat} \ f \in \text{nat} \ \text{env} \in \text{list}(A)$
 $\text{nth}(P, \text{env}) = PP \ \text{nth}(\text{leq}, \text{env}) = \text{lleq} \ \text{nth}(\text{fnnc}, \text{env}) = \text{ffnnc} \ \text{nth}(f, \text{env}) = \text{ff}$
shows
 $\text{is_Hfrc}(\#\#A, PP, \text{lleq}, \text{ffnnc}, \text{ff})$
 $\longleftrightarrow \text{sats}(A, \text{Hfrc_fm}(P, \text{leq}, \text{fnnc}, f), \text{env})$
 ⟨proof⟩

definition
 $\text{is_Hfrc_at} :: [i \Rightarrow o, i, i, i, i, i] \Rightarrow o$ **where**
 $\text{is_Hfrc_at}(M, P, \text{leq}, \text{fnnc}, f, z) \equiv$
 $(\text{empty}(M, z) \wedge \neg \text{is_Hfrc}(M, P, \text{leq}, \text{fnnc}, f))$
 $\vee (\text{number1}(M, z) \wedge \text{is_Hfrc}(M, P, \text{leq}, \text{fnnc}, f))$

definition
 $\text{Hfrc_at_fm} :: [i, i, i, i, i] \Rightarrow i$ **where**
 $\text{Hfrc_at_fm}(P, \text{leq}, \text{fnnc}, f, z) \equiv \text{Or}(\text{And}(\text{empty_fm}(z), \text{Neg}(\text{Hfrc_fm}(P, \text{leq}, \text{fnnc}, f))),$
 $\text{And}(\text{number1_fm}(z), \text{Hfrc_fm}(P, \text{leq}, \text{fnnc}, f)))$

lemma *arity_Hfrc_at_fm* :
assumes
 $P \in \text{nat} \ \text{leq} \in \text{nat} \ \text{fnnc} \in \text{nat} \ f \in \text{nat} \ z \in \text{nat}$
shows
 $\text{arity}(\text{Hfrc_at_fm}(P, \text{leq}, \text{fnnc}, f, z)) = \text{succ}(P) \cup \text{succ}(\text{leq}) \cup \text{succ}(\text{fnnc}) \cup \text{succ}(f)$
 $\cup \text{succ}(z)$
 ⟨proof⟩

lemma *Hfrc_at_fm_type*[TC] :

$\llbracket P \in \text{nat}; \text{leq} \in \text{nat}; \text{fnnc} \in \text{nat}; f \in \text{nat}; z \in \text{nat} \rrbracket \implies \text{Hfrc_at_fm}(P, \text{leq}, \text{fnnc}, f, z) \in \text{formula}$
 $\langle \text{proof} \rangle$

lemma *sats_Hfrc_at_fm*:

assumes

$P \in \text{nat} \ \text{leq} \in \text{nat} \ \text{fnnc} \in \text{nat} \ f \in \text{nat} \ z \in \text{nat} \ \text{env} \in \text{list}(A)$

shows

$\text{sats}(A, \text{Hfrc_at_fm}(P, \text{leq}, \text{fnnc}, f, z), \text{env})$

$\longleftrightarrow \text{is_Hfrc_at}(\#\#A, \text{nth}(P, \text{env}), \text{nth}(\text{leq}, \text{env}), \text{nth}(\text{fnnc}, \text{env}), \text{nth}(f, \text{env}), \text{nth}(z, \text{env}))$

$\langle \text{proof} \rangle$

lemma *is_Hfrc_at_iff_sats*:

assumes

$P \in \text{nat} \ \text{leq} \in \text{nat} \ \text{fnnc} \in \text{nat} \ f \in \text{nat} \ z \in \text{nat} \ \text{env} \in \text{list}(A)$

$\text{nth}(P, \text{env}) = PP \ \text{nth}(\text{leq}, \text{env}) = \text{lleq} \ \text{nth}(\text{fnnc}, \text{env}) = \text{ffnnc}$

$\text{nth}(f, \text{env}) = \text{ff} \ \text{nth}(z, \text{env}) = \text{zz}$

shows

$\text{is_Hfrc_at}(\#\#A, PP, \text{lleq}, \text{ffnnc}, \text{ff}, \text{zz})$

$\longleftrightarrow \text{sats}(A, \text{Hfrc_at_fm}(P, \text{leq}, \text{fnnc}, f, z), \text{env})$

$\langle \text{proof} \rangle$

lemma *arity_tran_closure_fm* :

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

$\langle \text{proof} \rangle$

17.3 The well-founded relation *forcere*

definition

forcere :: $i \Rightarrow i \Rightarrow i$ **where**

$\text{forcere}(P, x) \equiv \text{frecrel}(\text{names_below}(P, x)) \hat{+}$

definition

is_forcere :: $[i \Rightarrow o, i, i, i] \Rightarrow o$ **where**

$\text{is_forcere}(M, P, x, z) \equiv \exists r[M]. \exists \text{nb}[M]. \text{tran_closure}(M, r, z) \wedge$
 $(\text{is_names_below}(M, P, x, \text{nb}) \wedge \text{is_frecrel}(M, \text{nb}, r))$

definition

forcere_fm :: $i \Rightarrow i \Rightarrow i \Rightarrow i$ **where**

$\text{forcere_fm}(p, x, z) \equiv \text{Exists}(\text{Exists}(\text{And}(\text{trans_closure_fm}(1, z\#+2),$
 $\text{And}(\text{is_names_below_fm}(p\#+2, x\#+2, 0), \text{frecrel_fm}(0, 1))))))$

lemma *arity_forcere_fm*:

$\llbracket p \in \text{nat}; x \in \text{nat}; z \in \text{nat} \rrbracket \implies \text{arity}(\text{forcere_fm}(p, x, z)) = \text{succ}(p) \cup \text{succ}(x) \cup \text{succ}(z)$

$\langle \text{proof} \rangle$

lemma *forcere_fm_type[TC]*:

$\llbracket p \in \text{nat}; x \in \text{nat}; z \in \text{nat} \rrbracket \implies \text{forcere_fm}(p, x, z) \in \text{formula}$

$\langle \text{proof} \rangle$

lemma *sats_forcerel_fm*:

assumes

$p \in \text{nat } x \in \text{nat } z \in \text{nat } \text{env} \in \text{list}(A)$

shows

$\text{sats}(A, \text{forcerel_fm}(p, x, z), \text{env}) \longleftrightarrow \text{is_forcerel}(\#\#A, \text{nth}(p, \text{env}), \text{nth}(x, \text{env}), \text{nth}(z, \text{env}))$
 $\langle \text{proof} \rangle$

17.4 *frc_at*, forcing for atomic formulas

definition

$\text{frc_at} :: [i, i, i] \Rightarrow i$ **where**

$\text{frc_at}(P, \text{leq}, \text{fnnc}) \equiv \text{wfrec}(\text{frecrel}(\text{names_below}(P, \text{fnnc})), \text{fnnc},$
 $\lambda x f. \text{bool_of_o}(\text{Hfrc}(P, \text{leq}, x, f)))$

definition

$\text{is_frc_at} :: [i \Rightarrow o, i, i, i, i] \Rightarrow o$ **where**

$\text{is_frc_at}(M, P, \text{leq}, x, z) \equiv \exists r[M]. \text{is_forcerel}(M, P, x, r) \wedge$
 $\text{is_wfrec}(M, \text{is_Hfrc_at}(M, P, \text{leq}), r, x, z)$

definition

$\text{frc_at_fm} :: [i, i, i, i] \Rightarrow i$ **where**

$\text{frc_at_fm}(p, l, x, z) \equiv \text{Exists}(\text{And}(\text{forcerel_fm}(\text{succ}(p), \text{succ}(x), 0),$
 $\text{is_wfrec_fm}(\text{Hfrc_at_fm}(6\#+p, 6\#+l, 2, 1, 0), 0, \text{succ}(x), \text{succ}(z))))$

lemma *frc_at_fm_type* [TC] :

$\llbracket p \in \text{nat}; l \in \text{nat}; x \in \text{nat}; z \in \text{nat} \rrbracket \Longrightarrow \text{frc_at_fm}(p, l, x, z) \in \text{formula}$
 $\langle \text{proof} \rangle$

lemma *arity_frc_at_fm* :

assumes $p \in \text{nat } l \in \text{nat } x \in \text{nat } z \in \text{nat}$

shows $\text{arity}(\text{frc_at_fm}(p, l, x, z)) = \text{succ}(p) \cup \text{succ}(l) \cup \text{succ}(x) \cup \text{succ}(z)$
 $\langle \text{proof} \rangle$

lemma *sats_frc_at_fm* :

assumes

$p \in \text{nat } l \in \text{nat } i \in \text{nat } j \in \text{nat } \text{env} \in \text{list}(A) \ i < \text{length}(\text{env}) \ j < \text{length}(\text{env})$

shows

$\text{sats}(A, \text{frc_at_fm}(p, l, i, j), \text{env}) \longleftrightarrow$
 $\text{is_frc_at}(\#\#A, \text{nth}(p, \text{env}), \text{nth}(l, \text{env}), \text{nth}(i, \text{env}), \text{nth}(j, \text{env}))$
 $\langle \text{proof} \rangle$

definition

$\text{forces_eq}' :: [i, i, i, i, i] \Rightarrow o$ **where**

$\text{forces_eq}'(P, l, p, t1, t2) \equiv \text{frc_at}(P, l, \langle 0, t1, t2, p \rangle) = 1$

definition

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

definition

$forces_neq' :: [i,i,i,i] \Rightarrow o$ **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

$forces_nmem' :: [i,i,i,i] \Rightarrow o$ **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

$is_forces_eq' :: [i \Rightarrow o, i, i, i, i] \Rightarrow o$ **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

$is_forces_mem' :: [i \Rightarrow o, i, i, i, i] \Rightarrow o$ **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

$is_forces_neq' :: [i \Rightarrow o, i, i, i, i] \Rightarrow o$ **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

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

definition

$forces_eq_fm :: [i,i,i,i,i] \Rightarrow i$ **where**
 $forces_eq_fm(p,l,q,t1,t2) \equiv$
 $Exists(Exists(Exists(And(number1_fm(2),And(empty_fm(1),$
 $And(is_tuple_fm(1,t1\#\#3,t2\#\#3,q\#\#3,0),frc_at_fm(p\#\#3,l\#\#3,0,2)$
 $))))))$

definition

$forces_mem_fm :: [i,i,i,i,i] \Rightarrow i$ **where**
 $forces_mem_fm(p,l,q,t1,t2) \equiv Exists(Exists(And(number1_fm(1),$
 $And(is_tuple_fm(1,t1\#\#2,t2\#\#2,q\#\#2,0),frc_at_fm(p\#\#2,l\#\#2,0,1))))))$

definition

$forces_neq_fm :: [i,i,i,i,i] \Rightarrow i$ **where**
 $forces_neq_fm(p,l,q,t1,t2) \equiv Neg(Exists(Exists(And(Member(1,p\#\#2),$
 $And(pair_fm(1,q\#\#2,0),And(Member(0,l\#\#2),forces_eq_fm(p\#\#2,l\#\#2,1,t1\#\#2,t2\#\#2))))))))$

definition

$forces_nmem_fm :: [i,i,i,i] \Rightarrow i$ **where**
 $forces_nmem_fm(p,l,q,t1,t2) \equiv Neg(Exists(Exists(And(Member(1,p\#+2),$
 $And(pair_fm(1,q\#+2,0),And(Member(0,l\#+2),forces_mem_fm(p\#+2,l\#+2,1,t1\#+2,t2\#+2))))))))$

lemma $forces_eq_fm_type$ [TC]:

$\llbracket p \in nat; l \in nat; q \in nat; t1 \in nat; t2 \in nat \rrbracket \Longrightarrow forces_eq_fm(p,l,q,t1,t2) \in formula$
 $\langle proof \rangle$

lemma $forces_mem_fm_type$ [TC]:

$\llbracket p \in nat; l \in nat; q \in nat; t1 \in nat; t2 \in nat \rrbracket \Longrightarrow forces_mem_fm(p,l,q,t1,t2) \in formula$
 $\langle proof \rangle$

lemma $forces_neq_fm_type$ [TC]:

$\llbracket p \in nat; l \in nat; q \in nat; t1 \in nat; t2 \in nat \rrbracket \Longrightarrow forces_neq_fm(p,l,q,t1,t2) \in formula$
 $\langle proof \rangle$

lemma $forces_nmem_fm_type$ [TC]:

$\llbracket p \in nat; l \in nat; q \in nat; t1 \in nat; t2 \in nat \rrbracket \Longrightarrow forces_nmem_fm(p,l,q,t1,t2) \in formula$
 $\langle proof \rangle$

lemma $arity_forces_eq_fm$:

$p \in nat \Longrightarrow l \in nat \Longrightarrow q \in nat \Longrightarrow t1 \in nat \Longrightarrow t2 \in nat \Longrightarrow$
 $arity(forces_eq_fm(p,l,q,t1,t2)) = succ(t1) \cup succ(t2) \cup succ(q) \cup succ(p) \cup$
 $succ(l)$
 $\langle proof \rangle$

lemma $arity_forces_mem_fm$:

$p \in nat \Longrightarrow l \in nat \Longrightarrow q \in nat \Longrightarrow t1 \in nat \Longrightarrow t2 \in nat \Longrightarrow$
 $arity(forces_mem_fm(p,l,q,t1,t2)) = succ(t1) \cup succ(t2) \cup succ(q) \cup succ(p)$
 $\cup succ(l)$
 $\langle proof \rangle$

lemma $sats_forces_eq'_fm$:

assumes $p \in nat \ l \in nat \ q \in nat \ t1 \in nat \ t2 \in nat \ env \in list(M)$
shows $sats(M, forces_eq_fm(p,l,q,t1,t2), env) \longleftrightarrow$
 $is_forces_eq'(\#\#M, nth(p, env), nth(l, env), nth(q, env), nth(t1, env), nth(t2, env))$
 $\langle proof \rangle$

lemma $sats_forces_mem'_fm$:

assumes $p \in nat \ l \in nat \ q \in nat \ t1 \in nat \ t2 \in nat \ env \in list(M)$
shows $sats(M, forces_mem_fm(p,l,q,t1,t2), env) \longleftrightarrow$
 $is_forces_mem'(\#\#M, nth(p, env), nth(l, env), nth(q, env), nth(t1, env), nth(t2, env))$
 $\langle proof \rangle$

lemma $sats_forces_neq'_fm$:

assumes $p \in nat \ l \in nat \ q \in nat \ t1 \in nat \ t2 \in nat \ env \in list(M)$

shows $\text{sats}(M, \text{forces_neq_fm}(p, l, q, t1, t2), \text{env}) \longleftrightarrow$
 $\text{is_forces_neq}'(\#\#M, \text{nth}(p, \text{env}), \text{nth}(l, \text{env}), \text{nth}(q, \text{env}), \text{nth}(t1, \text{env}), \text{nth}(t2, \text{env}))$
 $\langle \text{proof} \rangle$

lemma $\text{sats_forces_nmem}'_fm:$

assumes $p \in \text{nat } l \in \text{nat } q \in \text{nat } t1 \in \text{nat } t2 \in \text{nat } \text{env} \in \text{list}(M)$

shows $\text{sats}(M, \text{forces_nmem_fm}(p, l, q, t1, t2), \text{env}) \longleftrightarrow$

$\text{is_forces_nmem}'(\#\#M, \text{nth}(p, \text{env}), \text{nth}(l, \text{env}), \text{nth}(q, \text{env}), \text{nth}(t1, \text{env}), \text{nth}(t2, \text{env}))$
 $\langle \text{proof} \rangle$

context forcing_data

begin

lemma $\text{fst_abs} [\text{simp}]:$

$\llbracket x \in M; y \in M \rrbracket \implies \text{is_fst}(\#\#M, x, y) \longleftrightarrow y = \text{fst}(x)$

$\langle \text{proof} \rangle$

lemma $\text{snd_abs} [\text{simp}]:$

$\llbracket x \in M; y \in M \rrbracket \implies \text{is_snd}(\#\#M, x, y) \longleftrightarrow y = \text{snd}(x)$

$\langle \text{proof} \rangle$

lemma $\text{ftype_abs} [\text{simp}] :$

$\llbracket x \in M; y \in M \rrbracket \implies \text{is_ftype}(\#\#M, x, y) \longleftrightarrow y = \text{ftype}(x) \langle \text{proof} \rangle$

lemma $\text{name1_abs} [\text{simp}] :$

$\llbracket x \in M; y \in M \rrbracket \implies \text{is_name1}(\#\#M, x, y) \longleftrightarrow y = \text{name1}(x)$

$\langle \text{proof} \rangle$

lemma $\text{snd_snd_abs}:$

$\llbracket x \in M; y \in M \rrbracket \implies \text{is_snd_snd}(\#\#M, x, y) \longleftrightarrow y = \text{snd}(\text{snd}(x))$

$\langle \text{proof} \rangle$

lemma $\text{name2_abs} [\text{simp}]:$

$\llbracket x \in M; y \in M \rrbracket \implies \text{is_name2}(\#\#M, x, y) \longleftrightarrow y = \text{name2}(x)$

$\langle \text{proof} \rangle$

lemma $\text{cond_of_abs} [\text{simp}]:$

$\llbracket x \in M; y \in M \rrbracket \implies \text{is_cond_of}(\#\#M, x, y) \longleftrightarrow y = \text{cond_of}(x)$

$\langle \text{proof} \rangle$

lemma $\text{tuple_abs} [\text{simp}]:$

$\llbracket z \in M; t1 \in M; t2 \in M; p \in M; t \in M \rrbracket \implies$

$\text{is_tuple}(\#\#M, z, t1, t2, p, t) \longleftrightarrow t = \langle z, t1, t2, p \rangle$

$\langle \text{proof} \rangle$

lemma $\text{oneN_in_M} [\text{simp}]: 1 \in M$

$\langle \text{proof} \rangle$

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

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

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

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

lemma *Hfrc_abs*:
 $\llbracket fnnc \in M; f \in M \rrbracket \implies$
 $is_Hfrc(\#\#M, P, leq, fnnc, f) \longleftrightarrow Hfrc(P, leq, fnnc, f)$
 ⟨proof⟩

lemma *Hfrc_at_abs*:
 $\llbracket fnnc \in M; f \in M; z \in M \rrbracket \implies$
 $is_Hfrc_at(\#\#M, P, leq, fnnc, f, z) \longleftrightarrow z = bool_of_o(Hfrc(P, leq, fnnc, f))$
 ⟨proof⟩

lemma *components_closed* :
 $x \in M \implies ftype(x) \in M$
 $x \in M \implies name1(x) \in M$
 $x \in M \implies name2(x) \in M$
 $x \in M \implies cond_of(x) \in M$
 ⟨proof⟩

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

lemma *is_eclose_n_abs* :

assumes $x \in M \ ec \in M$
shows $is_eclose_n(\#\#M, is_name1, ec, x) \longleftrightarrow ec = eclose_n(name1, x)$
 $is_eclose_n(\#\#M, is_name2, ec, x) \longleftrightarrow ec = eclose_n(name2, x)$
 $\langle proof \rangle$

lemma $is_ecloseN_abs$:
 $\llbracket x \in M; ec \in M \rrbracket \implies is_ecloseN(\#\#M, ec, x) \longleftrightarrow ec = ecloseN(x)$
 $\langle proof \rangle$

lemma $frecR_abs$:
 $x \in M \implies y \in M \implies frecR(x, y) \longleftrightarrow is_frecR(\#\#M, x, y)$
 $\langle proof \rangle$

lemma $frecrelP_abs$:
 $z \in M \implies frecrelP(\#\#M, z) \longleftrightarrow (\exists x y. z = \langle x, y \rangle \wedge frecR(x, y))$
 $\langle proof \rangle$

lemma $frecrel_abs$:
assumes
 $A \in M \ r \in M$
shows
 $is_frecrel(\#\#M, A, r) \longleftrightarrow r = frecrel(A)$
 $\langle proof \rangle$

lemma $frecrel_closed$:
assumes
 $x \in M$
shows
 $frecrel(x) \in M$
 $\langle proof \rangle$

lemma $field_frecrel$: $field(frecrel(names_below(P, x))) \subseteq names_below(P, x)$
 $\langle proof \rangle$

lemma $forcerelD$: $uv \in forcerel(P, x) \implies uv \in names_below(P, x) \times names_below(P, x)$
 $\langle proof \rangle$

lemma $wf_forcerel$:
 $wf(forcerel(P, x))$
 $\langle proof \rangle$

lemma $restrict_trancI_forcerel$:
assumes $frecR(w, y)$
shows $restrict(f, frecrel(names_below(P, x)) - \{\{y\}\}) 'w$
 $= restrict(f, forcerel(P, x) - \{\{y\}\}) 'w$
 $\langle proof \rangle$

lemma $names_belowI$:

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

lemma names_below_tr :
assumes $x \in \text{names_below}(P, y)$
 $y \in \text{names_below}(P, z)$
shows $x \in \text{names_below}(P, z)$
 $\langle \text{proof} \rangle$

lemma $\text{arg_into_names_below2}$:
assumes $\langle x, y \rangle \in \text{frecrel}(\text{names_below}(P, z))$
shows $x \in \text{names_below}(P, y)$
 $\langle \text{proof} \rangle$

lemma $\text{arg_into_names_below}$:
assumes $\langle x, y \rangle \in \text{frecrel}(\text{names_below}(P, z))$
shows $x \in \text{names_below}(P, x)$
 $\langle \text{proof} \rangle$

lemma $\text{forcere1_arg_into_names_below}$:
assumes $\langle x, y \rangle \in \text{forcere1}(P, z)$
shows $x \in \text{names_below}(P, x)$
 $\langle \text{proof} \rangle$

lemma names_below_mono :
assumes $\langle x, y \rangle \in \text{frecrel}(\text{names_below}(P, z))$
shows $\text{names_below}(P, x) \subseteq \text{names_below}(P, y)$
 $\langle \text{proof} \rangle$

lemma frecrel_mono :
assumes $\langle x, y \rangle \in \text{frecrel}(\text{names_below}(P, z))$
shows $\text{frecrel}(\text{names_below}(P, x)) \subseteq \text{frecrel}(\text{names_below}(P, y))$
 $\langle \text{proof} \rangle$

lemma forcere1_mono2 :
assumes $\langle x, y \rangle \in \text{frecrel}(\text{names_below}(P, z))$
shows $\text{forcere1}(P, x) \subseteq \text{forcere1}(P, y)$
 $\langle \text{proof} \rangle$

lemma forcere1_mono_aux :
assumes $\langle x, y \rangle \in \text{frecrel}(\text{names_below}(P, w)) \wedge +$
shows $\text{forcere1}(P, x) \subseteq \text{forcere1}(P, y)$
 $\langle \text{proof} \rangle$

lemma forcere1_mono :
assumes $\langle x, y \rangle \in \text{forcere1}(P, z)$
shows $\text{forcere1}(P, x) \subseteq \text{forcere1}(P, y)$
 $\langle \text{proof} \rangle$

lemma aux: $x \in \text{names_below}(P, w) \implies \langle x, y \rangle \in \text{forcerel}(P, z) \implies$
 $(y \in \text{names_below}(P, w) \longrightarrow \langle x, y \rangle \in \text{forcerel}(P, w))$
 $\langle \text{proof} \rangle$

lemma forcerel_eq :
assumes $\langle z, x \rangle \in \text{forcerel}(P, x)$
shows $\text{forcerel}(P, z) = \text{forcerel}(P, x) \cap \text{names_below}(P, z) \times \text{names_below}(P, z)$
 $\langle \text{proof} \rangle$

lemma forcerel_below_aux :
assumes $\langle z, x \rangle \in \text{forcerel}(P, x)$ $\langle u, z \rangle \in \text{forcerel}(P, x)$
shows $u \in \text{names_below}(P, z)$
 $\langle \text{proof} \rangle$

lemma forcerel_below :
assumes $\langle z, x \rangle \in \text{forcerel}(P, x)$
shows $\text{forcerel}(P, x) - \{z\} \subseteq \text{names_below}(P, z)$
 $\langle \text{proof} \rangle$

lemma relation_forcerel :
shows $\text{relation}(\text{forcerel}(P, z)) \text{ trans}(\text{forcerel}(P, z))$
 $\langle \text{proof} \rangle$

lemma Hfrc_restrict_trancl: $\text{bool_of_o}(\text{Hfrc}(P, \text{leq}, y, \text{restrict}(f, \text{frcrel}(\text{names_below}(P, x)) - \{y\})))$
 $= \text{bool_of_o}(\text{Hfrc}(P, \text{leq}, y, \text{restrict}(f, (\text{frcrel}(\text{names_below}(P, x)) \hat{+}) - \{y\})))$
 $\langle \text{proof} \rangle$

lemma frc_at_trancl: $\text{frc_at}(P, \text{leq}, z) = \text{wfrc}(\text{forcerel}(P, z), z, \lambda x f. \text{bool_of_o}(\text{Hfrc}(P, \text{leq}, x, f)))$
 $\langle \text{proof} \rangle$

lemma forcerelI1 :
assumes $n1 \in \text{domain}(b) \vee n1 \in \text{domain}(c)$ $p \in P$ $d \in P$
shows $\langle \langle 1, n1, b, p \rangle, \langle 0, b, c, d \rangle \rangle \in \text{forcerel}(P, \langle 0, b, c, d \rangle)$
 $\langle \text{proof} \rangle$

lemma forcerelI2 :
assumes $n1 \in \text{domain}(b) \vee n1 \in \text{domain}(c)$ $p \in P$ $d \in P$
shows $\langle \langle 1, n1, c, p \rangle, \langle 0, b, c, d \rangle \rangle \in \text{forcerel}(P, \langle 0, b, c, d \rangle)$
 $\langle \text{proof} \rangle$

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

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

forcereI2[*THEN vimage_singleton_iff*[*THEN iffD2*]]
forcereI3[*THEN vimage_singleton_iff*[*THEN iffD2*]]

lemma *aux_def_frc_at*:
assumes $z \in \text{forcereI}(P,x) - \{x\}$
shows $\text{wfrec}(\text{forcereI}(P,x), z, H) = \text{wfrec}(\text{forcereI}(P,z), z, H)$
<proof>

17.5 Recursive expression of *frc_at*

lemma *def_frc_at* :
assumes $p \in P$
shows
 $\text{frc_at}(P, \text{leq}, \langle \text{ft}, n1, n2, p \rangle) =$
 $\text{bool_of_o}(p \in P \wedge$
 $(\text{ft} = 0 \wedge (\forall s. s \in \text{domain}(n1) \cup \text{domain}(n2) \longrightarrow$
 $(\forall q. q \in P \wedge q \preceq p \longrightarrow (\text{frc_at}(P, \text{leq}, \langle 1, s, n1, q \rangle) = 1 \longleftrightarrow \text{frc_at}(P, \text{leq}, \langle 1, s, n2, q \rangle)$
 $= 1))))$
 $\vee \text{ft} = 1 \wedge (\forall v \in P. v \preceq p \longrightarrow$
 $(\exists q. \exists s. \exists r. r \in P \wedge q \in P \wedge q \preceq v \wedge \langle s, r \rangle \in n2 \wedge q \preceq r \wedge \text{frc_at}(P, \text{leq}, \langle 0, n1, s, q \rangle)$
 $= 1))))$
<proof>

17.6 Absoluteness of *frc_at*

lemma *trans_forcereI_t* : *trans*(*forcereI*(*P*, *x*))
<proof>

lemma *relation_forcereI_t* : *relation*(*forcereI*(*P*, *x*))
<proof>

lemma *forcereI_in_M* :
assumes
 $x \in M$
shows
 $\text{forcereI}(P,x) \in M$
<proof>

lemma *relation2_Hfrc_at_abs*:
 $\text{relation2}(\#\#M, \text{is_Hfrc_at}(\#\#M, P, \text{leq}), \lambda x f. \text{bool_of_o}(\text{Hfrc}(P, \text{leq}, x, f)))$
<proof>

lemma *Hfrc_at_closed* :
 $\forall x \in M. \forall g \in M. \text{function}(g) \longrightarrow \text{bool_of_o}(\text{Hfrc}(P, \text{leq}, x, g)) \in M$
<proof>

lemma *wfrec_Hfrc_at* :
assumes
 $X \in M$

shows
 $wfrec_replacement(\#\#M, is_Hfrc_at(\#\#M, P, leq), forcere1(P, X))$
 $\langle proof \rangle$

lemma names_below_abs :
 $\llbracket Q \in M ; x \in M ; nb \in M \rrbracket \implies is_names_below(\#\#M, Q, x, nb) \longleftrightarrow nb = names_below(Q, x)$
 $\langle proof \rangle$

lemma names_below_closed:
 $\llbracket Q \in M ; x \in M \rrbracket \implies names_below(Q, x) \in M$
 $\langle proof \rangle$

lemma names_below_productE :
assumes $Q \in M \ x \in M$
 $\bigwedge A1 \ A2 \ A3 \ A4. A1 \in M \implies A2 \in M \implies A3 \in M \implies A4 \in M \implies R(A1$
 $\times A2 \times A3 \times A4)$
shows $R(names_below(Q, x))$
 $\langle proof \rangle$

lemma forcere1_abs :
 $\llbracket x \in M ; z \in M \rrbracket \implies is_forcere1(\#\#M, P, x, z) \longleftrightarrow z = forcere1(P, x)$
 $\langle proof \rangle$

lemma frc_at_abs:
assumes $fnc \in M \ z \in M$
shows $is_frc_at(\#\#M, P, leq, fnc, z) \longleftrightarrow z = frc_at(P, leq, fnc)$
 $\langle proof \rangle$

lemma forces_eq'_abs :
 $\llbracket p \in M ; t1 \in M ; t2 \in M \rrbracket \implies is_forces_eq'(\#\#M, P, leq, p, t1, t2) \longleftrightarrow forces_eq'(P, leq, p, t1, t2)$
 $\langle proof \rangle$

lemma forces_mem'_abs :
 $\llbracket p \in M ; t1 \in M ; t2 \in M \rrbracket \implies is_forces_mem'(\#\#M, P, leq, p, t1, t2) \longleftrightarrow forces_mem'(P, leq, p, t1, t2)$
 $\langle proof \rangle$

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

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

definition

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

lemma (in forcing_data) leq_abs[simp]:

$\llbracket l \in M ; q \in M ; p \in M \rrbracket \Longrightarrow is_leq(\#\#M, l, q, p) \longleftrightarrow \langle q, p \rangle \in l$
 $\langle proof \rangle$

definition

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

lemma arity_leq_fm :

$\llbracket leq \in nat ; q \in nat ; p \in nat \rrbracket \Longrightarrow arity(leq_fm(leq, q, p)) = succ(q) \cup succ(p) \cup succ(leq)$
 $\langle proof \rangle$

lemma leq_fm_type[TC] :

$\llbracket leq \in nat ; q \in nat ; p \in nat \rrbracket \Longrightarrow leq_fm(leq, q, p) \in formula$
 $\langle proof \rangle$

lemma sats_leq_fm :

$\llbracket leq \in nat ; q \in nat ; p \in nat ; env \in list(A) \rrbracket \Longrightarrow$
 $sats(A, leq_fm(leq, q, p), env) \longleftrightarrow is_leq(\#\#A, nth(leq, env), nth(q, env), nth(p, env))$
 $\langle proof \rangle$

17.7.1 The primitive recursion

consts $forces' :: i \Rightarrow i$

primrec

$forces'(Member(x, y)) = forces_mem_fm(1, 2, 0, x\#+4, y\#+4)$
 $forces'(Equal(x, y)) = forces_eq_fm(1, 2, 0, x\#+4, y\#+4)$
 $forces'(Nand(p, q)) =$
 $Neg(Exists(And(Member(0, 2), And(leq_fm(3, 0, 1), And(ren_forces_nand(forces'(p)),$
 $ren_forces_nand(forces'(q)))))))$
 $forces'(Forall(p)) = Forall(ren_forces_forall(forces'(p)))$

definition

$forces :: i \Rightarrow i$ **where**
 $forces(\varphi) \equiv And(Member(0, 1), forces'(\varphi))$

lemma forces'_type [TC]: $\varphi \in formula \Longrightarrow forces'(\varphi) \in formula$

$\langle proof \rangle$

lemma forces_type[TC] : $\varphi \in formula \Longrightarrow forces(\varphi) \in formula$

$\langle proof \rangle$

context $forcing_data$

begin

17.8 Forcing for atomic formulas in context

definition

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

definition

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

definition

$is_forces_eq :: [i,i,i] \Rightarrow o$ **where**
 $is_forces_eq \equiv is_forces_eq'(\#\#M,P,leq)$

definition

$is_forces_mem :: [i,i,i] \Rightarrow o$ **where**
 $is_forces_mem \equiv is_forces_mem'(\#\#M,P,leq)$

lemma def_forces_eq : $p \in P \implies forces_eq(p,t1,t2) \longleftrightarrow$
 $(\forall s \in domain(t1) \cup domain(t2). \forall q. q \in P \wedge q \preceq p \longrightarrow$
 $(forces_mem(q,s,t1) \longleftrightarrow forces_mem(q,s,t2)))$
 $\langle proof \rangle$

lemma def_forces_mem : $p \in P \implies forces_mem(p,t1,t2) \longleftrightarrow$
 $(\forall v \in P. v \preceq p \longrightarrow$
 $(\exists q. \exists s. \exists r. r \in P \wedge q \in P \wedge q \preceq v \wedge \langle s,r \rangle \in t2 \wedge q \preceq r \wedge forces_eq(q,t1,s)))$
 $\langle proof \rangle$

lemma $forces_eq_abs$:
 $\llbracket p \in M ; t1 \in M ; t2 \in M \rrbracket \implies is_forces_eq(p,t1,t2) \longleftrightarrow forces_eq(p,t1,t2)$
 $\langle proof \rangle$

lemma $forces_mem_abs$:
 $\llbracket p \in M ; t1 \in M ; t2 \in M \rrbracket \implies is_forces_mem(p,t1,t2) \longleftrightarrow forces_mem(p,t1,t2)$
 $\langle proof \rangle$

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 $sats(M, forces_eq_fm(p,l,q,t1,t2), env) \longleftrightarrow$
 $is_forces_eq(nth(q,env), nth(t1,env), nth(t2,env))$
 $\langle proof \rangle$

lemma $sats_forces_mem_fm$:

assumes $p \in \text{nat } l \in \text{nat } q \in \text{nat } t1 \in \text{nat } t2 \in \text{nat } \text{env} \in \text{list}(M)$
 $\text{nth}(p, \text{env}) = P \text{ nth}(l, \text{env}) = \text{leq}$
shows $\text{sats}(M, \text{forces_mem_fm}(p, l, q, t1, t2), \text{env}) \longleftrightarrow$
 $\text{is_forces_mem}(\text{nth}(q, \text{env}), \text{nth}(t1, \text{env}), \text{nth}(t2, \text{env}))$
 $\langle \text{proof} \rangle$

definition

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

definition

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

lemma forces_neq :

$\text{forces_neq}(p, t1, t2) \longleftrightarrow \text{forces_neq}'(P, \text{leq}, p, t1, t2)$
 $\langle \text{proof} \rangle$

lemma forces_nmem :

$\text{forces_nmem}(p, t1, t2) \longleftrightarrow \text{forces_nmem}'(P, \text{leq}, p, t1, t2)$
 $\langle \text{proof} \rangle$

lemma $\text{sats_forces_Member}$:

assumes $x \in \text{nat } y \in \text{nat } \text{env} \in \text{list}(M)$
 $\text{nth}(x, \text{env}) = xx \text{ nth}(y, \text{env}) = yy \ q \in M$
shows $\text{sats}(M, \text{forces}(\text{Member}(x, y)), [q, P, \text{leq}, \text{one}]@ \text{env}) \longleftrightarrow$
 $(q \in P \wedge \text{is_forces_mem}(q, xx, yy))$
 $\langle \text{proof} \rangle$

lemma sats_forces_Equal :

assumes $x \in \text{nat } y \in \text{nat } \text{env} \in \text{list}(M)$
 $\text{nth}(x, \text{env}) = xx \text{ nth}(y, \text{env}) = yy \ q \in M$
shows $\text{sats}(M, \text{forces}(\text{Equal}(x, y)), [q, P, \text{leq}, \text{one}]@ \text{env}) \longleftrightarrow$
 $(q \in P \wedge \text{is_forces_eq}(q, xx, yy))$
 $\langle \text{proof} \rangle$

lemma sats_forces_Nand :

assumes $\varphi \in \text{formula } \psi \in \text{formula } \text{env} \in \text{list}(M) \ p \in M$
shows $\text{sats}(M, \text{forces}(\text{Nand}(\varphi, \psi)), [p, P, \text{leq}, \text{one}]@ \text{env}) \longleftrightarrow$
 $(p \in P \wedge \neg (\exists q \in M. q \in P \wedge \text{is_leq}(\#\#M, \text{leq}, q, p) \wedge$
 $(\text{sats}(M, \text{forces}'(\varphi), [q, P, \text{leq}, \text{one}]@ \text{env}) \wedge \text{sats}(M, \text{forces}'(\psi), [q, P, \text{leq}, \text{one}]@ \text{env}))))$
 $\langle \text{proof} \rangle$

lemma sats_forces_Neg :

assumes $\varphi \in \text{formula } \text{env} \in \text{list}(M) \ p \in M$
shows $\text{sats}(M, \text{forces}(\text{Neg}(\varphi)), [p, P, \text{leq}, \text{one}]@ \text{env}) \longleftrightarrow$

$(p \in P \wedge \neg(\exists q \in M. q \in P \wedge \text{is_leq}(\#\#M, \text{leq}, q, p) \wedge$
 $\text{sats}(M, \text{forces}'(\varphi), [q, P, \text{leq}, \text{one}]@env)))$
 <proof>

lemma *sats_forces_Forall* :
assumes $\varphi \in \text{formula}$ $env \in \text{list}(M)$ $p \in M$
shows $\text{sats}(M, \text{forces}(\text{Forall}(\varphi)), [p, P, \text{leq}, \text{one}]@env) \longleftrightarrow$
 $p \in P \wedge (\forall x \in M. \text{sats}(M, \text{forces}'(\varphi), [p, P, \text{leq}, \text{one}, x]@env))$
 <proof>

end

17.9 The arity of forces

lemma *arity_forces_at*:
assumes $x \in \text{nat}$ $y \in \text{nat}$
shows $\text{arity}(\text{forces}(\text{Member}(x, y))) = (\text{succ}(x) \cup \text{succ}(y)) \# + 4$
 $\text{arity}(\text{forces}(\text{Equal}(x, y))) = (\text{succ}(x) \cup \text{succ}(y)) \# + 4$
 <proof>

lemma *arity_forces'*:
assumes $\varphi \in \text{formula}$
shows $\text{arity}(\text{forces}'(\varphi)) \leq \text{arity}(\varphi) \# + 4$
 <proof>

lemma *arity_forces* :
assumes $\varphi \in \text{formula}$
shows $\text{arity}(\text{forces}(\varphi)) \leq 4 \# + \text{arity}(\varphi)$
 <proof>

lemma *arity_forces_le* :
assumes $\varphi \in \text{formula}$ $n \in \text{nat}$ $\text{arity}(\varphi) \leq n$
shows $\text{arity}(\text{forces}(\varphi)) \leq 4 \# + n$
 <proof>

end

18 The Forcing Theorems

theory *Forcing_Theorems*
imports
Forces_Definition

begin

context *forcing_data*
begin

18.1 The forcing relation in context

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

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*: $p \in P \implies forces_mem(p, t1, t2) \longleftrightarrow dense_below(\{q \in P. \exists s. \exists r. r \in P \wedge \langle s, r \rangle \in t2 \wedge q \preceq r \wedge forces_eq(q, t1, s)\}$

$, p)$

<proof>

18.2 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_eq*:

assumes $p \in P \ r \in P \ r \preceq p \ forces_eq(p, t1, t2)$

shows $forces_eq(r, t1, t2)$

<proof>

18.3 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_mem*:

assumes $p \in P \ r \in P \ r \preceq p \ forces_mem(p, t1, t2)$

shows $forces_mem(r, t1, t2)$

<proof>

18.4 Kunen 2013, Lemma IV.2.37(b)

lemma *density_mem*:

assumes $p \in P$

shows $forces_mem(p, t1, t2) \longleftrightarrow dense_below(\{q \in P. forces_mem(q, t1, t2)\}, p)$

<proof>

lemma *aux_density_eq*:

assumes

$dense_below(\{q' \in P. \forall q. q \in P \wedge q \preceq q' \implies forces_mem(q, s, t1) \longleftrightarrow forces_mem(q, s, t2)\}$

$, p)$

$forces_mem(q, s, t1) \ q \in P \ p \in P \ q \preceq p$

shows

$dense_below(\{r \in P. forces_mem(r, s, t2)\}, q)$

<proof>

lemma *density_eq*:

assumes $p \in P$

shows $\text{forces_eq}(p, t1, t2) \longleftrightarrow \text{dense_below}(\{q \in P. \text{forces_eq}(q, t1, t2)\}, p)$
<proof>

18.5 Kunen 2013, Lemma IV.2.38

lemma *not_forces_neq*:

assumes $p \in P$

shows $\text{forces_eq}(p, t1, t2) \longleftrightarrow \neg (\exists q \in P. q \preceq p \wedge \text{forces_neq}(q, t1, t2))$

<proof>

lemma *not_forces_nmem*:

assumes $p \in P$

shows $\text{forces_mem}(p, t1, t2) \longleftrightarrow \neg (\exists q \in P. q \preceq p \wedge \text{forces_nmem}(q, t1, t2))$

<proof>

lemma *sats_forces_Nand'*:

assumes

$p \in P \ \varphi \in \text{formula} \ \psi \in \text{formula} \ \text{env} \in \text{list}(M)$

shows

$M, [p, P, \text{leq}, \text{one}] @ \text{env} \models \text{forces}(\text{Nand}(\varphi, \psi)) \longleftrightarrow$

$\neg (\exists q \in M. q \in P \wedge \text{is_leq}(\#\#M, \text{leq}, q, p) \wedge$

$M, [q, P, \text{leq}, \text{one}] @ \text{env} \models \text{forces}(\varphi) \wedge$

$M, [q, P, \text{leq}, \text{one}] @ \text{env} \models \text{forces}(\psi))$

<proof>

lemma *sats_forces_Neg'*:

assumes

$p \in P \ \text{env} \in \text{list}(M) \ \varphi \in \text{formula}$

shows

$M, [p, P, \text{leq}, \text{one}] @ \text{env} \models \text{forces}(\text{Neg}(\varphi)) \longleftrightarrow$

$\neg (\exists q \in M. q \in P \wedge \text{is_leq}(\#\#M, \text{leq}, q, p) \wedge$

$M, [q, P, \text{leq}, \text{one}] @ \text{env} \models \text{forces}(\varphi))$

<proof>

lemma *sats_forces_Forall'*:

assumes

$p \in P \ \text{env} \in \text{list}(M) \ \varphi \in \text{formula}$

shows

$M, [p, P, \text{leq}, \text{one}] @ \text{env} \models \text{forces}(\text{Forall}(\varphi)) \longleftrightarrow$

$(\forall x \in M. M, [p, P, \text{leq}, \text{one}, x] @ \text{env} \models \text{forces}(\varphi))$

<proof>

18.6 The relation of forcing and atomic formulas

lemma *Forces_Equal*:

assumes

$$p \in P \ t1 \in M \ t2 \in M \ env \in list(M) \ nth(n, env) = t1 \ nth(m, env) = t2 \ n \in nat \ m \in nat$$

shows

$$(p \Vdash Equal(n, m) \ env) \longleftrightarrow forces_eq(p, t1, t2)$$

<proof>

lemma *Forces_Member*:

assumes

$$p \in P \ t1 \in M \ t2 \in M \ env \in list(M) \ nth(n, env) = t1 \ nth(m, env) = t2 \ n \in nat \ m \in nat$$

shows

$$(p \Vdash Member(n, m) \ env) \longleftrightarrow forces_mem(p, t1, t2)$$

<proof>

lemma *Forces_Neg*:

assumes

$$p \in P \ env \in list(M) \ \varphi \in formula$$

shows

$$(p \Vdash Neg(\varphi) \ env) \longleftrightarrow \neg(\exists q \in M. q \in P \wedge q \preceq p \wedge (q \Vdash \varphi \ env))$$

<proof>

18.7 The relation of forcing and connectives

lemma *Forces_Nand*:

assumes

$$p \in P \ env \in list(M) \ \varphi \in formula \ \psi \in formula$$

shows

$$(p \Vdash Nand(\varphi, \psi) \ env) \longleftrightarrow \neg(\exists q \in M. q \in P \wedge q \preceq p \wedge (q \Vdash \varphi \ env) \wedge (q \Vdash \psi \ env))$$

<proof>

lemma *Forces_And_aux*:

assumes

$$p \in P \ env \in list(M) \ \varphi \in formula \ \psi \in formula$$

shows

$$p \Vdash And(\varphi, \psi) \ env \longleftrightarrow (\forall q \in M. q \in P \wedge q \preceq p \longrightarrow (\exists r \in M. r \in P \wedge r \preceq q \wedge (r \Vdash \varphi \ env) \wedge (r \Vdash \psi \ env)))$$

<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

$(p \Vdash \text{Forall}(\varphi) \text{ env}) \longleftrightarrow (\forall x \in M. (p \Vdash \varphi ([x] @ \text{env})))$
 $\langle \text{proof} \rangle$

bundle $\text{some_rules} = \text{elem_of_val_pair} [\text{dest}] \text{SepReplace_iff} [\text{simp del}] \text{SepReplace_iff} [\text{iff}]$

context

includes some_rules

begin

lemma $\text{elem_of_valI}: \exists \vartheta. \exists p \in P. p \in G \wedge \langle \vartheta, p \rangle \in \pi \wedge \text{val}(G, \vartheta) = x \implies x \in \text{val}(G, \pi)$
 $\langle \text{proof} \rangle$

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

lemma $\text{left_in_M} : \tau \in M \implies \langle a, b \rangle \in \tau \implies a \in M$
 $\langle \text{proof} \rangle$

18.8 Kunen 2013, Lemma IV.2.29

lemma $\text{generic_inter_dense_below}$:

assumes $D \in M \ M_generic(G) \ \text{dense_below}(D, p) \ p \in G$

shows $D \cap G \neq \emptyset$

$\langle \text{proof} \rangle$

18.9 Auxiliary results for Lemma IV.2.40(a)

lemma $\text{IV240a_mem_Collect}$:

assumes

$\pi \in M \ \tau \in M$

shows

$\{q \in P. \exists \sigma. \exists r. r \in P \wedge \langle \sigma, r \rangle \in \tau \wedge q \preceq r \wedge \text{forces_eq}(q, \pi, \sigma)\} \in M$

$\langle \text{proof} \rangle$

lemma IV240a_mem :

assumes

$M_generic(G) \ p \in G \ \pi \in M \ \tau \in M \ \text{forces_mem}(p, \pi, \tau)$

$\bigwedge q \ \sigma. q \in P \implies q \in G \implies \sigma \in \text{domain}(\tau) \implies \text{forces_eq}(q, \pi, \sigma) \implies$
 $\text{val}(G, \pi) = \text{val}(G, \sigma)$

shows

$\text{val}(G, \pi) \in \text{val}(G, \tau)$

$\langle \text{proof} \rangle$

lemma $\text{refl_forces_eq}: p \in P \implies \text{forces_eq}(p, x, x)$

$\langle \text{proof} \rangle$

lemma *forces_memI*: $\langle \sigma, \tau \rangle \in \tau \implies p \in P \implies r \in P \implies p \preceq r \implies \text{forces_mem}(p, \sigma, \tau)$
 $\langle \text{proof} \rangle$

lemma *IV240a_eq_1st_incl*:

assumes

$M_generic(G) \ p \in G \ \text{forces_eq}(p, \tau, \vartheta)$

and

$IH: \bigwedge q \ \sigma. \ q \in P \implies q \in G \implies \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta) \implies$
 $(\text{forces_mem}(q, \sigma, \tau) \longrightarrow \text{val}(G, \sigma) \in \text{val}(G, \tau)) \wedge$
 $(\text{forces_mem}(q, \sigma, \vartheta) \longrightarrow \text{val}(G, \sigma) \in \text{val}(G, \vartheta))$

shows

$\text{val}(G, \tau) \subseteq \text{val}(G, \vartheta)$

$\langle \text{proof} \rangle$

lemma *IV240a_eq_2nd_incl*:

assumes

$M_generic(G) \ p \in G \ \text{forces_eq}(p, \tau, \vartheta)$

and

$IH: \bigwedge q \ \sigma. \ q \in P \implies q \in G \implies \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta) \implies$
 $(\text{forces_mem}(q, \sigma, \tau) \longrightarrow \text{val}(G, \sigma) \in \text{val}(G, \tau)) \wedge$
 $(\text{forces_mem}(q, \sigma, \vartheta) \longrightarrow \text{val}(G, \sigma) \in \text{val}(G, \vartheta))$

shows

$\text{val}(G, \vartheta) \subseteq \text{val}(G, \tau)$

$\langle \text{proof} \rangle$

lemma *IV240a_eq*:

assumes

$M_generic(G) \ p \in G \ \text{forces_eq}(p, \tau, \vartheta)$

and

$IH: \bigwedge q \ \sigma. \ q \in P \implies q \in G \implies \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta) \implies$
 $(\text{forces_mem}(q, \sigma, \tau) \longrightarrow \text{val}(G, \sigma) \in \text{val}(G, \tau)) \wedge$
 $(\text{forces_mem}(q, \sigma, \vartheta) \longrightarrow \text{val}(G, \sigma) \in \text{val}(G, \vartheta))$

shows

$\text{val}(G, \tau) = \text{val}(G, \vartheta)$

$\langle \text{proof} \rangle$

18.10 Induction on names

lemma *core_induction*:

assumes

$\bigwedge \tau \ \vartheta \ p. \ p \in P \implies \llbracket \bigwedge q \ \sigma. \ [q \in P ; \sigma \in \text{domain}(\vartheta)] \implies Q(\vartheta, \tau, \sigma, q) \rrbracket \implies Q(1, \tau, \vartheta, p)$
 $\bigwedge \tau \ \vartheta \ p. \ p \in P \implies \llbracket \bigwedge q \ \sigma. \ [q \in P ; \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta)] \implies Q(1, \sigma, \tau, q)$
 $\wedge Q(1, \sigma, \vartheta, q) \rrbracket \implies Q(\vartheta, \tau, \vartheta, p)$

$ft \in 2 p \in P$
shows
 $Q(ft, \tau, \vartheta, p)$
 $\langle proof \rangle$

lemma forces_induction_with_conds:

assumes
 $\bigwedge \tau \vartheta p. p \in P \implies \llbracket \bigwedge q \sigma. [q \in P ; \sigma \in \text{domain}(\vartheta)] \implies Q(q, \tau, \sigma) \rrbracket \implies R(p, \tau, \vartheta)$
 $\bigwedge \tau \vartheta p. p \in P \implies \llbracket \bigwedge q \sigma. [q \in P ; \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta)] \implies R(q, \sigma, \tau) \wedge R(q, \sigma, \vartheta) \rrbracket \implies Q(p, \tau, \vartheta)$
 $p \in P$
shows
 $Q(p, \tau, \vartheta) \wedge R(p, \tau, \vartheta)$
 $\langle proof \rangle$

lemma forces_induction:

assumes
 $\bigwedge \tau \vartheta. \llbracket \bigwedge \sigma. \sigma \in \text{domain}(\vartheta) \implies Q(\tau, \sigma) \rrbracket \implies R(\tau, \vartheta)$
 $\bigwedge \tau \vartheta. \llbracket \bigwedge \sigma. \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta) \implies R(\sigma, \tau) \wedge R(\sigma, \vartheta) \rrbracket \implies Q(\tau, \vartheta)$
shows
 $Q(\tau, \vartheta) \wedge R(\tau, \vartheta)$
 $\langle proof \rangle$

18.11 Lemma IV.2.40(a), in full

lemma IV240a:

assumes
 $M_generic(G)$
shows
 $(\tau \in M \longrightarrow \vartheta \in M \longrightarrow (\forall p \in G. \text{forces_eq}(p, \tau, \vartheta) \longrightarrow \text{val}(G, \tau) = \text{val}(G, \vartheta))) \wedge$
 $(\tau \in M \longrightarrow \vartheta \in M \longrightarrow (\forall p \in G. \text{forces_mem}(p, \tau, \vartheta) \longrightarrow \text{val}(G, \tau) \in \text{val}(G, \vartheta)))$
 $(\text{is } ?Q(\tau, \vartheta) \wedge ?R(\tau, \vartheta))$
 $\langle proof \rangle$

18.12 Lemma IV.2.40(b)

lemma IV240b_mem:

assumes
 $M_generic(G) \text{ val}(G, \pi) \in \text{val}(G, \tau) \pi \in M \tau \in M$
and
 $IH: \bigwedge \sigma. \sigma \in \text{domain}(\tau) \implies \text{val}(G, \pi) = \text{val}(G, \sigma) \implies$
 $\exists p \in G. \text{forces_eq}(p, \pi, \sigma)$
shows
 $\exists p \in G. \text{forces_mem}(p, \pi, \tau)$
 $\langle proof \rangle$

end

lemma Collect_forces_eq_in_M:

assumes $\tau \in M \vartheta \in M$

shows $\{p \in P. \text{forces_eq}(p, \tau, \vartheta)\} \in M$
 $\langle \text{proof} \rangle$

lemma *IV240b_eq_Collects*:

assumes $\tau \in M \ \vartheta \in M$

shows $\{p \in P. \exists \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta). \text{forces_mem}(p, \sigma, \tau) \wedge \text{forces_nmem}(p, \sigma, \vartheta)\} \in M$
and

$\{p \in P. \exists \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta). \text{forces_nmem}(p, \sigma, \tau) \wedge \text{forces_mem}(p, \sigma, \vartheta)\} \in M$
 $\langle \text{proof} \rangle$

lemma *IV240b_eq*:

assumes

$M_generic(G) \ \text{val}(G, \tau) = \text{val}(G, \vartheta) \ \tau \in M \ \vartheta \in M$

and

$IH: \bigwedge \sigma. \sigma \in \text{domain}(\tau) \cup \text{domain}(\vartheta) \implies$

$(\text{val}(G, \sigma) \in \text{val}(G, \tau) \longrightarrow (\exists q \in G. \text{forces_mem}(q, \sigma, \tau))) \wedge$

$(\text{val}(G, \sigma) \in \text{val}(G, \vartheta) \longrightarrow (\exists q \in G. \text{forces_mem}(q, \sigma, \vartheta)))$

shows

$\exists p \in G. \text{forces_eq}(p, \tau, \vartheta)$

$\langle \text{proof} \rangle$

lemma *IV240b*:

assumes

$M_generic(G)$

shows

$(\tau \in M \longrightarrow \vartheta \in M \longrightarrow \text{val}(G, \tau) = \text{val}(G, \vartheta) \longrightarrow (\exists p \in G. \text{forces_eq}(p, \tau, \vartheta))) \wedge$

$(\tau \in M \longrightarrow \vartheta \in M \longrightarrow \text{val}(G, \tau) \in \text{val}(G, \vartheta) \longrightarrow (\exists p \in G. \text{forces_mem}(p, \tau, \vartheta)))$

(is $?Q(\tau, \vartheta) \wedge ?R(\tau, \vartheta)$)

$\langle \text{proof} \rangle$

lemma *map_val_in_MG*:

assumes

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

shows

$\text{map}(\text{val}(G), \text{env}) \in \text{list}(M[G])$

$\langle \text{proof} \rangle$

lemma *truth_lemma_mem*:

assumes

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

$n \in \text{nat} \ m \in \text{nat} \ n < \text{length}(\text{env}) \ m < \text{length}(\text{env})$

shows

$(\exists p \in G. p \Vdash \text{Member}(n, m) \ \text{env}) \longleftrightarrow M[G], \text{map}(\text{val}(G), \text{env}) \models \text{Member}(n, m)$

$\langle \text{proof} \rangle$

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

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

lemma *arities_at_aux*:

assumes

$n \in nat$ $m \in nat$ $env \in list(M)$ $succ(n) \cup succ(m) \leq length(env)$

shows

$n < length(env)$ $m < length(env)$
 $\langle proof \rangle$

18.13 The Strengthening Lemma

lemma *strengthening_lemma*:

assumes

$p \in P$ $\varphi \in formula$ $r \in P$ $r \leq p$

shows

$\bigwedge env. env \in list(M) \implies arity(\varphi) \leq length(env) \implies p \Vdash \varphi env \implies r \Vdash \varphi env$
 $\langle proof \rangle$

18.14 The Density Lemma

lemma *arity_Nand_le*:

assumes $\varphi \in formula$ $\psi \in formula$ $arity(Nand(\varphi, \psi)) \leq length(env)$ $env \in list(A)$

shows $arity(\varphi) \leq length(env)$ $arity(\psi) \leq length(env)$

$\langle proof \rangle$

lemma *dense_below_imp_forces*:

assumes

$p \in P$ $\varphi \in formula$

shows

$\bigwedge env. env \in list(M) \implies arity(\varphi) \leq length(env) \implies$
 $dense_below(\{q \in P. (q \Vdash \varphi env)\}, p) \implies (p \Vdash \varphi env)$
 $\langle proof \rangle$

lemma *density_lemma*:

assumes

$p \in P$ $\varphi \in formula$ $env \in list(M)$ $arity(\varphi) \leq length(env)$

shows

$p \Vdash \varphi env \iff dense_below(\{q \in P. (q \Vdash \varphi env)\}, p)$
 $\langle proof \rangle$

18.15 The Truth Lemma

lemma *Forces_And*:

assumes

$p \in P$ $env \in list(M)$ $\varphi \in formula$ $\psi \in formula$

$arity(\varphi) \leq length(env) \quad arity(\psi) \leq length(env)$
shows
 $p \Vdash And(\varphi, \psi) \ env \iff (p \Vdash \varphi \ env) \wedge (p \Vdash \psi \ env)$
 ⟨proof⟩

lemma *Forces_Nand_alt*:

assumes
 $p \in P \quad env \in list(M) \quad \varphi \in formula \quad \psi \in formula$
 $arity(\varphi) \leq length(env) \quad arity(\psi) \leq length(env)$
shows
 $(p \Vdash Nand(\varphi, \psi) \ env) \iff (p \Vdash Neg(And(\varphi, \psi)) \ env)$
 ⟨proof⟩

lemma *truth_lemma_Neg*:

assumes
 $\varphi \in formula \quad M_generic(G) \quad env \in list(M) \quad arity(\varphi) \leq length(env)$ **and**
 $IH: (\exists p \in G. p \Vdash \varphi \ env) \iff M[G], map(val(G), env) \models \varphi$
shows
 $(\exists p \in G. p \Vdash Neg(\varphi) \ env) \iff M[G], map(val(G), env) \models Neg(\varphi)$
 ⟨proof⟩

lemma *truth_lemma_And*:

assumes
 $env \in list(M) \quad \varphi \in formula \quad \psi \in formula$
 $arity(\varphi) \leq length(env) \quad arity(\psi) \leq length(env) \quad M_generic(G)$
and
 $IH: (\exists p \in G. p \Vdash \varphi \ env) \iff M[G], map(val(G), env) \models \varphi$
 $(\exists p \in G. p \Vdash \psi \ env) \iff M[G], map(val(G), env) \models \psi$
shows
 $(\exists p \in G. (p \Vdash And(\varphi, \psi) \ env)) \iff M[G], map(val(G), env) \models And(\varphi, \psi)$
 ⟨proof⟩

definition

$ren_truth_lemma :: i \Rightarrow i$ **where**
 $ren_truth_lemma(\varphi) \equiv$
 $Exists(Exists(Exists(Exists(Exists($
 $And(Equal(0, 5), And(Equal(1, 8), And(Equal(2, 9), And(Equal(3, 10), And(Equal(4, 6),$
 $iterates(\lambda p. incr_bv(p) '5 , 6, \varphi))))))))))$

lemma *ren_truth_lemma_type*[TC] :

$\varphi \in formula \implies ren_truth_lemma(\varphi) \in formula$
 ⟨proof⟩

lemma *arity_ren_truth* :

assumes $\varphi \in formula$
shows $arity(ren_truth_lemma(\varphi)) \leq 6 \cup succ(arity(\varphi))$
 ⟨proof⟩

lemma *sats_ren_truth_lemma*:

$[q,b,d,a1,a2,a3] @ env \in list(M) \implies \varphi \in formula \implies$
 $(M, [q,b,d,a1,a2,a3] @ env \models ren_truth_lemma(\varphi)) \longleftrightarrow$
 $(M, [q,a1,a2,a3,b] @ env \models \varphi)$
 <proof>

lemma *truth_lemma'* :

assumes

$\varphi \in formula \ env \in list(M) \ arity(\varphi) \leq succ(length(env))$

shows

$separation(\#\#M, \lambda d. \exists b \in M. \forall q \in P. q \preceq d \longrightarrow \neg(q \Vdash \varphi ([b]@env)))$

<proof>

lemma *truth_lemma*:

assumes

$\varphi \in formula \ M_generic(G)$

shows

$\bigwedge env. env \in list(M) \implies arity(\varphi) \leq length(env) \implies$

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

<proof>

18.16 The “Definition of forcing”

lemma *definition_of_forcing*:

assumes

$p \in P \ \varphi \in formula \ env \in list(M) \ arity(\varphi) \leq length(env)$

shows

$(p \Vdash \varphi \ env) \longleftrightarrow$

$(\forall G. M_generic(G) \wedge p \in G \longrightarrow M[G], map(val(G), env) \models \varphi)$

<proof>

lemmas *definability = forces_type*

end

end

19 Auxiliary renamings for Separation

theory *Separation_Rename*

imports *Interface Renaming*

begin

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

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

<proof>

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

$\langle proof \rangle$

lemma *nth_concat3* : $env \in list(A) \implies u = nth(succ(length(env)), env @ [pi, u])$
 $\langle proof \rangle$

definition

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

definition

sep_env :: $i \Rightarrow i$ **where**
 $sep_env(n) \equiv \lambda i \in (5\#+n)-5 . i\#+2$

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

$weak(n,m) \equiv \{i\#+m . i \in n\}$

lemma *weakD* :

assumes $n \in nat \ k \in nat \ x \in weak(n,k)$
shows $\exists i \in n . x = i\#+k$
 $\langle proof \rangle$

lemma *weak_equal* :

assumes $n \in nat \ m \in nat$
shows $weak(n,m) = (m\#+n) - m$
 $\langle proof \rangle$

lemma *weak_zero*:

shows $weak(0,n) = 0$
 $\langle proof \rangle$

lemma *weakening_diff* :

assumes $n \in nat$
shows $weak(n,7) - weak(n,5) \subseteq \{5\#+n, 6\#+n\}$
 $\langle proof \rangle$

lemma *in_add_del* :

assumes $x \in j\#+n \ n \in nat \ j \in nat$
shows $x < j \vee x \in weak(n,j)$
 $\langle proof \rangle$

lemma *sep_env_action*:

assumes
 $[t,p,u,P,leq,o,pi] \in list(M)$
 $env \in list(M)$
shows $\forall i . i \in weak(length(env),5) \longrightarrow$
 $nth(sep_env(length(env))\ 'i, [t,p,u,P,leq,o,pi]@env) = nth(i, [p,P,leq,o,t] @ env$
 $@ [pi,u])$
 $\langle proof \rangle$

lemma *sep_env_type* :

assumes $n \in \text{nat}$

shows $\text{sep_env}(n) : (5\#+n)-5 \rightarrow (7\#+n)-7$

<proof>

lemma *sep_var_fn_type* :

assumes $n \in \text{nat}$

shows $\text{sep_var}(n) : 7\#+n \dashv\vdash 7\#+n$

<proof>

lemma *sep_var_domain* :

assumes $n \in \text{nat}$

shows $\text{domain}(\text{sep_var}(n)) = 7\#+n - \text{weak}(n,5)$

<proof>

lemma *sep_var_type* :

assumes $n \in \text{nat}$

shows $\text{sep_var}(n) : (7\#+n)-\text{weak}(n,5) \rightarrow 7\#+n$

<proof>

lemma *sep_var_action* :

assumes

$[t,p,u,P,\text{leq},o,\text{pi}] \in \text{list}(M)$

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

shows $\forall i . i \in (7\#+\text{length}(\text{env})) - \text{weak}(\text{length}(\text{env}),5) \longrightarrow$

$\text{nth}(\text{sep_var}(\text{length}(\text{env}))\ 'i, [t,p,u,P,\text{leq},o,\text{pi}]) @ \text{env} = \text{nth}(i, [p,P,\text{leq},o,t]) @ \text{env}$

$@ [pi,u]$

<proof>

definition

$\text{rensep} :: i \Rightarrow i$ **where**

$\text{rensep}(n) \equiv \text{union_fun}(\text{sep_var}(n), \text{sep_env}(n), 7\#+n-\text{weak}(n,5), \text{weak}(n,5))$

lemma *rensep_aux* :

assumes $n \in \text{nat}$

shows $(7\#+n-\text{weak}(n,5)) \cup \text{weak}(n,5) = 7\#+n \ 7\#+n \cup (7\#+n - 7) =$

$7\#+n$

<proof>

lemma *rensep_type* :

assumes $n \in \text{nat}$

shows $\text{rensep}(n) \in 7\#+n \rightarrow 7\#+n$

<proof>

lemma *rensep_action* :

assumes $[t,p,u,P,\text{leq},o,\text{pi}] @ \text{env} \in \text{list}(M)$

shows $\forall i . i < 7\#+\text{length}(\text{env}) \longrightarrow \text{nth}(\text{rensep}(\text{length}(\text{env}))\ 'i, [t,p,u,P,\text{leq},o,\text{pi}]) @ \text{env}$

$= \text{nth}(i, [p,P,\text{leq},o,t]) @ \text{env} @ [pi,u]$

<proof>

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

lemma *arity_rensep*: **assumes** $\varphi \in formula$ $env \in list(M)$
 $arity(\varphi) \leq 7\#+length(env)$
shows $arity(sep_ren(length(env),\varphi)) \leq 7\#+length(env)$
<proof>

lemma *type_rensep* [TC]:
assumes $\varphi \in formula$ $env \in list(M)$
shows $sep_ren(length(env),\varphi) \in formula$
<proof>

lemma *sepren_action*:
assumes $arity(\varphi) \leq 7\#+length(env)$
 $[t,p,u,P,leq,o,pi] \in list(M)$
 $env \in list(M)$
 $\varphi \in formula$
shows $sats(M, sep_ren(length(env),\varphi), [t,p,u,P,leq,o,pi] @ env) \longleftrightarrow sats(M, \varphi, [p,P,leq,o,t] @ env @ [pi,u])$
<proof>

end

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

theory *Separation_Axiom*
imports *Forcing_Theorems Separation_Rename*
begin

context *G_generic*
begin

lemma *map_val* :
assumes $env \in list(M[G])$
shows $\exists nenv \in list(M). env = map(val(G), nenv)$
<proof>

lemma *Collect_sats_in_MG* :
assumes
 $c \in M[G]$
 $\varphi \in formula$ $env \in list(M[G])$ $arity(\varphi) \leq 1\#+length(env)$
shows
 $\{x \in c. (M[G], [x] @ env \models \varphi)\} \in M[G]$
<proof>

```

theorem separation_in_MG:
  assumes
     $\varphi \in \text{formula}$  and  $\text{arity}(\varphi) \leq 1 \ \#\!+\ \text{length}(\text{env})$  and  $\text{env} \in \text{list}(M[G])$ 
  shows
     $\text{separation}(\#\!#\!M[G], \lambda x. (M[G], [x] \ @ \ \text{env} \models \varphi))$ 
   $\langle \text{proof} \rangle$ 

end

end

```

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

```

theory Pairing_Axiom imports Names begin

  context forcing_data
  begin

    lemma val_Upair :
       $\text{one} \in G \implies \text{val}(G, \{\langle \tau, \text{one} \rangle, \langle \varrho, \text{one} \rangle\}) = \{\text{val}(G, \tau), \text{val}(G, \varrho)\}$ 
       $\langle \text{proof} \rangle$ 

    lemma pairing_in_MG :
      assumes  $M\_generic(G)$ 
      shows  $\text{upair\_ax}(\#\!#\!M[G])$ 
       $\langle \text{proof} \rangle$ 

  end
end

```

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

```

theory Union_Axiom
  imports Names
begin

  context forcing_data
  begin

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

    definition Union_name_fm ::  $i$  where
       $\text{Union\_name\_fm} \equiv$ 

```

```

Exists(
  Exists(And(pair_fm(1,0,2),
    Exists (
      Exists (And(Member(0,7),
        Exists (And(And(pair_fm(2,1,0),Member(0,6)),
          Exists (And(Member(0,9),
            Exists (And(And(pair_fm(6,1,0),Member(0,4)),
              Exists (And(And(pair_fm(6,2,0),Member(0,10)),
                Exists (And(pair_fm(7,5,0),Member(0,11))))))))))))))))))

```

lemma *Union_name_fm_type* [TC]:

```

Union_name_fm ∈ formula
⟨proof⟩

```

lemma *arity_Union_name_fm* :

```

arity(Union_name_fm) = 4
⟨proof⟩

```

lemma *sats_Union_name_fm* :

```

[[ a ∈ M; b ∈ M; P' ∈ M; p ∈ M; ∅ ∈ M; τ ∈ M; leq' ∈ M ]] ⇒
  sats(M, Union_name_fm, [⟨∅, p⟩, τ, leq', P']∧@[a, b]) ↔
  Union_name_body(P', leq', τ, ⟨∅, p⟩)
⟨proof⟩

```

lemma *domD* :

```

assumes τ ∈ M σ ∈ domain(τ)
shows σ ∈ M
⟨proof⟩

```

definition *Union_name* :: *i* ⇒ *i* **where**

```

Union_name(τ) ≡
  {u ∈ domain(⋃(domain(τ))) × P . Union_name_body(P, leq, τ, u)}

```

lemma *Union_name_M* : **assumes** *τ* ∈ *M*

```

shows {u ∈ domain(⋃(domain(τ))) × P . Union_name_body(P, leq, τ, u)} ∈ M
⟨proof⟩

```

lemma *Union_MG_Eq* :

```

assumes a ∈ M[G] and a = val(G, τ) and filter(G) and τ ∈ M
shows ⋃ a = val(G, Union_name(τ))
⟨proof⟩

```

lemma *union_in_MG* : **assumes** *filter*(*G*)

```

shows Union_ax(##M[G])

```

<proof>

theorem *Union_MG* : $M_generic(G) \implies Union_ax(\#\#M[G])$
<proof>

end
end

23 The Powerset Axiom in $M[G]$

theory *Powerset_Axiom*

imports *Renaming_Auto Separation_Axiom Pairing_Axiom Union_Axiom*
begin

<ML>

lemma *Collect_inter_Transset*:

assumes

$Transset(M) \ b \in M$

shows

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

<proof>

context *G_generic* **begin**

lemma *name_components_in_M*:

assumes $\langle \sigma, p \rangle \in \vartheta \ \vartheta \in M$

shows $\sigma \in M \ p \in M$

<proof>

lemma *sats_fst_snd_in_M*:

assumes

$A \in M \ B \in M \ \varphi \in \text{formula} \ p \in M \ l \in M \ o \in M \ \chi \in M$

$\text{arity}(\varphi) \leq 6$

shows

$\{sq \in A \times B . \text{sats}(M, \varphi, [\text{snd}(sq), p, l, o, \text{fst}(sq), \chi])\} \in M$

(**is** $\vartheta \in M$)

<proof>

lemma *Pow_inter_MG*:

assumes

$a \in M[G]$

shows

$Pow(a) \cap M[G] \in M[G]$

<proof>

end

context *G_generic* **begin**

interpretation *mgtriv*: $M_trivial \#\#M[G]$
<proof>

theorem *power_in_MG* : $power_ax(\#\#(M[G]))$
<proof>
end
end

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

theory *Extensionality_Axiom*
imports
 Names
begin

context *forcing_data*
begin

lemma *extensionality_in_MG* : $extensionality(\#\#(M[G]))$
<proof>

end
end

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

theory *Foundation_Axiom*
imports
 Names
begin

context *forcing_data*
begin

lemma *foundation_in_MG* : $foundation_ax(\#\#(M[G]))$
<proof>

lemma *foundation_ax*($\#\#(M[G])$)
<proof>

end
end

26 The binder *Least*

theory *Least*
imports
Names

begin

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

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

lemma *Ord_Least_cong*:
assumes $\bigwedge y. Ord(y) \implies R(y) \longleftrightarrow Q(y)$
shows $(\mu \alpha. R(\alpha)) = (\mu \alpha. Q(\alpha))$
 $\langle proof \rangle$

definition

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

definition

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

lemma *least_fm_type*[*TC*] : $i \in nat \implies q \in formula \implies least_fm(q, i) \in formula$
 $\langle proof \rangle$

lemmas *basic_fm_simps* = *sats_subset_fm' sats_transset_fm' sats_ordinal_fm'*

lemma *sats_least_fm* :

assumes *p_iff_sats*:
 $\bigwedge a. a \in A \implies P(a) \longleftrightarrow sats(A, p, Cons(a, env))$
shows
 $\llbracket y \in nat; env \in list(A) ; 0 \in A \rrbracket$
 $\implies sats(A, least_fm(p, y), env) \longleftrightarrow$
 $least(\#\#A, P, nth(y, env))$
 $\langle proof \rangle$

lemma *least_iff_sats*:

assumes *is_Q_iff_sats*:
 $\bigwedge a. a \in A \implies is_Q(a) \longleftrightarrow sats(A, q, Cons(a, env))$
shows

$\llbracket nth(j, env) = y; j \in nat; env \in list(A); 0 \in A \rrbracket$
 $\implies least(\#\#A, is_Q, y) \longleftrightarrow sats(A, least_fm(q, j), env)$
 <proof>

lemma *least_conj*: $a \in M \implies least(\#\#M, \lambda x. x \in M \wedge Q(x), a) \longleftrightarrow least(\#\#M, Q, a)$
 <proof>

lemma (in *M_ctm*) *unique_least*: $a \in M \implies b \in M \implies least(\#\#M, Q, a) \implies least(\#\#M, Q, b)$
 $\implies a = b$
 <proof>

context *M_trivial*
begin

26.1 Absoluteness and closure under *Least*

lemma *least_abs*:
assumes $\bigwedge x. Q(x) \implies M(x) \ M(a)$
shows $least(M, Q, a) \longleftrightarrow a = (\mu x. Q(x))$
 <proof>

lemma *Least_closed*:
assumes $\bigwedge x. Q(x) \implies M(x)$
shows $M(\mu x. Q(x))$
 <proof>

end

end

27 The Axiom of Replacement in $M[G]$

theory *Replacement_Axiom*
imports
Least Relative_Univ Separation_Axiom Renaming_Auto
begin

<ML>

definition *renrep_fn* :: $i \Rightarrow i$ **where**
 $renrep_fn(env) \equiv sum(renrep1_fn, id(length(env)), 6, 8, length(env))$

definition
 $renrep :: [i, i] \Rightarrow i$ **where**
 $renrep(\varphi, env) = ren(\varphi)('6\#+length(env))('8\#+length(env))'renrep_fn(env)$

lemma *renrep_type* [*TC*]:
assumes $\varphi \in formula \ env \in list(M)$

shows $\text{renrep}(\varphi, \text{env}) \in \text{formula}$
 ⟨proof⟩

lemma *arity_renrep*:

assumes $\varphi \in \text{formula}$ $\text{arity}(\varphi) \leq 6 \# + \text{length}(\text{env})$ $\text{env} \in \text{list}(M)$
shows $\text{arity}(\text{renrep}(\varphi, \text{env})) \leq 8 \# + \text{length}(\text{env})$
 ⟨proof⟩

lemma *renrep_sats* :

assumes $\text{arity}(\varphi) \leq 6 \# + \text{length}(\text{env})$
 $[P, \text{leq}, o, p, \varrho, \tau] @ \text{env} \in \text{list}(M)$
 $V \in M$ $\alpha \in M$
 $\varphi \in \text{formula}$
shows $\text{sats}(M, \varphi, [p, P, \text{leq}, o, \varrho, \tau] @ \text{env}) \longleftrightarrow \text{sats}(M, \text{renrep}(\varphi, \text{env}), [V, \tau, \varrho, p, \alpha, P, \text{leq}, o]$
 $@ \text{env})$
 ⟨proof⟩

⟨ML⟩

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

$\text{renpbdy_fn}(\text{env}) \equiv \text{sum}(\text{renpbdy1_fn}, \text{id}(\text{length}(\text{env})), 6, 7, \text{length}(\text{env}))$

definition

$\text{renpbdy} :: [i, i] \Rightarrow i$ **where**
 $\text{renpbdy}(\varphi, \text{env}) = \text{ren}(\varphi) '(6 \# + \text{length}(\text{env})) '(7 \# + \text{length}(\text{env})) '\text{renpbdy_fn}(\text{env})$

lemma

renpbdy_type [TC]: $\varphi \in \text{formula} \Longrightarrow \text{env} \in \text{list}(M) \Longrightarrow \text{renpbdy}(\varphi, \text{env}) \in \text{formula}$
 ⟨proof⟩

lemma *arity_renpbdy*: $\varphi \in \text{formula} \Longrightarrow \text{arity}(\varphi) \leq 6 \# + \text{length}(\text{env}) \Longrightarrow \text{env} \in \text{list}(M)$
 $\Longrightarrow \text{arity}(\text{renpbdy}(\varphi, \text{env})) \leq 7 \# + \text{length}(\text{env})$
 ⟨proof⟩

lemma

sats_renpbdy: $\text{arity}(\varphi) \leq 6 \# + \text{length}(\text{nenv}) \Longrightarrow [\varrho, p, x, \alpha, P, \text{leq}, o, \pi] @ \text{nenv} \in$
 $\text{list}(M) \Longrightarrow \varphi \in \text{formula} \Longrightarrow$
 $\text{sats}(M, \varphi, [\varrho, p, \alpha, P, \text{leq}, o] @ \text{nenv}) \longleftrightarrow \text{sats}(M, \text{renpbdy}(\varphi, \text{nenv}), [\varrho, p, x, \alpha, P, \text{leq}, o]$
 $@ \text{nenv})$
 ⟨proof⟩

⟨ML⟩

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

$\text{renbody_fn}(\text{env}) \equiv \text{sum}(\text{renbody1_fn}, \text{id}(\text{length}(\text{env})), 5, 6, \text{length}(\text{env}))$

definition

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

lemma

renbody_type [TC]: $\varphi \in \text{formula} \implies \text{env} \in \text{list}(M) \implies \text{renbody}(\varphi, \text{env}) \in \text{formula}$
 ⟨*proof*⟩

lemma *arity_renbody*: $\varphi \in \text{formula} \implies \text{arity}(\varphi) \leq 5 \# + \text{length}(\text{env}) \implies \text{env} \in \text{list}(M)$
 ⇒

$\text{arity}(\text{renbody}(\varphi, \text{env})) \leq 6 \# + \text{length}(\text{env})$
 ⟨*proof*⟩

lemma

sats_renbody: $\text{arity}(\varphi) \leq 5 \# + \text{length}(\text{nenv}) \implies [\alpha, x, m, P, \text{leq}, o] @ \text{nenv} \in \text{list}(M) \implies \varphi \in \text{formula} \implies$
 $\text{sats}(M, \varphi, [x, \alpha, P, \text{leq}, o] @ \text{nenv}) \longleftrightarrow \text{sats}(M, \text{renbody}(\varphi, \text{nenv}), [\alpha, x, m, P, \text{leq}, o] @ \text{nenv})$
 ⟨*proof*⟩

context *G_generic*

begin

lemma *pow_inter_M*:

assumes

$x \in M \ y \in M$

shows

$\text{powerset}(\#\#M, x, y) \longleftrightarrow y = \text{Pow}(x) \cap M$

⟨*proof*⟩

schematic_goal *sats_prebody_fm_auto*:

assumes

$\varphi \in \text{formula} [P, \text{leq}, \text{one}, p, \varrho, \pi] @ \text{nenv} \in \text{list}(M) \ \alpha \in M \ \text{arity}(\varphi) \leq 2 \# + \text{length}(\text{nenv})$

shows

$(\exists \tau \in M. \exists V \in M. \text{is_Vset}(\#\#M, \alpha, V) \wedge \tau \in V \wedge \text{sats}(M, \text{forces}(\varphi), [p, P, \text{leq}, \text{one}, \varrho, \tau]$

@ *nenv*)

$\longleftrightarrow \text{sats}(M, ?\text{prebody_fm}, [\varrho, p, \alpha, P, \text{leq}, \text{one}] @ \text{nenv})$

⟨*proof*⟩

⟨*ML*⟩

lemma *prebody_fm_type* [TC]:

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

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

shows $\text{prebody_fm}(\varphi, \text{env}) \in \text{formula}$

⟨*proof*⟩

lemmas *new_fm_defs* = *fm_defs* *is_transrec_fm_def* *is_eclose_fm_def* *mem_eclose_fm_def*

finite_ordinal_fm_def is_wfrec_fm_def Memrel_fm_def eclose_n_fm_def is_recfun_fm_def
is_iterates_fm_def
iterates_MH_fm_def is_nat_case_fm_def quasinat_fm_def pre_image_fm_def
restriction_fm_def

lemma *sats_prebody_fm*:

assumes

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

shows

$sats(M, prebody_fm(\varphi, nenv), [\varrho, p, \alpha, P, leq, one] @ nenv) \longleftrightarrow$
 $(\exists \tau \in M. \exists V \in M. is_Vset(\#\#M, \alpha, V) \wedge \tau \in V \wedge sats(M, forces(\varphi), [p, P, leq, one, \varrho, \tau]$
 $@ nenv))$

<proof>

lemma *arity_prebody_fm*:

assumes

$\varphi \in formula \ \alpha \in M \ env \in list(M) \ arity(\varphi) \leq 2 \ \# + length(env)$

shows

$arity(prebody_fm(\varphi, env)) \leq 6 \ \# + length(env)$

<proof>

definition

body_fm' :: $[i, i] \Rightarrow i$ **where**

$body_fm'(\varphi, env) \equiv Exists(Exists(And(pair_fm(0, 1, 2), renpbdy(prebody_fm(\varphi, env), env))))$

lemma *body_fm'_type[TC]*: $\varphi \in formula \Longrightarrow env \in list(M) \Longrightarrow body_fm'(\varphi, env) \in formula$

<proof>

lemma *arity_body_fm'*:

assumes

$\varphi \in formula \ \alpha \in M \ env \in list(M) \ arity(\varphi) \leq 2 \ \# + length(env)$

shows

$arity(body_fm'(\varphi, env)) \leq 5 \ \# + length(env)$

<proof>

lemma *sats_body_fm'*:

assumes

$\exists t \ p. \ x = \langle t, p \rangle \ x \in M \ [\alpha, P, leq, one, p, \varrho] @ nenv \in list(M) \ \varphi \in formula \ arity(\varphi) \leq 2$
 $\# + length(nenv)$

shows

$sats(M, body_fm'(\varphi, nenv), [x, \alpha, P, leq, one] @ nenv) \longleftrightarrow$
 $sats(M, renpbdy(prebody_fm(\varphi, nenv), nenv), [fst(x), snd(x), x, \alpha, P, leq, one] @ nenv)$

<proof>

definition

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

$body_fm(\varphi, env) \equiv renbody(body_fm'(\varphi, env), env)$

lemma *body_fm_type* [TC]: $env \in list(M) \implies \varphi \in formula \implies body_fm(\varphi, env) \in formula$
 ⟨proof⟩

lemma *sats_body_fm*:

assumes

$\exists t p. x = \langle t, p \rangle \ [\alpha, x, m, P, leq, one] \ @ \ nenv \in list(M)$

$\varphi \in formula \ \text{arity}(\varphi) \leq 2 \ \#\ + \ length(nenv)$

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

$\exists t p. x = \langle t, p \rangle \ x \in M \ [\alpha, m, P, leq, one] \ @ \ nenv \in list(M)$

$\varphi \in formula \ \text{arity}(\varphi) \leq 2 \ \#\ + \ length(nenv)$

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

$\exists t p. x = \langle t, p \rangle \ x \in M \ [x, \alpha, m, P, leq, one] \ @ \ nenv \in list(M)$

$\varphi \in formula \ \text{arity}(\varphi) \leq 2 \ \#\ + \ length(nenv)$

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

$c \in M[G] \ env \in list(M[G])$

$\varphi \in formula \ \text{arity}(\varphi) \leq 2 \ \#\ + \ length(env)$

$univalent(\#\#M[G], c, \lambda x v. (M[G], [x, v] @ env \models \varphi))$

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

$\varphi \in formula \ \text{and} \ \text{arity}(\varphi) \leq 2 \ \#\ + \ length(env) \ env \in list(M[G])$

shows

$strong_replacement(\#\#M[G], \lambda x v. sats(M[G], \varphi, [x, v] @ env))$

⟨proof⟩

end

end

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

theory *Infinity_Axiom*
 imports *Pairing_Axiom Union_Axiom Separation_Axiom*
begin

context *G_generic* **begin**

interpretation *mg_triv*: *M_trivial*##*M[G]*
 \langle *proof* \rangle

lemma *infinity_in_MG* : *infinity_ax*(##*M[G]*)
 \langle *proof* \rangle

end

end

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

theory *Choice_Axiom*
 imports *Powerset_Axiom Pairing_Axiom Union_Axiom Extensionality_Axiom*

Foundation_Axiom Powerset_Axiom Separation_Axiom
 Replacement_Axiom Interface Infinity_Axiom

begin

definition

induced_surj :: $i \Rightarrow i \Rightarrow i \Rightarrow i$ **where**
 induced_surj(*f*,*a*,*e*) $\equiv f^{-1}((\text{range}(f)-a) \times \{e\}) \cup \text{restrict}(f, f^{-1}a)$

lemma *domain_induced_surj*: $\text{domain}(\text{induced_surj}(f,a,e)) = \text{domain}(f)$
 \langle *proof* \rangle

lemma *range_restrict_vimage*:
 assumes *function*(*f*)
 shows $\text{range}(\text{restrict}(f, f^{-1}a)) \subseteq a$
 \langle *proof* \rangle

lemma *induced_surj_type*:
 assumes
 function(*f*)
 shows
 $\text{induced_surj}(f,a,e): \text{domain}(f) \rightarrow \{e\} \cup a$
 and
 $x \in f^{-1}a \implies \text{induced_surj}(f,a,e) \text{' } x = f \text{' } x$

<proof>

lemma *induced_surj_is_surj* :

assumes

$e \in a$ *function*(f) $\text{domain}(f) = \alpha \wedge y. y \in a \implies \exists x \in \alpha. f \ ' \ x = y$

shows

$\text{induced_surj}(f, a, e) \in \text{surj}(\alpha, a)$

<proof>

context *G_generic*

begin

definition

upair_name :: $i \Rightarrow i \Rightarrow i$ **where**

$\text{upair_name}(\tau, \varrho) \equiv \{\langle \tau, \text{one} \rangle, \langle \varrho, \text{one} \rangle\}$

definition

is_upair_name :: $[i, i, i] \Rightarrow o$ **where**

$\text{is_upair_name}(x, y, z) \equiv \exists xo \in M. \exists yo \in M. \text{pair}(\#\#M, x, \text{one}, xo) \wedge \text{pair}(\#\#M, y, \text{one}, yo)$

\wedge

$\text{upair}(\#\#M, xo, yo, z)$

lemma *upair_name_abs* :

assumes $x \in M$ $y \in M$ $z \in M$

shows $\text{is_upair_name}(x, y, z) \longleftrightarrow z = \text{upair_name}(x, y)$

<proof>

lemma *upair_name_closed* :

$\llbracket x \in M; y \in M \rrbracket \implies \text{upair_name}(x, y) \in M$

<proof>

definition

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

$\text{upair_name_fm}(x, y, o, z) \equiv \text{Exists}(\text{Exists}(\text{And}(\text{pair_fm}(x\#\#2, o\#\#2, 1),$
 $\text{And}(\text{pair_fm}(y\#\#2, o\#\#2, 0), \text{upair_fm}(1, 0, z\#\#2))))))$

lemma *upair_name_fm_type*[*TC*] :

$\llbracket s \in \text{nat}; x \in \text{nat}; y \in \text{nat}; o \in \text{nat} \rrbracket \implies \text{upair_name_fm}(s, x, y, o) \in \text{formula}$

<proof>

lemma *sats_upair_name_fm* :

assumes $x \in \text{nat}$ $y \in \text{nat}$ $z \in \text{nat}$ $o \in \text{nat}$ $\text{env} \in \text{list}(M)$ $\text{nth}(o, \text{env}) = \text{one}$

shows

$\text{sats}(M, \text{upair_name_fm}(x, y, o, z), \text{env}) \longleftrightarrow \text{is_upair_name}(\text{nth}(x, \text{env}), \text{nth}(y, \text{env}), \text{nth}(z, \text{env}))$

<proof>

definition

opair_name :: $i \Rightarrow i \Rightarrow i$ **where**

$opair_name(\tau, \rho) \equiv upair_name(upair_name(\tau, \tau), upair_name(\tau, \rho))$

definition

$is_opair_name :: [i, i, i] \Rightarrow o$ **where**

$is_opair_name(x, y, z) \equiv \exists upxx \in M. \exists upxy \in M. is_upair_name(x, x, upxx) \wedge is_upair_name(x, y, upxy) \wedge is_upair_name(upxx, upxy, z)$

lemma $opair_name_abs$:

assumes $x \in M \ y \in M \ z \in M$

shows $is_opair_name(x, y, z) \longleftrightarrow z = opair_name(x, y)$

$\langle proof \rangle$

lemma $opair_name_closed$:

$\llbracket x \in M; y \in M \rrbracket \Longrightarrow opair_name(x, y) \in M$

$\langle proof \rangle$

definition

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

$opair_name_fm(x, y, o, z) \equiv Exists(Exists(And(upair_name_fm(x\#\# + 2, x\#\# + 2, o\#\# + 2, 1), And(upair_name_fm(x\#\# + 2, y\#\# + 2, o\#\# + 2, 0), upair_name_fm(1, 0, o\#\# + 2, z\#\# + 2))))))$

lemma $opair_name_fm_type[TC]$:

$\llbracket s \in nat; x \in nat; y \in nat; o \in nat \rrbracket \Longrightarrow opair_name_fm(s, x, y, o) \in formula$

$\langle proof \rangle$

lemma $sats_opair_name_fm$:

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

shows

$sats(M, opair_name_fm(x, y, o, z), env) \longleftrightarrow is_opair_name(nth(x, env), nth(y, env), nth(z, env))$

$\langle proof \rangle$

lemma val_upair_name : $val(G, upair_name(\tau, \rho)) = \{val(G, \tau), val(G, \rho)\}$

$\langle proof \rangle$

lemma val_opair_name : $val(G, opair_name(\tau, \rho)) = \langle val(G, \tau), val(G, \rho) \rangle$

$\langle proof \rangle$

lemma val_RepFun_one : $val(G, \{\langle f(x), one \rangle . x \in a\}) = \{val(G, f(x)) . x \in a\}$

$\langle proof \rangle$

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

interpretation $mgzf$: $M_ZF_trans \ M[G]$

$\langle proof \rangle$

definition

$is_opname_check :: [i, i, i] \Rightarrow o$ **where**

$is_opname_check(s,x,y) \equiv \exists chx \in M. \exists sx \in M. is_check(x, chx) \wedge fun_apply(\#\#M, s, x, sx)$
 \wedge
 $is_opair_name(chx, sx, y)$

definition

$opname_check_fm :: [i, i, i, i] \Rightarrow i$ **where**
 $opname_check_fm(s, x, y, o) \equiv Exists(Exists(And(check_fm(2\#+x, 2\#+o, 1),$
 $And(fun_apply_fm(2\#+s, 2\#+x, 0), opair_name_fm(1, 0, 2\#+o, 2\#+y))))))$

lemma $opname_check_fm_type[TC]$:

$\llbracket s \in nat; x \in nat; y \in nat; o \in nat \rrbracket \Longrightarrow opname_check_fm(s, x, y, o) \in formula$
 $\langle proof \rangle$

lemma $sats_opname_check_fm$:

assumes $x \in nat$ $y \in nat$ $z \in nat$ $o \in nat$ $env \in list(M)$ $nth(o, env) = one$
 $y < length(env)$

shows

$sats(M, opname_check_fm(x, y, z, o), env) \longleftrightarrow is_opname_check(nth(x, env), nth(y, env), nth(z, env))$
 $\langle proof \rangle$

lemma $opname_check_abs$:

assumes $s \in M$ $x \in M$ $y \in M$

shows $is_opname_check(s, x, y) \longleftrightarrow y = opair_name(check(x), s'x)$

$\langle proof \rangle$

lemma $repl_opname_check$:

assumes

$A \in M$ $f \in M$

shows

$\{opair_name(check(x), f'x). x \in A\} \in M$

$\langle proof \rangle$

theorem $choice_in_MG$:

assumes $choice_ax(\#\#M)$

shows $choice_ax(\#\#M[G])$

$\langle proof \rangle$

end

end

30 Ordinals in generic extensions

theory $Ordinals_In_MG$

imports

$Forcing_Theorems_Relative_Univ$

```

begin

context G_generic
begin

lemma rank_val:  $\text{rank}(\text{val}(G,x)) \leq \text{rank}(x)$  (is ?Q(x))
⟨proof⟩

lemma Ord_MG_iff:
  assumes Ord( $\alpha$ )
  shows  $\alpha \in M \longleftrightarrow \alpha \in M[G]$ 
⟨proof⟩

end

end

```

31 Separative notions and proper extensions

```

theory Proper_Extension
  imports
    Names

```

```
begin
```

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

```

locale separative_notion = forcing_notion +
  assumes separative:  $p \in P \implies \exists q \in P. \exists r \in P. q \preceq p \wedge r \preceq p \wedge q \perp r$ 
begin

```

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

```

lemma filter_complement_dense:
  assumes filter(G) shows dense(P - G)
⟨proof⟩

```

```
end
```

```

locale ctm_separative = forcing_data + separative_notion
begin

```

```

lemma generic_not_in_M: assumes M_generic(G) shows  $G \notin M$ 
⟨proof⟩

```

```

theorem proper_extension: assumes M_generic(G) shows  $M \neq M[G]$ 
⟨proof⟩

```

end

end

32 A poset of successions

theory *Succession_Poset*

imports

Arities Proper_Extension Synthetic_Definition

Names

begin

32.1 The set of finite binary sequences

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

definition

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

lemma *seqspaceI[intro]*: $n \in nat \Longrightarrow f: n \rightarrow B \Longrightarrow f \in seqspace(B)$
<proof>

lemma *seqspaceD[dest]*: $f \in seqspace(B) \Longrightarrow \exists n \in nat. f: n \rightarrow B$
<proof>

lemma *seqspace_type*:

$f \in B \hat{<} \omega \Longrightarrow \exists n \in nat. f: n \rightarrow B$
<proof>

schematic_goal *seqspace_fm_auto*:

assumes

$nth(i, env) = n \quad nth(j, env) = z \quad nth(h, env) = B$

$i \in nat \quad j \in nat \quad h \in nat \quad env \in list(A)$

shows

$(\exists om \in A. \omega(\#\#A, om) \wedge n \in om \wedge is_funspace(\#\#A, n, B, z)) \longleftrightarrow (A, env \models (?sqsprp(i, j, h)))$
<proof>

<ML>

locale *M_seqspace* = *M_trancl* +

assumes

seqspace_replacement: $M(B) \Longrightarrow strong_replacement(M, \lambda n z. n \in nat \wedge is_funspace(M, n, B, z))$

begin

lemma *seqspace_closed*:

$M(B) \Longrightarrow M(B \hat{<} \omega)$

$\langle proof \rangle$

end

sublocale $M_ctm \subseteq M_seqspace \#\#M$
 $\langle proof \rangle$

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 $n \in nat \ f \in n \rightarrow A \ a \in A$
shows $seq_upd(f,a) \in succ(n) \rightarrow A$
 $\langle proof \rangle$

lemma $seq_upd_type :$
assumes $f \in A^{\hat{<}\omega} \ a \in A$
shows $seq_upd(f,a) \in A^{\hat{<}\omega}$
 $\langle proof \rangle$

lemma $seq_upd_apply_domain$ [simp]:
assumes $f : n \rightarrow A \ n \in nat$
shows $seq_upd(f,a)'n = a$
 $\langle proof \rangle$

lemma $zero_in_seqspace :$
shows $0 \in A^{\hat{<}\omega}$
 $\langle proof \rangle$

definition
 $seqleR :: i \Rightarrow i \Rightarrow o$ **where**
 $seqleR(f,g) \equiv g \subseteq f$

definition
 $seqlerel :: i \Rightarrow i$ **where**
 $seqlerel(A) \equiv Rrel(\lambda x y. y \subseteq x, A^{\hat{<}\omega})$

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

lemma $seqleI$ [intro!]:
 $\langle f,g \rangle \in 2^{\hat{<}\omega} \times 2^{\hat{<}\omega} \implies g \subseteq f \implies \langle f,g \rangle \in seqle$
 $\langle proof \rangle$

lemma $seqleD$ [dest!]:
 $z \in seqle \implies \exists x y. \langle x,y \rangle \in 2^{\hat{<}\omega} \times 2^{\hat{<}\omega} \wedge y \subseteq x \wedge z = \langle x,y \rangle$
 $\langle proof \rangle$

lemma *upd_leI* :
assumes $f \in 2^{\omega}$ $a \in 2$
shows $\langle \text{seq_upd}(f,a),f \rangle \in \text{seqle}$ (**is** $\langle ?f, _ \rangle \in _$)
 $\langle \text{proof} \rangle$

lemma *preorder_on_seqle*: $\text{preorder_on}(2^{\omega}, \text{seqle})$
 $\langle \text{proof} \rangle$

lemma *zero_seqle_max*: $x \in 2^{\omega} \implies \langle x, 0 \rangle \in \text{seqle}$
 $\langle \text{proof} \rangle$

interpretation *forcing_notion* 2^{ω} *seqle* 0
 $\langle \text{proof} \rangle$

abbreviation *SEQle* :: $[i, i] \Rightarrow o$ (**infixl** $\langle \preceq_s \rangle$ 50)
where $x \preceq_s y \equiv \text{Leq}(x,y)$

abbreviation *SEQIncompatible* :: $[i, i] \Rightarrow o$ (**infixl** $\langle \perp_s \rangle$ 50)
where $x \perp_s y \equiv \text{Incompatible}(x,y)$

lemma *seqspace_separative*:
assumes $f \in 2^{\omega}$
shows $\text{seq_upd}(f,0) \perp_s \text{seq_upd}(f,1)$ (**is** $?f \perp_s ?g$)
 $\langle \text{proof} \rangle$

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

definition *seqleR_fm* :: $i \Rightarrow i$ **where**
 $\text{seqleR_fm}(fg) \equiv \text{Exists}(\text{Exists}(\text{And}(\text{pair_fm}(0,1,fg\#\#2), \text{subset_fm}(1,0))))$

lemma *type_seqleR_fm* :
 $fg \in \text{nat} \implies \text{seqleR_fm}(fg) \in \text{formula}$
 $\langle \text{proof} \rangle$

lemma *arity_seqleR_fm* :
 $fg \in \text{nat} \implies \text{arity}(\text{seqleR_fm}(fg)) = \text{succ}(fg)$
 $\langle \text{proof} \rangle$

lemma (**in** *M_basic*) *seqleR_abs*:
assumes $M(f)$ $M(g)$
shows $\text{seqleR}(f,g) \longleftrightarrow \text{is_seqleR}(M,f,g)$
 $\langle \text{proof} \rangle$

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

lemma (in M_ctm) $seqleR_fm_sats$:
assumes $fg \in nat$ $env \in list(M)$
shows $sats(M, seqleR_fm(fg), env) \longleftrightarrow relP(\#\#M, is_seqleR, nth(fg, env))$
 $\langle proof \rangle$

lemma (in M_basic) $is_related_abs$:
assumes $\bigwedge f g . M(f) \implies M(g) \implies rel(f, g) \longleftrightarrow is_rel(M, f, g)$
shows $\bigwedge z . M(z) \implies relP(M, is_rel, z) \longleftrightarrow (\exists x y . z = \langle x, y \rangle \wedge rel(x, y))$
 $\langle proof \rangle$

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 $M(A)$ $M(r)$
 $\bigwedge f g . M(f) \implies M(g) \implies rel(f, g) \longleftrightarrow is_rel(M, f, g)$
shows $is_RRel(M, is_rel, A, r) \longleftrightarrow r = Rrel(rel, A)$
 $\langle proof \rangle$

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 $is_seqlerel(M, A, r) \longleftrightarrow r = Rrel(seqleR, A)$
 $\langle proof \rangle$

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 $Rrel_eq : RrelP(R, A) = Rrel(R, A)$
 $\langle proof \rangle$

context M_ctm
begin

lemma $Rrel_closed$:
assumes $A \in M$
 $\bigwedge a . a \in nat \implies rel_fm(a) \in formula$
 $\bigwedge f g . (\#\#M)(f) \implies (\#\#M)(g) \implies rel(f, g) \longleftrightarrow is_rel(\#\#M, f, g)$
 $arity(rel_fm(0)) = 1$
 $\bigwedge a . a \in M \implies sats(M, rel_fm(0), [a]) \longleftrightarrow relP(\#\#M, is_rel, a)$
shows $(\#\#M)(Rrel(rel, A))$
 $\langle proof \rangle$

lemma $seqle_in_M : seqle \in M$

<proof>

32.2 Cohen extension is proper

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

<proof>

lemma *cohen_extension_is_proper*: $\exists G. M_generic(G) \wedge M \neq GenExt(G)$

<proof>

end

end

33 The main theorem

theory *Forcing_Main*

imports

Internal_ZFC_Axioms

Choice_Axiom

Ordinals_In_MG

Succession_Poset

begin

33.1 The generic extension is countable

definition

minimum :: $i \Rightarrow i \Rightarrow i$ **where**

minimum(r, B) \equiv *THE* $b. b \in B \wedge (\forall y \in B. y \neq b \longrightarrow \langle b, y \rangle \in r)$

lemma *well_ord_imp_min*:

assumes

well_ord(A, r) $B \subseteq A$ $B \neq 0$

shows

minimum(r, B) $\in B$

<proof>

lemma *well_ord_surj_imp_lepoll*:

assumes *well_ord*(A, r) $h \in surj(A, B)$

shows $B \lesssim A$

<proof>

lemma (**in** *forcing_data*) *surj_nat_MG* :

$\exists f. f \in surj(nat, M[G])$

<proof>

lemma (**in** *G_generic*) *MG_eqpoll_nat*: $M[G] \approx nat$

<proof>

33.2 The main result

theorem *extensions_of_ctms*:

assumes

$$M \approx \text{nat } \text{Transset}(M) \quad M \models \text{ZF}$$

shows

$\exists N.$

$$M \subseteq N \wedge N \approx \text{nat} \wedge \text{Transset}(N) \wedge N \models \text{ZF} \wedge M \neq N \wedge$$

$$(\forall \alpha. \text{Ord}(\alpha) \longrightarrow (\alpha \in M \longleftrightarrow \alpha \in N)) \wedge$$

$$(M, \Vdash \models \text{AC} \longrightarrow N \models \text{ZFC})$$

<proof>

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).