The Independence of the Continuum Hypothesis in Isabelle/ZF

Emmanuel Gunther* Miguel Pagano* Pedro Sánchez Terraf*†
Matías Steinberg*

September 1, 2025

Abstract

We redeveloped our formalization 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 proper generic extensions that satisfy the Continuum Hypothesis and its negation.

Contents

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

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

1 Introduction

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

The main entry point of the present session is `Definitions_Main.thy` (Section 33), in which a path from fundamental set theoretic concepts formalized in Isabelle reaching to our main theorems is expounded. Cross-references to major milestones are provided there.

In order to provide evidence for the correctness of several of our relativized definitions, we needed to assume the Axiom of Choice (AC) during the aforementioned theory. Nevertheless, the whole of our development is independent of AC, and the theory `CH.thy` already provides all of our results and does not import that axiom.

Release notes

Previous versions of this 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
Delta_System_Lemma.ZF_Library
begin

hide_const (**open**) *Order.pred*

2.1 Basic concepts

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

definition *compat_in* :: $i⇒i⇒i⇒i⇒o$ **where**
$compat\_in(A,r,p,q) \equiv \exists\, d∈A\ .\ ⟨d,p⟩∈r \wedge ⟨d,q⟩∈r$

lemma *compat_inI* :
$⟦\ d∈A\ ;\ ⟨d,p⟩∈r\ ;\ ⟨d,g⟩∈r\ ⟧ ⟹ compat\_in(A,r,p,g)$
by (*auto simp add*: *compat_in_def*)

lemma *refl_compat*:
$⟦\ refl(A,r)\ ;\ ⟨p,q⟩ ∈ r \mid p=q \mid ⟨q,p⟩ ∈ r\ ;\ p∈A\ ;\ q∈A⟧ ⟹ compat\_in(A,r,p,q)$
by (*auto simp add*: *refl_def compat_inI*)

lemma *chain_compat*:
$refl(A,r) ⟹ linear(A,r) ⟹ (\forall\, p∈A.\forall\, q∈A.\ compat\_in(A,r,p,q))$
by (*simp add*: *refl_compat linear_def*)

lemma *subset_fun_image*: $f:N→P ⟹ f``N⊆P$
by (*auto simp add*: *image_fun apply_funtype*)

lemma *refl_monot_domain*: $refl(B,r) ⟹ A⊆B ⟹ refl(A,r)$
unfolding *refl_def* **by** *blast*

locale *forcing_notion* =
fixes P (‹ℙ›) **and** *leq* **and** *one* (‹**1**›)
assumes *one_in_P*: $\mathbf{1} ∈ ℙ$
and *leq_preord*: $preorder\_on(ℙ,leq)$
and *one_max*: $\forall\, p∈ℙ.\ ⟨p,\mathbf{1}⟩∈leq$
begin

abbreviation *Leq* :: $[i,\ i] ⇒ o$ (**infixl** ‹$\preceq$› *50*)
where $x \preceq y \equiv ⟨x,y⟩∈leq$

lemma *refl_leq*:
$r∈ℙ ⟹ r\preceq r$
using *leq_preord* **unfolding** *preorder_on_def refl_def* **by** *simp*

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

definition

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

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

lemma *P_dense*: *dense*($\mathbb{P}$)
by (*insert leq_preord*, *auto simp add*: *preorder_on_def refl_def dense_def*)

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

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

lemma *leq_transD*: $a{\preceq}b \Longrightarrow b{\preceq}c \Longrightarrow a \in \mathbb{P} \Longrightarrow b \in \mathbb{P} \Longrightarrow c \in \mathbb{P} \Longrightarrow a{\preceq}c$
using *leq_preord trans_onD* **unfolding** *preorder_on_def* **by** *blast*

lemma *leq_transD'*: $A{\subseteq}\mathbb{P} \Longrightarrow a{\preceq}b \Longrightarrow b{\preceq}c \Longrightarrow a \in A \Longrightarrow b \in \mathbb{P} \Longrightarrow c \in \mathbb{P} \Longrightarrow a{\preceq}c$
using *leq_preord trans_onD subsetD* **unfolding** *preorder_on_def* **by** *blast*

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

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

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

lemma *Incompatible_imp_not_eq*: $[\![\ p \perp q;\ p{\in}\mathbb{P};\ q{\in}\mathbb{P}\ ]\!] \Longrightarrow p \neq q$
using *refl_leq* **by** *blast*

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

lemma *denseI* [*intro*!]: $[\![\ \bigwedge p.\ p{\in}\mathbb{P} \Longrightarrow \exists\, d{\in}D.\ d{\preceq}\ p\ ]\!] \Longrightarrow dense(D)$
unfolding *dense_def* **by** *blast*

lemma *dense_belowD* [*dest*]:
assumes *dense_below*(D,p) $q{\in}\mathbb{P}$ $q{\preceq}p$
shows $\exists\, d{\in}D.\ d{\in}\mathbb{P} \wedge d{\preceq}q$
using *assms* **unfolding** *dense_below_def* **by** *simp*

lemma *dense_belowI* [*intro*!]:
assumes $\bigwedge q.\ q{\in}\mathbb{P} \Longrightarrow q{\preceq}p \Longrightarrow \exists\, d{\in}D.\ d{\in}\mathbb{P} \wedge d{\preceq}q$
shows *dense_below*(D,p)
using *assms* **unfolding** *dense_below_def* **by** *simp*

lemma *dense_below_cong*: $p{\in}\mathbb{P} \Longrightarrow D = D' \Longrightarrow$ *dense_below*(D,p) $\longleftrightarrow$ *dense_below*(D',p)
by *blast*

lemma *dense_below_cong'*: $p{\in}\mathbb{P} \Longrightarrow [\![\bigwedge x.\ x{\in}\mathbb{P} \Longrightarrow Q(x) \longleftrightarrow Q'(x)]\!] \Longrightarrow$
dense_below($\{q{\in}\mathbb{P}.\ Q(q)\}$,$p$) $\longleftrightarrow$ *dense_below*($\{q{\in}\mathbb{P}.\ Q'(q)\}$,$p$)
by *blast*

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

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

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

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

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

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

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

lemma *filter_subset_notion*[*dest*]: *filter*(G) $\Longrightarrow G \subseteq \mathbb{P}$
by (*auto dest*:*filterD*)

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

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

lemma *low_bound_filter*: — says the compatibility is attained inside G

assumes *filter*(G) **and** $p{\in}G$ **and** $q{\in}G$
shows $\exists\, r{\in}G.\ r{\preceq}p \wedge r{\preceq}q$
using *assms*
unfolding *compat_in_def filter_def* **by** *blast*

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

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

lemma *upclosureI* [*intro*] : $p{\in}\mathbb{P} \Longrightarrow a{\in}A \Longrightarrow a{\preceq}p \Longrightarrow p{\in}$*upclosure*($A$)
by (*simp add*:*upclosure_def*, *auto*)

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

lemma *upclosureD* [*dest*] :
$p{\in}$*upclosure*(A) $\Longrightarrow \exists\, a{\in}A.(a{\preceq}p) \wedge p{\in}\mathbb{P}$
by (*simp add*:*upclosure_def*)

lemma *upclosure_increasing* :
assumes $A{\subseteq}\mathbb{P}$
shows *increasing*(*upclosure*(A))
unfolding *increasing_def upclosure_def*
using *leq_transD'*[*OF* ‹$A{\subseteq}\mathbb{P}$›] **by** *auto*

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

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

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

lemma *closure_compat_filter*:
assumes $A{\subseteq}\mathbb{P}$ ($\forall\, p{\in}A.\forall\, q{\in}A.$ *compat_in*(A,*leq*,p,q))
shows *filter*(*upclosure*(A))
unfolding *filter_def*
proof(*auto*)
show *increasing*(*upclosure*(A))
using *assms upclosure_increasing* **by** *simp*
next
let *?UA*=*upclosure*(A)
show *compat_in*(*upclosure*(A), *leq*, p, q) **if** $p{\in}$*?UA* $q{\in}$*?UA* **for** p q
proof -

```
    from that
    obtain a b where 1:a∈A b∈A a≼p b≼q p∈ℙ q∈ℙ
      using upclosureD[OF ‹p∈?UA›] upclosureD[OF ‹q∈?UA›] by auto
    with assms(2)
    obtain d where d∈A d≼a d≼b
      unfolding compat_in_def by auto
    with 1
    have d≼p d≼q d∈?UA
      using A_sub_upclosure[THEN subsetD] ‹A⊆ℙ›
        leq_transD'[of A d a] leq_transD'[of A d b] by auto
    then
    show ?thesis unfolding compat_in_def by auto
  qed
qed

lemma aux_RS1: f ∈ N → ℙ ⟹ n∈N ⟹ f‘n ∈ upclosure(f ‘‘N)
  using elem_upclosure[OF subset_fun_image] image_fun
  by (simp, blast)

lemma decr_succ_decr:
  assumes f ∈ nat → ℙ preorder_on(ℙ,leq)
    ∀ n∈nat. ⟨f ‘ succ(n), f ‘ n⟩ ∈ leq
    m∈nat
  shows n∈nat ⟹ n≤m ⟹ ⟨f ‘ m, f ‘ n⟩ ∈ leq
  using ‹m∈_›
proof(induct m)
  case 0
  then show ?case using assms refl_leq by simp
next
  case (succ x)
  then
  have 1:f‘succ(x) ≼ f‘x f‘n∈ℙ f‘x∈ℙ f‘succ(x)∈ℙ
    using assms by simp_all
  consider (lt) n<succ(x) | (eq) n=succ(x)
    using succ le_succ_iff by auto
  then
  show ?case
  proof(cases)
    case lt
    with 1 show ?thesis using leI succ leq_transD by auto
  next
    case eq
    with 1 show ?thesis using refl_leq by simp
  qed
qed

lemma decr_seq_linear:
  assumes refl(ℙ,leq) f ∈ nat → ℙ
    ∀ n∈nat. ⟨f ‘ succ(n), f ‘ n⟩ ∈ leq
```

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

end — forcing_notion
```

2.2 Towards Rasiowa-Sikorski Lemma (RSL)

```
locale countable_generic = forcing_notion +
  fixes 𝒟
  assumes countable_subs_of_P:  𝒟 ∈ nat→Pow(ℙ)
    and     seq_of_denses:          ∀ n ∈ nat. dense(𝒟`n)

begin

definition
  D_generic :: i⇒o where
  D_generic(G) ≡ filter(G) ∧ (∀ n∈nat.(𝒟`n)∩G≠0)
```

The next lemma identifies a sufficient condition for obtaining RSL.

```
lemma RS_sequence_imp_rasiowa_sikorski:
  assumes
    p∈ℙ f : nat→ℙ f ` 0 = p
    ⋀n. n∈nat ⟹ f ` succ(n)≼ f ` n ∧ f ` succ(n) ∈ 𝒟 ` n
  shows
```

```
    ∃ G. p∈G ∧ D_generic(G)
proof -
  note assms
  moreover from this
  have f‘‘nat ⊆ ℙ
    by (simp add:subset_fun_image)
  moreover from calculation
  have refl(f‘‘nat, leq) ∧ trans[ℙ](leq)
   using leq_preord unfolding preorder_on_def by (blast intro:refl_monot_domain)
  moreover from calculation
  have ∀ n∈nat.  f ‘ succ(n)⪯ f ‘ n by (simp)
  moreover from calculation
  have linear(f‘‘nat, leq)
    using leq_preord and decr_seq_linear unfolding preorder_on_def by (blast)
  moreover from calculation
  have (∀ p∈f‘‘nat.∀ q∈f‘‘nat. compat_in(f‘‘nat,leq,p,q))
    using chain_compat by (auto)
  ultimately
  have filter(upclosure(f‘‘nat)) (is filter(?G))
    using closure_compat_filter by simp
  moreover
  have ∀ n∈nat. 𝒟 ‘ n ∩ ?G ≠ 0
  proof
    fix n
    assume n∈nat
    with assms
    have f‘succ(n) ∈ ?G ∧ f‘succ(n) ∈ 𝒟 ‘ n
      using aux_RS1 by simp
    then
    show 𝒟 ‘ n ∩ ?G ≠ 0  by blast
  qed
  moreover from assms
  have p ∈ ?G
    using aux_RS1 by auto
  ultimately
  show ?thesis unfolding D_generic_def by auto
qed

end — countable_generic
```

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] ⇒ i
primrec
  RS_seq(0,P,leq,p,enum,𝒟) = p
  RS_seq(succ(n),P,leq,p,enum,𝒟) =
    enum‘(μ m. ⟨enum‘m, RS_seq(n,P,leq,p,enum,𝒟)⟩ ∈ leq ∧ enum‘m ∈ 𝒟 ‘ n)

context countable_generic
```

begin

lemma *countable_RS_sequence_aux*:
fixes p *enum*
defines $f(n) \equiv RS\_seq(n,\mathbb{P},leq,p,enum,\mathcal{D})$
and $Q(q,k,m) \equiv enum`m \preceq q \wedge enum`m \in \mathcal{D} \text{ ` } k$
assumes $n \in nat$ $p \in \mathbb{P}$ $\mathbb{P} \subseteq range(enum)$ $enum:nat \to M$
$\bigwedge x\ k.\ x \in \mathbb{P} \Longrightarrow k \in nat \Longrightarrow \exists\, q \in \mathbb{P}.\ q \preceq x \wedge q \in \mathcal{D} \text{ ` } k$
shows
$f(succ(n)) \in \mathbb{P} \wedge f(succ(n)) \preceq f(n) \wedge f(succ(n)) \in \mathcal{D} \text{ ` } n$
using ‹$n \in nat$›
proof (*induct*)
case *0*
from *assms*
obtain q **where** $q \in \mathbb{P}$ $q \preceq p$ $q \in \mathcal{D} \text{ ` } 0$ **by** *blast*
moreover from *this* **and** ‹$\mathbb{P} \subseteq range(enum)$›
obtain m **where** $m \in nat$ $enum`m = q$
using *Pi_rangeD*[*OF* ‹$enum:nat \to M$›] **by** *blast*
moreover
have $\mathcal{D}`0 \subseteq \mathbb{P}$
using *apply_funtype*[*OF countable_subs_of_P*] **by** *simp*
moreover note ‹$p \in \mathbb{P}$›
ultimately
show *?case*
using *LeastI*[*of* $Q(p,0)$ m] **unfolding** *Q_def f_def* **by** *auto*
next
case (*succ n*)
with *assms*
obtain q **where** $q \in \mathbb{P}$ $q \preceq f(succ(n))$ $q \in \mathcal{D} \text{ ` } succ(n)$ **by** *blast*
moreover from *this* **and** ‹$\mathbb{P} \subseteq range(enum)$›
obtain m **where** $m \in nat$ $enum`m \preceq f(succ(n))$ $enum`m \in \mathcal{D} \text{ ` } succ(n)$
using *Pi_rangeD*[*OF* ‹$enum:nat \to M$›] **by** *blast*
moreover note *succ*
moreover from *calculation*
have $\mathcal{D}`succ(n) \subseteq \mathbb{P}$
using *apply_funtype*[*OF countable_subs_of_P*] **by** *auto*
ultimately
show *?case*
using *LeastI*[*of* $Q(f(succ(n)),succ(n))$ m] **unfolding** *Q_def f_def* **by** *auto*
qed

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

from *assms*
show *f‘0 = p* **by** *simp*
{
fix *x k*
assume *x∈ℙ k∈nat*
then
have *∃ q∈ℙ. q≼ x ∧ q ∈ 𝒟 ‘ k*
using *seq_of_denses apply_funtype[OF countable_subs_of_P]*
unfolding *dense_def* **by** *blast*
}
with *assms*
show *f‘succ(n)≼ f‘n ∧ f‘succ(n) ∈ 𝒟 ‘ n f‘succ(n)∈ℙ*
unfolding *f_def* **using** *countable_RS_sequence_aux* **by** *simp_all*
qed

lemma *RS_seq_type*:
assumes *n ∈ nat p∈ℙ ℙ ⊆ range(enum) enum:nat→M*
shows *RS_seq(n,ℙ,leq,p,enum,𝒟) ∈ ℙ*
using *assms countable_RS_sequence(1,3)*
by (*induct*;*simp*)

lemma *RS_seq_funtype*:
assumes *p∈ℙ ℙ ⊆ range(enum) enum:nat→M*
shows *(λn∈nat. RS_seq(n,ℙ,leq,p,enum,𝒟)): nat → ℙ*
using *assms lam_type RS_seq_type* **by** *auto*

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

end — *countable_generic*

end

3 Cohen forcing notions

theory *Cohen_Posets_Relative*
imports
Forcing_Notions
Transitive_Models.Delta_System_Relative
Transitive_Models.Partial_Functions_Relative
begin

locale *cohen_data =*
fixes *κ I J::i*
assumes *zero_lt_kappa*: *0<κ*
begin

lemmas *zero_lesspoll_kappa = zero_lesspoll[OF zero_lt_kappa]*

end — *cohen_data*

abbreviation
inj_dense :: $[i,i,i,i]\Rightarrow i$ **where**
inj_dense$(I,J,w,x) \equiv$
$\{\ p{\in}Fn(\omega,I{\times}\omega,J)\ .\ (\exists\, n{\in}\omega.\ \langle\langle w,n\rangle,1\rangle \in p \wedge \langle\langle x,n\rangle,0\rangle \in p)\ \}$

lemma *dense_inj_dense*:
assumes $w \in I\ x \in I\ w \neq x\ p{\in}Fn(\omega,I{\times}\omega,J)\ 0{\in}J\ 1{\in}J$
shows $\exists\, d{\in}inj\_dense(I,J,w,x).\ \langle d\ ,p\rangle \in Fnle(\omega,I{\times}\omega,J)$
proof -
obtain n **where** $\langle w,n\rangle \notin domain(p)\ \langle x,n\rangle \notin domain(p)\ n \in \omega$
proof -
{
assume $\langle w,n\rangle \in domain(p) \vee \langle x,n\rangle \in domain(p)$ **if** $n \in \omega$ **for** n
then
have $\omega \subseteq range(domain(p))$ **by** *blast*
then
have $\neg\ Finite(p)$
using *Finite_range Finite_domain subset_Finite nat_not_Finite*
by *auto*
with ‹$p \in \_$›
have *False*
using *Fn_nat_eq_FiniteFun FiniteFun.dom_subset*[*of* $I{\times}\omega\ J$]
Fin_into_Finite **by** *auto*
}
with *that*— the shape of the goal puts assumptions in this variable
show *?thesis* **by** *auto*
qed
moreover
note ‹$p \in \_$› *assms*
moreover from *calculation*
have *cons*$(\langle\langle x,n\rangle,0\rangle,\ p) \in Fn(\omega,I{\times}\omega,J)$
using *FiniteFun.consI*[*of* $\langle x,n\rangle\ I{\times}\omega\ 0\ J\ p$]
Fn_nat_eq_FiniteFun **by** *auto*
ultimately
have *cons*$(\langle\langle w,n\rangle,1\rangle,\ cons(\langle\langle x,n\rangle,0\rangle,\ p)\ ) \in Fn(\omega,I{\times}\omega,J)$
using *FiniteFun.consI*[*of* $\langle w,n\rangle\ I \times \omega\ 1\ J\ cons(\langle\langle x,n\rangle,0\rangle,\ p)$]
Fn_nat_eq_FiniteFun **by** *auto*
with ‹$n \in \omega$›
show *?thesis*
using ‹$p \in \_$› **by** (*intro bexI*) *auto*
qed

locale *add_reals* = *cohen_data nat _ 2*

3.1 Combinatorial results on Cohen posets

sublocale *cohen_data* $\subseteq$ *forcing_notion* $Fn(\kappa,I,J)\ Fnle(\kappa,I,J)\ 0$

proof
show $0 \in Fn(\kappa, I, J)$
using *zero_lt_kappa zero_in_Fn* **by** *simp*
then
show $\forall\, p{\in}Fn(\kappa, I, J).\ \langle p, 0\rangle \in Fnle(\kappa, I, J)$
unfolding *preorder_on_def refl_def trans_on_def*
by *auto*
next
show *preorder_on*$(Fn(\kappa, I, J), Fnle(\kappa, I, J))$
unfolding *preorder_on_def refl_def trans_on_def*
by *blast*
qed

context *cohen_data*
begin

lemma *compat_imp_Un_is_function*:
assumes $G \subseteq Fn(\kappa, I, J)$ $\bigwedge p\ q.\ p \in G \Longrightarrow q \in G \Longrightarrow compat(p,q)$
shows *function*$(\bigcup G)$
unfolding *function_def*
proof (*intro allI ballI impI*)
fix $x\ y\ y'$
assume $\langle x, y\rangle \in \bigcup G\ \langle x, y'\rangle \in \bigcup G$
then
obtain $p\ q$ **where** $\langle x, y\rangle \in p\ \langle x, y'\rangle \in q\ p \in G\ q \in G$
by *auto*
moreover from *this* **and** *assms*
obtain r **where** $r \in Fn(\kappa, I, J)\ r \supseteq p\ r \supseteq q$
using *compatD*[*of p q*] **by** *force*
moreover from *this*
have *function*(r)
using *Fn_is_function* **by** *simp*
ultimately
show $y = y'$
unfolding *function_def* **by** *fastforce*
qed

lemma *Un_filter_is_function*: *filter*$(G) \Longrightarrow$ *function*$(\bigcup G)$
using *compat_imp_Un_is_function filter_imp_compat*[*of G*]
filter_subset_notion
by *simp*

end — *cohen_data*

locale *M_cohen* = *M_delta* +
assumes
countable_lepoll_assms2:

$M(A') \Longrightarrow M(A) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow separation(M,\ \lambda y.\ \exists\, x{\in}A'.\ y = \langle x,\ \mu\ i.\ x \in if\_range\_F\_else\_F(\lambda a.\ \{p \in A\ .\ domain(p) = a\},\ b,\ f,\ i)\rangle)$

and

countable_lepoll_assms3:

$M(A) \Longrightarrow M(f) \Longrightarrow M(b) \Longrightarrow M(D) \Longrightarrow M(r') \Longrightarrow M(A') \Longrightarrow$
$separation(M,\ \lambda y.\ \exists\, x{\in}A'.\ y = \langle x,\ \mu\ i.\ x \in if\_range\_F\_else\_F(drSR\_Y(r',\ D,\ A),\ b,\ f,\ i)\rangle)$

lemma (**in** *M_library*) *Fnle_rel_Aleph_rel1_closed*[*intro*,*simp*]:
$M(Fnle^M(\aleph_1{}^M,\ \aleph_1{}^M,\ \omega \to^M 2))$
by *simp*

locale *M_add_reals* = *M_cohen* + *add_reals*
begin

lemmas *zero_lesspoll_rel_kappa* = *zero_lesspoll_rel*[*OF zero_lt_kappa*]

end — *M_add_reals*

relativize relational *compat_in is_compat_in* **external**
synthesize *compat_in* **from_definition** *is_compat_in* **assuming** *nonempty*
context
notes *Un_assoc*[*simp*] *Un_trasposition_aux2*[*simp*]
begin
arity_theorem for *compat_in_fm*
end

lemma (**in** *M_trivial*) *compat_in_abs*[*absolut*]:
assumes
$M(A)\ M(r)\ M(p)\ M(q)$
shows
$is\_compat\_in(M,A,r,p,q) \longleftrightarrow compat\_in(A,r,p,q)$
using *assms* **unfolding** *is_compat_in_def compat_in_def* **by** *simp*

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

relativize relational *antichain antichain_rel*

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

abbreviation
$antichain\_rel\_abbr :: [i{\Rightarrow}o,i,i,i] \Rightarrow o\ (\langle antichain{-}'(\_,\_,\_')\rangle)$ **where**
$antichain^M(P,leq,A) \equiv antichain\_rel(M,P,leq,A)$

abbreviation
antichain_r_set :: [*i*,*i*,*i*,*i*] ⇒ *o* (‹*antichain*–′(_,_,_′)›) **where**
antichain^M(*P*,*leq*,*A*) ≡ *antichain_rel*(##*M*,*P*,*leq*,*A*)

context *M_trivial*
begin

lemma *antichain_abs* [*absolut*]:
⟦ *M*(*A*); *M*(*P*); *M*(*leq*) ⟧ ⟹ *antichain*^{M}(*P*,*leq*,*A*) ⟷ *antichain*(*P*,*leq*,*A*)
unfolding *antichain_rel_def antichain_def* **by** (*simp add*:*absolut*)

end — *M_trivial*

relativize relational *ccc ccc_rel*

abbreviation
ccc_rel_abbr :: [*i*⇒*o*,*i*,*i*]⇒*o* (‹*ccc*–′(_,_′)›) **where**
ccc_rel_abbr(*M*) ≡ *ccc_rel*(*M*)

abbreviation
ccc_r_set :: [*i*,*i*,*i*]⇒*o* (‹*ccc*–′(_,_′)›) **where**
ccc_r_set(*M*) ≡ *ccc_rel*(##*M*)

context *M_cardinals*
begin

lemma *def_ccc_rel*:
shows
ccc^M(*P*,*leq*) ⟷ (∀ *A*[*M*]. *antichain*^{M}(*P*,*leq*,*A*) ⟶ |*A*|^M ≤ *ω*)
using *is_cardinal_iff*
unfolding *ccc_rel_def* **by** (*simp add*:*absolut*)

end — *M_cardinals*

context *M_FiniteFun*
begin

lemma *Fnle_nat_closed*[*intro*,*simp*]:
assumes *M*(*I*) *M*(*J*)
shows *M*(*Fnle*(*ω*,*I*,*J*))
unfolding *Fnle_def Fnlerel_def Rrel_def*
using *supset_separation FiniteFun_closed Fn_nat_eq_FiniteFun assms* **by** *simp*

lemma *Fn_nat_closed*:
assumes *M*(*A*) *M*(*B*) **shows** *M*(*Fn*(*ω*,*A*,*B*))
using *assms Fn_nat_eq_FiniteFun*
by *simp*

end — *M_FiniteFun*

context *M_add_reals*
begin

lemma *lam_replacement_drSR_Y*: $M(A) \Longrightarrow M(D) \Longrightarrow M(r') \Longrightarrow$ *lam_replacement*(M, *drSR_Y*(r',D,A))
  using *lam_replacement_drSR_Y*
  by *simp*

lemma (**in** *M_trans*) *mem_F_bound3*:
  fixes F A
  defines $F \equiv dC\_F$
  shows $x{\in}F(A,c) \Longrightarrow c \in (range(f) \cup \{domain(x).\ x{\in}A\})$
  using *apply_0* **unfolding** *F_def*
  by (*cases* $M(c)$, *auto simp*:*F_def drSR_Y_def dC_F_def*)

lemma *ccc_rel_Fn_nat*:
  assumes $M(I)$
  shows $ccc^M(Fn(nat,I,2),\ Fnle(nat,I,2))$
proof -
  have *repFun_dom_closed*:$M(\{domain(p)\ .\ p \in A\})$ **if** $M(A)$ **for** A
    using *RepFun_closed domain_replacement_simp transM*[*OF* _ ‹$M(A)$›] *that*
    by *auto*
  from *assms*
  have $M(Fn(nat,I,2))$ **using** *Fn_nat_eq_FiniteFun* **by** *simp*
  {
    fix A
    assume $\neg\ |A|^M \le nat\ M(A)\ A \subseteq Fn(nat,\ I,\ 2)$
    moreover from *this*
    have *countable_rel*(M,$\{p{\in}A.\ domain(p) = d\}$) **if** $M(d)$ **for** d
    proof (*cases* $d{\prec}^M nat \wedge d \subseteq I$)
      case *True*
      with ‹$A \subseteq Fn(nat,\ I,\ 2)$› ‹$M(A)$›
      have $\{p \in A\ .\ domain(p) = d\} \subseteq d \rightarrow^M 2$
        using *domain_of_fun function_space_rel_char*[*of* _ *2*]
        by (*auto*,*subgoal_tac* $M(domain(x))$,*simp_all add*:*transM*[*of* _ A],*force*)
      moreover from *True* ‹$M(d)$›
      have *Finite*($d \rightarrow^M 2$)
        using *Finite_Pi*[*THEN* [*2*] *subset_Finite*, *of* _ d λ_. *2*]
          lesspoll_rel_nat_is_Finite_rel function_space_rel_char[*of* _ *2*]
        by *auto*
      moreover from ‹$M(d)$›
      have $M(d \rightarrow^M 2)$
        by *simp*
      moreover from ‹$M(A)$›
      have $M(\{p \in A\ .\ domain(p) = d\})$
        using *separation_closed domain_eq_separation*[*OF* ‹$M(d)$›] **by** *simp*
      ultimately
      show *?thesis* **using** *subset_Finite*[*of* _ $d{\rightarrow}^M 2$]

```
      by (auto intro!:Finite_imp_countable_rel)
    next
      case False
      with ‹A ⊆ Fn(nat, I, 2)› ‹M(A)›
      have domain(p) ≠ d if p∈A for p
      proof -
        note False that ‹M(A)›
        moreover from this
        obtain d' where d' ⊆ I p∈d' → 2 d' ≺ ω
          using FnD[OF subsetD[OF ‹A⊆_› ‹p∈A›]]
          by auto
        moreover from this
        have p ≈ d' domain(p) = d'
          using function_eqpoll Pi_iff
          by auto
        ultimately
        show ?thesis
          using lesspoll_nat_imp_lesspoll_rel transM[of p]
          by auto
      qed
      then
      show ?thesis
        using empty_lepoll_relI by auto
    qed
    have 2:M(x) ⟹ x ∈ dC_F(X, i) ⟹ M(i) for x X i
      unfolding dC_F_def
      by auto
    moreover
    have uncountable_rel(M,{domain(p) . p ∈ A})
    proof
      interpret M_replacement_lepoll M dC_F
     using lam_replacement_dC_F domain_eq_separation lam_replacement_inj_rel
lam_replacement_minimum
        unfolding dC_F_def
      proof(unfold_locales,simp_all)
        fix X b f
        assume M(X) M(b) M(f)
        with 2[of X]
        show lam_replacement(M, λx. μ i. x ∈ if_range_F_else_F(λd. {p ∈ X .
domain(p) = d}, b, f, i))
          using lam_replacement_dC_F domain_eq_separation
            mem_F_bound3 countable_lepoll_assms2 repFun_dom_closed
         by (rule_tac lam_Least_assumption_general[where U=λ_. {domain(x).
x∈X}],auto)
      qed (auto)
      have ∃ a∈A. x = domain(a) ⟹ M(dC_F(A,x)) for x
        using ‹M(A)› transM[OF _ ‹M(A)›] by (auto)
      moreover
      have w ∈ A ∧ domain(w) = x ⟹ M(x) for w x
```

using *transM*[*OF* _ ‹*M*(*A*)›] **by** *auto*
ultimately
interpret *M_cardinal_UN_lepoll* _ *dC_F*(*A*) {*domain*(*p*). *p*∈*A*}
using *lam_replacement_dC_F lam_replacement_inj_rel* ‹*M*(*A*)›
lepoll_assumptions domain_eq_separation lam_replacement_minimum
countable_lepoll_assms2 repFun_dom_closed
lepoll_assumptions1[*OF* ‹*M*(*A*)› *repFun_dom_closed*[*OF* ‹*M*(*A*)›]]
apply(*unfold_locales*)
by(*simp_all del*:*if_range_F_else_F_def*,
rule_tac lam_Least_assumption_general[**where** *U*=λ_. {*domain*(*x*). *x*∈*A*}])
(*auto simp del*:*if_range_F_else_F_def simp add*:*dC_F_def*)
from ‹*A* ⊆ *Fn*(*nat*, *I*, *2*)›
have *x*:(⋃ *d*∈{*domain*(*p*) . *p* ∈ *A*}. {*p*∈*A*. *domain*(*p*) = *d*}) = *A*
by *auto*
moreover
assume *countable_rel*(*M*,{*domain*(*p*) . *p* ∈ *A*})
moreover
note ‹⋀*d*. *M*(*d*) ⟹ *countable_rel*(*M*,{*p*∈*A*. *domain*(*p*) = *d*})›
moreover from ‹*M*(*A*)›
have *p*∈*A* ⟹ *M*(*domain*(*p*)) **for** *p*
by (*auto dest*: *transM*)
ultimately
have *countable_rel*(*M*,*A*)
using *countable_rel_imp_countable_rel_UN*
unfolding *dC_F_def*
by *auto*
with ‹¬ |*A*|^M ≤ *nat*› ‹*M*(*A*)›
show *False*
using *countable_rel_iff_cardinal_rel_le_nat* **by** *simp*
qed
moreover from ‹*A* ⊆ *Fn*(*nat*, *I*, *2*)› ‹*M*(*A*)›
have *p* ∈ *A* ⟹ *Finite*(*domain*(*p*)) **for** *p*
using *lesspoll_rel_nat_is_Finite_rel*[*of domain*(*p*)]
lesspoll_nat_imp_lesspoll_rel[*of domain*(*p*)]
domain_of_fun[*of p* _ λ_. *2*] **by** (*auto dest*:*transM*)
moreover
note *repFun_dom_closed*[*OF* ‹*M*(*A*)›]
ultimately
obtain *D* **where** *delta_system*(*D*) *D* ⊆ {*domain*(*p*) . *p* ∈ *A*} *D* ≈^{M} ℵ$_1^M$ *M*(*D*)
using *delta_system_uncountable_rel*[*of* {*domain*(*p*) . *p* ∈ *A*}] **by** *auto*
then
have *delta*:∀ *d1*∈*D*. ∀ *d2*∈*D*. *d1* ≠ *d2* ⟶ *d1* ∩ *d2* = ⋂ *D*
using *delta_system_root_eq_Inter*
by *simp*
moreover from ‹*D* ≈^{M} ℵ$_1^M$› ‹*M*(*D*)›
have *uncountable_rel*(*M*,*D*)
using *uncountable_rel_iff_subset_eqpoll_rel_Aleph_rel1* **by** *auto*
moreover from *this* **and** ‹*D* ⊆ {*domain*(*p*) . *p* ∈ *A*}›

```
    obtain p1 where p1 ∈ A domain(p1) ∈ D
      using uncountable_rel_not_empty[of D] by blast
    moreover from this and ‹p1 ∈ A ⟹ Finite(domain(p1))›
    have Finite(domain(p1))
      using Finite_domain by simp
    moreover
    define r where r ≡ ⋂ D
    moreover from ‹M(D)›
    have M(r) M(r×2)
      unfolding r_def by simp_all
    ultimately
    have Finite(r) using subset_Finite[of r domain(p1)]
      by auto
    have countable_rel(M,{restrict(p,r) . p∈A})
    proof -
      note ‹M(Fn(nat, I, 2))› ‹M(r)›
      moreover from this
      have f∈Fn(nat, I, 2) ⟹ M(restrict(f,r)) for f
        by (blast dest: transM)
      ultimately
      have f∈Fn(nat, I, 2) ⟹ restrict(f,r) ∈ Pow_rel(M,r × 2) for f
        using restrict_subset_Sigma[of f _ λ_. 2 r] Pow_rel_char
        by (auto del:FnD dest!:FnD simp: Pi_def) (auto dest:transM)
      with ‹A ⊆ Fn(nat, I, 2)›
      have {restrict(f,r) . f ∈ A } ⊆ Pow_rel(M,r × 2)
        by fast
      moreover from ‹M(A)› ‹M(r)›
      have M({restrict(f,r) . f ∈ A })
        using RepFun_closed restrict_strong_replacement transM[OF _ ‹M(A)›]
by auto
      moreover
      note ‹Finite(r)› ‹M(r)›
      ultimately
      show ?thesis
        using Finite_Sigma[THEN Finite_Pow_rel, of r λ_. 2]
        by (intro Finite_imp_countable_rel) (auto intro:subset_Finite)
    qed
    moreover from ‹M(A)› ‹M(D)›
    have M({p∈A. domain(p) ∈ D})
      using domain_mem_separation by simp
    have uncountable_rel(M,{p∈A. domain(p) ∈ D}) (is uncountable_rel(M,?X))
    proof
      from ‹D ⊆ {domain(p) . p ∈ A}›
      have (λp∈?X. domain(p)) ∈ surj(?X, D)
        using lam_type unfolding surj_def by auto
      moreover from ‹M(A)› ‹M(?X)›
      have M(λp∈?X. domain(p))
      using lam_closed[OF domain_replacement ‹M(?X)›] transM[OF _ ‹M(?X)›]
by simp
```

```
    moreover
    note ‹M(?X)› ‹M(D)›
    moreover from calculation
    have surjection:(λp∈?X. domain(p)) ∈ surj^M(?X, D)
      using surj_rel_char by simp
    moreover
    assume countable_rel(M,?X)
    moreover
    note ‹uncountable_rel(M,D)›
    ultimately
    show False
      using surj_rel_countable_rel[OF _ surjection] by auto
  qed
  moreover
  have D = (⋃f∈Pow_rel(M,r×2) . {y . p∈A, restrict(p,r) = f ∧ y=domain(p)
∧ domain(p) ∈ D})
  proof -
    {
      fix z
      assume z ∈ D
      with ‹M(D)›
      have ‹M(z)› by (auto dest:transM)
      from ‹z∈D› ‹D ⊆ _› ‹M(A)›
      obtain p  where domain(p) = z p ∈ A M(p)
        by (auto dest:transM)
      moreover from ‹A ⊆ Fn(nat, I, 2)› ‹M(z)› and this
      have p ∈ z →^M 2
            using domain_of_fun function_space_rel_char by (auto del:FnD
dest!:FnD)
      moreover from this ‹M(z)›
      have p ∈ z → 2
        using domain_of_fun function_space_rel_char by (auto)
      moreover from this ‹M(r)›
      have restrict(p,r) ⊆ r × 2
        using function_restrictI[of p r] fun_is_function[of p z λ_. 2]
          restrict_subset_Sigma[of p z λ_. 2 r] function_space_rel_char
        by (auto simp:Pi_def)
      moreover from ‹M(p)› ‹M(r)›
      have M(restrict(p,r)) by simp
      moreover
      note ‹M(r)›
      ultimately
      have ∃p∈A.  restrict(p,r) ∈ Pow_rel(M,r×2) ∧ domain(p) = z
        using Pow_rel_char by auto
    }
    then
    show ?thesis
      by (intro equalityI) (force)+
  qed
```

```
    from ‹M(D)›‹M(r)›
     have M({y . p∈A, restrict(p,r) = f ∧ y=domain(p) ∧ domain(p) ∈ D}) (is
M(?Y(A,f)))
      if M(f) M(A) for f A
      using drSR_Y_closed[unfolded drSR_Y_def] that
      by simp
    then
    obtain f where uncountable_rel(M,?Y(A,f)) M(f)
    proof -
      have 1:M(i)
        if M(B) M(x)
          x ∈ {y . x ∈ B, restrict(x, r) = i ∧ y = domain(x) ∧ domain(x) ∈ D}
        for B x i
        using that ‹M(r)›
        by (auto dest:transM)
      note ‹M(r)›
      moreover from this
      have M(Pow^M(r × 2)) by simp
      moreover
      note ‹M(A)› ‹⋀f A. M(f) ⟹ M(A) ⟹ M(?Y(A,f))› ‹M(D)›
      moreover from calculation
      interpret M_replacement_lepoll M drSR_Y(r,D)
     using countable_lepoll_assms3 lam_replacement_inj_rel lam_replacement_drSR_Y
      drSR_Y_closed lam_Least_assumption_drSR_Y lam_replacement_minimum
        by (unfold_locales,simp_all add:drSR_Y_def,rule_tac 1,simp_all)
      from calculation
      have x ∈ Pow^M(r × 2) ⟹ M(drSR_Y(r, D, A, x)) for x
        unfolding drSR_Y_def by (auto dest:transM)
      ultimately
      interpret M_cardinal_UN_lepoll _ ?Y(A) Pow_rel(M,r×2)
        using countable_lepoll_assms3 lam_replacement_drSR_Y
       lepoll_assumptions[where S=Pow_rel(M,r×2), unfolded lepoll_assumptions_defs]
       lam_Least_assumption_drSR_Y[unfolded drSR_Y_def] lam_replacement_minimum
        unfolding drSR_Y_def
        apply unfold_locales
         apply (simp_all add:lam_replacement_inj_rel del: if_range_F_else_F_def,rule_tac
1,simp_all)
        by ((fastforce dest:transM[OF _ ‹M(A)›])+)
      {
        from ‹Finite(r)› ‹M(r)›
        have countable_rel(M,Pow_rel(M,r×2))
          using Finite_Sigma[THEN Finite_Pow_rel] Finite_imp_countable_rel
          by simp
        moreover
        assume M(f) ⟹ countable_rel(M,?Y(A,f)) for f
        moreover
        note ‹D = (⋃f∈Pow_rel(M,r×2) .?Y(A,f))› ‹M(r)›
        moreover
        note ‹uncountable_rel(M,D)›
```

```
      ultimately
      have False
        using countable_rel_imp_countable_rel_UN by (auto dest: transM)
    }
    with that
    show ?thesis
      by auto
  qed
  moreover from this ‹M(A)› and ‹M(f) ⟹ M(A) ⟹ M(?Y(A,f))›
  have M(?Y(A,f))
    by blast
  ultimately
  obtain j where j ∈ inj_rel(M,nat, ?Y(A,f)) M(j)
    using uncountable_rel_iff_nat_lt_cardinal_rel[THEN iffD1, THEN leI,
        THEN cardinal_rel_le_imp_lepoll_rel, THEN lepoll_relD]
    by auto
  with ‹M(?Y(A,f))›
  have j'0 ≠ j'1 j'0 ∈ ?Y(A,f) j'1 ∈ ?Y(A,f)
    using inj_is_fun[THEN apply_type, of j nat ?Y(A,f)]
      inj_rel_char
    unfolding inj_def by auto
  then
  obtain p q where domain(p) ≠ domain(q) p ∈ A q ∈ A
    domain(p) ∈ D domain(q) ∈ D
    restrict(p,r) = restrict(q,r) by auto
  moreover from this and delta
  have domain(p) ∩ domain(q) = r
    unfolding r_def by simp
  moreover
  note ‹A ⊆ Fn(nat, I, 2)› Fn_nat_abs[OF ‹M(I)› nat_into_M[of 2],simplified]
  moreover from calculation
  have p ∈ Fn^M(nat, I, 2) q ∈ Fn^M(nat, I, 2)
    by auto
  moreover from calculation
  have p ∪ q ∈ Fn(nat, I, 2)
    using restrict_eq_imp_compat_rel InfCard_rel_nat
    by simp
  ultimately
  have ∃ p∈A. ∃ q∈A. p ≠ q ∧ compat_in(Fn(nat, I, 2), Fnle(nat, I, 2), p, q)
    unfolding compat_in_def
    by (rule_tac bexI[of _ p], rule_tac bexI[of _ q]) blast
}
moreover from assms
have M(Fnle(ω,I,2))
  by simp
moreover note ‹M(Fn(ω,I,2))›
ultimately
show ?thesis using def_ccc_rel by (auto simp:absolut antichain_def) fastforce
qed
```

end — *M_add_reals*

end
theory *Edrel*
imports
Transitive_Models.ZF_Miscellanea
Transitive_Models.Recursion_Thms

begin

3.2 The well-founded relation *ed*

lemma *eclose_sing* : $x \in eclose(a) \Longrightarrow x \in eclose(\{a\})$
using *subsetD*[*OF mem_eclose_subset*]
by *simp*

lemma *ecloseE* :
assumes $x \in eclose(A)$
shows $x \in A \vee (\exists\ B \in A\ .\ x \in eclose(B))$
using *assms*
proof (*induct rule*:*eclose_induct_down*)
case (*1 y*)
then
show *?case*
using *arg_into_eclose* **by** *auto*
next
case (*2 y z*)
from ‹$y \in A \vee (\exists\, B{\in}A.\ y \in eclose(B))$›
consider (*inA*) $y \in A$ | (*exB*) $(\exists\, B{\in}A.\ y \in eclose(B))$
by *auto*
then show *?case*
proof (*cases*)
case *inA*
then
show *?thesis* **using** *2 arg_into_eclose* **by** *auto*
next
case *exB*
then obtain *B* **where** $y \in eclose(B)$ $B{\in}A$
by *auto*
then
show *?thesis* **using** *2 ecloseD*[*of y B z*] **by** *auto*
qed
qed

lemma *eclose_singE* : $x \in eclose(\{a\}) \Longrightarrow x = a \vee x \in eclose(a)$
by(*blast dest*: *ecloseE*)

lemma *in_eclose_sing* :

assumes $x \in eclose(\{a\})$ $a \in eclose(z)$
shows $x \in eclose(\{z\})$
proof -
from ‹$x \in eclose(\{a\})$›
consider $x{=}a \mid x \in eclose(a)$
using *eclose_singE* **by** *auto*
then
show *?thesis*
using *eclose_sing mem_eclose_trans assms*
by (*cases*, *auto*)
qed

lemma *in_dom_in_eclose* :
assumes $x \in domain(z)$
shows $x \in eclose(z)$
proof -
from *assms*
obtain y **where** $\langle x,y\rangle \in z$
unfolding *domain_def* **by** *auto*
then
show *?thesis*
unfolding *Pair_def*
using *ecloseD*[*of* $\{x,x\}$] *ecloseD*[*of* $\{\{x,x\},\{x,y\}\}$] *arg_into_eclose*
by *auto*
qed

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

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

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

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

lemma *edD*[*dest*!]: $ed(t,x) \Longrightarrow t \in domain(x)$
unfolding *ed_def* **.**

lemma *rank_ed*:
assumes $ed(y,x)$
shows $succ(rank(y)) \leq rank(x)$
proof
from *assms*
obtain p **where** $\langle y,p\rangle \in x$ **by** *auto*
moreover
obtain s **where** $y \in s$ $s \in \langle y,p\rangle$ **unfolding** *Pair_def* **by** *auto*
ultimately
have $rank(y) < rank(s)$ $rank(s) < rank(\langle y,p\rangle)$ $rank(\langle y,p\rangle) < rank(x)$

```
    using rank_lt by blast+
  then
  show rank(y) < rank(x)
    using lt_trans by blast
qed

lemma edrel_dest [dest]: x ∈ edrel(A) ⟹ ∃ a ∈ A. ∃ b ∈ A. x =⟨a,b⟩
  by(auto simp add:ed_def edrel_def Rrel_def)

lemma edrelD : x ∈ edrel(A) ⟹ ∃ a∈ A. ∃ b ∈ A. x =⟨a,b⟩ ∧ a ∈ domain(b)
  by(auto simp add:ed_def edrel_def Rrel_def)

lemma edrelI [intro!]: x∈A ⟹ y∈A ⟹ x ∈ domain(y) ⟹ ⟨x,y⟩∈edrel(A)
  by (simp add:ed_def edrel_def Rrel_def)

lemma edrel_trans: Transset(A) ⟹ y∈A ⟹ x ∈ domain(y) ⟹ ⟨x,y⟩∈edrel(A)
  by (rule edrelI, auto simp add:Transset_def domain_def Pair_def)

lemma domain_trans: Transset(A) ⟹ y∈A ⟹ x ∈ domain(y) ⟹ x∈A
  by (auto simp add: Transset_def domain_def Pair_def)

lemma relation_edrel : relation(edrel(A))
  by(auto simp add: relation_def)

lemma field_edrel : field(edrel(A))⊆A
  by blast

lemma edrel_sub_memrel: edrel(A) ⊆ trancl(Memrel(eclose(A)))
proof
  let
    ?r=trancl(Memrel(eclose(A)))
  fix z
  assume z∈edrel(A)
  then
  obtain x y where x∈A y∈A z=⟨x,y⟩ x∈domain(y)
    using edrelD
    by blast
  moreover from this
  obtain u v where x∈u u∈v v∈y
    unfolding domain_def Pair_def by auto
  moreover from calculation
  have x∈eclose(A) y∈eclose(A) y⊆eclose(A)
    using arg_into_eclose Transset_eclose[unfolded Transset_def]
    by simp_all
  moreover from calculation
  have v∈eclose(A)
    by auto
  moreover from calculation
  have u∈eclose(A)
```

```
    using Transset_eclose[unfolded Transset_def]
    by auto
  moreover from calculation
  have⟨x,u⟩∈?r ⟨u,v⟩∈?r ⟨v,y⟩∈?r
    by (auto simp add: r_into_trancl)
  moreover from this
  have ⟨x,y⟩∈?r
    using trancl_trans[OF _ trancl_trans[of _ v _ y]]
    by simp
  ultimately
  show z∈?r
    by simp
qed

lemma wf_edrel : wf(edrel(A))
  using wf_subset[of trancl(Memrel(eclose(A)))] edrel_sub_memrel
    wf_trancl wf_Memrel
  by auto

lemma ed_induction:
  assumes ⋀x. ⟦⋀y.  ed(y,x) ⟹ Q(y) ⟧ ⟹ Q(x)
  shows Q(a)
proof(induct rule: wf_induct2[OF wf_edrel[of eclose({a})] ,of a eclose({a})])
  case 1
  then show ?case using arg_into_eclose by simp
next
  case 2
  then show ?case using field_edrel .
next
  case (3 x)
  then
  show ?case
    using assms[of x] edrelI domain_trans[OF Transset_eclose 3(1)] by blast
qed

lemma dom_under_edrel_eclose: edrel(eclose({x})) -`` {x} = domain(x)
proof(intro equalityI)
  show edrel(eclose({x})) -`` {x} ⊆ domain(x)
    unfolding edrel_def Rrel_def ed_def
    by auto
next
  show domain(x) ⊆ edrel(eclose({x})) -`` {x}
    unfolding edrel_def Rrel_def
    using in_dom_in_eclose eclose_sing arg_into_eclose
    by blast
qed

lemma ed_eclose : ⟨y,z⟩ ∈ edrel(A) ⟹ y ∈ eclose(z)
  by(drule edrelD,auto simp add:domain_def in_dom_in_eclose)
```

lemma *tr_edrel_eclose* : $\langle y,z\rangle \in edrel(eclose(\{x\}))^+ \Longrightarrow y \in eclose(z)$
by(*rule trancl_induct*,(*simp add*: *ed_eclose mem_eclose_trans*)+)

lemma *restrict_edrel_eq* :
assumes $z \in domain(x)$
shows $edrel(eclose(\{x\})) \cap eclose(\{z\})\times eclose(\{z\}) = edrel(eclose(\{z\}))$
proof(*intro equalityI subsetI*)
let *?ec*=λ y . $edrel(eclose(\{y\}))$
let *?ez*=$eclose(\{z\})$
let *?rr*=*?ec*$(x) \cap$ *?ez* $\times$ *?ez*
fix y
assume $y \in$ *?rr*
then
obtain a b **where** $\langle a,b\rangle \in$ *?rr* $a \in$ *?ez* $b \in$ *?ez* $\langle a,b\rangle \in$ *?ec*(x) y=$\langle a,b\rangle$
by *blast*
moreover from *this*
have $a \in domain(b)$
using *edrelD* **by** *blast*
ultimately
show $y \in edrel(eclose(\{z\}))$
by *blast*
next
let *?ec*=λ y . $edrel(eclose(\{y\}))$
let *?ez*=$eclose(\{z\})$
let *?rr*=*?ec*$(x) \cap$ *?ez* $\times$ *?ez*
fix y
assume $y \in edrel($*?ez*$)$
then
obtain a b **where** $a \in$ *?ez* $b \in$ *?ez* y=$\langle a,b\rangle$ $a \in domain(b)$
using *edrelD* **by** *blast*
moreover
from *this assms*
have $z \in eclose(x)$
using *in_dom_in_eclose* **by** *simp*
moreover
from *assms calculation*
have $a \in eclose(\{x\})$ $b \in eclose(\{x\})$
using *in_eclose_sing* **by** *simp_all*
moreover from *calculation*
have $\langle a,b\rangle \in edrel(eclose(\{x\}))$
by *blast*
ultimately
show $y \in$ *?rr*
by *simp*
qed

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

shows *tr_down(edrel(eclose({x})),z) ⊆ eclose({z})*
proof(*intro subsetI*)
let *?r=λ x . edrel(eclose({x}))*
fix *y*
assume *y ∈ tr_down(?r(x),z)*
then
have *⟨y,z⟩ ∈ ?r(x)^+*
using *tr_downD* **by** *simp*
with *assms*
show *y ∈ eclose({z})*
using *tr_edrel_eclose eclose_sing* **by** *simp*
qed

end

4 Well-founded relation on names

theory *FrecR*
imports
Transitive_Models.Discipline_Function
Edrel
begin

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

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

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

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

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

lemma *components_simp*:
ftype(⟨f,n1,n2,c⟩) = f
name1(⟨f,n1,n2,c⟩) = n1
name2(⟨f,n1,n2,c⟩) = n2
cond_of(⟨f,n1,n2,c⟩) = c
unfolding *ftype_def name1_def name2_def cond_of_def*
by *simp_all*

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

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

lemma *components_in_eclose* :
n1 ∈ *ecloseN*(⟨*f*,*n1*,*n2*,*c*⟩)
n2 ∈ *ecloseN*(⟨*f*,*n1*,*n2*,*c*⟩)
unfolding *ecloseN_def eclose_n_def*
using *components_simp arg_into_eclose* **by** *auto*

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

lemma *ecloseNI1* :
assumes *x* ∈ *eclose*(*n1*) ∨ *x*∈*eclose*(*n2*)
shows *x* ∈ *ecloseN*(⟨*f*,*n1*,*n2*,*c*⟩)
unfolding *ecloseN_def eclose_n_def*
using *assms eclose_sing names_simp*
by *auto*

lemmas *ecloseNI* = *ecloseNI1*

lemma *ecloseN_mono* :
assumes *u* ∈ *ecloseN*(*x*) *name1*(*x*) ∈ *ecloseN*(*y*) *name2*(*x*) ∈ *ecloseN*(*y*)
shows *u* ∈ *ecloseN*(*y*)
proof -
from ‹*u*∈_›
consider (*a*) *u*∈*eclose*({*name1*(*x*)}) | (*b*) *u* ∈ *eclose*({*name2*(*x*)})
unfolding *ecloseN_def eclose_n_def* **by** *auto*
then
show *?thesis*
proof *cases*
case *a*
with ‹*name1*(*x*) ∈ _›
show *?thesis*
unfolding *ecloseN_def eclose_n_def*
using *eclose_singE*[*OF a*] *mem_eclose_trans*[*of u name1*(*x*)] **by** *auto*
next
case *b*
with ‹*name2*(*x*) ∈ _›
show *?thesis*
unfolding *ecloseN_def eclose_n_def*
using *eclose_singE*[*OF b*] *mem_eclose_trans*[*of u name2*(*x*)] **by** *auto*
qed
qed

definition

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

definition
ftype_fm :: $[i,i] \Rightarrow i$ **where**
ftype_fm $\equiv$ *fst_fm*

lemma *is_ftype_iff_sats* [*iff_sats*]:
assumes
$nth(a,env) = x$ $nth(b,env) = y$ $a \in nat$ $b \in nat$ $env \in list(A)$
shows
is_ftype$(\#\#A,x,y) \longleftrightarrow sats(A,ftype\_fm(a,b),\ env)$
unfolding *ftype_fm_def is_ftype_def*
using *assms sats_fst_fm*
by *simp*

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* [*simp*]:
$[\![\ x \in nat;\ y \in nat; env \in list(A)\ ]\!] \Longrightarrow$
$(A,\ env \models name1\_fm(x,y)) \longleftrightarrow is\_name1(\#\#A,\ nth(x,env),\ nth(y,env))$
unfolding *name1_fm_def is_name1_def*
using *sats_fst_fm sats_snd_fm sats_hcomp_fm*[*of A is_fst*$(\#\#A)$ *_ fst_fm is_snd*$(\#\#A)$]
by *simp*

lemma *is_name1_iff_sats* [*iff_sats*]:
assumes
$nth(a,env) = x$ $nth(b,env) = y$ $a \in nat$ $b \in nat$ $env \in list(A)$
shows
is_name1$(\#\#A,x,y) \longleftrightarrow A\ ,\ env \models name1\_fm(a,b)$
using *assms sats_name1_fm*
by *simp*

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* [*simp*]:

⟦ $x \in nat$; $y \in nat$;$env \in list(A)$ ⟧ ⟹
(A, env ⊨ $snd\_snd\_fm(x,y)$) ⟷ $is\_snd\_snd(\#\#A, nth(x,env), nth(y,env))$
unfolding *snd_snd_fm_def is_snd_snd_def*
using *sats_snd_fm sats_hcomp_fm*[*of A is_snd*(*##A*) *_ snd_fm is_snd*(*##A*)]
by *simp*

definition
$is\_name2 :: (i⇒o)⇒i⇒i⇒o$ **where**
$is\_name2(M,x,t3) \equiv is\_hcomp(M,is\_fst(M),is\_snd\_snd(M),x,t3)$

definition
$name2\_fm :: [i,i] ⇒ i$ **where**
$name2\_fm(x,t3) \equiv hcomp\_fm(fst\_fm,snd\_snd\_fm,x,t3)$

lemma *sats_name2_fm* :
⟦ $x \in nat$; $y \in nat$;$env \in list(A)$ ⟧
⟹ (A , env ⊨ $name2\_fm(x,y)$) ⟷ $is\_name2(\#\#A, nth(x,env), nth(y,env))$
unfolding *name2_fm_def is_name2_def*
using *sats_fst_fm sats_snd2_fm sats_hcomp_fm*[*of A is_fst*(*##A*) *_ fst_fm is_snd_snd*(*##A*)]
by *simp*

lemma *is_name2_iff_sats* [*iff_sats*]:
assumes
$nth(a,env) = x$ $nth(b,env) = y$ $a∈nat$ $b∈nat$ $env \in list(A)$
shows
$is\_name2(\#\#A,x,y)$ ⟷ A, env ⊨ $name2\_fm(a,b)$
using *assms sats_name2_fm*
by *simp*

definition
$is\_cond\_of :: (i⇒o)⇒i⇒i⇒o$ **where**
$is\_cond\_of(M,x,t4) \equiv is\_hcomp(M,is\_snd(M),is\_snd\_snd(M),x,t4)$

definition
$cond\_of\_fm :: [i,i] ⇒ i$ **where**
$cond\_of\_fm(x,t4) \equiv hcomp\_fm(snd\_fm,snd\_snd\_fm,x,t4)$

lemma *sats_cond_of_fm* :
⟦ $x \in nat$; $y \in nat$;$env \in list(A)$ ⟧ ⟹
(A, env ⊨ $cond\_of\_fm(x,y)$) ⟷ $is\_cond\_of(\#\#A, nth(x,env), nth(y,env))$
unfolding *cond_of_fm_def is_cond_of_def*
using *sats_snd_fm sats_snd2_fm sats_hcomp_fm*[*of A is_snd*(*##A*) *_ snd_fm is_snd_snd*(*##A*)]
by *simp*

lemma *is_cond_of_iff_sats* [*iff_sats*]:
assumes
$nth(a,env) = x$ $nth(b,env) = y$ $a∈nat$ $b∈nat$ $env \in list(A)$

shows
is_cond_of(##*A*,*x*,*y*) ⟷ *A*, *env* ⊨ *cond_of_fm*(*a*,*b*)
using *assms sats_cond_of_fm*
by *simp*

lemma *components_type*[*TC*] :
assumes *a*∈*nat b*∈*nat*
shows
ftype_fm(*a*,*b*)∈*formula*
name1_fm(*a*,*b*)∈*formula*
name2_fm(*a*,*b*)∈*formula*
cond_of_fm(*a*,*b*)∈*formula*
using *assms*
unfolding *ftype_fm_def fst_fm_def snd_fm_def snd_snd_fm_def name1_fm_def name2_fm_def*
cond_of_fm_def hcomp_fm_def
by *simp_all*

lemmas *components_iff_sats* = *is_ftype_iff_sats is_name1_iff_sats is_name2_iff_sats is_cond_of_iff_sats*

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

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

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

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

definition
is_ecloseN :: [*i*⇒*o*,*i*,*i*] ⇒ *o* **where**
is_ecloseN(*N*,*t*,*en*) ≡ ∃ *en1*[*N*].∃ *en2*[*N*].
is_eclose_n(*N*,*is_name1*,*en1*,*t*) ∧ *is_eclose_n*(*N*,*is_name2*,*en2*,*t*)∧
union(*N*,*en1*,*en2*,*en*)

definition
ecloseN_fm :: [*i*,*i*] ⇒ *i* **where**
ecloseN_fm(*en*,*t*) ≡ *Exists*(*Exists*(*And*(*eclose_n1_fm*(*1*,*t*+ω*2*),
And(*eclose_n2_fm*(*0*,*t*+ω*2*),*union_fm*(*1*,*0*,*en*+ω*2*)))))

lemma *ecloseN_fm_type* [*TC*] :
 ⟦ *en* ∈ *nat* ; *t* ∈ *nat* ⟧ ⟹ *ecloseN_fm*(*en*,*t*) ∈ *formula*
 unfolding *ecloseN_fm_def eclose_n1_fm_def eclose_n2_fm_def* **by** *simp*

lemma *sats_ecloseN_fm* [*simp*]:
 ⟦ *en* ∈ *nat*; *t* ∈ *nat* ; *env* ∈ *list*(*A*) ⟧
 ⟹ (*A*, *env* ⊨ *ecloseN_fm*(*en*,*t*)) ⟷ *is_ecloseN*(##*A*,*nth*(*t*,*env*),*nth*(*en*,*env*))
 unfolding *ecloseN_fm_def is_ecloseN_def eclose_n1_fm_def eclose_n2_fm_def is_eclose_n_def*
 using *nth_0 nth_ConsI sats_name1_fm sats_name2_fm singleton_iff_sats*[*symmetric*]
 by *auto*

lemma *is_ecloseN_iff_sats* [*iff_sats*]:
 ⟦ *nth*(*en*, *env*) = *ena*; *nth*(*t*, *env*) = *ta*; *en* ∈ *nat*; *t* ∈ *nat* ; *env* ∈ *list*(*A*) ⟧
 ⟹ *is_ecloseN*(##*A*,*ta*,*ena*) ⟷ *A*, *env* ⊨ *ecloseN_fm*(*en*,*t*)
 by *simp*

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

lemma *frecR_ftypeD* :
 assumes *frecR*(*x*,*y*)
 shows (*ftype*(*x*) = *0* ∧ *ftype*(*y*) = *1*) ∨ (*ftype*(*x*) = *1* ∧ *ftype*(*y*) = *0*)
 using *assms* **unfolding** *frecR_def* **by** *auto*

lemma *frecRI1*: *s* ∈ *domain*(*n1*) ∨ *s* ∈ *domain*(*n2*) ⟹ *frecR*(⟨*1*, *s*, *n1*, *q*⟩, ⟨*0*, *n1*, *n2*, *q'*⟩)
 unfolding *frecR_def* **by** (*simp add*:*components_simp*)

lemma *frecRI1'*: *s* ∈ *domain*(*n1*) ∪ *domain*(*n2*) ⟹ *frecR*(⟨*1*, *s*, *n1*, *q*⟩, ⟨*0*, *n1*, *n2*, *q'*⟩)
 unfolding *frecR_def* **by** (*simp add*:*components_simp*)

lemma *frecRI2*: *s* ∈ *domain*(*n1*) ∨ *s* ∈ *domain*(*n2*) ⟹ *frecR*(⟨*1*, *s*, *n2*, *q*⟩, ⟨*0*, *n1*, *n2*, *q'*⟩)
 unfolding *frecR_def* **by** (*simp add*:*components_simp*)

lemma *frecRI2'*: *s* ∈ *domain*(*n1*) ∪ *domain*(*n2*) ⟹ *frecR*(⟨*1*, *s*, *n2*, *q*⟩, ⟨*0*, *n1*, *n2*, *q'*⟩)
 unfolding *frecR_def* **by** (*simp add*:*components_simp*)

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

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

lemma *frecR_D1* :
$frecR(x,y) \Longrightarrow ftype(y) = 0 \Longrightarrow ftype(x) = 1 \wedge$
$(name1(x) \in domain(name1(y)) \cup domain(name2(y)) \wedge (name2(x) = name1(y)$
$\vee\ name2(x) = name2(y)))$
unfolding *frecR_def*
by *auto*

lemma *frecR_D2* :
$frecR(x,y) \Longrightarrow ftype(y) = 1 \Longrightarrow ftype(x) = 0 \wedge$
$ftype(x) = 0 \wedge ftype(y) = 1 \wedge name1(x) = name1(y) \wedge name2(x) \in$
$domain(name2(y))$
unfolding *frecR_def*
by *auto*

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

reldb_add *ftype is_ftype*
reldb_add *name1 is_name1*
reldb_add *name2 is_name2*

relativize *frecR is_frecR*

schematic_goal *sats_frecR_fm_auto*:
assumes
$i{\in}nat\ j{\in}nat\ env{\in}list(A)$
shows
$is\_frecR(\#\#A,nth(i,env),nth(j,env)) \longleftrightarrow A, env \models ?fr\_fm(i,j)$
unfolding *is_frecR_def*
by (*insert assms* ; (*rule sep_rules' cartprod_iff_sats components_iff_sats*
| *simp del*:*sats_cartprod_fm*)+)

synthesize *frecR* **from_schematic** *sats_frecR_fm_auto*

Third item of Kunen's observations (p. 257) about the trcl relation.

lemma *eq_ftypep_not_frecrR*:
assumes $ftype(x) = ftype(y)$
shows $\neg\ frecR(x,y)$
using *assms frecR_ftypeD* **by** *force*

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

lemma *rank_names_types* [*TC*]:
shows *Ord*(*rank_names*(x))
unfolding *rank_names_def max_def* **using** *Ord_rank Ord_Un* **by** *auto*

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

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

lemma *type_form_tc* [*TC*]:
shows *type_form*(x) $\in$ *3*
unfolding *type_form_def mtype_form_def* **by** *auto*

lemma *frecR_le_rnk_names* :
assumes *frecR*(x,y)
shows *rank_names*(x)$\leq$*rank_names*(y)
proof -
obtain a b c d **where**
H: a $=$ *name1*(x) b $=$ *name2*(x)
c $=$ *name1*(y) d $=$ *name2*(y)
($a \in$ *domain*(c)$\cup$*domain*(d) $\wedge$ (b=c $\vee$ b $=$ d)) $\vee$ (a $=$ c $\wedge$ $b \in$ *domain*(d))
using *assms*
unfolding *frecR_def*
by *force*
then
consider
(m) $a \in$ *domain*(c) $\wedge$ (b $=$ c $\vee$ b $=$ d)
| (n) $a \in$ *domain*(d) $\wedge$ (b $=$ c $\vee$ b $=$ d)
| (o) $b \in$ *domain*(d) $\wedge$ a $=$ c
by *auto*
then
show *?thesis*
proof(*cases*)
case m
then
have *rank*(a) $<$ *rank*(c)
using *eclose_rank_lt in_dom_in_eclose*
by *simp*
with ‹*rank*(a) $<$ *rank*(c)› H m
show *?thesis*
unfolding *rank_names_def*

```
    using Ord_rank max_cong max_cong2 leI
    by auto
  next
    case n
    then
    have rank(a) < rank(d)
      using eclose_rank_lt in_dom_in_eclose
      by simp
    with ‹rank(a) < rank(d)› H n
    show ?thesis
      unfolding rank_names_def
       using Ord_rank max_cong2 max_cong max_commutes[of rank(c) rank(d)]
leI
      by auto
  next
    case o
    then
    have rank(b) < rank(d) (is ?b < ?d) rank(a) = rank(c) (is ?a = _)
      using eclose_rank_lt in_dom_in_eclose
      by simp_all
    with H
    show ?thesis
      unfolding rank_names_def
      using Ord_rank max_commutes max_cong2[OF leI[OF ‹?b < ?d›], of ?a]
      by simp
  qed
qed

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

lemma Γ_type [TC]:
  shows Ord(Γ(x))
  unfolding Γ_def by simp

lemma Γ_mono :
  assumes frecR(x,y)
  shows Γ(x) < Γ(y)
proof -
  have F: type_form(x) < 3 type_form(y) < 3
    using ltI
    by simp_all
  from assms
  have A: rank_names(x) ≤ rank_names(y) (is ?x ≤ ?y)
    using frecR_le_rnk_names
    by simp
  then
  have Ord(?y)
```

```
    unfolding rank_names_def
    using Ord_rank max_def
    by simp
  note leE[OF ‹?x≤?y›]
  then
  show ?thesis
  proof(cases)
    case 1
    then
    show ?thesis
      unfolding Γ_def
      using oadd_lt_mono2 ‹?x < ?y› F
      by auto
  next
    case 2
    consider (a) ftype(x) = 0 ∧ ftype(y) = 1 | (b) ftype(x) = 1 ∧ ftype(y) = 0
      using frecR_ftypeD[OF ‹frecR(x,y)›]
      by auto
    then show ?thesis
    proof(cases)
      case b
      moreover from this
      have type_form(y) = 1
        using type_form_def by simp
      moreover from calculation
      have name2(x) = name1(y) ∨ name2(x) = name2(y)   (is ?τ = ?σ' ∨ ?τ =
?τ')
        name1(x) ∈ domain(name1(y)) ∪ domain(name2(y)) (is ?σ ∈ domain(?σ')
∪ domain(?τ'))
        using assms unfolding type_form_def frecR_def by auto
      moreover from calculation
      have E: rank(?τ) = rank(?σ') ∨ rank(?τ) = rank(?τ') by auto
      from calculation
      consider (c) rank(?σ) < rank(?σ') |  (d) rank(?σ) < rank(?τ')
        using eclose_rank_lt in_dom_in_eclose by force
      then
      have rank(?σ) < rank(?τ)
      proof (cases)
        case c
        with ‹rank_names(x) = rank_names(y) ›
        show ?thesis
          unfolding rank_names_def mtype_form_def type_form_def
          using max_D2[OF E c] E assms Ord_rank
          by simp
      next
        case d
        with ‹rank_names(x) = rank_names(y) ›
        show ?thesis
          unfolding rank_names_def mtype_form_def type_form_def
```

```
        using max_D2[OF _ d] max_commutes E assms Ord_rank disj_commute
        by simp
    qed
    with b
   have type_form(x) = 0 unfolding type_form_def mtype_form_def by simp
   with ‹rank_names(x) = rank_names(y) › ‹type_form(y) = 1› ‹type_form(x)
= 0›
    show ?thesis
      unfolding Γ_def by auto
  next
    case a
    then
    have name1(x) = name1(y) (is ?σ = ?σ′)
      name2(x) ∈ domain(name2(y)) (is ?τ ∈ domain(?τ′))
      type_form(x) = 1
      using assms
      unfolding type_form_def frecR_def
      by auto
    then
    have rank(?σ) = rank(?σ′) rank(?τ) < rank(?τ′)
      using  eclose_rank_lt in_dom_in_eclose
      by simp_all
    with ‹rank_names(x) = rank_names(y) ›
    have rank(?τ′) ≤ rank(?σ′)
      using Ord_rank max_D1
      unfolding rank_names_def
      by simp
    with a
    have type_form(y) = 2
      unfolding type_form_def mtype_form_def
      using not_lt_iff_le assms
      by simp
   with ‹rank_names(x) = rank_names(y) › ‹type_form(y) = 2› ‹type_form(x)
= 1›
    show ?thesis
      unfolding Γ_def by auto
  qed
 qed
qed

definition
 frecrel :: i ⇒ i where
 frecrel(A) ≡ Rrel(frecR,A)

lemma frecrelI :
 assumes x ∈ A y∈A frecR(x,y)
 shows ⟨x,y⟩∈frecrel(A)
 using assms unfolding frecrel_def Rrel_def by auto
```

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

lemma *wf_frecrel* :
shows $wf(frecrel(A))$
proof -
have $frecrel(A) \subseteq measure(A,\Gamma)$
unfolding *frecrel_def Rrel_def measure_def*
using Γ*_mono*
by *force*
then
show *?thesis*
using *wf_subset wf_measure* **by** *auto*
qed

lemma *core_induction_aux*:
fixes *A1 A2* :: *i*
assumes
Transset(*A1*)
$\bigwedge\tau\ \vartheta\ p.\ \ p \in A2 \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![\ q\in A2\ ;\ \sigma\in domain(\vartheta)]\!] \Longrightarrow Q(0,\tau,\sigma,q)]\!] \Longrightarrow Q(1,\tau,\vartheta,p)$
$\bigwedge\tau\ \vartheta\ p.\ \ p \in A2 \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![\ q\in A2\ ;\ \sigma\in domain(\tau)\cup domain(\vartheta)]\!] \Longrightarrow Q(1,\sigma,\tau,q) \wedge Q(1,\sigma,\vartheta,q)]\!] \Longrightarrow Q(0,\tau,\vartheta,p)$
shows $a\in 2\times A1\times A1\times A2 \Longrightarrow Q(ftype(a),name1(a),name2(a),cond\_of(a))$
proof (*induct a rule*:*wf_induct*[*OF wf_frecrel*[*of* $2\times A1\times A1\times A2$]])
case (*1 x*)
let $?\tau = name1(x)$
let $?\vartheta = name2(x)$
let $?D = 2\times A1\times A1\times A2$
assume $x \in ?D$
then
have $cond\_of(x)\in A2$
by (*auto simp add*:*components_simp*)
from ‹$x\in ?D$›
consider (*eq*) $ftype(x)=0$ | (*mem*) $ftype(x)=1$
by (*auto simp add*:*components_simp*)
then
show *?case*
proof *cases*
case *eq*
then

```
    have Q(1, σ, ?τ, q) ∧ Q(1, σ, ?ϑ, q) if σ ∈ domain(?τ) ∪ domain(?ϑ) and
q∈A2 for q σ
    proof -
      from 1
      have ?τ∈A1 ?ϑ∈A1 ?τ∈eclose(A1) ?ϑ∈eclose(A1)
        using  arg_into_eclose
        by (auto simp add:components_simp)
      moreover from ‹Transset(A1)› that(1)
      have σ∈eclose(?τ) ∪ eclose(?ϑ)
        using in_dom_in_eclose
        by auto
      then
      have σ∈A1
       using mem_eclose_subset[OF ‹?τ∈A1›] mem_eclose_subset[OF ‹?ϑ∈A1›]
         Transset_eclose_eq_arg[OF ‹Transset(A1)›]
        by auto
      with ‹q∈A2› ‹?ϑ ∈ A1› ‹cond_of(x)∈A2› ‹?τ∈A1›
      have frecR(⟨1, σ, ?τ, q⟩, x) (is frecR(?T,_))
           frecR(⟨1, σ, ?ϑ, q⟩, x) (is frecR(?U,_))
        using frecRI1'[OF that(1)] frecR_DI  ‹ftype(x) = 0›
          frecRI2'[OF that(1)]
        by (auto simp add:components_simp)
      with ‹x∈?D› ‹σ∈A1› ‹q∈A2›
      have ⟨?T,x⟩∈ frecrel(?D) ⟨?U,x⟩∈ frecrel(?D)
        using frecrelI[of ?T ?D x]  frecrelI[of ?U ?D x]
        by (auto simp add:components_simp)
      with ‹q∈A2› ‹σ∈A1› ‹?τ∈A1› ‹?ϑ∈A1›
      have Q(1, σ, ?τ, q)
        using 1
        by (force simp add:components_simp)
      moreover from ‹q∈A2› ‹σ∈A1› ‹?τ∈A1› ‹?ϑ∈A1› ‹⟨?U,x⟩∈ frecrel(?D)›
      have Q(1, σ, ?ϑ, q)
        using 1 by (force simp add:components_simp)
      ultimately
      show ?thesis
        by simp
    qed
    with assms(3) ‹ftype(x) = 0› ‹cond_of(x)∈A2›
    show ?thesis
      by auto
  next
    case mem
    have Q(0, ?τ,  σ, q) if σ ∈ domain(?ϑ) and q∈A2 for q σ
    proof -
      from 1 assms
      have ?τ∈A1 ?ϑ∈A1 cond_of(x)∈A2 ?τ∈eclose(A1) ?ϑ∈eclose(A1)
        using arg_into_eclose
        by (auto simp add:components_simp)
      with  ‹Transset(A1)› that(1)
```

```
    have σ∈ eclose(?ϑ)
      using in_dom_in_eclose
      by auto
    then
    have σ∈A1
    using mem_eclose_subset[OF ‹?ϑ∈A1›] Transset_eclose_eq_arg[OF ‹Trans-
set(A1)›]
      by auto
    with ‹q∈A2› ‹?ϑ ∈ A1› ‹cond_of(x)∈A2› ‹?τ∈A1› ‹ftype(x) = 1›
    have frecR(⟨0, ?τ, σ, q⟩, x) (is frecR(?T,_))
      using frecRI3'[OF that(1)] frecR_DI
      by (auto simp add:components_simp)
    with ‹x∈?D› ‹σ∈A1› ‹q∈A2› ‹?τ∈A1›
    have ⟨?T,x⟩∈ frecrel(?D) ?T∈?D
      using frecrelI[of ?T ?D x]
      by (auto simp add:components_simp)
    with ‹q∈A2› ‹σ∈A1› ‹?τ∈A1› ‹?ϑ∈A1› 1
    show ?thesis
      by (force simp add:components_simp)
  qed
  with assms(2) ‹ftype(x) = 1› ‹cond_of(x)∈A2›
  show ?thesis
    by auto
 qed
qed

lemma def_frecrel : frecrel(A) = {z∈A×A. ∃x y. z = ⟨x, y⟩ ∧ frecR(x,y)}
 unfolding frecrel_def Rrel_def ..

lemma frecrel_fst_snd:
 frecrel(A) = {z ∈ A×A .
          ftype(fst(z)) = 1 ∧
           ftype(snd(z)) = 0 ∧ name1(fst(z)) ∈ domain(name1(snd(z))) ∪ do-
main(name2(snd(z))) ∧
          (name2(fst(z)) = name1(snd(z)) ∨ name2(fst(z)) = name2(snd(z)))
        ∨ (ftype(fst(z)) = 0 ∧
        ftype(snd(z)) = 1 ∧  name1(fst(z)) = name1(snd(z)) ∧ name2(fst(z)) ∈
domain(name2(snd(z))))}
 unfolding def_frecrel frecR_def
 by (intro equalityI subsetI CollectI; elim CollectE; auto)

end
theory FrecR_Arities
 imports
   FrecR
begin

context
 notes FOL_arities[simp]
```

begin

arity_theorem intermediate for *fst_fm*
lemma *arity_fst_fm* [*arity*] :
 ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*fst_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
 using *arity_fst_fm'*
 by *auto*

arity_theorem intermediate for *snd_fm*
lemma *arity_snd_fm* [*arity*] :
 ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*snd_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
 using *arity_snd_fm'*
 by *auto*

lemma *arity_snd_snd_fm* [*arity*] :
 ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*snd_snd_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
 unfolding *snd_snd_fm_def hcomp_fm_def*
 using *arity_snd_fm arity_empty_fm union_abs2 pred_Un_distrib*
 by *auto*

lemma *arity_ftype_fm* [*arity*] :
 ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*ftype_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
 unfolding *ftype_fm_def*
 using *arity_fst_fm*
 by *auto*

lemma *arity_name1_fm* [*arity*] :
 ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*name1_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
 unfolding *name1_fm_def hcomp_fm_def*
 using *arity_fst_fm arity_snd_fm union_abs2 pred_Un_distrib*
 by *auto*

lemma *arity_name2_fm* [*arity*] :
 ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*name2_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
 unfolding *name2_fm_def hcomp_fm_def*
 using *arity_fst_fm arity_snd_snd_fm union_abs2 pred_Un_distrib*
 by *auto*

lemma *arity_cond_of_fm* [*arity*] :
 ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*cond_of_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
 unfolding *cond_of_fm_def hcomp_fm_def*
 using *arity_snd_fm arity_snd_snd_fm union_abs2 pred_Un_distrib*
 by *auto*

lemma *arity_eclose_n1_fm* [*arity*] :
 ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*eclose_n1_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
 unfolding *eclose_n1_fm_def*
 using *arity_is_eclose_fm arity_singleton_fm arity_name1_fm union_abs2 pred_Un_distrib*
 by *auto*

lemma *arity_eclose_n2_fm* [*arity*] :
  ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*eclose_n2_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
  unfolding *eclose_n2_fm_def*
 using *arity_is_eclose_fm arity_singleton_fm arity_name2_fm union_abs2 pred_Un_distrib*
  by *auto*

lemma *arity_ecloseN_fm* [*arity*] :
  ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*ecloseN_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
  unfolding *ecloseN_fm_def*
   using *arity_eclose_n1_fm arity_eclose_n2_fm arity_union_fm union_abs2*
pred_Un_distrib
  by *auto*

lemma *arity_frecR_fm* [*arity*]:
  ⟦*a*∈*nat*;*b*∈*nat*⟧ ⟹ *arity*(*frecR_fm*(*a*,*b*)) = *succ*(*a*) ∪ *succ*(*b*)
  unfolding *frecR_fm_def*
  using *arity_ftype_fm arity_name1_fm arity_name2_fm arity_domain_fm*
     arity_empty_fm arity_union_fm pred_Un_distrib arity_succ_fm
  by *auto*

end — *FOL_arities*

end

5 Concepts involved in instances of Replacement

theory *Fm_Definitions*
  imports
    Transitive_Models.Renaming_Auto
    Transitive_Models.Aleph_Relative
    FrecR_Arities
begin

no_notation *Aleph* (‹ℵ_› [*90*] *90*)

In this theory we put every concept that should be synthesized in a formula to have an instance of replacement.

The automatic synthesis of a concept /foo/ requires that every concept used to define /foo/ is already synthesized. We try to use our meta-programs to synthesize concepts: given the absolute concept /foo/ we relativize in relational form obtaining /is_foo/ and the we synthesize the formula /is_foo_fm/. The meta-program that synthesizes formulas also produce satisfactions lemmas.

Having one file to collect every formula needed for replacements breaks the reading flow: we need to introduce the concept in this theory in order to use

the meta-programs; moreover there are some concepts for which we prove here the satisfaction lemmas manually, while for others we prove them on its theory.

declare *arity_subset_fm* [*simp del*] *arity_ordinal_fm*[*simp del*, *arity*] *arity_transset_fm*[*simp del*]
FOL_arities[*simp del*]

synthesize *setdiff* **from_definition** *setdiff* **assuming** *nonempty*
arity_theorem for *setdiff_fm*

synthesize *is_converse* **from_definition assuming** *nonempty*
arity_theorem for *is_converse_fm*

relationalize *first_rel is_first* **external**
synthesize *first_fm* **from_definition** *is_first* **assuming** *nonempty*

relationalize *minimum_rel is_minimum* **external**
definition *is_minimum'* **where**
is_minimum'(*M*,*R*,*X*,*u*) ≡ (*M*(*u*) ∧ *u* ∈ *X* ∧ (∀ *v*[*M*]. ∃ *a*[*M*]. (*v* ∈ *X* ⟶ *v* ≠ *u* ⟶ *a* ∈ *R*) ∧ *pair*(*M*, *u*, *v*, *a*))) ∧
(∃ *x*[*M*].
(*M*(*x*) ∧ *x* ∈ *X* ∧ (∀ *v*[*M*]. ∃ *a*[*M*]. (*v* ∈ *X* ⟶ *v* ≠ *x* ⟶ *a* ∈ *R*) ∧ *pair*(*M*, *x*, *v*, *a*))) ∧
(∀ *y*[*M*]. *M*(*y*) ∧ *y* ∈ *X* ∧ (∀ *v*[*M*]. ∃ *a*[*M*]. (*v* ∈ *X* ⟶ *v* ≠ *y* ⟶ *a* ∈ *R*) ∧ *pair*(*M*, *y*, *v*, *a*)) ⟶ *y* = *x*)) ∨
¬ (∃ *x*[*M*]. (*M*(*x*) ∧ *x* ∈ *X* ∧ (∀ *v*[*M*]. ∃ *a*[*M*]. (*v* ∈ *X* ⟶ *v* ≠ *x* ⟶ *a* ∈ *R*) ∧ *pair*(*M*, *x*, *v*, *a*))) ∧
(∀ *y*[*M*]. *M*(*y*) ∧ *y* ∈ *X* ∧ (∀ *v*[*M*]. ∃ *a*[*M*]. (*v* ∈ *X* ⟶ *v* ≠ *y* ⟶ *a* ∈ *R*) ∧ *pair*(*M*, *y*, *v*, *a*)) ⟶ *y* = *x*)) ∧
empty(*M*, *u*)

synthesize *minimum* **from_definition** *is_minimum'* **assuming** *nonempty*
arity_theorem for *minimum_fm*

lemma *is_lambda_iff_sats*[*iff_sats*]:
assumes *is_F_iff_sats*:
!!*a0 a1 a2*.
[|*a0*∈*Aa*; *a1*∈*Aa*; *a2*∈*Aa*|]
==> *is_F*(*a1*, *a0*) ⟷ *sats*(*Aa*, *is_F_fm*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*env*))))
shows
nth(*A*, *env*) = *Ab* ⟹
nth(*r*, *env*) = *ra* ⟹
A ∈ *nat* ⟹
r ∈ *nat* ⟹
env ∈ *list*(*Aa*) ⟹
is_lambda(##*Aa*, *Ab*, *is_F*, *ra*) ⟷ *Aa*, *env* ⊨ *lambda_fm*(*is_F_fm*,*A*, *r*)
using *sats_lambda_fm*[*OF assms*, *of A r*] **by** *simp*

— same as *sats_is_wfrec_fm*, but changing length assumptions to *0* being in the

model
lemma *sats_is_wfrec_fm'*:
assumes *MH_iff_sats*:
!!*a0 a1 a2 a3 a4*.
[|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*; *a3*∈*A*; *a4*∈*A*|]
==> *MH*(*a2*, *a1*, *a0*) ⟷ *sats*(*A*, *p*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*Cons*(*a3*,*Cons*(*a4*,*env*))))))
shows
[|*x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*); *0* ∈ *A*|]
==> *sats*(*A*, *is_wfrec_fm*(*p*,*x*,*y*,*z*), *env*) ⟷
is_wfrec(##*A*, *MH*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
using *MH_iff_sats* [*THEN iff_sym*] *nth_closed sats_is_recfun_fm*
by (*simp add*: *is_wfrec_fm_def is_wfrec_def*) *blast*

lemma *is_wfrec_iff_sats'*[*iff_sats*]:
assumes *MH_iff_sats*:
!!*a0 a1 a2 a3 a4*.
[|*a0*∈*Aa*; *a1*∈*Aa*; *a2*∈*Aa*; *a3*∈*Aa*; *a4*∈*Aa*|]
==> *MH*(*a2*, *a1*, *a0*) ⟷ *sats*(*Aa*, *p*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*Cons*(*a3*,*Cons*(*a4*,*env*))))))
nth(*x*, *env*) = *xx* *nth*(*y*, *env*) = *yy* *nth*(*z*, *env*) = *zz*
x ∈ *nat* *y* ∈ *nat* *z* ∈ *nat* *env* ∈ *list*(*Aa*) *0* ∈ *Aa*
shows
is_wfrec(##*Aa*, *MH*, *xx*, *yy*, *zz*) ⟷ *Aa*, *env* ⊨ *is_wfrec_fm*(*p*,*x*,*y*,*z*)
using *assms*(*2-4*) *sats_is_wfrec_fm'*[*OF assms*(*1*,*5-9*)] **by** *simp*

lemma *is_wfrec_on_iff_sats*[*iff_sats*]:
assumes *MH_iff_sats*:
!!*a0 a1 a2 a3 a4*.
[|*a0*∈*Aa*; *a1*∈*Aa*; *a2*∈*Aa*; *a3*∈*Aa*; *a4*∈*Aa*|]
==> *MH*(*a2*, *a1*, *a0*) ⟷ *sats*(*Aa*, *p*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*Cons*(*a3*,*Cons*(*a4*,*env*))))))
shows
nth(*x*, *env*) = *xx* ⟹
nth(*y*, *env*) = *yy* ⟹
nth(*z*, *env*) = *zz* ⟹
x ∈ *nat* ⟹
y ∈ *nat* ⟹
z ∈ *nat* ⟹
env ∈ *list*(*Aa*) ⟹
0 ∈ *Aa* ⟹ *is_wfrec_on*(##*Aa*, *MH*, *aa*,*xx*, *yy*, *zz*) ⟷ *Aa*, *env* ⊨ *is_wfrec_fm*(*p*,*x*,*y*,*z*)
using *assms sats_is_wfrec_fm'*[*OF assms*] **unfolding** *is_wfrec_on_def* **by** *simp*

Formulas for particular replacement instances

Now we introduce some definitions used in the definition of check; which is defined by well-founded recursion using replacement in the recursive call.

definition
rcheck :: *i* ⇒ *i* **where**
rcheck(*x*) ≡ *Memrel*(*eclose*({*x*}))^+

relativize *rcheck is_rcheck*

synthesize *is_rcheck* **from_definition**
arity_theorem for *is_rcheck_fm*

— The function used for the replacement.
definition
PHcheck :: $[i⇒o,i,i,i,i] ⇒ o$ **where**
$PHcheck(M,o,f,y,p) ≡ M(p) ∧ (∃ fy[M].\ fun\_apply(M,f,y,fy) ∧ pair(M,fy,o,p))$

synthesize *PHcheck* **from_definition assuming** *nonempty*
arity_theorem for *PHcheck_fm*

— The recursive call for check. We could use the meta-program relationalize for this; but it makes some proofs more involved.
definition
is_Hcheck :: $[i⇒o,i,i,i,i] ⇒ o$ **where**
$is\_Hcheck(M,o,z,f,hc) \ ≡ is\_Replace(M,z,PHcheck(M,o,f),hc)$

synthesize *is_Hcheck* **from_definition assuming** *nonempty*

lemma *arity_is_Hcheck_fm*:
assumes $m∈nat\ n∈nat\ p∈nat\ o∈nat$
shows $arity(is\_Hcheck\_fm(m,n,p,o)) = succ(o) ∪ succ(n) ∪ succ(p) ∪ succ(m)$
unfolding *is_Hcheck_fm_def*
using *assms arity_Replace_fm*[*rule_format*,*OF PHcheck_fm_type _ _ _ arity_PHcheck_fm*]
pred_Un_distrib Un_assoc Un_nat_type
by *simp*

— The relational version of check is hand-made because our automatic tool does not handle *wfrec*.
definition
is_check :: $[i⇒o,i,i,i] ⇒ o$ **where**
$is\_check(M,o,x,z) ≡ ∃ rch[M].\ is\_rcheck(M,x,rch) ∧$
$is\_wfrec(M,is\_Hcheck(M,o),rch,x,z)$

— Finally, we internalize the formula.
definition
check_fm :: $[i,i,i] ⇒ i$ **where**
$check\_fm(o,x,z) ≡ Exists(And(is\_rcheck\_fm(1+_ω x,0),$
$is\_wfrec\_fm(is\_Hcheck\_fm(6+_ω o,2,1,0),0,1+_ω x,1+_ω z)))$

lemma *check_fm_type*[*TC*]: $x∈nat ⟹ o∈nat ⟹ z∈nat ⟹ check\_fm(x,o,z) ∈ formula$
by (*simp add*:*check_fm_def*)

lemma *sats_check_fm* :
assumes
$o∈nat\ x∈nat\ z∈nat\ env∈list(M)\ 0∈M$
shows

$(M\ ,\ env \models check\_fm(o,x,z)) \longleftrightarrow is\_check(\#\#M,nth(o,env),nth(x,env),nth(z,env))$
proof -
have *sats_is_Hcheck_fm*:
$\bigwedge a0\ a1\ a2\ a3\ a4\ a6.\ [\![\ a0{\in}M;\ a1{\in}M;\ a2{\in}M;\ a3{\in}M;\ a4{\in}M;a6\ {\in}M]\!] \Longrightarrow$
$is\_Hcheck(\#\#M,a6,a2,\ a1,\ a0) \longleftrightarrow$
$(M\ ,\ [a0,a1,a2,a3,a4,r,a6]@env \models is\_Hcheck\_fm(6,2,1,0))$ **if** $r{\in}M$ **for** r
using *that assms*
by *simp*
then
have $(M\ ,\ [r]@env \models is\_wfrec\_fm(is\_Hcheck\_fm(6{+_\omega}o,2,1,0),0,1{+_\omega}x,1{+_\omega}z))$
$\longleftrightarrow is\_wfrec(\#\#M,is\_Hcheck(\#\#M,nth(o,env)),r,nth(x,env),nth(z,env))$
if $r{\in}M$ **for** r
using *that assms is_wfrec_iff_sats'*[*symmetric*]
by *simp*
then
show *?thesis*
unfolding *is_check_def check_fm_def*
using *assms is_rcheck_iff_sats*[*symmetric*]
by *simp*
qed

lemma *iff_sats_check_fm*[*iff_sats*] :
assumes
$nth(o,\ env) = oa\ nth(x,\ env) = xa\ nth(z,\ env) = za\ o \in nat\ x \in nat\ z \in nat$
$env \in list(A)\ 0 \in A$
shows $is\_check(\#\#A,\ oa,xa,\ za) \longleftrightarrow A,\ env \models check\_fm(o,x,z)$
using *assms sats_check_fm*[*symmetric*]
by *auto*

lemma *arity_check_fm*[*arity*]:
assumes $m{\in}nat\ n{\in}nat\ o{\in}nat$
shows $arity(check\_fm(m,n,o)) = succ(o) \cup succ(n) \cup succ(m)$
unfolding *check_fm_def*
using *assms arity_is_wfrec_fm*[*rule_format*,*OF _ _ _ _ _ _ arity_is_Hcheck_fm*]
pred_Un_distrib Un_assoc arity_tran_closure_fm
by (*auto simp add*:*arity*)

notation *check_fm* ($\langle\cdot\_\ {}^{v}\_\ is\ \_\cdot\rangle$)

— The pair of elements belongs to some set. The intended set is the preorder.
definition
$is\_leq :: [i{\Rightarrow}o,i,i,i] \Rightarrow o$ **where**
$is\_leq(A,l,q,p) \equiv \exists\, qp[A].\ (pair(A,q,p,qp) \wedge qp{\in}l)$

synthesize *is_leq* **from_definition assuming** *nonempty*
arity_theorem for *is_leq_fm*

abbreviation
$fm\_leq :: [i,i,i] \Rightarrow i$ ($\langle\cdot\_\preceq\!-\_\cdot\rangle$) **where**

fm_leq(*A*,*l*,*B*) ≡ *is_leq_fm*(*l*,*A*,*B*)

5.1 Formulas used to prove some generic instances.

definition *ϱ_repl* :: *i*⇒*i* **where**
ϱ_repl(*l*) ≡ *rsum*({⟨0, 1⟩, ⟨1, 0⟩}, *id*(*l*), 2, 3, *l*)

lemma *f_type* : {⟨0, 1⟩, ⟨1, 0⟩} ∈ 2 → 3
using *Pi_iff* **unfolding** *function_def* **by** *auto*

— thm*Internalize.sum_type* clashes with thm*Renaming.sum_type*.
hide_fact *Internalize.sum_type*

lemma *ren_type* :
assumes *l*∈*nat*
shows *ϱ_repl*(*l*) : 2+$_\omega$*l* → 3+$_\omega$*l*
using *sum_type*[*of* 2 3 *l l* {⟨0, 1⟩, ⟨1, 0⟩} *id*(*l*)] *f_type assms id_type*
unfolding *ϱ_repl_def* **by** *auto*

definition *Lambda_in_M_fm* **where** [*simp*]:*Lambda_in_M_fm*(*φ*,*len*) ≡
·(·∃·*pair_fm*(1, 0, 2) ∧
ren(*φ*) ‘ (2 +$_\omega$ *len*) ‘ (3 +$_\omega$ *len*) ‘ *ϱ_repl*(*len*) ··) ∧ ·0 ∈ *len* +$_\omega$ 2··

lemma *Lambda_in_M_fm_type*[*TC*]: *φ*∈*formula* ⟹ *len*∈*nat* ⟹ *Lambda_in_M_fm*(*φ*,*len*)
∈*formula*
using *ren_tc*[*of φ* 2+$_\omega$*len* 3+$_\omega$*len ϱ_repl*(*len*)] *ren_type*
unfolding *Lambda_in_M_fm_def*
by *simp*

definition *ϱ_pair_repl* :: *i*⇒*i* **where**
ϱ_pair_repl(*l*) ≡ *rsum*({⟨0, 0⟩, ⟨1, 1⟩, ⟨2, 3⟩}, *id*(*l*), 3, 4, *l*)

definition *LambdaPair_in_M_fm* **where** *LambdaPair_in_M_fm*(*φ*,*len*) ≡
·(·∃·*pair_fm*(1, 0, 2) ∧
ren((·∃(·∃··*fst*(2) *is* 0· ∧ ··*snd*(2) *is* 1· ∧ *ren*(*φ*) ‘ (3 +$_\omega$ *len*) ‘ (4 +$_\omega$
len) ‘ *ϱ_pair_repl*(*len*) ···)·)) ‘ (2 +$_\omega$ *len*) ‘
(3 +$_\omega$ *len*) ‘
ϱ_repl(*len*) ··) ∧
·0 ∈ *len* +$_\omega$ 2··

lemma *f_type′* : {⟨0,0 ⟩, ⟨1, 1⟩, ⟨2, 3⟩} ∈ 3 → 4
using *Pi_iff* **unfolding** *function_def* **by** *auto*

lemma *ren_type′* :
assumes *l*∈*nat*
shows *ϱ_pair_repl*(*l*) : 3+$_\omega$*l* → 4+$_\omega$*l*
using *sum_type*[*of* 3 4 *l l* {⟨0, 0⟩, ⟨1, 1⟩, ⟨2, 3⟩} *id*(*l*)] *f_type′ assms id_type*
unfolding *ϱ_pair_repl_def* **by** *auto*

lemma *LambdaPair_in_M_fm_type*[*TC*]: $\varphi \in formula \Longrightarrow len \in nat \Longrightarrow$ *Lambda-Pair_in_M_fm*(φ,*len*) $\in$*formula*
using *ren_tc*[*OF _ _ _ ren_type'*,*of* φ *len*] *Lambda_in_M_fm_type*
unfolding *LambdaPair_in_M_fm_def*
by *simp*

5.2 The relation *frecrel*

definition
frecrelP :: [*i*⇒*o*,*i*] ⇒ *o* **where**
frecrelP(*M*,*xy*) ≡ (∃ *x*[*M*]. ∃ *y*[*M*]. *pair*(*M*,*x*,*y*,*xy*) ∧ *is_frecR*(*M*,*x*,*y*))

synthesize *frecrelP* **from_definition**
arity_theorem for *frecrelP_fm*

definition
is_frecrel :: [*i*⇒*o*,*i*,*i*] ⇒ *o* **where**
is_frecrel(*M*,*A*,*r*) ≡ ∃ *A2*[*M*]. *cartprod*(*M*,*A*,*A*,*A2*) ∧ *is_Collect*(*M*,*A2*, *frecrelP*(*M*),*r*)

synthesize *frecrel* **from_definition** *is_frecrel*
arity_theorem for *frecrel_fm*

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

lemma *names_belowsD*:
assumes *x* ∈ *names_below*(*P*,*z*)
obtains *f n1 n2 p* **where**
x = ⟨*f*,*n1*,*n2*,*p*⟩ *f*∈*2 n1*∈*ecloseN*(*z*) *n2*∈*ecloseN*(*z*) *p*∈*P*
using *assms* **unfolding** *names_below_def* **by** *auto*

synthesize *number2* **from_definition**

lemma *number2_iff* :
(*A*)(*c*) ⟹ *number2*(*A*,*c*) ⟷ (∃ *b*[*A*]. ∃ *a*[*A*]. *successor*(*A*, *b*, *c*) ∧ *successor*(*A*, *a*, *b*) ∧ *empty*(*A*, *a*))
unfolding *number2_def number1_def* **by** *auto*
arity_theorem for *number2_fm*

reldb_add *ecloseN is_ecloseN*
relativize *names_below is_names_below*
synthesize *is_names_below* **from_definition**
arity_theorem for *is_names_below_fm*

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

$\wedge$

$pair(M,z,t1t2p,t)$

synthesize *is_tuple* **from_definition**
arity_theorem for *is_tuple_fm*

5.3 Definition of Forces

5.3.1 Definition of *forces* for equality and membership

$p \Vdash \tau = \theta$ if for every $q \leqslant p$ both $q \Vdash \sigma \in \tau$ and $q \Vdash \sigma \in \theta$ hold for all $\sigma \in \mathsf{dom}(\tau) \cup \mathsf{dom}(\theta)$.

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

relativize *eq_case is_eq_case*
synthesize *eq_case* **from_definition** *is_eq_case*

$p \Vdash \tau \in \theta$ if for every $v \leqslant p$ there exist q, r, and σ such that $v \leqslant q$, $q \leqslant r$, $\langle \sigma, r\rangle \in \tau$, and $q \Vdash \pi = \sigma$.

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

relativize *mem_case is_mem_case*
synthesize *mem_case* **from_definition** *is_mem_case*
arity_theorem intermediate for *eq_case_fm*
lemma *arity_eq_case_fm*[*arity*]:
assumes
n1∈*nat n2*∈*nat p*∈*nat P*∈*nat leq*∈*nat f*∈*nat*
shows
$arity(eq\_case\_fm(n1,n2,p,P,leq,f)) =$
$succ(n1) \cup succ(n2) \cup succ(p) \cup succ(P) \cup succ(leq) \cup succ(f)$
using *assms arity_eq_case_fm′*
by *auto*

arity_theorem intermediate for *mem_case_fm*
lemma *arity_mem_case_fm*[*arity*] :
assumes
n1∈*nat n2*∈*nat p*∈*nat P*∈*nat leq*∈*nat f*∈*nat*
shows
$arity(mem\_case\_fm(n1,n2,p,P,leq,f)) =$
$succ(n1) \cup succ(n2) \cup succ(p) \cup succ(P) \cup succ(leq) \cup succ(f)$
using *assms arity_mem_case_fm′*
by *auto*

definition
Hfrc :: $[i,i,i,i] \Rightarrow o$ **where**
$Hfrc(P,leq,fnnc,f) \equiv \exists ft.\ \exists \tau.\ \exists \vartheta.\ \exists p.\ p{\in}P \wedge fnnc = \langle ft,\tau,\vartheta,p\rangle \wedge$
$(\ ft = 0 \wedge\ eq\_case(\tau,\vartheta,p,P,leq,f)$
$\vee\ ft = 1 \wedge mem\_case(\tau,\vartheta,p,P,leq,f))$

relativize *Hfrc is_Hfrc*
synthesize *Hfrc* **from_definition** *is_Hfrc*

definition
is_Hfrc_at :: $[i{\Rightarrow}o,i,i,i,i,i] \Rightarrow o$ **where**
$is\_Hfrc\_at(M,P,leq,fnnc,f,b) \equiv$
$(empty(M,b) \wedge \neg\ is\_Hfrc(M,P,leq,fnnc,f))$
$\vee\ (number1(M,b) \wedge is\_Hfrc(M,P,leq,fnnc,f))$

synthesize *Hfrc_at* **from_definition** *is_Hfrc_at*
arity_theorem intermediate for *Hfrc_fm*

lemma *arity_Hfrc_fm*$[arity]$:
assumes
$P{\in}nat\ leq{\in}nat\ fnnc{\in}nat\ f{\in}nat$
shows
$arity(Hfrc\_fm(P,leq,fnnc,f)) = succ(P) \cup succ(leq) \cup succ(fnnc) \cup succ(f)$
using *assms arity_Hfrc_fm′*
by *auto*

arity_theorem for *Hfrc_at_fm*

5.3.2 The well-founded relation *forcerel*

definition
forcerel :: $i \Rightarrow i \Rightarrow i$ **where**
$forcerel(P,x) \equiv frecrel(names\_below(P,x))^{+}$

definition
is_forcerel :: $[i{\Rightarrow}o,i,i,i] \Rightarrow o$ **where**
$is\_forcerel(M,P,x,z) \equiv \exists r[M].\ \exists nb[M].\ tran\_closure(M,r,z) \wedge$
$(is\_names\_below(M,P,x,nb) \wedge is\_frecrel(M,nb,r))$
synthesize *is_forcerel* **from_definition**
arity_theorem for *is_forcerel_fm*

5.4 *frc_at*, forcing for atomic formulas

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

— The relational form is defined manually because it uses *wfrec*.

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

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

lemma *frc_at_fm_type* [*TC*] :
⟦*p*∈*nat*;*l*∈*nat*;*x*∈*nat*;*z*∈*nat*⟧ ⟹ *frc_at_fm*(*p*,*l*,*x*,*z*)∈*formula*
unfolding *frc_at_fm_def* **by** *simp*

lemma *arity_frc_at_fm*[*arity*] :
assumes *p*∈*nat* *l*∈*nat* *x*∈*nat* *z*∈*nat*
shows *arity*(*frc_at_fm*(*p*,*l*,*x*,*z*)) = *succ*(*p*) ∪ *succ*(*l*) ∪ *succ*(*x*) ∪ *succ*(*z*)
proof -
let *?φ* = *Hfrc_at_fm*(6 +ω *p*, 6 +ω *l*, 2, 1, 0)
note *assms*
moreover from *this*
have *arity*(*?φ*) = (7+ω*p*) ∪ (7+ω*l*) *?φ* ∈ *formula*
using *arity_Hfrc_at_fm ord_simp_union*
by *auto*
moreover from *calculation*
have *arity*(*is_wfrec_fm*(*?φ*, 0, *succ*(*x*), *succ*(*z*))) = 2+ω*p* ∪ (2+ω*l*) ∪ (2+ω*x*)
∪ (2+ω*z*)
using *arity_is_wfrec_fm*[*OF* ‹*?φ*∈_› _ _ _ _ ‹*arity*(*?φ*) = _›] *pred_Un_distrib*
pred_succ_eq
union_abs1
by *auto*
moreover from *assms*
have *arity*(*is_forcerel_fm*(*succ*(*p*),*succ*(*x*),0)) = 2+ω*p* ∪ (2+ω*x*)
using *arity_is_forcerel_fm ord_simp_union*
by *auto*
ultimately
show *?thesis*
unfolding *frc_at_fm_def*
using *arity_is_forcerel_fm pred_Un_distrib*
by (*auto simp*:*FOL_arities*)
qed

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

```
  {
    fix r pp ll
    assume r∈A
    have is_Hfrc_at(##A,nth(p,env),nth(l,env),a2, a1, a0) ⟷
         (A, [a0,a1,a2,a3,a4,r]@env ⊨ Hfrc_at_fm(6+ω p,6+ω l,2,1,0))
      if a0∈A a1∈A a2∈A a3∈A a4∈A for a0 a1 a2 a3 a4
      using  that assms ‹r∈A›
       Hfrc_at_iff_sats[of 6+ω p 6+ω l 2 1 0 [a0,a1,a2,a3,a4,r]@env A]  by simp
    with ‹r∈A›
      have (A,[r]@env ⊨ is_wfrec_fm(Hfrc_at_fm(6+ω p, 6+ω l,2,1,0),0, i+ω 1,
j+ω 1)) ⟷
          is_wfrec(##A, is_Hfrc_at(##A, nth(p,env), nth(l,env)), r,nth(i, env),
nth(j, env))
       using assms sats_is_wfrec_fm
       by simp
  }
  moreover
  have (A, Cons(r, env) ⊨ is_forcerel_fm(succ(p), succ(i), 0)) ⟷
        is_forcerel(##A,nth(p,env),nth(i,env),r) if r∈A for r
    using assms sats_is_forcerel_fm that
    by simp
  ultimately
  show ?thesis
    unfolding is_frc_at_def frc_at_fm_def
    using assms
    by simp
qed

lemma frc_at_fm_iff_sats:
  assumes nth(i,env) = w nth(j,env) = x nth(k,env) = y nth(l,env) = z
    i ∈ nat j ∈ nat k ∈ nat l∈nat env ∈ list(A) k<length(env) l<length(env)
  shows is_frc_at(##A, w, x, y,z) ⟷ (A , env ⊨ frc_at_fm(i,j,k,l))
  using assms sats_frc_at_fm
  by simp

declare frc_at_fm_iff_sats [iff_sats]

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

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

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

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

— The following definitions are explicitly defined to avoid the expansion of concepts.
definition
is_forces_eq′ :: $[i{\Rightarrow}o,i,i,i,i,i] \Rightarrow o$ **where**
$is\_forces\_eq'(M,P,l,p,t1,t2) \equiv \exists\, o[M].\ \exists\, z[M].\ \exists\, t[M].\ number1(M,o) \wedge empty(M,z)$
$\wedge$
$is\_tuple(M,z,t1,t2,p,t) \wedge is\_frc\_at(M,P,l,t,o)$

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

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

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

synthesize *forces_eq* **from_definition** *is_forces_eq′*
synthesize *forces_mem* **from_definition** *is_forces_mem′*
synthesize *forces_neq* **from_definition** *is_forces_neq′* **assuming** *nonempty*
synthesize *forces_nmem* **from_definition** *is_forces_nmem′* **assuming** *nonempty*

context
notes *Un_assoc*[*simp*] *Un_trasposition_aux2*[*simp*]
begin
arity_theorem for *forces_eq_fm*
arity_theorem for *forces_mem_fm*
arity_theorem for *forces_neq_fm*
arity_theorem for *forces_nmem_fm*
end

5.5 Forcing for general formulas

definition
ren_forces_nand :: $i{\Rightarrow}i$ **where**
$ren\_forces\_nand(\varphi) \equiv Exists(And(Equal(0,1),iterates(\lambda p.\ incr\_bv(p)`1\ ,\ 2,\ \varphi)))$

lemma *ren_forces_nand_type*[*TC*] :
$\varphi{\in}formula \Longrightarrow ren\_forces\_nand(\varphi)\ {\in}formula$
unfolding *ren_forces_nand_def*

by *simp*

lemma *arity_ren_forces_nand* :
assumes $\varphi \in formula$
shows $arity(ren\_forces\_nand(\varphi)) \leq succ(arity(\varphi))$
proof -
consider (*lt*) $1<arity(\varphi)$ | (*ge*) $\neg\ 1 < arity(\varphi)$
by *auto*
then
show *?thesis*
proof *cases*
case *lt*
with ‹$\varphi \in \_$›
have $2 < succ(arity(\varphi))$ $2<arity(\varphi)+_\omega 2$
using *succ_ltI* **by** *auto*
with ‹$\varphi \in \_$›
have $arity(iterates(\lambda p.\ incr\_bv(p)`1,2,\varphi)) = 2+_\omega arity(\varphi)$
using *arity_incr_bv_lemma lt*
by *auto*
with ‹$\varphi \in \_$›
show *?thesis*
unfolding *ren_forces_nand_def*
using *lt pred_Un_distrib union_abs1 Un_assoc*[*symmetric*] *Un_le_compat*
by (*simp add:FOL_arities*)
next
case *ge*
with ‹$\varphi \in \_$›
have $arity(\varphi) \leq 1$ $pred(arity(\varphi)) \leq 1$
using *not_lt_iff_le le_trans*[*OF le_pred*]
by *simp_all*
with ‹$\varphi \in \_$›
have $arity(iterates(\lambda p.\ incr\_bv(p)`1,2,\varphi)) = (arity(\varphi))$
using *arity_incr_bv_lemma ge*
by *simp*
with ‹$arity(\varphi) \leq 1$› ‹$\varphi \in \_$› ‹$pred(\_) \leq 1$›
show *?thesis*
unfolding *ren_forces_nand_def*
using *pred_Un_distrib union_abs1 Un_assoc*[*symmetric*] *union_abs2*
by (*simp add:FOL_arities*)
qed
qed

lemma *sats_ren_forces_nand*:
$[q,P,leq,o,p] \ @\ env \in list(M) \Longrightarrow \varphi \in formula \Longrightarrow$
$(M,\ [q,p,P,leq,o] \ @\ env \models ren\_forces\_nand(\varphi)) \longleftrightarrow (M,\ [q,P,leq,o] \ @\ env \models \varphi)$
unfolding *ren_forces_nand_def*
using *sats_incr_bv_iff* [*of* _ _ *M* _ [*q*]]
by *simp*

```
definition
  ren_forces_forall :: i⇒i where
  ren_forces_forall(φ) ≡
      Exists(Exists(Exists(Exists(Exists(
        And(Equal(0,6),And(Equal(1,7),And(Equal(2,8),And(Equal(3,9),
        And(Equal(4,5),iterates(λp. incr_bv(p)‘5 , 5, φ))))))))))

lemma arity_ren_forces_all :
  assumes φ∈formula
  shows arity(ren_forces_forall(φ)) = 5 ∪ arity(φ)
proof -
  consider (lt) 5<arity(φ) | (ge) ¬ 5 < arity(φ)
    by auto
  then
  show ?thesis
  proof cases
    case lt
    with ‹φ∈_›
    have 5 < succ(arity(φ)) 5<arity(φ)+ω2  5<arity(φ)+ω3  5<arity(φ)+ω4
      using succ_ltI by auto
    with ‹φ∈_›
    have arity(iterates(λp. incr_bv(p)‘5,5,φ)) = 5+ωarity(φ)
      using arity_incr_bv_lemma lt
      by simp
    with ‹φ∈_›
    show ?thesis
      unfolding ren_forces_forall_def
      using pred_Un_distrib union_abs1 Un_assoc[symmetric] union_abs2
      by (simp add:FOL_arities)
  next
    case ge
    with ‹φ∈_›
    have arity(φ) ≤ 5 pred^5(arity(φ)) ≤ 5
      using not_lt_iff_le le_trans[OF le_pred]
      by simp_all
    with ‹φ∈_›
    have arity(iterates(λp. incr_bv(p)‘5,5,φ)) = arity(φ)
      using arity_incr_bv_lemma ge
      by simp
    with ‹arity(φ) ≤ 5› ‹φ∈_› ‹pred^5(_) ≤ 5›
    show ?thesis
      unfolding ren_forces_forall_def
      using  pred_Un_distrib union_abs1 Un_assoc[symmetric] union_abs2
      by (simp add:FOL_arities)
  qed
qed
```

lemma *ren_forces_forall_type*[*TC*] :
φ∈*formula* ⟹ *ren_forces_forall*(*φ*) ∈*formula*
unfolding *ren_forces_forall_def* **by** *simp*

lemma *sats_ren_forces_forall* :
[*x*,*P*,*leq*,*o*,*p*] @ *env* ∈ *list*(*M*) ⟹ *φ*∈*formula* ⟹
(*M*, [*x*,*p*,*P*,*leq*,*o*] @ *env* ⊨ *ren_forces_forall*(*φ*)) ⟷ (*M*, [*p*,*P*,*leq*,*o*,*x*] @ *env*
⊨ *φ*)
unfolding *ren_forces_forall_def*
using *sats_incr_bv_iff* [*of* _ _ *M* _ [*p*,*P*,*leq*,*o*,*x*]]
by *simp*

5.5.1 The primitive recursion

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

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

lemma *forces′_type* [*TC*]: *φ*∈*formula* ⟹ *forces′*(*φ*) ∈ *formula*
by (*induct φ set*:*formula*; *simp*)

lemma *forces_type*[*TC*] : *φ*∈*formula* ⟹ *forces*(*φ*) ∈ *formula*
unfolding *forces_def* **by** *simp*

5.6 The arity of *forces*

lemma *arity_forces_at*:
assumes *x* ∈ *nat y* ∈ *nat*
shows *arity*(*forces*(*Member*(*x*, *y*))) = (*succ*(*x*) ∪ *succ*(*y*)) +*ω* *4*
arity(*forces*(*Equal*(*x*, *y*))) = (*succ*(*x*) ∪ *succ*(*y*)) +*ω* *4*
unfolding *forces_def*
using *assms arity_forces_mem_fm arity_forces_eq_fm succ_Un_distrib ord_simp_union*
by (*auto simp*:*FOL_arities*,(*rule_tac le_anti_sym*,*simp_all*,(*rule_tac not_le_anti_sym*,*simp_all*))+)

lemma *arity_forces′*:
assumes *φ*∈*formula*
shows *arity*(*forces′*(*φ*)) ≤ *arity*(*φ*) +*ω* *4*
using *assms*
proof (*induct set*:*formula*)
case (*Member x y*)

```
  then
  show ?case
   using arity_forces_mem_fm succ_Un_distrib ord_simp_union leI not_le_iff_lt
    by simp
next
  case (Equal x y)
  then
  show ?case
   using arity_forces_eq_fm succ_Un_distrib ord_simp_union leI not_le_iff_lt
    by simp
next
  case (Nand φ ψ)
  let ?φ' = ren_forces_nand(forces'(φ))
  let ?ψ' = ren_forces_nand(forces'(ψ))
  have arity(is_leq_fm(3, 0, 1)) = 4
    using arity_is_leq_fm succ_Un_distrib ord_simp_union
    by simp
  have 3 ≤ (4+ω arity(φ)) ∪ (4+ω arity(ψ)) (is _ ≤ ?rhs)
    using ord_simp_union by simp
  from ‹φ∈_› Nand
  have pred(arity(?φ')) ≤ ?rhs  pred(arity(?ψ')) ≤ ?rhs
  proof -
    from ‹φ∈_› ‹ψ∈_›
    have A:pred(arity(?φ')) ≤ arity(forces'(φ))
      pred(arity(?ψ')) ≤ arity(forces'(ψ))
      using pred_mono[OF _  arity_ren_forces_nand] pred_succ_eq
      by simp_all
    from Nand
    have 3 ∪ arity(forces'(φ)) ≤ arity(φ) +ω 4
      3 ∪ arity(forces'(ψ)) ≤ arity(ψ) +ω 4
      using Un_le by simp_all
    with Nand
    show pred(arity(?φ')) ≤ ?rhs
      pred(arity(?ψ')) ≤ ?rhs
      using le_trans[OF A(1)] le_trans[OF A(2)] le_Un_iff
      by simp_all
  qed
  with Nand ‹_=4›
  show ?case
     using pred_Un_distrib Un_assoc[symmetric] succ_Un_distrib union_abs1
Un_leI3[OF ‹3 ≤ ?rhs›]
   by (simp add:FOL_arities)
next
  case (Forall φ)
  let ?φ' = ren_forces_forall(forces'(φ))
  show ?case
  proof (cases arity(φ) = 0)
    case True
    with Forall
```

```
    show ?thesis
    proof -
      from Forall True
      have arity(forces'(φ)) ≤ 5
        using le_trans[of _ 4 5] by auto
      with ‹φ∈_›
      have arity(?φ') ≤ 5
        using arity_ren_forces_all[OF forces'_type[OF ‹φ∈_›]] union_abs2
        by auto
      with Forall True
      show ?thesis
        using pred_mono[OF _ ‹arity(?φ') ≤ 5›]
        by simp
    qed
  next
    case False
    with Forall
    show ?thesis
    proof -
      from Forall False
      have arity(?φ') = 5 ∪ arity(forces'(φ))
        arity(forces'(φ)) ≤ 5 +ω arity(φ)
        4 ≤ 3+ωarity(φ)
        using Ord_0_lt arity_ren_forces_all
          le_trans[OF _ add_le_mono[of 4 5, OF _ le_refl]]
        by auto
      with ‹φ∈_›
      have 5 ∪ arity(forces'(φ)) ≤ 5+ωarity(φ)
        using ord_simp_union by auto
      with ‹φ∈_› ‹arity(?φ') = 5 ∪ _›
      show ?thesis
        using pred_Un_distrib succ_pred_eq[OF _ ‹arity(φ)≠0›]
          pred_mono[OF _ Forall(2)] Un_le[OF ‹4≤3+ωarity(φ)›]
        by simp
    qed
  qed
qed

lemma arity_forces :
  assumes φ∈formula
  shows arity(forces(φ)) ≤ 4+ωarity(φ)
  unfolding forces_def
  using assms arity_forces' le_trans ord_simp_union FOL_arities by auto

lemma arity_forces_le :
  assumes φ∈formula n∈nat arity(φ) ≤ n
  shows arity(forces(φ)) ≤ 4+ωn
  using assms le_trans[OF _ add_le_mono[OF le_refl[of 5] ‹arity(φ)≤_›]] ar-
ity_forces
```

by *auto*

definition *rename_split_fm* **where**
rename_split_fm(φ) ≡ ($\cdot\exists$($\cdot\exists$($\cdot\exists$($\cdot\exists$($\cdot\exists\cdot\cdot$*snd*(9) *is 0*$\cdot \wedge \cdot\cdot$*fst*(9) *is 4*$\cdot \wedge \cdot\cdot$*1=11*$\cdot$
$\wedge$
$\cdot\cdot$*2=12*$\cdot \wedge \cdot\cdot$*3=13*$\cdot \wedge \cdot\cdot$*5=7*$\cdot \wedge$
(λ*p*. *incr_bv*(*p*)‘*6*)^*8*(*forces*(φ)) ·····)·)·)·)·)·)

lemma *rename_split_fm_type*[*TC*]: φ∈*formula* ⟹ *rename_split_fm*(φ)∈*formula*
unfolding *rename_split_fm_def* **by** *simp*

schematic_goal *arity_rename_split_fm*: φ∈*formula* ⟹ *arity*(*rename_split_fm*(φ))
= *?m*
using *arity_forces*[*of* φ] *forces_type* **unfolding** *rename_split_fm_def*
by (*simp add*:*arity Un_assoc*[*symmetric*] *union_abs1*)

lemma *arity_rename_split_fm_le*:
assumes φ∈*formula*
shows *arity*(*rename_split_fm*(φ)) ≤ *8* ∪ (*6* +$_\omega$ *arity*(φ))
proof -
from *assms*
have *arity_forces_6*: ¬ *1* < *arity*(φ) ⟹ *6* ≤ *n* ⟹ *arity*(*forces*(φ)) ≤ *n* **for** *n*
using *le_trans lt_trans*[*of* _ *5 n*] *not_lt_iff_le*[*of 1 arity*(φ)]
by (*auto intro*!:*le_trans*[*OF arity_forces*])
have *pred1_arity_forces*: ¬ *1* < *arity*(φ) ⟹ *pred*^*n*(*arity*(*forces*(φ))) ≤ *8* **if**
n∈*nat* **for** *n*
using *that pred_le*[*of 7*] *le_succ*[*THEN* [*2*] *le_trans*] *arity_forces_6*
by (*induct rule*:*nat_induct*) *auto*
have *arity_forces_le_succ6*: *pred*^*n*(*arity*(*forces*(φ))) ≤ *succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*arity*(φ)))))))
if *n*∈*nat* **for** *n*
using *that assms arity_forces*[*of* φ, *THEN le_trans*,
OF _ *le_succ*, *THEN le_trans*, *OF* _ _ *le_succ*] *le_trans*[*OF pred_le*[*OF*
_ *le_succ*]]
by (*induct rule*:*nat_induct*) *auto*
note *trivial_arities* = *arity_forces_6*
arity_forces_le_succ6[*of 1*, *simplified*] *arity_forces_le_succ6*[*of 2*, *simplified*]
arity_forces_le_succ6[*of 3*, *simplified*] *arity_forces_le_succ6*[*of 4*, *simplified*]
arity_forces_le_succ6[*of 5*, *simplified*] *arity_forces_le_succ6*[*of 6*, *simplified*]
pred1_arity_forces[*of 1*, *simplified*] *pred1_arity_forces*[*of 2*, *simplified*]
pred1_arity_forces[*of 3*, *simplified*] *pred1_arity_forces*[*of 4*, *simplified*]
pred1_arity_forces[*of 5*, *simplified*] *pred1_arity_forces*[*of 6*, *simplified*]
show *?thesis*
using *assms arity_forces*[*of* φ] *arity_forces*[*of* φ, *THEN le_trans*, *OF* _
le_succ]
arity_forces[*of* φ, *THEN le_trans*, *OF* _ *le_succ*, *THEN le_trans*, *OF* _ _
le_succ]
unfolding *rename_split_fm_def*
by (*simp add*:*arity Un_assoc*[*symmetric*] *union_abs1 arity_forces*[*of* φ] *forces_type*)
((*subst arity_incr_bv_lemma*; *auto simp*: *arity ord_simp_union forces_type*

trivial_arities)+)
qed

definition *body_ground_repl_fm* **where**
body_ground_repl_fm(*φ*) ≡ (·∃(·∃·*is_Vset_fm*(*2*, *0*) ∧ ··*1* ∈ *0*· ∧ *rename_split_fm*(*φ*)
···)·)

lemma *body_ground_repl_fm_type*[*TC*]: *φ*∈*formula* ⟹ *body_ground_repl_fm*(*φ*)∈*formula*
unfolding *body_ground_repl_fm_def* **by** *simp*

lemma *arity_body_ground_repl_fm_le*:
notes *le_trans*[*trans*]
assumes *φ*∈*formula*
shows *arity*(*body_ground_repl_fm*(*φ*)) ≤ *6* ∪ (*arity*(*φ*) +$_\omega$ *4*)
proof -
from ‹*φ*∈*formula*›
have *ineq*: *n* ∪ *pred*(*pred*(*arity*(*rename_split_fm*(*φ*))))
≤ *m* ∪ *pred*(*pred*(*8* ∪ (*arity*(*φ*) +$_\omega$ *6*))) **if** *n* ≤ *m* *n*∈*nat* *m*∈*nat* **for** *n m*
using *that arity_rename_split_fm_le*[*of φ*, *THEN* [*2*] *pred_mono*, *THEN* [*2*]
pred_mono,
THEN [*2*] *Un_mono*[*THEN subset_imp_le*, *OF _ le_imp_subset*]] *le_imp_subset*
by *auto*
moreover
have *eq1*: *pred*(*pred*(*pred*(*4* ∪ *2* ∪ *pred*(*pred*(*pred*(
pred(*pred*(*pred*(*pred*(*pred*(*9* ∪ *1* ∪ *3* ∪ *2*)))))))))) = *1*
by (*auto simp*:*pred_Un_distrib*)
ultimately
have *pred*(*pred*(*pred*(*4* ∪ *2* ∪ *pred*(*pred*(*pred*(
pred(*pred*(*pred*(*pred*(*pred*(*9* ∪ *1* ∪ *3* ∪ *2*)))))))))) ∪
pred(*pred*(*arity*(*rename_split_fm*(*φ*)))) ≤
1 ∪ *pred*(*pred*(*8* ∪ (*arity*(*φ*) +$_\omega$ *6*)))
by *auto*
also from ‹*φ*∈*formula*›
have *1* ∪ *pred*(*pred*(*8* ∪ (*arity*(*φ*) +$_\omega$ *6*))) ≤ *6* ∪ (*4*+$_\omega$ *arity*(*φ*))
by (*auto simp*:*pred_Un_distrib Un_assoc*[*symmetric*] *ord_simp_union*)
finally
show *?thesis*
using ‹*φ*∈*formula*› **unfolding** *body_ground_repl_fm_def*
by (*simp add*:*arity pred_Un_distrib*, *subst arity_transrec_fm*[*of is_HVfrom_fm*(*8*,*2*,*1*,*0*)
3 1])
(*simp add*:*arity pred_Un_distrib*,*simp_all*,
auto simp add:*eq1 arity_is_HVfrom_fm*[*of 8 2 1 0*])
qed

definition *ground_repl_fm* **where**
ground_repl_fm(*φ*) ≡ *least_fm*(*body_ground_repl_fm*(*φ*), *1*)

lemma *ground_repl_fm_type*[*TC*]:
φ∈*formula* ⟹ *ground_repl_fm*(*φ*) ∈ *formula*

unfolding *ground_repl_fm_def* **by** *simp*

lemma *arity_ground_repl_fm*:
assumes φ∈*formula*
shows *arity*(*ground_repl_fm*(φ)) ≤ 5 ∪ (3 +$_\omega$ *arity*(φ))
proof -
from *assms*
have *pred*(*arity*(*body_ground_repl_fm*(φ))) ≤ 5 ∪ (3 +$_\omega$ *arity*(φ))
using *arity_body_ground_repl_fm_le pred_mono succ_Un_distrib*
by (*rule_tac pred_le*) *auto*
with *assms*
have 2 ∪ *pred*(*arity*(*body_ground_repl_fm*(φ))) ≤ 5 ∪ (3 +$_\omega$ *arity*(φ))
using *Un_le le_Un_iff* **by** *auto*
then
show *?thesis*
using *assms arity_forces arity_body_ground_repl_fm_le*
unfolding *least_fm_def ground_repl_fm_def*
apply (*auto simp add*:*arity Un_assoc*[*symmetric*])
apply (*simp add*: *pred_Un Un_assoc*, *simp add*: *Un_assoc*[*symmetric*] *union_abs1 pred_Un*)
by(*simp only*: *Un_commute*, *subst Un_commute*, *simp add*:*ord_simp_union*,*force*)
qed

synthesize *is_ordermap* **from_definition assuming** *nonempty*

synthesize *is_ordertype* **from_definition assuming** *nonempty*

synthesize *is_order_body* **from_definition assuming** *nonempty*
arity_theorem for *is_order_body_fm*

definition *omap_wfrec_body* **where**
omap_wfrec_body(*A*,*r*) ≡ (·∃·*image_fm*(2, 0, 1) ∧ *pred_set_fm*(9+$_\omega$*A*, 3,9+$_\omega$*r*, 0) ··)

lemma *type_omap_wfrec_body_fm* :*A*∈*nat* ⟹ *r*∈*nat* ⟹ *omap_wfrec_body*(*A*,*r*)∈*formula*
unfolding *omap_wfrec_body_def* **by** *simp*

lemma *arity_omap_wfrec_aux* : *A*∈*nat* ⟹ *r*∈*nat* ⟹ *arity*(*omap_wfrec_body*(*A*,*r*)) = (9+$_\omega$*A*) ∪ (9+$_\omega$*r*)
unfolding *omap_wfrec_body_def*
using *arity_image_fm arity_pred_set_fm pred_Un_distrib union_abs2*[*of 3*] *union_abs1*
by (*simp add*:*FOL_arities*, *auto simp add*:*Un_assoc*[*symmetric*] *union_abs1*)

lemma *arity_omap_wfrec*: *A*∈*nat* ⟹ *r*∈*nat* ⟹
arity(*is_wfrec_fm*(*omap_wfrec_body*(*A*,*r*),*r*+$_\omega$3, 1, 0)) = (4+$_\omega$*A*) ∪ (4+$_\omega$*r*)
using *Arities.arity_is_wfrec_fm*[*OF* _ _ _ _ _ *arity_omap_wfrec_aux*,*of A r* 3+$_\omega$*r* 1 0]
pred_Un_distrib union_abs1 union_abs2 type_omap_wfrec_body_fm

by *auto*

lemma *arity_isordermap*: $A{\in}nat \Longrightarrow r{\in}nat \Longrightarrow d{\in}nat \Longrightarrow$
$arity(is\_ordermap\_fm(A,r,d)) = succ(d) \cup (succ(A) \cup succ(r))$
unfolding *is_ordermap_fm_def*
using *arity_lambda_fm*[**where** $i{=}(4{+}_\omega A) \cup (4{+}_\omega r), OF\ \_\ \_\ \_\ \_\ \_$ *arity_omap_wfrec*,
unfolded omap_wfrec_body_def] *pred_Un_distrib union_abs1*
by *auto*

lemma *arity_is_ordertype*: $A{\in}nat \Longrightarrow r{\in}nat \Longrightarrow d{\in}nat \Longrightarrow$
$arity(is\_ordertype\_fm(A,r,d)) = succ(d) \cup (succ(A) \cup succ(r))$
unfolding *is_ordertype_fm_def*
using *arity_isordermap arity_image_fm pred_Un_distrib FOL_arities*
by *auto*

lemma *arity_is_order_body*: $arity(is\_order\_body\_fm(0,1)) = 2$
using *arity_is_order_body_fm arity_is_ordertype ord_simp_union*
by (*simp add*:*FOL_arities*)

definition *H_order_pred* **where**
$H\_order\_pred(A,r) \equiv \ \lambda x\ f\ .\ f\ ``\ Order.pred(A,\ x,\ r)$

relationalize *H_order_pred is_H_order_pred*

synthesize *is_H_order_pred* **from_definition assuming** *nonempty*

definition *order_pred_wfrec_body* **where**
$order\_pred\_wfrec\_body(M,A,r,z,x) \equiv \exists\, y[M].$
$pair(M,\ x,\ y,\ z) \wedge$
$(\exists\, f[M].$
$(\forall\, z[M].$
$z \in f \longleftrightarrow$
$(\exists\, xa[M].$
$\exists\, y[M].$
$\exists\, xaa[M].$
$\exists\, sx[M].$
$\exists\, r\_sx[M].$
$\exists\, f\_r\_sx[M].$
$pair(M,\ xa,\ y,\ z) \wedge$
$pair(M,\ xa,\ x,\ xaa) \wedge$
$upair(M,\ xa,\ xa,\ sx) \wedge$
$pre\_image(M,\ r,\ sx,\ r\_sx) \wedge$
$restriction(M,\ f,\ r\_sx,\ f\_r\_sx) \wedge$
$xaa \in r \wedge (\exists\, a[M].\ image(M,\ f\_r\_sx,\ a,\ y) \wedge$
$pred\_set(M,\ A,\ xa,\ r,\ a)))) \wedge$
$(\exists\, a[M].\ image(M,\ f,\ a,\ y) \wedge pred\_set(M,\ A,\ x,\ r,\ a)))$

synthesize *order_pred_wfrec_body* **from_definition**

arity_theorem for *order_pred_wfrec_body_fm*

definition *ordtype_replacement_fm* **where** *ordtype_replacement_fm* ≡ (·∃·*is_order_body_fm*(*1*, *0*) ∧ ·⟨*1*,*0*⟩ *is 2* ··)
definition *wfrec_ordertype_fm* **where** *wfrec_ordertype_fm* ≡ *order_pred_wfrec_body_fm*(*3*,*2*,*1*,*0*)
definition *replacement_is_aleph_fm* **where** *replacement_is_aleph_fm* ≡ ··0 *is ordinal*· ∧ ·ℵ(*0*) *is 1*··

definition
funspace_succ_rep_intf **where**
funspace_succ_rep_intf ≡ λ*p z n*. ∃*f b*. *p* = <*f*,*b*> & *z* = {*cons*(<*n*,*b*>, *f*)}

relativize functional *funspace_succ_rep_intf funspace_succ_rep_intf_rel*

— The definition obtained next uses *is_cons* instead of *upair* as in Paulson's `~~/src/ZF/Constructible/Relative.thy`.
relationalize *funspace_succ_rep_intf_rel is_funspace_succ_rep_intf*

synthesize *is_funspace_succ_rep_intf* **from_definition**

arity_theorem for *is_funspace_succ_rep_intf_fm*

definition *wfrec_Hfrc_at_fm* **where** *wfrec_Hfrc_at_fm* ≡ (·∃·*pair_fm*(*1*, *0*, *2*) ∧ *is_wfrec_fm*(*Hfrc_at_fm*(*8*, *9*, *2*, *1*, *0*), *5*, *1*, *0*) ··)
definition *list_repl1_intf_fm* **where** *list_repl1_intf_fm* ≡ (·∃·*pair_fm*(*1*, *0*, *2*) ∧ *is_wfrec_fm*(*iterates_MH_fm*(*list_functor_fm*(*13*, *1*, *0*), *10*, *2*, *1*, *0*), *3*, *1*, *0*) ··)
definition *list_repl2_intf_fm* **where** *list_repl2_intf_fm* ≡ ··0 ∈ *4*· ∧ *is_iterates_fm*(*list_functor_fm*(*13*, *1*, *0*), *3*, *0*, *1*) ·
definition *formula_repl2_intf_fm* **where** *formula_repl2_intf_fm* ≡ ··0 ∈ *3*· ∧ *is_iterates_fm*(*formula_functor_fm*(*1*, *0*), *2*, *0*, *1*) ·
definition *eclose_abs_fm* **where** *eclose_abs_fm* ≡ ··0 ∈ *3*· ∧ *is_iterates_fm*(·⋃ *1 is 0*·, *2*, *0*, *1*) ·
definition *powapply_repl_fm* **where** *powapply_repl_fm* ≡ *is_Powapply_fm*(*2*,*0*,*1*)
definition *wfrec_rank_fm* **where** *wfrec_rank_fm* ≡ (·∃·*pair_fm*(*1*, *0*, *2*) ∧ *is_wfrec_fm*(*is_Hrank_fm*(*2*, *1*, *0*), *3*, *1*, *0*) ··)
definition *transrec_VFrom_fm* **where** *transrec_VFrom_fm* ≡ (·∃·*pair_fm*(*1*, *0*, *2*) ∧ *is_wfrec_fm*(*is_HVfrom_fm*(*8*, *2*, *1*, *0*), *4*, *1*, *0*) ··)
definition *wfrec_Hcheck_fm* **where** *wfrec_Hcheck_fm* ≡ (·∃·*pair_fm*(*1*, *0*, *2*) ∧ *is_wfrec_fm*(*is_Hcheck_fm*(*8*, *2*, *1*, *0*), *4*, *1*, *0*) ··)
definition *repl_PHcheck_fm* **where** *repl_PHcheck_fm* ≡ *PHcheck_fm*(*2*,*3*,*0*,*1*)
definition *tl_repl_intf_fm* **where** *tl_repl_intf_fm* ≡ (·∃·*pair_fm*(*1*, *0*, *2*) ∧ *is_wfrec_fm*(*iterates_MH_fm*(*tl_fm*(*1*,*0*), *9*, *2*, *1*, *0*), *3*, *1*, *0*) ··)
definition *formula_repl1_intf_fm* **where** *formula_repl1_intf_fm* ≡ (·∃·*pair_fm*(*1*, *0*, *2*) ∧ *is_wfrec_fm*(*iterates_MH_fm*(*formula_functor_fm*(*1*,*0*), *9*, *2*, *1*, *0*), *3*, *1*, *0*) ··)
definition *eclose_closed_fm* **where** *eclose_closed_fm* ≡ (·∃·*pair_fm*(*1*, *0*, *2*) ∧ *is_wfrec_fm*(*iterates_MH_fm*(·⋃ *1 is 0*·, *9*, *2*, *1*, *0*), *3*, *1*, *0*) ··)

definition *replacement_assm* **where**
replacement_assm(*M*,*env*,φ) $\equiv$ $\varphi \in$ *formula* $\longrightarrow$ *env* $\in$ *list*(*M*) $\longrightarrow$
arity(φ) $\leq$ *2* $+_\omega$ *length*(*env*) $\longrightarrow$
strong_replacement(##*M*,$\lambda x\ y.\ (M\ ,\ [x,y]@env \models \varphi)$)

definition *ground_replacement_assm* **where**
ground_replacement_assm(*M*,*env*,φ) $\equiv$ *replacement_assm*(*M*,*env*,*ground_repl_fm*(φ))

end

6 The ZFC axioms, internalized

theory *Internal_ZFC_Axioms*
imports
Fm_Definitions

begin

schematic_goal *ZF_union_auto*:
$Union\_ax(\#\#A) \longleftrightarrow (A,\ [] \models ?zfunion)$
unfolding *Union_ax_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_union* **from_schematic** *ZF_union_auto*
notation *ZF_union_fm* ($\langle\cdot Union\ Ax\cdot\rangle$)

schematic_goal *ZF_power_auto*:
$power\_ax(\#\#A) \longleftrightarrow (A,\ [] \models ?zfpow)$
unfolding *power_ax_def powerset_def subset_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_power* **from_schematic** *ZF_power_auto*
notation *ZF_power_fm* ($\langle\cdot Powerset\ Ax\cdot\rangle$)

schematic_goal *ZF_pairing_auto*:
$upair\_ax(\#\#A) \longleftrightarrow (A,\ [] \models ?zfpair)$
unfolding *upair_ax_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_pairing* **from_schematic** *ZF_pairing_auto*
notation *ZF_pairing_fm* ($\langle\cdot Pairing\cdot\rangle$)

schematic_goal *ZF_foundation_auto*:
$foundation\_ax(\#\#A) \longleftrightarrow (A,\ [] \models ?zffound)$
unfolding *foundation_ax_def*
by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_foundation* **from_schematic** *ZF_foundation_auto*
notation *ZF_foundation_fm* ($\langle\cdot Foundation\cdot\rangle$)

schematic_goal *ZF_extensionality_auto*:
  extensionality(##*A*) ⟷ (*A*, [] ⊨ *?zfext*)
 unfolding *extensionality_def*
 by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_extensionality* **from_schematic** *ZF_extensionality_auto*
notation *ZF_extensionality_fm* (‹·*Extensionality*·›)

schematic_goal *ZF_infinity_auto*:
  infinity_ax(##*A*) ⟷ (*A*, [] ⊨ (*?φ*(*i*,*j*,*h*)))
 unfolding *infinity_ax_def*
 by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_infinity* **from_schematic** *ZF_infinity_auto*
notation *ZF_infinity_fm* (‹·*Infinity*·›)

schematic_goal *ZF_choice_auto*:
  choice_ax(##*A*) ⟷ (*A*, [] ⊨ (*?φ*(*i*,*j*,*h*)))
 unfolding *choice_ax_def*
 by ((*rule sep_rules* | *simp*)+)

synthesize *ZF_choice* **from_schematic** *ZF_choice_auto*
notation *ZF_choice_fm* (‹·*AC*·›)

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 ≡ {·*Extensionality*·, ·*Foundation*·, ·*Pairing*·,
    ·*Union Ax*·, ·*Infinity*·, ·*Powerset Ax*·}

6.1 The Axiom of Separation, internalized

lemma *iterates_Forall_type* [*TC*]:
  ⟦ *n* ∈ *nat*; *p* ∈ *formula* ⟧ ⟹ *Forall*^*n*(*p*) ∈ *formula*
 by (*induct set*:*nat*, *auto*)

lemma *last_init_eq* :
 assumes *l* ∈ *list*(*A*) *length*(*l*) = *succ*(*n*)
 shows ∃ *a*∈*A*. ∃ *l′*∈*list*(*A*). *l* = *l′*@[*a*]
proof-
 from ‹*l*∈_› ‹*length*(_) = _›
 have *rev*(*l*) ∈ *list*(*A*) *length*(*rev*(*l*)) = *succ*(*n*)
  by *simp_all*

```
  then
  obtain a l' where a∈A l'∈list(A) rev(l) = Cons(a,l')
    by (cases;simp)
  then
  have l = rev(l') @ [a] rev(l') ∈ list(A)
    using rev_rev_ident[OF ‹l∈_›] by auto
  with ‹a∈_›
  show ?thesis by blast
qed

lemma take_drop_eq :
  assumes l∈list(M)
  shows ⋀ n . n < succ(length(l)) ⟹ l = take(n,l) @ drop(n,l)
  using ‹l∈list(M)›
proof induct
  case Nil
  then show ?case by auto
next
  case (Cons a l)
  then show ?case
  proof -
    {
      fix i
      assume i<succ(succ(length(l)))
      with ‹l∈list(M)›
      consider (lt) i = 0 | (eq) ∃ k∈nat. i = succ(k) ∧ k < succ(length(l))
        using ‹l∈list(M)›  le_natI nat_imp_quasinat
        by (cases rule:nat_cases[of i];auto)
      then
      have take(i,Cons(a,l)) @ drop(i,Cons(a,l)) = Cons(a,l)
        using Cons
        by (cases;auto)
    }
    then show ?thesis using Cons by auto
  qed
qed

lemma list_split :
assumes n ≤ succ(length(rest)) rest ∈ list(M)
shows  ∃ re∈list(M). ∃ st∈list(M). rest = re @ st ∧ length(re) = pred(n)
proof -
  from assms
  have pred(n) ≤ length(rest)
    using pred_mono[OF _ ‹n≤_›] pred_succ_eq by auto
  with ‹rest∈_›
  have pred(n)∈nat rest = take(pred(n),rest) @ drop(pred(n),rest) (is _ = ?re @
?st)
    using take_drop_eq[OF ‹rest∈_›] le_natI by auto
  then
```

```
  have length(?re) = pred(n) ?re∈list(M) ?st∈list(M)
    using length_take[rule_format,OF _ ‹pred(n)∈_›] ‹pred(n) ≤ _› ‹rest∈_›
    unfolding min_def
    by auto
  then
  show ?thesis
   using rev_bexI[of _ _ λ re. ∃ st∈list(M). rest = re @ st ∧ length(re) = pred(n)]
     ‹length(?re) = _› ‹rest = _›
    by auto
qed

lemma sats_nForall:
  assumes
    φ ∈ formula
  shows
    n∈nat ⟹ ms ∈ list(M) ⟹
      (M, ms ⊨ (Forall^n(φ))) ⟷
      (∀ rest ∈ list(M). length(rest) = n ⟶ M, rest @ ms ⊨ φ)
proof (induct n arbitrary:ms set:nat)
  case 0
  with assms
  show ?case by simp
next
  case (succ n)
  have (∀ rest∈list(M). length(rest) = succ(n) ⟶ P(rest,n)) ⟷
       (∀ t∈M. ∀ res∈list(M). length(res) = n ⟶ P(res @ [t],n))
    if n∈nat for n P
    using that last_init_eq by force
  from this[of _ λrest _. (M, rest @ ms ⊨ φ)] ‹n∈nat›
  have (∀ rest∈list(M). length(rest) = succ(n) ⟶ M, rest @ ms ⊨ φ) ⟷
       (∀ t∈M. ∀ res∈list(M). length(res) = n ⟶  M, (res @ [t]) @ ms ⊨ φ)
    by simp
    with assms succ(1,3) succ(2)[of Cons(_,ms)]
  show ?case
    using arity_sats_iff[of φ _ M Cons(_, ms @ _)] app_assoc
    by (simp)
qed

definition
  sep_body_fm :: i ⇒ i where
  sep_body_fm(p) ≡ (·∀ (·∃ (·∀ ··0 ∈ 1· ↔ ··0 ∈ 2· ∧ incr_bv1^2 (p) ···)·)·)

lemma sep_body_fm_type [TC]: p ∈ formula ⟹ sep_body_fm(p) ∈ formula
  by (simp add: sep_body_fm_def)

lemma sats_sep_body_fm:
  assumes
    φ ∈ formula ms∈list(M) rest∈list(M)
  shows
```

```
    (M, rest @ ms ⊨ sep_body_fm(φ)) ⟷
     separation(##M,λx. M, [x] @ rest @ ms ⊨ φ)
  using assms formula_add_params1[of _ 2 _ _ [_,_] ]
  unfolding sep_body_fm_def separation_def by simp

definition
  ZF_separation_fm :: i ⇒ i (‹·Separation′(_′)·›) where
  ZF_separation_fm(p) ≡ Forall^(pred(arity(p)))(sep_body_fm(p))

lemma ZF_separation_fm_type [TC]: p ∈ formula ⟹ ZF_separation_fm(p) ∈
formula
  by (simp add: ZF_separation_fm_def)

lemma sats_ZF_separation_fm_iff:
  assumes
    φ∈formula
  shows
  (M, [] ⊨ ·Separation(φ)·)
   ⟷
   (∀ env∈list(M). arity(φ) ≤ 1 +ω length(env) ⟶
      separation(##M,λx. M, [x] @ env ⊨ φ))
proof (intro iffI ballI impI)
  let ?n=pred(arity(φ))
  fix env
  assume M, [] ⊨ ZF_separation_fm(φ)
  assume arity(φ) ≤ 1 +ω length(env) env∈list(M)
  moreover from this
  have arity(φ) ≤ succ(length(env)) by simp
  then
  obtain some rest where some∈list(M) rest∈list(M)
    env = some @ rest length(some) = pred(arity(φ))
    using list_split[OF ‹arity(φ) ≤ succ(_)› ‹env∈_›] by force
  moreover from ‹φ∈_›
  have arity(φ) ≤ succ(pred(arity(φ)))
   using succpred_leI by simp
  moreover
  note assms
  moreover
  assume M, [] ⊨ ZF_separation_fm(φ)
  moreover from calculation
  have M, some ⊨ sep_body_fm(φ)
    using sats_nForall[of sep_body_fm(φ) ?n]
    unfolding ZF_separation_fm_def by simp
  ultimately
  show separation(##M, λx. M, [x] @ env ⊨ φ)
    unfolding ZF_separation_fm_def
    using sats_sep_body_fm[of φ [] M some]
      arity_sats_iff[of φ rest M [_] @ some]
      separation_cong[of ##M λx. M, Cons(x, some @ rest) ⊨ φ _ ]
```

by *simp*
next — almost equal to the previous implication
let *?n=pred(arity(φ))*
assume *asm*:∀ *env*∈*list*(*M*). *arity*(*φ*) ≤ *1* +$_\omega$ *length*(*env*) ⟶
separation(##*M*, λ*x*. *M*, [*x*] @ *env* ⊨ *φ*)
{
fix *some*
assume *some*∈*list*(*M*) *length*(*some*) = *pred*(*arity*(*φ*))
moreover
note ‹*φ*∈_›
moreover from *calculation*
have *arity*(*φ*) ≤ *1* +$_\omega$ *length*(*some*)
using *le_trans*[*OF succpred_leI*] *succpred_leI* **by** *simp*
moreover from *calculation* **and** *asm*
have *separation*(##*M*, λ*x*. *M*, [*x*] @ *some* ⊨ *φ*) **by** *blast*
ultimately
have *M*, *some* ⊨ *sep_body_fm*(*φ*)
using *sats_sep_body_fm*[*of φ* [] *M some*]
arity_sats_iff[*of φ _ M* [_,_] @ *some*]
strong_replacement_cong[*of* ##*M* λ*x y*. *M*, *Cons*(*x*, *Cons*(*y*, *some* @ _)) ⊨
φ _]
by *simp*
}
with ‹*φ*∈_›
show *M*, [] ⊨ *ZF_separation_fm*(*φ*)
using *sats_nForall*[*of sep_body_fm*(*φ*) *?n*]
unfolding *ZF_separation_fm_def*
by *simp*
qed

6.2 The Axiom of Replacement, internalized

schematic_goal *sats_univalent_fm_auto*:
assumes

Q_iff_sats:⋀*x y z*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *z*∈*A* ⟹
Q(*x*,*z*) ⟷ (*A*,*Cons*(*z*,*Cons*(*y*,*Cons*(*x*,*env*))) ⊨ *Q1_fm*)
⋀*x y z*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *z*∈*A* ⟹
Q(*x*,*y*) ⟷ (*A*,*Cons*(*z*,*Cons*(*y*,*Cons*(*x*,*env*))) ⊨ *Q2_fm*)
and
asms: *nth*(*i*,*env*) = *B* *i* ∈ *nat* *env* ∈ *list*(*A*)
shows
univalent(##*A*,*B*,*Q*) ⟷ *A*,*env* ⊨ *?ufm*(*i*)
unfolding *univalent_def*
by (*insert asms*; (*rule sep_rules Q_iff_sats* | *simp*)+)

synthesize_notc *univalent* **from_schematic** *sats_univalent_fm_auto*

lemma *univalent_fm_type* [*TC*]: *q1*∈ *formula* ⟹ *q2*∈*formula* ⟹ *i*∈*nat* ⟹

univalent_fm(q2,q1,i) ∈*formula*
by (*simp add:univalent_fm_def*)

lemma *sats_univalent_fm* :
assumes
Q_iff_sats:⋀*x y z*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *z*∈*A* ⟹
Q(*x*,*z*) ⟷ (*A*,*Cons*(*z*,*Cons*(*y*,*Cons*(*x*,*env*))) ⊨ *Q1_fm*)
⋀*x y z*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *z*∈*A* ⟹
Q(*x*,*y*) ⟷ (*A*,*Cons*(*z*,*Cons*(*y*,*Cons*(*x*,*env*))) ⊨ *Q2_fm*)
and
asms: *nth*(*i*,*env*) = *B i* ∈ *nat env* ∈ *list*(*A*)
shows
(*A*,*env* ⊨ *univalent_fm*(*Q1_fm*,*Q2_fm*,*i*)) ⟷ *univalent*(##*A*,*B*,*Q*)
unfolding *univalent_fm_def* **using** *asms sats_univalent_fm_auto*[*OF Q_iff_sats*]
by *simp*

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

lemma *swap_vars_type*[*TC*] :
φ∈*formula* ⟹ *swap_vars*(*φ*) ∈*formula*
unfolding *swap_vars_def* **by** *simp*

lemma *sats_swap_vars* :
[*x*,*y*] @ *env* ∈ *list*(*M*) ⟹ *φ*∈*formula* ⟹
(*M*, [*x*,*y*] @ *env* ⊨ *swap_vars*(*φ*)) ⟷ *M*,[*y*,*x*] @ *env* ⊨ *φ*
unfolding *swap_vars_def*
using *sats_incr_bv_iff* [*of* _ _ *M* _ [*y*,*x*]] **by** *simp*

definition
univalent_Q1 :: *i* ⇒ *i* **where**
univalent_Q1(*φ*) ≡ *incr_bv1*(*swap_vars*(*φ*))

definition
univalent_Q2 :: *i* ⇒ *i* **where**
univalent_Q2(*φ*) ≡ *incr_bv*(*swap_vars*(*φ*))‘*0*

lemma *univalent_Qs_type* [*TC*]:
assumes *φ*∈*formula*
shows *univalent_Q1*(*φ*) ∈ *formula univalent_Q2*(*φ*) ∈ *formula*
unfolding *univalent_Q1_def univalent_Q2_def* **using** *assms* **by** *simp_all*

lemma *sats_univalent_fm_assm*:
assumes
x ∈ *A y* ∈ *A z*∈*A env*∈ *list*(*A*) *φ* ∈ *formula*
shows

$(A, ([x,z] @ env) \models \varphi) \longleftrightarrow (A, Cons(z,Cons(y,Cons(x,env))) \models (univalent\_Q1(\varphi)))$
$(A, ([x,y] @ env) \models \varphi) \longleftrightarrow (A, Cons(z,Cons(y,Cons(x,env))) \models (univalent\_Q2(\varphi)))$
unfolding *univalent_Q1_def univalent_Q2_def*
using
sats_incr_bv_iff[*of* _ _ *A* _ []] — simplifies iterates of $\lambda x.\ incr\_bv(x)$ ‘ *0*
sats_incr_bv1_iff[*of* _ *Cons*(*x*,*env*) *A z y*]
sats_swap_vars assms
by *simp_all*

definition
rep_body_fm :: $i \Rightarrow i$ **where**
rep_body_fm(*p*) ≡ *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* ∈ *formula* ⟹ *rep_body_fm*(*p*) ∈ *formula*
by (*simp add*: *rep_body_fm_def*)

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

lemma *sats_rep_body_fm*:
assumes
$\varphi \in formula$ *ms*∈*list*(*M*) *rest*∈*list*(*M*)
shows
(*M*, *rest* @ *ms* ⊨ *rep_body_fm*(φ)) ⟷
strong_replacement(##*M*,$\lambda x\ y.\ M$, [*x*,*y*] @ *rest* @ *ms* ⊨ φ)
using *assms ZF_replacement_simps*
unfolding *rep_body_fm_def strong_replacement_def univalent_def*
unfolding *univalent_fm_def univalent_Q1_def univalent_Q2_def*
by *simp*

definition
ZF_replacement_fm :: $i \Rightarrow i$ (‹·*Replacement*′(_′)·›) **where**
ZF_replacement_fm(*p*) ≡ *Forall*^(*pred*(*pred*(*arity*(*p*))))(*rep_body_fm*(*p*))

lemma *ZF_replacement_fm_type* [*TC*]: *p* ∈ *formula* ⟹ *ZF_replacement_fm*(*p*)
∈ *formula*
by (*simp add*: *ZF_replacement_fm_def*)

lemma *sats_ZF_replacement_fm_iff*:
assumes
φ∈*formula*
shows
(*M*, [] ⊨ ·*Replacement*(φ)·) ⟷ ($\forall$ *env*. *replacement_assm*(*M*,*env*,φ))
unfolding *replacement_assm_def*

proof (*intro iffI allI impI*)
let *?n=pred(pred(arity(φ)))*
fix *env*
assume $M, [] \models$ *ZF_replacement_fm(φ) arity(φ)* $\leq$ *2* $+_\omega$ *length(env) env∈list(M)*
moreover from *this*
have *arity(φ)* $\leq$ *succ(succ(length(env)))* **by** (*simp*)
moreover from *calculation*
have *pred(arity(φ))* $\leq$ *succ(length(env))*
using *pred_mono[OF _ ‹arity(φ)≤succ(_)›] pred_succ_eq* **by** *simp*
moreover from *calculation*
obtain *some rest* **where** *some∈list(M) rest∈list(M)*
env = some @ rest length(some) = pred(pred(arity(φ)))
using *list_split[OF ‹pred(_) ≤ _› ‹env∈_›]* **by** *auto*
moreover
note *‹φ∈_›*
moreover from *this*
have *arity(φ)* $\leq$ *succ(succ(pred(pred(arity(φ)))))*
using *le_trans[OF succpred_leI] succpred_leI* **by** *simp*
moreover from *calculation*
have *M, some* $\models$ *rep_body_fm(φ)*
using *sats_nForall[of rep_body_fm(φ) ?n]*
unfolding *ZF_replacement_fm_def*
by *simp*
ultimately
show *strong_replacement(##M, λx y. M, [x, y] @ env* $\models$ *φ)*
using *sats_rep_body_fm[of φ [] M some]*
arity_sats_iff[of φ rest M [_,_] @ some]
strong_replacement_cong[of ##M λx y. M, Cons(x, Cons(y, some @ rest))
$\models$ *φ _]*
by *simp*
next — almost equal to the previous implication
let *?n=pred(pred(arity(φ)))*
assume *asm:*$\forall$ *env. φ* $\in$ *formula* $\longrightarrow$
env $\in$ *list(M)* $\longrightarrow$ *arity(φ)* $\leq$ *2* $+_\omega$ *length(env)* $\longrightarrow$
strong_replacement(##M, λx y. M, [x, y] @ env $\models$ *φ)*
{
fix *some*
assume *some∈list(M) length(some) = pred(pred(arity(φ)))*
moreover
note *‹φ∈_›*
moreover from *calculation*
have *arity(φ)* $\leq$ *2* $+_\omega$ *length(some)*
using *le_trans[OF succpred_leI] succpred_leI* **by** *simp*
moreover from *calculation* **and** *asm*
have *strong_replacement(##M, λx y. M, [x, y] @ some* $\models$ *φ)* **by** *blast*
ultimately
have *M, some* $\models$ *rep_body_fm(φ)*
using *sats_rep_body_fm[of φ [] M some]*
arity_sats_iff[of φ _ M [_,_] @ some]

strong_replacement_cong[*of* ##*M* λ*x y*. *M*, *Cons*(*x*, *Cons*(*y*, *some* @ _)) ⊨ φ _]
by *simp*
}
with ‹φ∈_›
show *M*, [] ⊨ *ZF_replacement_fm*(φ)
using *sats_nForall*[*of rep_body_fm*(φ) *?n*]
unfolding *ZF_replacement_fm_def*
by *simp*
qed

definition
ZF_schemes :: *i* **where**
ZF_schemes ≡ {·*Separation*(*p*)· . *p* ∈ *formula* } ∪ {·*Replacement*(*p*)· . *p* ∈ *formula* }

lemma *Un_subset_formula* [*TC*]: *A*⊆*formula* ∧ *B*⊆*formula* ⟹ *A*∪*B* ⊆ *formula*
by *auto*

lemma *ZF_schemes_subset_formula* [*TC*]: *ZF_schemes* ⊆ *formula*
unfolding *ZF_schemes_def* **by** *auto*

lemma *ZF_fin_subset_formula* [*TC*]: *ZF_fin* ⊆ *formula*
unfolding *ZF_fin_def* **by** *simp*

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

lemma *ZF_subset_formula* [*TC*]: *ZF* ⊆ *formula*
unfolding *ZF_def* **by** *auto*

definition
ZFC :: *i* **where**
ZFC ≡ *ZF* ∪ {·*AC*·}

definition
ZF_minus_P :: *i* **where**
ZF_minus_P ≡ *ZF* - { ·*Powerset Ax*· }

definition
Zermelo_fms :: *i* (‹·*Z*·›) **where**
Zermelo_fms ≡ *ZF_fin* ∪ {·*Separation*(*p*)· . *p* ∈ *formula* }

definition
ZC :: *i* **where**
ZC ≡ *Zermelo_fms* ∪ {·*AC*·}

lemma *ZFC_subset_formula*: *ZFC* ⊆ *formula*

by (*simp add*:*ZFC_def Un_subset_formula*)

Satisfaction of a set of sentences

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

lemma *satTI* [*intro*!]:
assumes ⋀φ. φ∈Φ ⟹ *A*,[] ⊨ φ
shows *A* ⊨ Φ
using *assms* **unfolding** *satT_def* **by** *simp*

lemma *satTD* [*dest*] :*A* ⊨ Φ ⟹ φ∈Φ ⟹ *A*,[] ⊨ φ
unfolding *satT_def* **by** *simp*

lemma *satT_mono*: *A* ⊨ Φ ⟹ Ψ ⊆ Φ ⟹ *A* ⊨ Ψ
by *blast*

lemma *satT_Un_iff*: *M* ⊨ Φ ∪ Ψ ⟷ *M* ⊨ Φ ∧ *M* ⊨ Ψ **by** *auto*

lemma *sats_ZFC_iff_sats_ZF_AC*:
(*N* ⊨ *ZFC*) ⟷ (*N* ⊨ *ZF*) ∧ (*N*, [] ⊨ ·*AC*·)
unfolding *ZFC_def ZF_def* **by** *auto*

lemma *satT_ZF_imp_satT_Z*: *M* ⊨ *ZF* ⟹ *M* ⊨ ·*Z*·
unfolding *ZF_def ZF_schemes_def Zermelo_fms_def ZF_fin_def* **by** *auto*

lemma *satT_ZFC_imp_satT_ZC*: *M* ⊨ *ZFC* ⟹ *M* ⊨ *ZC*
unfolding *ZFC_def ZF_def ZF_schemes_def ZC_def Zermelo_fms_def* **by** *auto*

lemma *satT_Z_ZF_replacement_imp_satT_ZF*: *N* ⊨ ·*Z*· ⟹ *N* ⊨ {·*Replacement*(*x*)· . *x* ∈ *formula*} ⟹ *N* ⊨ *ZF*
unfolding *ZF_def ZF_schemes_def Zermelo_fms_def ZF_fin_def* **by** *auto*

lemma *satT_ZC_ZF_replacement_imp_satT_ZFC*: *N* ⊨ *ZC* ⟹ *N* ⊨ {·*Replacement*(*x*)· . *x* ∈ *formula*} ⟹ *N* ⊨ *ZFC*
unfolding *ZFC_def ZF_def ZF_schemes_def ZC_def Zermelo_fms_def* **by** *auto*

lemma *ground_repl_fm_sub_ZF*: {·*Replacement*(*ground_repl_fm*(φ))· . φ ∈ *formula*} ⊆ *ZF*
unfolding *ZF_def ZF_schemes_def* **by** *auto*

lemma *ZF_replacement_fms_sub_ZFC*: {·*Replacement*(φ)· . φ ∈ *formula*} ⊆ *ZFC*
unfolding *ZFC_def ZF_def ZF_schemes_def* **by** *auto*

lemma *ground_repl_fm_sub_ZFC*: {·*Replacement*(*ground_repl_fm*(φ))· . φ ∈

formula} ⊆ *ZFC*
unfolding *ZFC_def ZF_def ZF_schemes_def* **by** *auto*

lemma *ZF_replacement_ground_repl_fm_type*: {·*Replacement*(*ground_repl_fm*(*φ*))·
. *φ* ∈ *formula*} ⊆ *formula*
by *auto*

end

7 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-Constructible` (*M_trivial*, *M_basic*, *M_eclose*, etc).

In order to interpret the locales in `ZF-Constructible` we introduce new locales, each stronger than the previous one, assuming only the instances of Replacement needed to interpret the subsequent locales of that session. From the start we assume Separation for every internalized formula (with one parameter, but this is not a problem since we can use pairing).

theory *Interface*
imports
Fm_Definitions
Transitive_Models.Cardinal_AC_Relative
begin

locale *M_Z_basic* =
fixes *M*
assumes
upair_ax: *upair_ax*(##*M*) **and**
Union_ax: *Union_ax*(##*M*) **and**
power_ax: *power_ax*(##*M*) **and**
extensionality:*extensionality*(##*M*) **and**
foundation_ax: *foundation_ax*(##*M*) **and**
infinity_ax: *infinity_ax*(##*M*) **and**
separation_ax: *φ* ∈ *formula* ⟹ *env* ∈ *list*(*M*) ⟹
arity(*φ*) ≤ 1 +$_\omega$ *length*(*env*) ⟹
separation(##*M*,λ*x*. (*M*, [*x*] @ *env* ⊨ *φ*))

locale *M_transset* =
fixes *M*
assumes
trans_M: *Transset*(*M*)

locale *M_Z_trans* = *M_Z_basic* + *M_transset*

locale *M_ZF1 = M_Z_basic +*
  assumes
    replacement_ax1:
    replacement_assm(M,env,eclose_closed_fm)
    replacement_assm(M,env,eclose_abs_fm)
    replacement_assm(M,env,wfrec_rank_fm)
    replacement_assm(M,env,transrec_VFrom_fm)

definition *instances1_fms* **where** *instances1_fms ≡*
  { *eclose_closed_fm*,
    eclose_abs_fm,
    wfrec_rank_fm,
    transrec_VFrom_fm
  }

This set has 4 internalized formulas.

lemmas *replacement_instances1_defs =*
  list_repl1_intf_fm_def list_repl2_intf_fm_def
  formula_repl1_intf_fm_def formula_repl2_intf_fm_def
  eclose_closed_fm_def eclose_abs_fm_def
  wfrec_rank_fm_def transrec_VFrom_fm_def tl_repl_intf_fm_def

lemma *instances1_fms_type*[*TC*]: *instances1_fms ⊆ formula*
  using *Lambda_in_M_fm_type*
  unfolding *replacement_instances1_defs instances1_fms_def* **by** *simp*

declare (**in** *M_ZF1*) *replacement_instances1_defs*[*simp*]

locale *M_ZF1_trans = M_ZF1 + M_Z_trans*

context *M_Z_trans*
begin

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

7.1 Interface with *M_trivial*

lemma *zero_in_M*: *0 ∈ M*
proof -
  obtain *z* **where** *empty*(*##M*,*z*) *z∈M*
    using *empty_intf*[*OF infinity_ax*]
    by *auto*
  moreover from *this*
  have *z=0*
    using *transitivity empty_def*
    by *auto*
  ultimately
  show *?thesis*

by *simp*
qed

lemma *separation_in_ctm* :
assumes
$\varphi \in$ *formula* *env*$\in$*list*(M)
arity(φ) $\leq$ 1 $+_\omega$ *length*(*env*) **and**
satsQ: $\bigwedge x.\ x{\in}M \Longrightarrow (M,\ [x]@env \models \varphi) \longleftrightarrow Q(x)$
shows
separation(*##M*,*Q*)
using *assms separation_ax satsQ transitivity*
separation_cong[*of* *##M* $\lambda y.\ (M,\ [y]@env \models \varphi)$ *Q*]
by *simp*

end — *M_Z_trans*

locale *M_ZC_basic* = *M_Z_basic* + *M_AC* *##M*

locale *M_ZFC1* = *M_ZF1* + *M_ZC_basic*

locale *M_ZFC1_trans* = *M_ZF1_trans* + *M_ZFC1*

sublocale *M_Z_trans* $\subseteq$ *M_trans* *##M*
using *transitivity zero_in_M exI*[*of* $\lambda x.\ x{\in}M$]
by *unfold_locales simp_all*

sublocale *M_Z_trans* $\subseteq$ *M_trivial* *##M*
using *upair_ax Union_ax* **by** *unfold_locales*

7.2 Interface with *M_basic*

definition *Intersection* **where**
Intersection(N,B,x) $\equiv (\forall\, y[N].\ y{\in}B \longrightarrow x{\in}y)$

synthesize *Intersection* **from_definition** *Intersection* **assuming** *nonempty*
arity_theorem for *Intersection_fm*

definition *CartProd* **where**
CartProd(N,B,C,z) $\equiv (\exists\, x[N].\ x{\in}B \wedge (\exists\, y[N].\ y{\in}C \wedge$ *pair*(N,x,y,z)))

synthesize *CartProd* **from_definition** *CartProd* **assuming** *nonempty*
arity_theorem for *CartProd_fm*

definition *ImageSep* **where**
ImageSep(N,B,r,y) $\equiv (\exists\, p[N].\ p{\in}r \wedge (\exists\, x[N].\ x{\in}B \wedge$ *pair*(N,x,y,p)))

synthesize *ImageSep* **from_definition** **assuming** *nonempty*
arity_theorem for *ImageSep_fm*

definition *Converse* **where**
  $Converse(N,R,z) \equiv \exists\, p[N].\ p{\in}R \wedge (\exists\, x[N].\exists\, y[N].\ pair(N,x,y,p) \wedge pair(N,y,x,z))$

synthesize *Converse* **from_definition** *Converse* **assuming** *nonempty*
arity_theorem for *Converse_fm*

definition *Restrict* **where**
  $Restrict(N,A,z) \equiv \exists\, x[N].\ x{\in}A \wedge (\exists\, y[N].\ pair(N,x,y,z))$

synthesize *Restrict* **from_definition** *Restrict* **assuming** *nonempty*
arity_theorem for *Restrict_fm*

definition *Comp* **where**
  $Comp(N,R,S,xz) \equiv \exists\, x[N].\ \exists\, y[N].\ \exists\, z[N].\ \exists\, xy[N].\ \exists\, yz[N].$
  $pair(N,x,z,xz) \wedge pair(N,x,y,xy) \wedge pair(N,y,z,yz) \wedge xy{\in}S \wedge yz{\in}R$

synthesize *Comp* **from_definition** *Comp* **assuming** *nonempty*
arity_theorem for *Comp_fm*

definition *Pred* **where**
  $Pred(N,R,X,y) \equiv \exists\, p[N].\ p{\in}R \wedge pair(N,y,X,p)$

synthesize *Pred* **from_definition** *Pred* **assuming** *nonempty*
arity_theorem for *Pred_fm*

definition *is_Memrel* **where**
  $is\_Memrel(N,z) \equiv \exists\, x[N].\ \exists\, y[N].\ pair(N,x,y,z) \wedge x \in y$

synthesize *is_Memrel* **from_definition** *is_Memrel* **assuming** *nonempty*
arity_theorem for *is_Memrel_fm*

definition *RecFun* **where**
  $RecFun(N,r,f,g,a,b,x) \equiv \exists\, xa[N].\ \exists\, xb[N].$
  $pair(N,x,a,xa) \wedge xa \in r \wedge pair(N,x,b,xb) \wedge xb \in r \wedge$
  $(\exists\, fx[N].\ \exists\, gx[N].\ fun\_apply(N,f,x,fx) \wedge fun\_apply(N,g,x,gx) \wedge$
  $fx \neq gx)$

synthesize *RecFun* **from_definition** *RecFun* **assuming** *nonempty*
arity_theorem for *RecFun_fm*

arity_theorem for *rtran_closure_mem_fm*

synthesize *wellfounded_trancl* **from_definition assuming** *nonempty*
arity_theorem for *wellfounded_trancl_fm*

context *M_Z_trans*
begin

lemma *inter_sep_intf* :

assumes $A{\in}M$
shows *separation*$(\#\#M,\lambda x\ .\ \forall\, y{\in}M\ .\ y{\in}A \longrightarrow x{\in}y)$
using *assms separation_in_ctm*[*of Intersection_fm*(1,0) [A] *Intersection*$(\#\#M,A)$]
Intersection_iff_sats[*of 1* [_,A] *A 0 _ M*] *arity_Intersection_fm Intersection_fm_type*
ord_simp_union zero_in_M
unfolding *Intersection_def*
by *simp*

lemma *diff_sep_intf* :
assumes $B{\in}M$
shows *separation*$(\#\#M,\lambda x\ .\ x{\notin}B)$
using *assms separation_in_ctm*[*of Neg*(*Member*(0,1)) [B] $\lambda x\ .\ x{\notin}B$] *ord_simp_union*
by *simp*

lemma *cartprod_sep_intf* :
assumes $A{\in}M$ **and** $B{\in}M$
shows *separation*$(\#\#M,\lambda z.\ \exists\, x{\in}M.\ x{\in}A \wedge (\exists\, y{\in}M.\ y{\in}B \wedge pair(\#\#M,x,y,z)))$
using *assms separation_in_ctm*[*of CartProd_fm*(1,2,0) [A,B] *CartProd*$(\#\#M,A,B)$]
CartProd_iff_sats[*of 1* [_,A,B] *A 2 B 0 _ M*] *arity_CartProd_fm CartProd_fm_type*
ord_simp_union zero_in_M
unfolding *CartProd_def*
by *simp*

lemma *image_sep_intf* :
assumes $A{\in}M$ **and** $B{\in}M$
shows *separation*$(\#\#M,\ \lambda y.\ \exists\, p{\in}M.\ p{\in}B \wedge (\exists\, x{\in}M.\ x{\in}A \wedge pair(\#\#M,x,y,p)))$
using *assms separation_in_ctm*[*of ImageSep_fm*(1,2,0) [A,B] *ImageSep*$(\#\#M,A,B)$]
ImageSep_iff_sats[*of 1* [_,A,B] *_ 2 _ 0 _ M*] *arity_ImageSep_fm ImageSep_fm_type*
ord_simp_union zero_in_M
unfolding *ImageSep_def*
by *simp*

lemma *converse_sep_intf* :
assumes $R{\in}M$
shows *separation*$(\#\#M,\lambda z.\ \exists\, p{\in}M.\ p{\in}R \wedge (\exists\, x{\in}M.\exists\, y{\in}M.\ pair(\#\#M,x,y,p) \wedge pair(\#\#M,y,x,z)))$
using *assms separation_in_ctm*[*of Converse_fm*(1,0) [R] *Converse*$(\#\#M,R)$]
Converse_iff_sats[*of 1* [_,R] *_ 0 _ M*] *arity_Converse_fm Converse_fm_type*
ord_simp_union zero_in_M
unfolding *Converse_def*
by *simp*

lemma *restrict_sep_intf* :
assumes $A{\in}M$
shows *separation*$(\#\#M,\lambda z.\ \exists\, x{\in}M.\ x{\in}A \wedge (\exists\, y{\in}M.\ pair(\#\#M,x,y,z)))$
using *assms separation_in_ctm*[*of Restrict_fm*(1,0) [A] *Restrict*$(\#\#M,A)$]

Restrict_iff_sats[*of 1* [_,*A*] _ *0* _ *M*] *arity_Restrict_fm Restrict_fm_type*
ord_simp_union zero_in_M
unfolding *Restrict_def*
by *simp*

lemma *comp_sep_intf* :
assumes $R{\in}M$ **and** $S{\in}M$
shows *separation*(##*M*,$\lambda xz.\ \exists\, x{\in}M.\ \exists\, y{\in}M.\ \exists\, z{\in}M.\ \exists\, xy{\in}M.\ \exists\, yz{\in}M.$
pair(##*M*,*x*,*z*,*xz*) $\wedge$ *pair*(##*M*,*x*,*y*,*xy*) $\wedge$ *pair*(##*M*,*y*,*z*,*yz*) $\wedge$ $xy{\in}S$ $\wedge$ $yz{\in}R$)
using *assms separation_in_ctm*[*of Comp_fm*(*1*,*2*,*0*) [*R*,*S*] *Comp*(##*M*,*R*,*S*)]
Comp_iff_sats[*of 1* [_,*R*,*S*] _ *2* _ *0* _ *M*] *arity_Comp_fm Comp_fm_type*
ord_simp_union zero_in_M
unfolding *Comp_def*
by *simp*

lemma *pred_sep_intf*:
assumes $R{\in}M$ **and** $X{\in}M$
shows *separation*(##*M*, $\lambda y.\ \exists\, p{\in}M.\ p{\in}R \wedge$ *pair*(##*M*,*y*,*X*,*p*))
using *assms separation_in_ctm*[*of Pred_fm*(*1*,*2*,*0*) [*R*,*X*] *Pred*(##*M*,*R*,*X*)]
Pred_iff_sats[*of 1* [_,*R*,*X*] _ *2* _ *0* _ *M*] *arity_Pred_fm Pred_fm_type*
ord_simp_union zero_in_M
unfolding *Pred_def*
by *simp*

lemma *memrel_sep_intf*:
separation(##*M*, $\lambda z.\ \exists\, x{\in}M.\ \exists\, y{\in}M.$ *pair*(##*M*,*x*,*y*,*z*) $\wedge$ $x \in y$)
using *separation_in_ctm*[*of is_Memrel_fm*(*0*) [] *is_Memrel*(##*M*)]
is_Memrel_iff_sats[*of 0* [_] _ *M*] *arity_is_Memrel_fm is_Memrel_fm_type*
ord_simp_union zero_in_M
unfolding *is_Memrel_def*
by *simp*

lemma *is_recfun_sep_intf* :
assumes $r{\in}M$ $f{\in}M$ $g{\in}M$ $a{\in}M$ $b{\in}M$
shows *separation*(##*M*,$\lambda x.\ \exists\, xa{\in}M.\ \exists\, xb{\in}M.$
pair(##*M*,*x*,*a*,*xa*) $\wedge$ $xa \in r \wedge$ *pair*(##*M*,*x*,*b*,*xb*) $\wedge$ $xb \in r \wedge$
$(\exists\, fx{\in}M.\ \exists\, gx{\in}M.$ *fun_apply*(##*M*,*f*,*x*,*fx*) $\wedge$ *fun_apply*(##*M*,*g*,*x*,*gx*) $\wedge$
$fx \neq gx$))
using *assms separation_in_ctm*[*of RecFun_fm*(*1*,*2*,*3*,*4*,*5*,*0*) [*r*,*f*,*g*,*a*,*b*] *RecFun*(##*M*,*r*,*f*,*g*,*a*,*b*)]
RecFun_iff_sats[*of 1* [_,*r*,*f*,*g*,*a*,*b*] _ *2* _ *3* _ *4* _ *5* _ *0* _ *M*] *arity_RecFun_fm RecFun_fm_type*
ord_simp_union zero_in_M
unfolding *RecFun_def*
by *simp*

lemmas *M_basic_sep_instances* =
inter_sep_intf diff_sep_intf cartprod_sep_intf

image_sep_intf converse_sep_intf restrict_sep_intf
pred_sep_intf memrel_sep_intf comp_sep_intf is_recfun_sep_intf

end — *M_Z_trans*

sublocale *M_Z_trans* ⊆ *M_basic_no_repl* ##*M*
using *power_ax M_basic_sep_instances*
by *unfold_locales simp_all*

lemma *Replace_eq_Collect*:
assumes ⋀*x y y'*. *x*∈*A* ⟹ *P*(*x*,*y*) ⟹ *P*(*x*,*y'*) ⟹ *y*=*y'* {*y* . *x* ∈ *A*, *P*(*x*, *y*)} ⊆ *B*
shows {*y* . *x* ∈ *A*, *P*(*x*, *y*)} = {*y*∈*B* . ∃ *x*∈*A*. *P*(*x*,*y*)}
using *assms* **by** *blast*

context *M_Z_trans*
begin

lemma *Pow_inter_M_closed*: **assumes** *A* ∈ *M* **shows** *Pow*(*A*) ∩ *M* ∈ *M*
proof -
have {*a* ∈ *Pow*(*A*) . *a* ∈ *M*} = *Pow*(*A*) ∩ *M* **by** *auto*
then
show *?thesis*
using *power_ax powerset_abs assms* **unfolding** *power_ax_def*
by *auto*
qed

lemma *Pow'_inter_M_closed*: **assumes** *A* ∈ *M* **shows** {*a* ∈ *Pow*(*A*) . *a* ∈ *M*} ∈ *M*
using *power_ax powerset_abs assms* **unfolding** *power_ax_def* **by** *auto*

end — *M_Z_trans*

context *M_basic_no_repl*
begin

lemma *Replace_funspace_succ_rep_intf_sub*:
assumes
M(*A*) *M*(*n*)
shows
{*z* . *p* ∈ *A*, *funspace_succ_rep_intf_rel*(*M*,*p*,*z*,*n*)}
⊆ *Pow*^{M}(*Pow*^{M}(⋃ *domain*(*A*) ∪ ({*n*} × *range*(*A*)) ∪ (⋃ ({*n*} × *range*(*A*)))))
unfolding *funspace_succ_rep_intf_rel_def* **using** *assms mem_Pow_rel_abs*
by *clarsimp* (*auto simp*: *cartprod_def*)

lemma *funspace_succ_rep_intf_uniq*:
assumes
funspace_succ_rep_intf_rel(*M*,*p*,*z*,*n*) *funspace_succ_rep_intf_rel*(*M*,*p*,*z'*,*n*)
shows

$z = z'$
using *assms* **unfolding** *funspace_succ_rep_intf_rel_def* **by** *auto*

lemma *Replace_funspace_succ_rep_intf_eq*:
assumes
$M(A)$ $M(n)$
shows
$\{z \,.\, p \in A, \mathit{funspace\_succ\_rep\_intf\_rel}(M,p,z,n)\} =$
$\{z \in Pow^M(Pow^M(\bigcup \mathit{domain}(A) \cup (\{n\} \times \mathit{range}(A)) \cup (\bigcup(\{n\} \times \mathit{range}(A)))))$
.
$\exists\, p{\in}A.\ \mathit{funspace\_succ\_rep\_intf\_rel}(M,p,z,n)\}$
using *assms Replace_eq_Collect*[*OF funspace_succ_rep_intf_uniq*, *of A*,
OF _ _ Replace_funspace_succ_rep_intf_sub[*of A n*], *of* $\lambda x\ y\ z.\ x$ $\lambda x\ y\ z.\ n$]
by (*intro equalityI*)
(*auto dest*:*transM simp*:*funspace_succ_rep_intf_rel_def*)

end — *M_basic_no_repl*

definition *fsri* **where**
$\mathit{fsri}(N,A,B) \equiv \lambda z.\ \exists\, p{\in}A.\ \exists\, f[N].\ \exists\, b[N].\ p = \langle f,\ b\rangle \wedge z = \{\mathit{cons}(\langle B,\ b\rangle,\ f)\}$

relationalize *fsri is_fsri*
synthesize *is_fsri* **from_definition assuming** *nonempty*
arity_theorem for *is_fsri_fm*

context *M_Z_trans*
begin

lemma *separation_fsri*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow \mathit{separation}(\#\#M,\ \mathit{is\_fsri}(\#\#M,A,B))$
using *separation_in_ctm*[**where** *env*=[*A*,*B*] **and** φ=*is_fsri_fm*(*1*,*2*,*0*)]
zero_in_M is_fsri_iff_sats[*symmetric*] *arity_is_fsri_fm is_fsri_fm_type*
by (*simp_all add*: *ord_simp_union*)

lemma *separation_funspace_succ_rep_intf_rel*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow \mathit{separation}(\#\#M, \lambda z.\ \exists\, p{\in}A.\ \mathit{funspace\_succ\_rep\_intf\_rel}(\#\#M,p,z,B))$
using *separation_fsri zero_in_M*
by (*rule_tac separation_cong*[*THEN iffD1*, *of _ is_fsri*(##*M*,*A*,*B*)])
(*auto simp flip*:*setclass_iff dest*:*transM*
simp:*is_fsri_def funspace_succ_rep_intf_rel_def*, *force*)

lemma *Replace_funspace_succ_rep_intf_in_M*:
assumes
$A \in M$ $n \in M$
shows
$\{z \,.\, p \in A, \mathit{funspace\_succ\_rep\_intf\_rel}(\#\#M,p,z,n)\} \in M$
proof -
have $(\#\#M)(\{z \in Pow^M(Pow^M(\bigcup \mathit{domain}(A) \cup (\{n\} \times \mathit{range}(A)) \cup (\bigcup(\{n\}$

$\times$ *range*(*A*))))) .
        $\exists\, p{\in}A.$ *funspace_succ_rep_intf_rel*(*##M*,*p*,*z*,*n*)})
    using *assms separation_funspace_succ_rep_intf_rel*
    by (*intro separation_closed*) (*auto simp flip*:*setclass_iff*)
  with *assms*
  show *?thesis*
    using *Replace_funspace_succ_rep_intf_eq* **by** *auto*
qed

lemma *funspace_succ_rep_intf*:
  assumes *n*∈*M*
  shows
    strong_replacement(*##M*,
          $\lambda p\ z.\ \exists\, f{\in}M.\ \exists\, b{\in}M.\ \exists\, nb{\in}M.\ \exists\, cnbf{\in}M.$
               pair(*##M*,*f*,*b*,*p*) $\wedge$ *pair*(*##M*,*n*,*b*,*nb*) $\wedge$ *is_cons*(*##M*,*nb*,*f*,*cnbf*) $\wedge$
                upair(*##M*,*cnbf*,*cnbf*,*z*))
  using *assms pair_in_M_iff*[*simplified*] *cons_closed*[*simplified*]
  unfolding *strong_replacement_def univalent_def*
  apply (*clarsimp*, *rename_tac A*)
   apply (*rule_tac x*={*z* . *p* ∈ *A*, *funspace_succ_rep_intf_rel*(*##M*,*p*,*z*,*n*)} **in** *bexI*)
    apply (*auto simp*:*funspace_succ_rep_intf_rel_def*
    Replace_funspace_succ_rep_intf_in_M[*unfolded funspace_succ_rep_intf_rel_def*, *simplified*])
  done

end — *M_Z_trans*

sublocale *M_Z_trans* ⊆ *M_basic ##M*
  using *power_ax M_basic_sep_instances funspace_succ_rep_intf*
  by *unfold_locales auto*

7.3 Interface with *M_trancl*

context *M_ZF1_trans*
begin

lemma *rtrancl_separation_intf*:
  assumes *r*∈*M A*∈*M*
  shows *separation* (*##M*, *rtran_closure_mem*(*##M*,*A*,*r*))
 using *assms separation_in_ctm*[*of rtran_closure_mem_fm*(*1*,*2*,*0*) [*A*,*r*] *rtran_closure_mem*(*##M*,*A*,*r*)]
    arity_rtran_closure_mem_fm ord_simp_union zero_in_M
  by *simp*

lemma *wftrancl_separation_intf*:
  assumes *r*∈*M* **and** *Z*∈*M*
  shows *separation* (*##M*, *wellfounded_trancl*(*##M*,*Z*,*r*))
   using *assms separation_in_ctm*[*of wellfounded_trancl_fm*(*1*,*2*,*0*) [*Z*,*r*] *well-founded_trancl*(*##M*,*Z*,*r*)]

arity_wellfounded_trancl_fm ord_simp_union zero_in_M
by *simp*

To prove $\omega \in M$ we get an infinite set I from *infinity_ax* closed under *0* and *succ*; that shows $\omega \subseteq I$. Then we can separate I with the predicate $\lambda x.\ x \in \omega$.

lemma *finite_sep_intf*: *separation*(##*M*, $\lambda x.$ *x*∈*nat*)
proof -
have ($\forall$ *v*∈*M*. *separation*(##*M*,$\lambda x.$ (*M*, [*x*,*v*] ⊨ *finite_ordinal_fm*(*0*))))
using *separation_ax arity_finite_ordinal_fm*
by *simp*
then
have ($\forall$ *v*∈*M*. *separation*(##*M*,*finite_ordinal*(##*M*)))
unfolding *separation_def*
by *simp*
then
have *separation*(##*M*,*finite_ordinal*(##*M*))
using *separation_in_ctm zero_in_M*
by *auto*
then
show *?thesis*
unfolding *separation_def*
by *simp*
qed

lemma *nat_subset_I*: $\exists$ *I*∈*M*. *nat* ⊆ *I*
proof -
have *nat* ⊆ *I*
if *I*∈*M* **and** *0*∈*I* **and** $\bigwedge x.$ *x*∈*I* ⟹ *succ*(*x*)∈*I* **for** *I*
using *that*
by (*rule_tac subsetI*,*induct_tac x*,*simp_all*)
moreover
obtain *I* **where**
I∈*M* *0*∈*I* $\bigwedge x.$ *x*∈*I* ⟹ *succ*(*x*)∈*I*
using *infinity_ax transitivity*
unfolding *infinity_ax_def*
by *auto*
ultimately
show *?thesis*
by *auto*
qed

lemma *nat_in_M*: *nat* ∈ *M*
proof -
have {*x*∈*B* . *x*∈*A*}=*A* **if** *A*⊆*B* **for** *A B*
using *that* **by** *auto*
moreover
obtain *I* **where**
I∈*M* *nat*⊆*I*

using *nat_subset_I* **by** *auto*
moreover from *this*
have $\{x{\in}I\ .\ x{\in}nat\} \in M$
using *finite_sep_intf separation_closed*[*of* $\lambda x\ .\ x{\in}nat$]
by *simp*
ultimately
show *?thesis*
by *simp*
qed

end — *M_ZF1_trans*

sublocale *M_ZF1_trans* $\subseteq$ *M_trancl* $\#\#M$
using *rtrancl_separation_intf wftrancl_separation_intf nat_in_M*
wellfounded_trancl_def
by *unfold_locales auto*

7.4 Interface with *M_eclose*

lemma *repl_sats*:
assumes
sat:$\bigwedge x\ z.\ x{\in}M \Longrightarrow z{\in}M \Longrightarrow (M,\ Cons(x,Cons(z,env)) \models \varphi) \longleftrightarrow P(x,z)$
shows
strong_replacement$(\#\#M,\lambda x\ z.\ (M,\ Cons(x,Cons(z,env)) \models \varphi)) \longleftrightarrow$
strong_replacement$(\#\#M,P)$
by (*rule strong_replacement_cong*,*simp add*:*sat*)

arity_theorem for *list_functor_fm*
arity_theorem for *formula_functor_fm*
arity_theorem for *Inl_fm*
arity_theorem for *Inr_fm*
arity_theorem for *Nil_fm*
arity_theorem for *Cons_fm*
arity_theorem for *quasilist_fm*
arity_theorem for *tl_fm*
arity_theorem for *big_union_fm*

context *M_ZF1_trans*
begin

This lemma obtains *iterates_replacement* for predicates without parameters.

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

```
  ⋀env. (·∃··⟨1,0⟩ is 2· ∧ is_wfrec_fm(iterates_MH_fm(is_F_fm,9,2,1,0),3,1,0)
··) ∈ formula ⟹ env ∈ list(M) ⟹
    arity((·∃··⟨1,0⟩ is 2· ∧ is_wfrec_fm(iterates_MH_fm(is_F_fm,9,2,1,0),3,1,0)
··)) ≤ 2 +ω length(env) ⟹
    strong_replacement(##M,λx y.
     M, [x,y] @ env ⊨ (·∃··⟨1,0⟩ is 2· ∧ is_wfrec_fm(iterates_MH_fm(is_F_fm,9,2,1,0),3,1,0)
··))
  shows
   iterates_replacement(##M,is_F,v)
proof -
  let ?f=(·∃··⟨1,0⟩ is 2· ∧ is_wfrec_fm(iterates_MH_fm(is_F_fm,9,2,1,0),3,1,0)
··)
  have arity(?f) = 4 ?f∈formula
    using arity_iterates_MH_fm[where isF=is_F_fm and i=2]
     arity_wfrec_replacement_fm[where i=10] isfm arty ord_simp_union
    by simp_all
  {
    fix n
    assume n∈nat
    then
    have Memrel(succ(n))∈M
      using nat_into_M Memrel_closed
      by simp
    moreover
    {
      fix a0 a1 a2 a3 a4 y x z
      assume [a0,a1,a2,a3,a4,y,x,z]∈list(M)
      moreover
      note ‹v∈M› ‹Memrel(succ(n))∈M›
      moreover from calculation
      have (M, [b,a,c,d,a0,a1,a2,a3,a4,y,x,z,Memrel(succ(n)),v] ⊨ is_F_fm) ⟷
          is_F(a,b)
        if a∈M b∈M c∈M d∈M for a b c d
        using that satsf[of a b [c,d,a0,a1,a2,a3,a4,y,x,z,Memrel(succ(n)),v]]
        by simp
      moreover from calculation
    have (M, [a0,a1,a2,a3,a4,y,x,z,Memrel(succ(n)),v] ⊨ iterates_MH_fm(is_F_fm,9,2,1,0))
⟷
          iterates_MH(##M,is_F,v,a2, a1, a0)
        using sats_iterates_MH_fm[of M is_F is_F_fm]
        by simp
    }
    moreover from calculation
   have (M, [y,x,z,Memrel(succ(n)),v] ⊨ is_wfrec_fm(iterates_MH_fm(is_F_fm,9,2,1,0),3,1,0))
⟷
        is_wfrec(##M, iterates_MH(##M,is_F,v),Memrel(succ(n)), x, y)
      if y∈M x∈M z∈M for y x z
      using that sats_is_wfrec_fm ‹v∈M› by simp
    moreover from calculation
```

```
    have (M, [x,z,Memrel(succ(n)),v] ⊨ ?f) ⟷

        (∃ y∈M. pair(##M,x,y,z) ∧
        is_wfrec(##M, iterates_MH(##M,is_F,v) , Memrel(succ(n)), x, y))
      if x∈M z∈M for x z
      using that ‹v∈M›
      by (simp del:pair_abs)
    moreover
    note ‹arity(?f) = 4› ‹?f∈formula›
    moreover from calculation ‹v∈_›
    have strong_replacement(##M,λx z. (M, [x,z,Memrel(succ(n)),v] ⊨ ?f))
      using is_F_fm_replacement
      by simp
    ultimately
    have strong_replacement(##M,λx z.
          ∃ y∈M. pair(##M,x,y,z) ∧ is_wfrec(##M, iterates_MH(##M,is_F,v)
,
                Memrel(succ(n)), x, y))
      using repl_sats[of M ?f [Memrel(succ(n)),v]]
      by (simp del:pair_abs)
  }
  then
  show ?thesis
    unfolding iterates_replacement_def wfrec_replacement_def
    by simp
qed

lemma eclose_repl1_intf:
  assumes A∈M
  shows iterates_replacement(##M, big_union(##M), A)
  using assms arity_big_union_fm
    iterates_repl_intf[where is_F_fm=big_union_fm(1,0)]
    replacement_ax1(1)[unfolded replacement_assm_def]
    ord_simp_union
  by simp

lemma eclose_repl2_intf:
  assumes A∈M
 shows strong_replacement(##M,λn y. n∈nat ∧ is_iterates(##M, big_union(##M),
A, n, y))
proof -
  let ?f = And(Member(0,3),is_iterates_fm(big_union_fm(1,0),2,0,1))
  note nat_in_M ‹A∈M›
  moreover from this
  have big_union(##M,a,b) ⟷
        (M, [b,a,c,d,e,f,g,h,i,j,k,n,y,A,nat] ⊨ big_union_fm(1,0))
    if a∈M b∈M c∈M d∈M e∈M f∈Mg∈Mh∈Mi∈Mj∈M k∈M n∈M y∈M
    for a b c d e f g h i j k n y
    using that by simp
```

moreover from *calculation*
have $(M, [n,y,A,nat] \models is\_iterates\_fm(big\_union\_fm(1,0),2,0,1)) \longleftrightarrow$
$is\_iterates(\#\#M,\ big\_union(\#\#M),\ A,\ n\ ,\ y)$
if $n\in M$ $y\in M$ **for** n y
using *that sats_is_iterates_fm*[*of M big_union*(*##M*)]
by *simp*
moreover from *calculation*
have $(M, [n,y,A,nat] \models ?f) \longleftrightarrow$
$n\in nat \wedge is\_iterates(\#\#M,\ big\_union(\#\#M),\ A,\ n,\ y)$
if $n\in M$ $y\in M$ **for** n y
using *that*
by *simp*
moreover
have $arity(?f) = 4$
using *arity_is_iterates_fm*[**where** p=*big_union_fm*(*1,0*) **and** i=*2*]
arity_big_union_fm arity_And ord_simp_union
by *simp*
ultimately
show *?thesis*
using *repl_sats*[*of M ?f* [*A,nat*]] *replacement_ax1*(*2*)[*unfolded replacement_assm_def*]
by *simp*
qed

end — *M_ZF1_trans*

sublocale *M_ZF1_trans* ⊆ *M_eclose ##M*
using *eclose_repl1_intf eclose_repl2_intf*
by *unfold_locales auto*

Interface with *M_eclose*.

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
$is\_Vset(\#\#A,nth(i,\ env),nth(v,\ env)) \longleftrightarrow (A,\ env \models ?ivs\_fm(i,v))$
unfolding *is_Vset_def is_Vfrom_def*
by (*insert assms*; (*rule sep_rules is_HVfrom_iff_sats is_transrec_iff_sats* | *simp*)+)

synthesize *is_Vset* **from_schematic** *sats_is_Vset_fm_auto*
arity_theorem for *is_Vset_fm*

declare *is_Hrank_fm_def*[*fm_definitions add*]

context *M_ZF1_trans*
begin

lemma *wfrec_rank* :

assumes $X{\in}M$
shows *wfrec_replacement*(##M,*is_Hrank*(##M),*rrank*(X))
proof -
let *?f*=*Exists*(*And*(*pair_fm*(1,0,2),*is_wfrec_fm*(*is_Hrank_fm*(2,1,0),3,1,0)))
note *assms zero_in_M*
moreover from *this*
have
is_Hrank(##M,*a2*, *a1*, *a0*) ⟷
(M, [*a0*,*a1*,*a2*,*a3*,*a4*,*y*,*x*,*z*,*rrank*(X)] ⊨ *is_Hrank_fm*(2,1,0))
if *a4*∈M *a3*∈M *a2*∈M *a1*∈M *a0*∈M *y*∈M *x*∈M *z*∈M **for** *a4 a3 a2 a1 a0 y x z*
using *that rrank_in_M is_Hrank_iff_sats*
by *simp*
moreover from *calculation*
have (M, [*y*,*x*,*z*,*rrank*(X)] ⊨ *is_wfrec_fm*(*is_Hrank_fm*(2,1,0),3,1,0)) ⟷
is_wfrec(##M, *is_Hrank*(##M) ,*rrank*(X), *x*, *y*)
if *y*∈M *x*∈M *z*∈M **for** *y x z*
using *that rrank_in_M sats_is_wfrec_fm*
by *simp*
moreover from *calculation*
have (M, [*x*,*z*,*rrank*(X)] ⊨ *?f*) ⟷
(∃ *y*∈M. *pair*(##M,*x*,*y*,*z*) ∧ *is_wfrec*(##M, *is_Hrank*(##M) , *rrank*(X), *x*, *y*))
if *x*∈M *z*∈M **for** *x z*
using *that rrank_in_M*
by (*simp del*:*pair_abs*)
moreover
have *arity*(*?f*) = 3
using *arity_wfrec_replacement_fm*[**where** *p*=*is_Hrank_fm*(2,1,0) **and** *i*=3,*simplified*]
arity_is_Hrank_fm[*of 2 1 0*,*simplified*] *ord_simp_union*
by *simp*
moreover from *calculation*
have *strong_replacement*(##M,λ*x z*. (M, [*x*,*z*,*rrank*(X)] ⊨ *?f*))
using *replacement_ax1*(3)[*unfolded replacement_assm_def*] *rrank_in_M*
by *simp*
ultimately
show *?thesis*
using *repl_sats*[*of M ?f* [*rrank*(X)]]
unfolding *wfrec_replacement_def*
by (*simp del*:*pair_abs*)
qed

lemma *trans_repl_HVFrom* :
assumes $A{\in}M$ $i{\in}M$
shows *transrec_replacement*(##M,*is_HVfrom*(##M,A),i)
proof -
let *?f*=*Exists*(*And*(*pair_fm*(1,0,2),*is_wfrec_fm*(*is_HVfrom_fm*(8,2,1,0),4,1,0)))
note *facts* = *assms zero_in_M*
moreover
have ∃ *sa*∈M. ∃ *esa*∈M. ∃ *mesa*∈M.

```
    upair(##M,a,a,sa) ∧ is_eclose(##M,sa,esa) ∧ membership(##M,esa,mesa)
   if a∈M for a
   using that upair_ax eclose_closed Memrel_closed
   unfolding upair_ax_def
   by (simp del:upair_abs)
  moreover
  {
   fix mesa
   assume mesa∈M
   moreover
   note facts
   moreover from calculation
   have is_HVfrom(##M,A,a2, a1, a0) ⟷
     (M, [a0,a1,a2,a3,a4,y,x,z,A,mesa] ⊨ is_HVfrom_fm(8,2,1,0))
     if a4∈M a3∈M a2∈M a1∈M a0∈M y∈M x∈M z∈M for a4 a3 a2 a1 a0 y x z
     using that sats_is_HVfrom_fm
     by simp
   moreover from calculation
   have (M, [y,x,z,A,mesa] ⊨ is_wfrec_fm(is_HVfrom_fm(8,2,1,0),4,1,0)) ⟷
       is_wfrec(##M, is_HVfrom(##M,A),mesa, x, y)
     if y∈M x∈M z∈M for y x z
     using that sats_is_wfrec_fm
     by simp
   moreover from calculation
   have (M, [x,z,A,mesa] ⊨ ?f) ⟷
            (∃ y∈M. pair(##M,x,y,z) ∧ is_wfrec(##M, is_HVfrom(##M,A) ,
mesa, x, y))
     if x∈M z∈M for x z
     using that
     by (simp del:pair_abs)
   moreover
   have arity(?f) = 4
     using arity_wfrec_replacement_fm[where p=is_HVfrom_fm(8,2,1,0) and
i=9]
       arity_is_HVfrom_fm ord_simp_union
     by simp
   moreover from calculation
   have strong_replacement(##M,λx z. (M, [x,z,A,mesa] ⊨ ?f))
     using replacement_ax1(4)[unfolded replacement_assm_def]
     by simp
   ultimately
   have wfrec_replacement(##M,is_HVfrom(##M,A),mesa)
     using repl_sats[of M ?f [A,mesa]]
     unfolding wfrec_replacement_def
     by (simp del:pair_abs)
  }
  ultimately
  show ?thesis
   unfolding transrec_replacement_def
```

by *simp*
qed

end — *M_ZF1_trans*

7.5 Interface for proving Collects and Replace in M.

context *M_ZF1_trans*
begin

lemma *Collect_in_M* :
assumes
$\varphi \in formula\ env{\in}list(M)$
$arity(\varphi) \leq 1 +_\omega length(env)\ A{\in}M$ **and**
$satsQ$: $\bigwedge x.\ x{\in}M \Longrightarrow (M,\ [x]@env \models \varphi) \longleftrightarrow Q(x)$
shows
$\{y{\in}A\ .\ Q(y)\}{\in}M$
proof -
have $separation(\#\#M,\lambda x.\ (M,\ [x]\ @\ env \models \varphi))$
using *assms separation_ax* **by** *simp*
then
show *?thesis*
using ‹$A{\in}M$› *satsQ transitivity separation_closed*
separation_cong[*of* $\#\#M\ \lambda y.\ (M,\ [y]@env \models \varphi)\ Q$]
by *simp*
qed

— This version has a weaker assumption.
lemma *separation_in_M* :
assumes
$\varphi \in formula\ env{\in}list(M)$
$arity(\varphi) \leq 1 +_\omega length(env)\ A{\in}M$ **and**
$satsQ$: $\bigwedge x.\ x{\in}A \Longrightarrow (M,\ [x]@env \models \varphi) \longleftrightarrow Q(x)$
shows
$\{y{\in}A\ .\ Q(y)\} \in M$
proof -
let $?\varphi' = And(\varphi,Member(0,length(env)+_\omega 1))$
note *assms*
moreover
have $arity(?\varphi') \leq 1 +_\omega length(env@[A])$
using *assms Un_le le_trans*[*of* $arity(\varphi)\ 1+_\omega length(env)\ 2+_\omega length(env)$]
by (*force simp*:*FOL_arities*)
moreover from *calculation*
have $?\varphi'{\in}formula\ nth(length(env),\ env\ @\ [A]) = A$
using *nth_append*
by *auto*
moreover from *calculation*
have $\bigwedge\ x\ .\ x \in M \Longrightarrow (M,\ [x]@env@[A] \models ?\varphi') \longleftrightarrow Q(x) \wedge x{\in}A$
using *arity_sats_iff*[*of* _ [A] _ [_]@*env*]

by *auto*
ultimately
show *?thesis*
using *Collect_in_M*[*of* *?φ′ env*@[*A*] _ λ*x* . *Q*(*x*) ∧ *x*∈*A*, *OF* _ _ _ ‹*A*∈*M*›]
by *auto*
qed

end — *M_ZF1_trans*

context *M_Z_trans*
begin

lemma *strong_replacement_in_ctm*:
assumes
f_fm: *φ* ∈ *formula* **and**
f_ar: *arity*(*φ*)≤ *2* +ω *length*(*env*) **and**
fsats: ⋀*x y*. *x*∈*M* ⟹ *y*∈*M* ⟹ (*M*,[*x*,*y*]@*env* ⊨ *φ*) ⟷ *y* = *f*(*x*) **and**
fclosed: ⋀*x*. *x*∈*M* ⟹ *f*(*x*) ∈ *M* **and**
phi_replacement:*replacement_assm*(*M*,*env*,*φ*) **and**
env∈*list*(*M*)
shows *strong_replacement*(##*M*, λ*x y* . *y* = *f*(*x*))
using *assms*
strong_replacement_cong[*of* ##*M* λ*x y*. *M*,[*x*,*y*]@*env*⊨*φ* λ*x y*. *y* = *f*(*x*)]
unfolding *replacement_assm_def*
by *auto*

lemma *strong_replacement_rel_in_ctm* :
assumes
f_fm: *φ* ∈ *formula* **and**
f_ar: *arity*(*φ*)≤ *2* +ω *length*(*env*) **and**
fsats: ⋀*x y*. *x*∈*M* ⟹ *y*∈*M* ⟹ (*M*,[*x*,*y*]@*env* ⊨ *φ*) ⟷ *f*(*x*,*y*) **and**
phi_replacement:*replacement_assm*(*M*,*env*,*φ*) **and**
env∈*list*(*M*)
shows *strong_replacement*(##*M*, *f*)
using *assms*
strong_replacement_cong[*of* ##*M* λ*x y*. *M*,[*x*,*y*]@*env*⊨*φ f*]
unfolding *replacement_assm_def*
by *auto*

lemma *Replace_in_M* :
assumes
f_fm: *φ* ∈ *formula* **and**
f_ar: *arity*(*φ*)≤ *2* +ω *length*(*env*) **and**
fsats: ⋀*x y*. *x*∈*A* ⟹ *y*∈*M* ⟹ (*M*,[*x*,*y*]@*env* ⊨ *φ*) ⟷ *y* = *f*(*x*) **and**
fclosed: ⋀*x*. *x*∈*A* ⟹ *f*(*x*) ∈ *M* **and**
A∈*M env*∈*list*(*M*) **and**
phi′_replacement:*replacement_assm*(*M*,*env*@[*A*], ·*φ* ∧ ·*0* ∈ *length*(*env*) +ω *2*··
)
shows {*f*(*x*) . *x*∈*A*}∈*M*

proof -
  let *?φ′* = *And*(*φ*,*Member*(*0*,*length*(*env*)+$_\omega$*2*))
  note *assms*
  moreover from *this*
  have *arity*(*?φ′*) ≤ *2* +$_\omega$ *length*(*env*@[*A*])
    using *Un_le le_trans*[*of arity*(*φ*) *2*+$_\omega$(*length*(*env*)) *3*+$_\omega$*length*(*env*)]
    by (*force simp*:*FOL_arities*)
  moreover from *calculation*
  have *?φ′*∈*formula nth*(*length*(*env*), *env* @ [*A*]) = *A*
    using *nth_append* **by** *auto*
  moreover from *calculation*
  have ⋀ *x y*. *x* ∈ *M* ⟹ *y*∈*M* ⟹ (*M*,[*x*,*y*]@*env*@[*A*]⊨ *?φ′*) ⟷ *y*=*f*(*x*) ∧*x*∈*A*
    using *arity_sats_iff*[*of* _ [*A*] _ [_,_]@*env*]
    by *auto*
  moreover from *calculation*
  have *strong_replacement*(##*M*, λ*x y*. *M*,[*x*,*y*]@*env*@[*A*] ⊨ *?φ′*)
   using *phi′_replacement assms*(*1-6*) **unfolding** *replacement_assm_def* **by** *simp*
  ultimately
  have *4*:*strong_replacement*(##*M*, λ*x y*. *y* = *f*(*x*) ∧ *x*∈*A*)
    using
      strong_replacement_cong[*of* ##*M* λ*x y*. *M*,[*x*,*y*]@*env*@[*A*]⊨*?φ′* λ*x y*. *y* =
f(*x*) ∧ *x*∈*A*]
    by *simp*
  then
  have {*y* . *x*∈*A* , *y* = *f*(*x*)} ∈ *M*
    using ‹*A*∈*M*› *strong_replacement_closed*[*OF 4*,*of A*] *fclosed* **by** *simp*
  moreover
  have {*f*(*x*). *x*∈*A*} = { *y* . *x*∈*A* , *y* = *f*(*x*)}
    by *auto*
  ultimately
  show *?thesis* **by** *simp*
qed

lemma *Replace_relativized_in_M* :
  assumes
    f_fm: *φ* ∈ *formula* **and**
    f_ar: *arity*(*φ*)≤ *2* +$_\omega$ *length*(*env*) **and**
    fsats: ⋀*x y*. *x*∈*A* ⟹ *y*∈*M* ⟹ (*M*,[*x*,*y*]@*env* ⊨ *φ*) ⟷ *is_f*(*x*,*y*) **and**
    fabs: ⋀*x y*. *x*∈*A* ⟹ *y*∈*M* ⟹ *is_f*(*x*,*y*) ⟷ *y* = *f*(*x*) **and**
    fclosed: ⋀*x*. *x*∈*A* ⟹ *f*(*x*) ∈ *M* **and**
    A∈*M env*∈*list*(*M*) **and**
    phi′_replacement:*replacement_assm*(*M*,*env*@[*A*], ·*φ* ∧ ·*0* ∈ *length*(*env*) +$_\omega$ *2*··
)
  shows {*f*(*x*) . *x*∈*A*}∈*M*
  using *assms Replace_in_M*[*of φ*] **by** *auto*

lemma *ren_action* :
  assumes
    env∈*list*(*M*) *x*∈*M y*∈*M z*∈*M*

```
  shows ∀ i . i < 2+ω length(env) ⟶
        nth(i,[x,z]@env) = nth(ϱ_repl(length(env))‘i,[z,x,y]@env)
proof -
  let ?f={⟨0, 1⟩, ⟨1, 0⟩}
  have 1:(⋀j. j < length(env) ⟹ nth(j, env) = nth(id(length(env)) ‘ j, env))
    using assms ltD by simp
  have 2:nth(j, [x,z]) = nth(?f ‘ j, [z,x,y]) if j<2 for j
  proof -
    consider j=0 | j=1 using  ltD[OF ‹j<2›] by auto
    then show ?thesis
    proof(cases)
      case 1
      then show ?thesis using apply_equality f_type by simp
    next
      case 2
      then show ?thesis using apply_equality f_type by simp
    qed
  qed
  show ?thesis
   using sum_action[OF _ _ _ _ f_type id_type _ _ _ _ _ _ _ _ 2 1,simplified]
assms
    unfolding ϱ_repl_def by simp
qed

lemma Lambda_in_M :
  assumes
    f_fm:  φ ∈ formula and
    f_ar:  arity(φ)≤ 2 +ω length(env) and
    fsats: ⋀x y. x∈A ⟹ y∈M ⟹ (M,[x,y]@env ⊨ φ) ⟷ is_f(x,y) and
    fabs:  ⋀x y. x∈A ⟹ y∈M ⟹ is_f(x,y) ⟷ y = f(x) and
    fclosed: ⋀x. x∈A ⟹ f(x) ∈ M and
    A∈M env∈list(M) and
   phi'_replacement2: replacement_assm(M,env@[A],Lambda_in_M_fm(φ,length(env)))
  shows (λx∈A . f(x)) ∈M
  unfolding lam_def
proof -
  let ?ren=ϱ_repl(length(env))
  let ?j=2+ω length(env)
  let ?k=3+ω length(env)
  let ?ψ=ren(φ)‘?j‘?k‘?ren
  let ?φ'=Exists(And(pair_fm(1,0,2),?ψ))
  let ?p=λx y. ∃z∈M. pair(##M,x,z,y) ∧ is_f(x,z)
  have ?φ'∈formula ?ψ∈formula
   using ‹env∈_› length_type f_fm ren_type ren_tc[of φ 2+ω length(env) 3+ω length(env)
?ren]
    by simp_all
  moreover from this
  have arity(?ψ)≤3+ω(length(env)) arity(?ψ)∈nat
    using assms arity_ren[OF f_fm _ _ ren_type,of length(env)] by simp_all
```

```
  then
  have arity(?φ') ≤ 2+ω(length(env))
    using Un_le pred_Un_distrib assms pred_le
    by (simp add:arity)
  moreover from this calculation
  have x∈A ⟹ y∈M ⟹ (M,[x,y]@env ⊨ ?φ') ⟷ ?p(x,y) for x y
    using ‹env∈_› length_type[OF ‹env∈_›] assms transitivity[OF _ ‹A∈M›]
     sats_iff_sats_ren[OF f_fm _ _ _ _ ren_type f_ar ren_action[rule_format,of
_ x y],of _ M ]
    by auto
  moreover
  have x∈A ⟹ y∈M ⟹ ?p(x,y) ⟷ y = <x,f(x)> for x y
    using assms transitivity[OF _ ‹A∈_›] fclosed
    by simp
  moreover
  have ⋀ x . x∈A ⟹ <x,f(x)> ∈ M
    using transitivity[OF _ ‹A∈M›] pair_in_M_iff fclosed by simp
  ultimately
  show {⟨x,f(x)⟩ . x∈A } ∈ M
    using Replace_in_M[of ?φ' env A] phi'_replacement2 ‹A∈M› ‹env∈_›
    by simp
qed

lemma ren_action' :
  assumes
    env∈list(M) x∈M y∈M z∈M u∈M
  shows ∀ i . i < 3+ω length(env) ⟶
        nth(i,[x,z,u]@env) = nth(ϱ_pair_repl(length(env)) `i,[x,z,y,u]@env)
proof -
  let ?f={⟨0, 0⟩, ⟨1, 1⟩, ⟨2,3⟩}
  have 1:(⋀j. j < length(env) ⟹ nth(j, env) = nth(id(length(env)) ` j, env))
    using assms ltD by simp
  have 2:nth(j, [x,z,u]) = nth(?f ` j, [x,z,y,u]) if j<3 for j
  proof -
    consider j=0 | j=1 | j=2 using  ltD[OF ‹j<3›] by auto
    then show ?thesis
    proof(cases)
      case 1
      then show ?thesis using apply_equality f_type' by simp
    next
      case 2
      then show ?thesis using apply_equality f_type' by simp
    next
      case 3
      then show ?thesis using apply_equality f_type' by simp
    qed
  qed
  show ?thesis
   using sum_action[OF _ _ _ _ f_type' id_type _ _ _ _ _ _ _ _ 2 1,simplified]
```

assms
unfolding *ϱ_pair_repl_def* **by** *simp*
qed

lemma *LambdaPair_in_M* :
assumes
f_fm: $\varphi \in formula$ **and**
f_ar: $arity(\varphi) \le 3 +_\omega length(env)$ **and**
fsats: $\bigwedge x\ z\ r.\ x{\in}M \Longrightarrow z{\in}M \Longrightarrow r{\in}M \Longrightarrow (M,[x,z,r]@env \models \varphi) \longleftrightarrow is\_f(x,z,r)$
and
fabs: $\bigwedge x\ z\ r.\ x{\in}M \Longrightarrow z{\in}M \Longrightarrow r{\in}M \Longrightarrow is\_f(x,z,r) \longleftrightarrow r = f(x,z)$ **and**
fclosed: $\bigwedge x\ z.\ x{\in}M \Longrightarrow z{\in}M \Longrightarrow f(x,z) \in M$ **and**
$A{\in}M\ env{\in}list(M)$ **and**
phi'_replacement3: *replacement_assm*$(M,env@[A],LambdaPair\_in\_M\_fm(\varphi,length(env)))$
shows $(\lambda x{\in}A\ .\ f(fst(x),snd(x))) \in M$
proof -
let *?ren*=$\varrho\_pair\_repl(length(env))$
let *?j*=$3+_\omega length(env)$
let *?k*=$4+_\omega length(env)$
let *?ψ*=$ren(\varphi)$ ‘*?j*‘*?k*‘*?ren*
let *?φ'*=$Exists(Exists(And(fst\_fm(2,0),(And(snd\_fm(2,1),?\psi)))))$
let *?p*=$\lambda x\ y.\ is\_f(fst(x),snd(x),y)$
have *?φ'*∈*formula* *?ψ*∈*formula*
using ‹*env*∈_› *length_type f_fm ren_type' ren_tc*[*of* φ *?j ?k ?ren*]
by *simp_all*
moreover from *this*
have $arity(?\psi) \le 4+_\omega (length(env))$ $arity(?\psi) \in nat$
using *assms arity_ren*[*OF f_fm _ _ ren_type'*,*of length*(*env*)] **by** *simp_all*
moreover from *calculation*
have *1*:$arity(?\varphi') \le 2+_\omega (length(env))$ *?φ'*∈*formula*
using *Un_le pred_Un_distrib assms pred_le*
by (*simp_all add*:*arity*)
moreover from *this calculation*
have *2*:$x{\in}A \Longrightarrow y{\in}M \Longrightarrow (M,[x,y]@env \models ?\varphi') \longleftrightarrow ?p(x,y)$ **for** *x y*
using
sats_iff_sats_ren[*OF f_fm _ _ _ _ ren_type' f_ar*
ren_action'[*rule_format*,*of _ fst*(*x*) *x snd*(*x*) *y*],*simplified*]
‹*env*∈_› *length_type*[*OF* ‹*env*∈_›] *transitivity*[*OF _* ‹*A*∈*M*›]
fst_snd_closed pair_in_M_iff fsats[*of fst*(*x*) *snd*(*x*) *y*,*symmetric*]
fst_abs snd_abs
by *auto*
moreover from *assms*
have *3*:$x{\in}A \Longrightarrow y{\in}M \Longrightarrow ?p(x,y) \longleftrightarrow y = f(fst(x),snd(x))$ **for** *x y*
using *fclosed fst_snd_closed pair_in_M_iff fabs transitivity*
by *auto*
moreover
have *4*:$\bigwedge x\ .\ x{\in}A \Longrightarrow <x,f(fst(x),snd(x))> \in M \bigwedge x\ .\ x{\in}A \Longrightarrow f(fst(x),snd(x)) \in M$
using *transitivity*[*OF _* ‹*A*∈*M*›] *pair_in_M_iff fclosed fst_snd_closed*

by *simp_all*
ultimately
show *?thesis*
using *Lambda_in_M*[*unfolded Lambda_in_M_fm_def*, *of ?φ'*, *OF* _ _ _ _ _ _ _ _ _
phi'_replacement3[*unfolded LambdaPair_in_M_fm_def*]]
‹*env*∈_› ‹*A*∈_› **by** *simp*

qed

lemma (**in** *M_ZF1_trans*) *lam_replacement2_in_ctm* :
assumes
f_fm: $\varphi \in formula$ **and**
f_ar: $arity(\varphi) \leq 3 +_\omega length(env)$ **and**
fsats: $\bigwedge x\ z\ r.\ x{\in}M \Longrightarrow z{\in}M \Longrightarrow r{\in}M \Longrightarrow (M,[x,z,r]@env \models \varphi) \longleftrightarrow is\_f(x,z,r)$
and
fabs: $\bigwedge x\ z\ r.\ x{\in}M \Longrightarrow z{\in}M \Longrightarrow r{\in}M \Longrightarrow is\_f(x,z,r) \longleftrightarrow r = f(x,z)$ **and**
fclosed: $\bigwedge x\ z.\ x{\in}M \Longrightarrow z{\in}M \Longrightarrow f(x,z) \in M$ **and**
$env{\in}list(M)$ **and**
phi'_replacement3: $\bigwedge A.\ A{\in}M \Longrightarrow replacement\_assm(M,env@[A],LambdaPair\_in\_M\_fm(\varphi,length(env)))$
shows $lam\_replacement(\#\#M\ ,\ \lambda x\ .\ f(fst(x),snd(x)))$
using
LambdaPair_in_M fabs
f_ar ord_simp_union transitivity assms fst_snd_closed
by (*rule_tac lam_replacement_iff_lam_closed*[*THEN iffD2*],*simp_all*)

simple_rename *ren_U* **src** [*z1*,*x_P*, *x_leq*, *x_o*, *x_t*, *z2_c*]
tgt [*z2_c*,*z1*,*z*,*x_P*, *x_leq*, *x_o*, *x_t*]

simple_rename *ren_V* **src** [*fz*,*x_P*, *x_leq*, *x_o*,*x_f*, *x_t*, *gz*]
tgt [*gz*,*fz*,*z*,*x_P*, *x_leq*, *x_o*,*x_f*, *x_t*]

simple_rename *ren_V3* **src** [*fz*,*x_P*, *x_leq*, *x_o*,*x_f*, *gz*, *hz*]
tgt [*hz*,*gz*,*fz*,*z*,*x_P*, *x_leq*, *x_o*,*x_f*]

lemma *separation_sat_after_function_1*:
assumes $[a,b,c,d]{\in}list(M)$ **and** $\chi{\in}formula$ **and** $arity(\chi) \leq 6$
and
f_fm: $f\_fm \in formula$ **and**
f_ar: $arity(f\_fm) \leq 6$ **and**
fsats: $\bigwedge fx\ x.\ fx{\in}M \Longrightarrow x{\in}M \Longrightarrow (M,[fx,x]@[a,\ b,\ c,\ d] \models f\_fm) \longleftrightarrow fx{=}f(x)$
and
fclosed: $\bigwedge x\ .\ x{\in}M \Longrightarrow f(x) \in M$ **and**
g_fm: $g\_fm \in formula$ **and**
g_ar: $arity(g\_fm) \leq 7$ **and**
gsats: $\bigwedge gx\ fx\ x.\ gx{\in}M \Longrightarrow fx{\in}M \Longrightarrow x{\in}M \Longrightarrow (M,[gx,fx,x]@[a,\ b,\ c,\ d] \models g\_fm) \longleftrightarrow gx{=}g(x)$ **and**
gclosed: $\bigwedge x\ .\ x{\in}M \Longrightarrow g(x) \in M$
shows $separation(\#\#M,\ \lambda r.\ M,\ [f(r),\ a,\ b,\ c,\ d,\ g(r)] \models \chi)$

proof -
note *types* = *assms*(*1-4*)
let *?ψ*=*ren*(*χ*)‘*6*‘*7*‘*ren_U_fn*
let *?ψ′*=*Exists*(*And*(*f_fm*,*Exists*(*And*(*g_fm*,*?ψ*))))
let *?ϱ*=*λz*.[*f*(*z*), *a*, *b*, *c*, *d*, *g*(*z*)]
let *?env*=[*a*, *b*, *c*, *d*]
let *?η*=*λz*.[*g*(*z*),*f*(*z*),*z*]@*?env*
note *types*
moreover from *this*
have *arity*(*χ*) ≤ *7* *?ψ*∈*formula*
using *ord_simp_union ren_tc ren_U_thm*(*2*)[*folded ren_U_fn_def*] *le_trans*[*of arity*(*χ*) *6*]
by *simp_all*
moreover from *calculation*
have *arity*(*?ψ*) ≤ *7* *?ψ′*∈*formula*
using *arity_ren ren_U_thm*(*2*)[*folded ren_U_fn_def*] *f_fm g_fm*
by *simp_all*
moreover from *calculation f_ar g_ar f_fm g_fm*
have *arity*(*?ψ′*) ≤ *5*
using *ord_simp_union pred_le arity_type*
by (*simp add*:*arity*)
moreover from *calculation fclosed gclosed*
have *0*:(*M*, [*f*(*z*), *a*, *b*, *c*, *d*, *g*(*z*)] ⊨ *χ*) ⟷ (*M*,*?η*(*z*)⊨ *?ψ*) **if** (*##M*)(*z*) **for** *z*
using *sats_iff_sats_ren*[*of χ 6 7 _ _ ?η*(*z*)]
ren_U_thm(*1*)[**where** *A*=*M*,*folded ren_U_fn_def*] *ren_U_thm*(*2*)[*folded ren_U_fn_def*] *that*
by *simp*
moreover from *calculation*
have *1*:(*M*,*?η*(*z*)⊨ *?ψ*) ⟷ *M*,[*z*]@*?env*⊨*?ψ′* **if** (*##M*)(*z*) **for** *z*
using *that fsats*[*OF fclosed*[*of z*],*of z*] *gsats*[*of g*(*z*) *f*(*z*) *z*] *fclosed gclosed f_fm g_fm*
proof(*rule_tac iffI*,*simp*,*rule_tac rev_bexI*[**where** *x*=*f*(*z*)],*simp*,(*auto*)[*1*])
assume *M*, [*z*] @ [*a*, *b*, *c*, *d*] ⊨ (·∃·*f_fm* ∧ (·∃·*g_fm* ∧ *ren*(*χ*) ‘ *6* ‘ *7* ‘ *ren_U_fn*··)··)
then
have ∃ *xa*∈*M*. (*M*, [*xa*, *z*, *a*, *b*, *c*, *d*] ⊨ *f_fm*) ∧
(∃ *x*∈*M*. (*M*, [*x*, *xa*, *z*, *a*, *b*, *c*, *d*] ⊨ *g_fm*) ∧
(*M*, [*x*, *xa*, *z*, *a*, *b*, *c*, *d*] ⊨ *ren*(*χ*) ‘ *6* ‘ *7* ‘ *ren_U_fn*))
using *that calculation* **by** *auto*
then
obtain *xa x* **where** *x*∈*M xa*∈*M M*, [*xa*, *z*, *a*, *b*, *c*, *d*] ⊨ *f_fm*
(*M*, [*x*, *xa*, *z*, *a*, *b*, *c*, *d*] ⊨ *g_fm*)
(*M*, [*x*, *xa*, *z*, *a*, *b*, *c*, *d*] ⊨ *ren*(*χ*) ‘ *6* ‘ *7* ‘ *ren_U_fn*)
using *that calculation* **by** *auto*
moreover from *this*
have *xa*=*f*(*z*) *x*=*g*(*z*) **using** *fsats*[*of xa*] *gsats*[*of x xa*] *that* **by** *simp_all*
ultimately
show *M*, [*g*(*z*), *f*(*z*), *z*] @ [*a*, *b*, *c*, *d*] ⊨ *ren*(*χ*) ‘ *6* ‘ *7* ‘ *ren_U_fn*
by *auto*

qed
moreover from *calculation*
have *separation*(##*M*, λ*z*. (*M*,[*z*]@*?env* ⊨ *?ψ′*))
using *separation_ax*
by *simp_all*
ultimately
show *?thesis*
by(*rule_tac separation_cong*[*THEN iffD2*,*OF iff_trans*[*OF 0 1*]],*clarify*,*force*)
qed

lemma *separation_sat_after_function3*:
assumes [*a*, *b*, *c*, *d*]∈*list*(*M*) **and** *χ*∈*formula* **and** *arity*(*χ*) ≤ *7*
and
f_fm: *f_fm* ∈ *formula* **and**
f_ar: *arity*(*f_fm*) ≤ *6* **and**
fsats: ⋀ *fx x*. *fx*∈*M* ⟹ *x*∈*M* ⟹ (*M*,[*fx*,*x*]@[*a*, *b*, *c*, *d*] ⊨ *f_fm*) ⟷ *fx*=*f*(*x*)
and
fclosed: ⋀*x* . *x*∈*M* ⟹ *f*(*x*) ∈ *M* **and**
g_fm: *g_fm* ∈ *formula* **and**
g_ar: *arity*(*g_fm*) ≤ *7* **and**
gsats: ⋀ *gx fx x*. *gx*∈*M* ⟹ *fx*∈*M* ⟹ *x*∈*M* ⟹ (*M*,[*gx*,*fx*,*x*]@[*a*, *b*, *c*, *d*] ⊨ *g_fm*) ⟷ *gx*=*g*(*x*) **and**
gclosed: ⋀*x* . *x*∈*M* ⟹ *g*(*x*) ∈ *M* **and**
h_fm: *h_fm* ∈ *formula* **and**
h_ar: *arity*(*h_fm*) ≤ *8* **and**
hsats: ⋀ *hx gx fx x*. *hx*∈*M* ⟹ *gx*∈*M* ⟹ *fx*∈*M* ⟹ *x*∈*M* ⟹ (*M*,[*hx*,*gx*,*fx*,*x*]@[*a*, *b*, *c*, *d*] ⊨ *h_fm*) ⟷ *hx*=*h*(*x*) **and**
hclosed: ⋀*x* . *x*∈*M* ⟹ *h*(*x*) ∈ *M*
shows *separation*(##*M*, λ*r*. *M*, [*f*(*r*), *a*, *b*, *c*, *d*, *g*(*r*), *h*(*r*)] ⊨ *χ*)
proof -
note *types* = *assms*(*1-3*)
let *?φ*=*χ*
let *?ψ*=*ren*(*?φ*)‘*7*‘*8*‘*ren_V3_fn*
let *?ψ′*=*Exists*(*And*(*f_fm*,*Exists*(*And*(*g_fm*,*Exists*(*And*(*h_fm*,*?ψ*))))))
let *?ϱ*=λ*z*.[*f*(*z*), *a*, *b*, *c*, *d*,*g*(*z*), *h*(*z*)]
let *?env*=[*a*, *b*, *c*, *d*]
let *?η*=λ*z*.[*h*(*z*),*g*(*z*),*f*(*z*),*z*]@*?env*
note *types*
moreover from *this*
have *?φ*∈*formula* **by** *simp*
moreover from *calculation*
have *arity*(*?φ*) ≤ *9* *?ψ*∈*formula*
using *ord_simp_union ren_tc ren_V3_thm*(*2*)[*folded ren_V3_fn_def*] *le_trans*[*of arity*(*χ*) *7*]
by *simp_all*
moreover from *calculation*
have *arity*(*?ψ*) ≤ *8* *?ψ′*∈*formula*
using *arity_ren ren_V3_thm*(*2*)[*folded ren_V3_fn_def*] *f_fm g_fm h_fm*
by (*simp_all*)

moreover from *this f_ar g_ar f_fm g_fm h_fm h_ar* ‹*?ψ′∈_*›
have *arity(?ψ′)* ≤ *5*
using *ord_simp_union arity_type nat_into_Ord*
by (*simp add:arity,*(*rule_tac pred_le,simp,rule_tac Un_le,simp*)+,*simp_all add*: ‹*?ψ∈_*›)
moreover from *calculation fclosed gclosed hclosed*
have *0*:(*M*, *?ϱ*(*z*) ⊨ *?φ*) ⟷ (*M*,*?η*(*z*)⊨ *?ψ*) **if** (##*M*)(*z*) **for** *z*
using *sats_iff_sats_ren*[*of ?φ 7 8 ?ϱ*(*z*) *M ?η*(*z*)]
ren_V3_thm(*1*)[**where** *A*=*M*,*folded ren_V3_fn_def*,*simplified*] *ren_V3_thm*(*2*)[*folded ren_V3_fn_def*] *that*
by *simp*
moreover from *calculation*
have *1*:(*M*,*?η*(*z*)⊨ *?ψ*) ⟷ *M*,[*z*]@*?env*⊨*?ψ′* **if** (##*M*)(*z*) **for** *z*
using *that fsats*[*OF fclosed*[*of z*],*of z*] *gsats*[*of g*(*z*) *f*(*z*) *z*]
hsats[*of h*(*z*) *g*(*z*) *f*(*z*) *z*]
fclosed gclosed hclosed f_fm g_fm h_fm
apply(*rule_tac iffI*,*simp*,*rule_tac rev_bexI*[**where** *x*=*f*(*z*)],*simp*)
apply(*rule_tac conjI*,*simp*,*rule_tac rev_bexI*[**where** *x*=*g*(*z*)],*simp*)
apply(*rule_tac conjI*,*simp*,*rule_tac rev_bexI*[**where** *x*=*h*(*z*)],*simp*,*rule_tac conjI*,*simp*,*simp*)
proof -
assume *M*, [*z*] @ [*a*, *b*, *c*, *d*] ⊨ (·∃·*f_fm* ∧ (·∃·*g_fm* ∧ (·∃·*h_fm* ∧ *ren*(*χ*) ‘ *7* ‘ *8* ‘ *ren_V3_fn*··)··)··)
with *calculation that*
have ∃ *x*∈*M*. (*M*, [*x*, *z*, *a*, *b*, *c*, *d*] ⊨ *f_fm*) ∧
(∃ *xa*∈*M*. (*M*, [*xa*, *x*, *z*, *a*, *b*, *c*, *d*] ⊨ *g_fm*) ∧ (∃ *xb*∈*M*. (*M*, [*xb*, *xa*, *x*, *z*, *a*, *b*, *c*, *d*] ⊨ *h_fm*) ∧ (*M*, [*xb*, *xa*, *x*, *z*, *a*, *b*, *c*, *d*] ⊨ *ren*(*χ*) ‘ *7* ‘ *8* ‘ *ren_V3_fn*)))
by *auto*
with *calculation*
obtain *x* **where** *x*∈*M* (*M*, [*x*, *z*, *a*, *b*, *c*, *d*] ⊨ *f_fm*)
(∃ *xa*∈*M*. (*M*, [*xa*, *x*, *z*, *a*, *b*, *c*, *d*] ⊨ *g_fm*) ∧ (∃ *xb*∈*M*. (*M*, [*xb*, *xa*, *x*, *z*, *a*, *b*, *c*, *d*] ⊨ *h_fm*) ∧ (*M*, [*xb*, *xa*, *x*, *z*, *a*, *b*, *c*, *d*] ⊨ *ren*(*χ*) ‘ *7* ‘ *8* ‘ *ren_V3_fn*)))
by *force*
moreover from *this*
have *x*=*f*(*z*) **using** *fsats*[*of x*] *that* **by** *simp*
moreover from *calculation*
obtain *xa* **where** *xa*∈*M* (*M*, [*xa*, *x*, *z*, *a*, *b*, *c*, *d*] ⊨ *g_fm*)
(∃ *xb*∈*M*. (*M*, [*xb*, *xa*, *x*, *z*, *a*, *b*, *c*, *d*] ⊨ *h_fm*) ∧ (*M*, [*xb*, *xa*, *x*, *z*, *a*, *b*, *c*, *d*] ⊨ *ren*(*χ*) ‘ *7* ‘ *8* ‘ *ren_V3_fn*))
by *auto*
moreover from *calculation*
have *xa*=*g*(*z*) **using** *gsats*[*of xa x*] *that* **by** *simp*
moreover from *calculation*
obtain *xb* **where** *xb*∈*M* (*M*, [*xb*, *xa*, *x*, *z*, *a*, *b*, *c*, *d*] ⊨ *h_fm*)
(*M*, [*xb*, *xa*, *x*, *z*, *a*, *b*, *c*, *d*] ⊨ *ren*(*χ*) ‘ *7* ‘ *8* ‘ *ren_V3_fn*)
by *auto*
moreover from *calculation*
have *xb*=*h*(*z*) **using** *hsats*[*of xb xa x*] *that* **by** *simp*
ultimately

show M, $[h(z),\ g(z),\ f(z),\ z]$ @ $[a,\ b,\ c,\ d] \models ren(\chi)$ ‘ 7 ‘ 8 ‘ *ren_V3_fn*
by *auto*
qed
moreover from *calculation* ‹*?ψ′∈_*›
have *separation*(*##M*, $\lambda z.\ (M,[z]@?env \models ?\psi')$)
using *separation_ax*
by *simp*
ultimately
show *?thesis*
by(*rule_tac separation_cong*[*THEN iffD2*,*OF iff_trans*[*OF 0 1*]],*clarify*,*force*)
qed

lemma *separation_sat_after_function*:
assumes $[a,\ b,\ c,\ d,\ \tau] \in list(M)$ **and** $\chi \in formula$ **and** $arity(\chi) \leq 7$
and
f_fm: $f\_fm \in formula$ **and**
f_ar: $arity(f\_fm) \leq 7$ **and**
fsats: $\bigwedge fx\ x.\ fx \in M \Longrightarrow x \in M \Longrightarrow (M,[fx,x]@[a,\ b,\ c,\ d,\ \tau] \models f\_fm) \longleftrightarrow fx = f(x)$ **and**
fclosed: $\bigwedge x\ .\ x \in M \Longrightarrow f(x) \in M$ **and**
g_fm: $g\_fm \in formula$ **and**
g_ar: $arity(g\_fm) \leq 8$ **and**
gsats: $\bigwedge gx\ fx\ x.\ gx \in M \Longrightarrow fx \in M \Longrightarrow x \in M \Longrightarrow (M,[gx,fx,x]@[a,\ b,\ c,\ d,\ \tau] \models g\_fm) \longleftrightarrow gx = g(x)$ **and**
gclosed: $\bigwedge x\ .\ x \in M \Longrightarrow g(x) \in M$
shows *separation*(*##M*, $\lambda r.\ M,\ [f(r),\ a,\ b,\ c,\ d,\ \tau,\ g(r)] \models \chi$)
proof -
note *types* = *assms*(*1-3*)
let $?\varphi = \chi$
let $?\psi = ren(?\varphi)$‘7‘8‘*ren_V_fn*
let $?\psi' = Exists(And(f\_fm, Exists(And(g\_fm, ?\psi))))$
let $?\varrho = \lambda z.[f(z),\ a,\ b,\ c,\ d,\ \tau,\ g(z)]$
let $?env = [a,\ b,\ c,\ d,\ \tau]$
let $?\eta = \lambda z.[g(z),f(z),z]@?env$
note *types*
moreover from *this*
have $?\varphi \in formula$ **by** *simp*
moreover from *calculation*
have $arity(?\varphi) \leq 8$ $?\psi \in formula$
using *ord_simp_union ren_tc ren_V_thm*(*2*)[*folded ren_V_fn_def*] *le_trans*[*of* $arity(\chi)$ 7]
by *simp_all*
moreover from *calculation*
have $arity(?\psi) \leq 8$ $?\psi' \in formula$
using *arity_ren ren_V_thm*(*2*)[*folded ren_V_fn_def*] *f_fm g_fm*
by (*simp_all*)
moreover from *calculation f_ar g_ar f_fm g_fm*
have $arity(?\psi') \leq 6$
using *ord_simp_union pred_le arity_type*

```
    by (simp add:arity)
  moreover from calculation fclosed gclosed
  have 0:(M, ?ϱ(z) ⊨ ?φ) ⟷ (M,?η(z)⊨ ?ψ) if (##M)(z) for z
    using sats_iff_sats_ren[of ?φ 7 8 ?ϱ(z) _ ?η(z)]
        ren_V_thm(1)[where A=M,folded ren_V_fn_def] ren_V_thm(2)[folded
ren_V_fn_def] that
    by simp
  moreover from calculation
  have 1:(M,?η(z)⊨ ?ψ) ⟷ M,[z]@?env⊨?ψ' if (##M)(z) for z
    using that fsats[OF fclosed[of z],of z] gsats[of g(z) f(z) z]
      fclosed gclosed f_fm g_fm
    apply(rule_tac iffI,simp,rule_tac rev_bexI[where x=f(z)],simp)
     apply(auto)[1]
  proof -
     assume M, [z] @ [a, b, c, d, τ] ⊨ (·∃·f_fm ∧ (·∃·g_fm ∧ ren(χ) ‘ 7 ‘ 8 ‘
ren_V_fn··)··)
    then have ∃ xa∈M. (M, [xa, z, a, b, c, d, τ] ⊨ f_fm) ∧
       (∃ x∈M. (M, [x, xa, z, a, b, c, d, τ] ⊨ g_fm) ∧ (M, [x, xa, z, a, b, c, d, τ]
⊨ ren(χ) ‘ 7 ‘ 8 ‘ ren_V_fn))
      using that calculation by auto
    then
    obtain xa where xa∈M M, [xa, z, a, b, c, d, τ] ⊨ f_fm
     (∃ x∈M. (M, [x, xa, z, a, b, c, d, τ] ⊨ g_fm) ∧ (M, [x, xa, z, a, b, c, d, τ] ⊨
ren(χ) ‘ 7 ‘ 8 ‘ ren_V_fn))
      by auto
    moreover from this
    have xa=f(z) using fsats[of xa] that by simp
    moreover from calculation
    obtain x where x∈M M, [x, xa, z, a, b, c, d, τ] ⊨ g_fm M, [x, xa, z, a, b, c,
d, τ] ⊨ ren(χ) ‘ 7 ‘ 8 ‘ ren_V_fn
      by auto
    moreover from calculation
    have x=g(z) using gsats[of x xa] that by simp
    ultimately
    show M, [g(z), f(z), z] @ [a, b, c, d, τ] ⊨ ren(χ) ‘ 7 ‘ 8 ‘ ren_V_fn
      by auto
  qed
  moreover from calculation
  have separation(##M, λz. (M,[z]@?env ⊨ ?ψ'))
    using separation_ax
    by simp_all
  ultimately
  show ?thesis
    by(rule_tac separation_cong[THEN iffD2,OF iff_trans[OF 0 1]],clarify,force)
qed
end

definition separation_assm_fm :: [i,i,i] ⇒ i
  where
```

separation_assm_fm(A,x,f_fm) ≡ (·∃ (·∃ ··0 ∈ A +$_\omega$ 2· ∧ ··⟨0,1⟩ *is* x+$_\omega$ 2 · ∧ *f_fm* ···)·)

lemma *separation_assm_fm_type*[*TC*]:
A ∈ ω ⟹ *y* ∈ ω ⟹ *f_fm* ∈ *formula* ⟹ *separation_assm_fm*(*A*, *y*,*f_fm*) ∈ *formula*
unfolding *separation_assm_fm_def*
by *simp*

lemma *arity_separation_assm_fm* : *A* ∈ ω ⟹ *x* ∈ ω ⟹ *f_fm* ∈ *formula* ⟹
arity(*separation_assm_fm*(*A*, *x*, *f_fm*)) = *succ*(*A*) ∪ *succ*(*x*) ∪ *pred*(*pred*(*arity*(*f_fm*)))
using *pred_Un_distrib*
unfolding *separation_assm_fm_def*
by (*auto simp add*:*arity*)

definition *separation_assm_bin_fm* **where**
separation_assm_bin_fm(*A*,*y*,*f_fm*) ≡
(·∃(·∃(·∃(·∃(·(··3 ∈ A +$_\omega$ 4· ∧ ·⟨3,2⟩ *is* y +$_\omega$ 4··) ∧ ·f_fm ∧ · ·fst(3) *is* 0 · ∧ ·snd(3) *is* 1····) ·)·)·)·)·)

lemma *separation_assm_bin_fm_type*[*TC*]:
A ∈ ω ⟹ *y* ∈ ω ⟹ *f_fm* ∈ *formula* ⟹ *separation_assm_bin_fm*(*A*, *y*,*f_fm*) ∈ *formula*
unfolding *separation_assm_bin_fm_def*
by *simp*

lemma *arity_separation_assm_bin_fm* : *A* ∈ ω ⟹ *x* ∈ ω ⟹ *f_fm* ∈ *formula* ⟹
arity(*separation_assm_bin_fm*(*A*, *x*, *f_fm*)) = *succ*(*A*) ∪ *succ*(*x*) ∪ (*pred*^4(*arity*(*f_fm*)))
using *pred_Un_distrib*
unfolding *separation_assm_bin_fm_def*
by (*auto simp add*:*arity*)

context *M_Z_trans*
begin

lemma *separation_assm_sats* :
assumes
f_fm: φ ∈ *formula* **and**
f_ar: *arity*(φ) = 2 **and**
fsats: ⋀*env x y*. *env*∈*list*(*M*) ⟹ *x*∈*M* ⟹ *y*∈*M* ⟹ (*M*,[*x*,*y*]@*env* ⊨ φ) ⟷ *is_f*(*x*,*y*) **and**
fabs: ⋀*x y*. *x*∈*M* ⟹ *y*∈*M* ⟹ *is_f*(*x*,*y*) ⟷ *y* = *f*(*x*) **and**
fclosed: ⋀*x*. *x*∈*M* ⟹ *f*(*x*) ∈ *M* **and**
A∈*M*
shows *separation*(##*M*, λ*y*. ∃ *x* ∈ *M* . *x*∈*A* ∧ *y* = ⟨*x*, *f*(*x*)⟩)
proof -
let *?φ'*=*separation_assm_fm*(1,0,φ)
let *?p*=λ*y*. ∃ *x*∈*M* . *x*∈*A* ∧ *y* = ⟨*x*, *f*(*x*)⟩

from *f_fm*
have *?φ′∈formula*
by *simp*
moreover from *this f_ar f_fm*
have *arity(?φ′) = 2*
using *arity_separation_assm_fm[of 1 0 φ] ord_simp_union*
by *simp*
moreover from ‹*A∈M*› *calculation*
have *separation(##M,λy . M,[y,A] ⊨ ?φ′)*
using *separation_ax* **by** *auto*
moreover
have *y∈M ⟹ (M,[y,A] ⊨ ?φ′) ⟷ ?p(y)* **for** *y*
using *assms transitivity[OF _ ‹A∈M›]*
unfolding *separation_assm_fm_def*
by *auto*
ultimately
show *?thesis*
by(*rule_tac separation_cong[THEN iffD1],auto*)
qed

lemma *separation_assm_bin_sats* :
assumes
f_fm: *φ ∈ formula* **and**
f_ar: *arity(φ) = 3* **and**
fsats: ⋀*env x z y. env∈list(M) ⟹ x∈M ⟹ z∈M ⟹ y∈M ⟹ (M,[x,z,y]@env ⊨ φ) ⟷ is_f(x,z,y)* **and**
fabs: ⋀*x z y. x∈M ⟹ z∈M ⟹ y∈M ⟹ is_f(x,z,y) ⟷ y = f(x,z)* **and**
fclosed: ⋀*x z . x∈M ⟹ z∈M ⟹ f(x,z) ∈ M* **and**
A∈M
shows *separation(##M, λy. ∃x ∈ M . x∈A ∧ y = ⟨x, f(fst(x),snd(x))⟩)*
proof -
let *?φ′=separation_assm_bin_fm(1,0,φ)*
let *?p=λy. ∃x∈M . x∈A ∧ y = ⟨x, f(fst(x),snd(x))⟩*
from *f_fm*
have *?φ′∈formula*
by *simp*
moreover from *this f_ar f_fm*
have *arity(?φ′) = 2*
using *arity_separation_assm_bin_fm[of 1 0 φ] ord_simp_union*
by *simp*
moreover from ‹*A∈M*› *calculation*
have *separation(##M,λy . M,[y,A] ⊨ ?φ′)*
using *separation_ax* **by** *auto*
moreover
have *y∈M ⟹ (M,[y,A] ⊨ ?φ′) ⟷ ?p(y)* **for** *y*
using *assms transitivity[OF _ ‹A∈M›] pair_in_M_iff fst_abs snd_abs fst_closed snd_closed*
unfolding *separation_assm_bin_fm_def*
by *auto*

ultimately
show *?thesis*
by(*rule_tac separation_cong*[*THEN iffD1*],*auto*)
qed

lemma *separation_Union*: *A*∈*M* ⟹
separation(##*M*, λ*y*. ∃ *x* ∈ *M* . *x*∈*A* ∧ *y* = ⟨*x*, *Union*(*x*)⟩)
using *separation_assm_sats*[*of big_union_fm*(*0*,*1*)] *arity_big_union_fm ord_simp_union*
Union_closed[*simplified*]
by *simp*

lemma *lam_replacement_Union*: *lam_replacement*(##*M*, *Union*)
using *lam_replacement_Union′ separation_Union transM* **by** *simp*

lemma *separation_fst*: *A*∈*M* ⟹
separation(##*M*, λ*y*. ∃ *x* ∈ *M* . *x*∈*A* ∧ *y* = ⟨*x*, *fst*(*x*)⟩)
using *separation_assm_sats*[*of fst_fm*(*0*,*1*)] *arity_fst_fm ord_simp_union*
fst_closed fst_abs
by *simp*

lemma *lam_replacement_fst*: *lam_replacement*(##*M*, *fst*)
using *lam_replacement_fst′ separation_fst transM* **by** *simp*

lemma *separation_snd*: *A*∈*M* ⟹
separation(##*M*, λ*y*. ∃ *x* ∈ *M* . *x*∈*A* ∧ *y* = ⟨*x*, *snd*(*x*)⟩)
using *separation_assm_sats*[*of snd_fm*(*0*,*1*)] *arity_snd_fm ord_simp_union*
snd_closed[*simplified*] *snd_abs*
by *simp*

lemma *lam_replacement_snd*: *lam_replacement*(##*M*, *snd*)
using *lam_replacement_snd′ separation_snd transM* **by** *simp*

Binary lambda-replacements

lemma *separation_Image*: *A*∈*M* ⟹
separation(##*M*, λ*y*. ∃ *x*∈*M*. *x* ∈ *A* ∧ *y* = ⟨*x*, *fst*(*x*) `` *snd*(*x*)⟩)
using *arity_image_fm ord_simp_union*
nonempty image_closed image_abs
by (*rule_tac separation_assm_bin_sats*[*of image_fm*(*0*,*1*,*2*)],*auto*)

lemma *lam_replacement_Image*: *lam_replacement*(##*M*, λ*x* . *fst*(*x*) `` *snd*(*x*))
using *lam_replacement_Image′ separation_Image*
by *simp*

lemma *separation_middle_del*: *A*∈*M* ⟹
separation(##*M*, λ*y*. ∃ *x*∈*M*. *x* ∈ *A* ∧ *y* = ⟨*x*, *middle_del*(*fst*(*x*), *snd*(*x*))⟩)
using *arity_is_middle_del_fm ord_simp_union nonempty*
fst_abs snd_abs fst_closed snd_closed pair_in_M_iff
by (*rule_tac separation_assm_bin_sats*[*of is_middle_del_fm*(*0*,*1*,*2*)],
auto simp:*is_middle_del_def middle_del_def*)

lemma *lam_replacement_middle_del*: *lam_replacement*(*##M*, *λr . middle_del*(*fst*(*r*),*snd*(*r*)))
using *lam_replacement_middle_del′ separation_middle_del*
by *simp*

lemma *separation_prodRepl*: *A∈M ⟹*
separation(*##M*, *λy*. *∃ x∈M*. *x ∈ A ∧ y = ⟨x*, *prodRepl*(*fst*(*x*), *snd*(*x*))⟩)
using *arity_is_prodRepl_fm ord_simp_union nonempty*
fst_abs snd_abs fst_closed snd_closed pair_in_M_iff
by (*rule_tac separation_assm_bin_sats*[*of is_prodRepl_fm*(*0*,*1*,*2*)],
auto simp:*is_prodRepl_def prodRepl_def*)

lemma *lam_replacement_prodRepl*: *lam_replacement*(*##M*, *λr . prodRepl*(*fst*(*r*),*snd*(*r*)))
using *lam_replacement_prodRepl′ separation_prodRepl*
by *simp*

end — *M_Z_trans*

context *M_trivial*
begin

lemma *first_closed*:
M(*B*) *⟹ M*(*r*) *⟹ first*(*u*,*r*,*B*) *⟹ M*(*u*)
using *transM*[*OF first_is_elem*] **by** *simp*

is_iff_rel for *first*
unfolding *is_first_def first_rel_def* **by** *auto*

is_iff_rel for *minimum*
unfolding *is_minimum_def minimum_rel_def*
using *is_first_iff The_abs nonempty*
by *force*

end — *M_trivial*

context *M_Z_trans*
begin

lemma (**in** *M_basic*) *is_minimum_equivalence* :
M(*R*) *⟹ M*(*X*) *⟹ M*(*u*) *⟹ is_minimum*(*M*,*R*,*X*,*u*) *⟷ is_minimum′*(*M*,*R*,*X*,*u*)
unfolding *is_minimum_def is_minimum′_def is_The_def is_first_def* **by**
simp

lemma *separation_minimum*: *A∈M ⟹*
separation(*##M*, *λy*. *∃ x∈M*. *x ∈ A ∧ y = ⟨x*, *minimum*(*fst*(*x*), *snd*(*x*))⟩)
using *arity_minimum_fm ord_simp_union is_minimum_iff minimum_abs*
is_minimum_equivalence nonempty minimum_closed minimum_abs
by (*rule_tac separation_assm_bin_sats*[*of minimum_fm*(*0*,*1*,*2*)], *auto*)

lemma *lam_replacement_minimum*: *lam_replacement*(*##M*, λx . *minimum*(*fst*(*x*),*snd*(*x*)))
using *lam_replacement_minimum'* *separation_minimum*
by *simp*

end — *M_Z_trans*

end

7.6 More Instances of Separation

theory *Separation_Instances*
imports
Interface
begin

The following instances are mostly the same repetitive task; and we just copied and pasted, tweaking some lemmas if needed (for example, we might have needed to use some closure results).

definition *radd_body* :: [*i*,*i*,*i*] ⇒ *o* **where**
radd_body(*R*,*S*) ≡ $\lambda z.$ ($\exists\, x\ y.\ z = \langle Inl(x),\ Inr(y)\rangle$) ∨
($\exists\, x'\ x.\ z = \langle Inl(x'),\ Inl(x)\rangle \wedge \langle x',\ x\rangle \in R$) ∨
($\exists\, y'\ y.\ z = \langle Inr(y'),\ Inr(y)\rangle \wedge \langle y',\ y\rangle \in S$)

relativize functional *radd_body* *radd_body_rel*
relationalize *radd_body_rel* *is_radd_body*

synthesize *is_radd_body* **from_definition**
arity_theorem for *is_radd_body_fm*

definition *rmult_body* :: [*i*,*i*,*i*] ⇒ *o* **where**
rmult_body(*b*,*d*) ≡ $\lambda z.\ \exists\, x'\ y'\ x\ y.\ z = \langle\langle x',\ y'\rangle,\ x,\ y\rangle \wedge (\langle x',\ x\rangle \in b \vee$
$x' = x \wedge \langle y',\ y\rangle \in d)$

relativize functional *rmult_body* *rmult_body_rel*
relationalize *rmult_body_rel* *is_rmult_body*

synthesize *is_rmult_body* **from_definition**
arity_theorem for *is_rmult_body_fm*

lemma (**in** *M_replacement*) *separation_well_ord_iso*:
(*M*)(*f*) ⟹ (*M*)(*r*) ⟹ (*M*)(*A*) ⟹ *separation*
(*M*, $\lambda x.\ x \in A \longrightarrow (\exists\, y[M].\ \exists\, p[M].$ *is_apply*(*M*, *f*, *x*, *y*) ∧ *pair*(*M*, *y*, *x*, *p*)
∧ *p* ∈ *r*))
using *separation_imp* *separation_in* *lam_replacement_identity* *lam_replacement_constant*
lam_replacement_apply[*of f*] *lam_replacement_product*
by *simp*

definition *is_obase_body* :: [*i*⇒*o*,*i*,*i*,*i*] ⇒ *o* **where**
is_obase_body(*N*,*A*,*r*,*x*) ≡ $x \in A \longrightarrow$

$$\neg\ (\exists\, y[N].\ \exists\, g[N].\ ordinal(N,\ y)\ \wedge\ (\exists\, my[N].\ \exists\, pxr[N].\ membership(N,\ y,\ my)\ \wedge\ pred\_set(N,\ A,\ x,\ r,\ pxr)\ \wedge\ order\_isomorphism(N,\ pxr,\ r,\ y,\ my,\ g)))$$

synthesize *is_obase_body* **from_definition**
arity_theorem for *is_obase_body_fm*

definition *is_obase_equals* :: $[i{\Rightarrow}o,i,i,i] \Rightarrow o$ **where**
$is\_obase\_equals(N,A,r,a) \equiv \exists\, x[N].\ \exists\, g[N].\ \exists\, mx[N].\ \exists\, par[N].\ ordinal(N,\ x)\ \wedge\ membership(N,\ x,\ mx)\ \wedge\ pred\_set(N,\ A,\ a,\ r,\ par)\ \wedge\ order\_isomorphism(N,\ par,\ r,\ x,\ mx,\ g)$

synthesize *is_obase_equals* **from_definition**
arity_theorem for *is_obase_equals_fm*

synthesize *PiP_rel* **from_definition assuming** *nonempty*
arity_theorem for *PiP_rel_fm*

synthesize *injP_rel* **from_definition assuming** *nonempty*
arity_theorem for *injP_rel_fm*

synthesize *surjP_rel* **from_definition assuming** *nonempty*
arity_theorem for *surjP_rel_fm*

context *M_ZF1_trans*
begin

lemma *radd_body_abs*:
assumes $(\#\#M)(R)$ $(\#\#M)(S)$ $(\#\#M)(x)$
shows $is\_radd\_body(\#\#M,R,S,x) \longleftrightarrow radd\_body(R,S,x)$
using *assms pair_in_M_iff Inl_in_M_iff Inr_in_M_iff*
unfolding *radd_body_def is_radd_body_def*
by (*auto*)

lemma *separation_radd_body*:
$(\#\#M)(R) \Longrightarrow (\#\#M)(S) \Longrightarrow separation\ (\#\#M,\ \lambda z.\ (\exists\, x\ y.\ z = \langle Inl(x),\ Inr(y)\rangle)\ \vee\ (\exists\, x'\ x.\ z = \langle Inl(x'),\ Inl(x)\rangle\ \wedge\ \langle x',\ x\rangle \in R)\ \vee$

$(\exists\, y'\ y.\ z = \langle Inr(y'),\ Inr(y)\rangle \wedge \langle y',\ y\rangle \in S))$
using *separation_in_ctm*[**where** φ=*is_radd_body_fm*(*1*,*2*,*0*) **and** *env*=[*R*,*S*]]
is_radd_body_def arity_is_radd_body_fm ord_simp_union is_radd_body_fm_type radd_body_abs
unfolding *radd_body_def*
by *simp*

lemma *rmult_body_abs*:
assumes $(\#\#M)(b)$ $(\#\#M)(d)$ $(\#\#M)(x)$
shows *is_rmult_body*($\#\#M$,*b*,*d*,*x*) $\longleftrightarrow$ *rmult_body*(*b*,*d*,*x*)
using *assms pair_in_M_iff apply_closed*
unfolding *rmult_body_def is_rmult_body_def*
by (*auto*)

lemma *separation_rmult_body*:
$(\#\#M)(b) \Longrightarrow (\#\#M)(d) \Longrightarrow$ *separation*
$(\#\#M,\ \lambda z.\ \exists\, x'\ y'\ x\ y.\ z = \langle\langle x',\ y'\rangle,\ x,\ y\rangle \wedge (\langle x',\ x\rangle \in b \vee x' = x \wedge \langle y',\ y\rangle \in d))$
using *separation_in_ctm*[**where** φ=*is_rmult_body_fm*(*1*,*2*,*0*) **and** *env*=[*b*,*d*]]
is_rmult_body_def arity_is_rmult_body_fm ord_simp_union is_rmult_body_fm_type rmult_body_abs
unfolding *rmult_body_def*
by *simp*

lemma *separation_is_obase*:
$(\#\#M)(f) \Longrightarrow (\#\#M)(r) \Longrightarrow (\#\#M)(A) \Longrightarrow$ *separation*
$(\#\#M,\ \lambda x.\ x \in A \longrightarrow$
$\neg\ (\exists\, y[\#\#M].$
$\exists\, g[\#\#M].$
ordinal($\#\#M$,\ *y*) $\wedge$
$(\exists\, my[\#\#M].$
$\exists\, pxr[\#\#M].$
membership($\#\#M$,\ *y*,\ *my*) $\wedge$
pred_set($\#\#M$,\ *A*,\ *x*,\ *r*,\ *pxr*) $\wedge$
order_isomorphism($\#\#M$,\ *pxr*,\ *r*,\ *y*,\ *my*,\ *g*))))
using *separation_in_ctm*[**where** φ=*is_obase_body_fm*(*1*,*2*,*0*) **and** *env*=[*A*,*r*]]
is_obase_body_def arity_is_obase_body_fm ord_simp_union is_obase_body_fm_type
by *simp*

lemma *separation_obase_equals*:
$(\#\#M)(f) \Longrightarrow (\#\#M)(r) \Longrightarrow (\#\#M)(A) \Longrightarrow$ *separation*
$(\#\#M,\ \lambda a.\ \exists\, x[\#\#M].$
$\exists\, g[\#\#M].$
$\exists\, mx[\#\#M].$
$\exists\, par[\#\#M].$
ordinal($\#\#M$,\ *x*) $\wedge$
membership($\#\#M$,\ *x*,\ *mx*) $\wedge$
pred_set($\#\#M$,\ *A*,\ *a*,\ *r*,\ *par*) $\wedge$ *order_isomorphism*($\#\#M$,\ *par*,\ *r*,\ *x*,\ *mx*,\ *g*))

using *separation_in_ctm*[**where** φ=*is_obase_equals_fm*(*1*,*2*,*0*) **and** *env*=[*A*,*r*]]
is_obase_equals_def arity_is_obase_equals_fm ord_simp_union is_obase_equals_fm_type
by *simp*

lemma *separation_PiP_rel*:
(##*M*)(*A*) $\Longrightarrow$ *separation*(##*M*, *PiP_rel*(##*M*,*A*))
using *separation_in_ctm*[**where** *env*=[*A*] **and** φ=*PiP_rel_fm*(*1*,*0*)]
nonempty PiP_rel_iff_sats[*symmetric*] *arity_PiP_rel_fm PiP_rel_fm_type*
by(*simp_all add*: *ord_simp_union*)

lemma *separation_injP_rel*:
(##*M*)(*A*) $\Longrightarrow$ *separation*(##*M*, *injP_rel*(##*M*,*A*))
using *separation_in_ctm*[**where** *env*=[*A*] **and** φ=*injP_rel_fm*(*1*,*0*)]
nonempty injP_rel_iff_sats[*symmetric*] *arity_injP_rel_fm injP_rel_fm_type*
by(*simp_all add*: *ord_simp_union*)

lemma *separation_surjP_rel*:
(##*M*)(*A*) $\Longrightarrow$ (##*M*)(*B*) $\Longrightarrow$ *separation*(##*M*, *surjP_rel*(##*M*,*A*,*B*))
using *separation_in_ctm*[**where** *env*=[*A*,*B*] **and** φ=*surjP_rel_fm*(*1*,*2*,*0*)]
nonempty surjP_rel_iff_sats[*symmetric*] *arity_surjP_rel_fm surjP_rel_fm_type*
by(*simp_all add*: *ord_simp_union*)

lemma *separation_is_function*:
separation(##*M*, *is_function*(##*M*))
using *separation_in_ctm*[**where** *env*=[] **and** φ=*function_fm*(*0*)] *arity_function_fm*
by *simp*

end — *M_ZF1_trans*

definition *fstsnd_in_sndsnd* :: [*i*] $\Rightarrow$ *o* **where**
fstsnd_in_sndsnd $\equiv \lambda x.$ *fst*(*snd*(*x*)) $\in$ *snd*(*snd*(*x*))
relativize *fstsnd_in_sndsnd is_fstsnd_in_sndsnd*
synthesize *is_fstsnd_in_sndsnd* **from_definition assuming** *nonempty*
arity_theorem for *is_fstsnd_in_sndsnd_fm*

definition *sndfst_eq_fstsnd* :: [*i*] $\Rightarrow$ *o* **where**
sndfst_eq_fstsnd $\equiv \lambda x.$ *snd*(*fst*(*x*)) = *fst*(*snd*(*x*))
relativize *sndfst_eq_fstsnd is_sndfst_eq_fstsnd*
synthesize *is_sndfst_eq_fstsnd* **from_definition assuming** *nonempty*
arity_theorem for *is_sndfst_eq_fstsnd_fm*

context *M_ZF1_trans*
begin

lemma *fstsnd_in_sndsnd_abs*:
assumes (##*M*)(*x*)
shows *is_fstsnd_in_sndsnd*(##*M*,*x*) $\longleftrightarrow$ *fstsnd_in_sndsnd*(*x*)

using *assms pair_in_M_iff fst_abs snd_abs fst_snd_closed*
unfolding *fstsnd_in_sndsnd_def is_fstsnd_in_sndsnd_def*
by *auto*

lemma *separation_fstsnd_in_sndsnd*:
separation(##*M*, λ*x*. *fst*(*snd*(*x*)) ∈ *snd*(*snd*(*x*)))
using *separation_in_ctm*[**where** *env*=[] **and** φ=*is_fstsnd_in_sndsnd_fm*(*0*) **and** *Q*=*fstsnd_in_sndsnd*]
nonempty fstsnd_in_sndsnd_abs arity_is_fstsnd_in_sndsnd_fm
unfolding *fstsnd_in_sndsnd_def*
by *simp*

lemma *sndfst_eq_fstsnd_abs*:
assumes (##*M*)(*x*)
shows *is_sndfst_eq_fstsnd*(##*M*,*x*) ⟷ *sndfst_eq_fstsnd*(*x*)
using *assms pair_in_M_iff fst_abs snd_abs fst_snd_closed*
unfolding *sndfst_eq_fstsnd_def is_sndfst_eq_fstsnd_def*
by *auto*

lemma *separation_sndfst_eq_fstsnd*:
separation(##*M*, λ*x*. *snd*(*fst*(*x*)) = *fst*(*snd*(*x*)))
using *separation_in_ctm*[**where** *env*=[] **and** φ=*is_sndfst_eq_fstsnd_fm*(*0*) **and** *Q*=*sndfst_eq_fstsnd*]
nonempty sndfst_eq_fstsnd_abs arity_is_sndfst_eq_fstsnd_fm
unfolding *sndfst_eq_fstsnd_def*
by *simp*

end — *M_ZF1_trans*

end

8 More Instances of Replacement

theory *Replacement_Instances*
imports
Separation_Instances
Transitive_Models.Pointed_DC_Relative
begin

lemma *composition_fm_type*[*TC*]: *a0* ∈ ω ⟹ *a1* ∈ ω ⟹ *a2* ∈ ω ⟹
composition_fm(*a0*,*a1*,*a2*) ∈ *formula*
unfolding *composition_fm_def* **by** *simp*

arity_theorem for *composition_fm*

definition *is_omega_funspace* :: [*i*⇒*o*,*i*,*i*,*i*]⇒*o* **where**
is_omega_funspace(*N*,*B*,*n*,*z*) ≡ ∃ *o*[*N*]. *omega*(*N*,*o*) ∧ *n*∈*o* ∧ *is_funspace*(*N*, *n*, *B*, *z*)

synthesize *omega_funspace* **from_definition** *is_omega_funspace* **assuming** *nonempty*
arity_theorem for *omega_funspace_fm*

definition *HAleph_wfrec_repl_body* **where**
HAleph_wfrec_repl_body(*N*,*mesa*,*x*,*z*) ≡ ∃ *y*[*N*].
pair(*N*, *x*, *y*, *z*) ∧
(∃ *g*[*N*].
(∀ *u*[*N*].
u ∈ *g* ⟷
(∃ *a*[*N*].
∃ *y*[*N*].
∃ *ax*[*N*].
∃ *sx*[*N*].
∃ *r_sx*[*N*].
∃ *f_r_sx*[*N*].
pair(*N*, *a*, *y*, *u*) ∧
pair(*N*, *a*, *x*, *ax*) ∧
upair(*N*, *a*, *a*, *sx*) ∧
pre_image(*N*, *mesa*, *sx*, *r_sx*) ∧
restriction(*N*, *g*, *r_sx*, *f_r_sx*) ∧ *ax* ∈ *mesa* ∧ *is_HAleph*(*N*, *a*, *f_r_sx*, *y*)))
∧
is_HAleph(*N*, *x*, *g*, *y*))

arity_theorem for *ordinal_fm*
arity_theorem for *is_Limit_fm*
arity_theorem for *empty_fm*
arity_theorem for *fun_apply_fm*

synthesize *HAleph_wfrec_repl_body* **from_definition assuming** *nonempty*
arity_theorem for *HAleph_wfrec_repl_body_fm*

definition *dcwit_repl_body* **where**
dcwit_repl_body(*N*,*mesa*,*A*,*a*,*s*,*R*) ≡ λ*x z*. ∃ *y*[*N*]. *pair*(*N*, *x*, *y*, *z*) ∧
is_wfrec
(*N*, λ*n f*. *is_nat_case*
(*N*, *a*,
λ*m bmfm*.
∃ *fm*[*N*].
∃ *cp*[*N*].
is_apply(*N*, *f*, *m*, *fm*) ∧
is_Collect(*N*, *A*, λ*x*. ∃ *fmx*[*N*]. (*N*(*x*)
∧ *fmx* ∈ *R*) ∧ *pair*(*N*, *fm*, *x*, *fmx*), *cp*) ∧
is_apply(*N*, *s*, *cp*, *bmfm*),
n),
mesa, *x*, *y*)

manual_schematic for *dcwit_repl_body* **assuming** *nonempty*
unfolding *dcwit_repl_body_def*

by (*rule iff_sats is_nat_case_iff_sats is_eclose_iff_sats sep_rules* | *simp*)+

synthesize *dcwit_repl_body* **from_schematic**

definition *dcwit_aux_fm* **where**
dcwit_aux_fm(*A*,*s*,*R*) ≡ (·∃··4‘2 *is* 0· ∧
(·∃·*Collect_fm*
(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*A*))))))))))),
(·∃··0 ∈
succ(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*R*))))))))))))
· ∧
pair_fm(3, 1, 0) ··),
0) ∧
· *succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*succ*(*s*))))))))))‘0 *is*
2···)··)

arity_theorem for *dcwit_aux_fm*

lemma *dcwit_aux_fm_type*[*TC*]: *A* ∈ ω ⟹ *s* ∈ ω ⟹ *R* ∈ ω ⟹ *dcwit_aux_fm*(*A*,*s*,*R*)
∈ *formula*
by (*simp_all add*: *dcwit_aux_fm_def*)

definition *is_nat_case_dcwit_aux_fm* **where**
is_nat_case_dcwit_aux_fm(*A*,*a*,*s*,*R*) ≡ *is_nat_case_fm*
(*succ*(*succ*(*succ*(*succ*(*succ*(*a*))))))*dcwit_aux_fm*(*A*,*s*,*R*),
2, 0)

lemma *is_nat_case_dcwit_aux_fm_type*[*TC*]: *A* ∈ ω ⟹ *a* ∈ ω ⟹ *s* ∈ ω ⟹
R ∈ ω ⟹ *is_nat_case_dcwit_aux_fm*(*A*,*a*,*s*,*R*) ∈ *formula*
by (*simp_all add*: *is_nat_case_dcwit_aux_fm_def*)

manual_arity for *is_nat_case_dcwit_aux_fm*
unfolding *is_nat_case_dcwit_aux_fm_def*
by (*rule arity_dcwit_aux_fm*[*THEN* [6] *arity_is_nat_case_fm*]) *simp_all*

manual_arity for *dcwit_repl_body_fm*
using *arity_is_nat_case_dcwit_aux_fm*[*THEN* [6] *arity_is_wfrec_fm*]
unfolding *dcwit_repl_body_fm_def is_nat_case_dcwit_aux_fm_def dcwit_aux_fm_def*
by (*auto simp add*: *arity*(1-33))

lemma *arity_dcwit_repl_body*: *arity*(*dcwit_repl_body_fm*(6,5,4,3,2,0,1)) = 7
by (*simp_all add*: *FOL_arities arity_dcwit_repl_body_fm ord_simp_union*)

definition *fst2_snd2*
where *fst2_snd2*(*x*) ≡ ⟨*fst*(*fst*(*x*)), *snd*(*snd*(*x*))⟩

relativize functional *fst2_snd2 fst2_snd2_rel*
relationalize *fst2_snd2_rel is_fst2_snd2*

lemma (**in** *M_trivial*) *fst2_snd2_abs*:
  assumes $M(x)$ $M(res)$
  shows $is\_fst2\_snd2(M,\ x,\ res) \longleftrightarrow res = fst2\_snd2(x)$
  unfolding *is_fst2_snd2_def fst2_snd2_def*
  using *fst_rel_abs snd_rel_abs fst_abs snd_abs assms*
  by *simp*

synthesize *is_fst2_snd2* **from_definition assuming** *nonempty*
arity_theorem for *is_fst2_snd2_fm*

definition *sndfst_fst2_snd2*
  where $sndfst\_fst2\_snd2(x) \equiv \langle snd(fst(x)),\ fst(fst(x)),\ snd(snd(x))\rangle$

relativize functional *sndfst_fst2_snd2 sndfst_fst2_snd2_rel*
relationalize *sndfst_fst2_snd2_rel is_sndfst_fst2_snd2*
synthesize *is_sndfst_fst2_snd2* **from_definition assuming** *nonempty*
arity_theorem for *is_sndfst_fst2_snd2_fm*

definition *order_eq_map* **where**
  $order\_eq\_map(M,A,r,a,z) \equiv \exists\, x[M].\ \exists\, g[M].\ \exists\, mx[M].\ \exists\, par[M].$
  $ordinal(M,x)\ \&\ pair(M,a,x,z)\ \&\ membership(M,x,mx)\ \&$
  $pred\_set(M,A,a,r,par)\ \&\ order\_isomorphism(M,par,r,x,mx,g)$

synthesize *order_eq_map* **from_definition assuming** *nonempty*
arity_theorem for *is_ord_iso_fm*
arity_theorem for *order_eq_map_fm*

synthesize *is_banach_functor* **from_definition assuming** *nonempty*
arity_theorem for *is_banach_functor_fm*

definition *banach_body_iterates* **where**
  $banach\_body\_iterates(M,X,Y,f,g,W,n,x,z) \equiv$
$\exists\, y[M].$
$pair(M,\ x,\ y,\ z)\ \wedge$
$(\exists\, fa[M].$
$(\forall\, z[M].$
$z \in fa \longleftrightarrow$
$(\exists\, xa[M].$
$\exists\, y[M].$
$\exists\, xaa[M].$
$\exists\, sx[M].$
$\exists\, r\_sx[M].$
$\exists\, f\_r\_sx[M].\ \exists\, sn[M].\ \exists\, msn[M].\ successor(M,n,sn)$
$\wedge$
$membership(M,sn,msn)\ \wedge$
$pair(M,\ xa,\ y,\ z)\ \wedge$
$pair(M,\ xa,\ x,\ xaa)\ \wedge$
$upair(M,\ xa,\ xa,\ sx)\ \wedge$

$pre\_image(M, msn, sx, r\_sx) \wedge$
$restriction(M, fa, r\_sx, f\_r\_sx) \wedge$
$xaa \in msn \wedge$
$(empty(M, xa) \longrightarrow y = W) \wedge$
$(\forall m[M].$
$successor(M, m, xa) \longrightarrow$
$(\exists gm[M].$
$is\_apply(M, f\_r\_sx, m, gm) \wedge$
$is\_banach\_functor(M, X, Y, f, g, gm, y))) \wedge$
$(is\_quasinat(M, xa) \vee empty(M, y)))) \wedge$
$(empty(M, x) \longrightarrow y = W) \wedge$
$(\forall m[M].$
$successor(M, m, x) \longrightarrow$
$(\exists gm[M]. is\_apply(M, fa, m, gm) \wedge is\_banach\_functor(M, X, Y, f, g, gm, y))) \wedge$
$(is\_quasinat(M, x) \vee empty(M, y)))$

synthesize *is_quasinat* **from_definition assuming** *nonempty*
arity_theorem for *is_quasinat_fm*

synthesize *banach_body_iterates* **from_definition assuming** *nonempty*
arity_theorem for *banach_body_iterates_fm*

definition *banach_is_iterates_body* **where**
$banach\_is\_iterates\_body(M,X,Y,f,g,W,n,y) \equiv \exists om[M]. omega(M,om) \wedge n \in om \wedge$
$(\exists sn[M].$
$\exists msn[M].$
$successor(M, n, sn) \wedge$
$membership(M, sn, msn) \wedge$
$(\exists fa[M].$
$(\forall z[M].$
$z \in fa \longleftrightarrow$
$(\exists x[M].$
$\exists y[M].$
$\exists xa[M].$
$\exists sx[M].$
$\exists r\_sx[M].$
$\exists f\_r\_sx[M].$
$pair(M, x, y, z) \wedge$
$pair(M, x, n, xa) \wedge$
$upair(M, x, x, sx) \wedge$
$pre\_image(M, msn, sx, r\_sx) \wedge$
$restriction(M, fa, r\_sx, f\_r\_sx) \wedge$
$xa \in msn \wedge$
$(empty(M, x) \longrightarrow y = W) \wedge$
$(\forall m[M].$
$successor(M, m, x) \longrightarrow$
$(\exists gm[M].$

$$fun\_apply(M,\ f\_r\_sx,\ m,\ gm) \wedge$$
$$is\_banach\_functor(M,\ X,\ Y,\ f,\ g,\ gm,\ y))) \wedge$$
$$(is\_quasinat(M,\ x) \vee empty(M,\ y)))) \wedge$$
$$(empty(M,\ n) \longrightarrow y = W) \wedge$$
$$(\forall\, m[M].$$
$$successor(M,\ m,\ n) \longrightarrow$$
$$(\exists\, gm[M].\ fun\_apply(M,\ fa,\ m,\ gm) \wedge is\_banach\_functor(M,$$
$$X,\ Y,\ f,\ g,\ gm,\ y))) \wedge$$
$$(is\_quasinat(M,\ n) \vee empty(M,\ y))))$$

synthesize *banach_is_iterates_body* **from_definition assuming** *nonempty*
arity_theorem for *banach_is_iterates_body_fm*

definition *trans_apply_image* **where**
trans_apply_image$(f) \equiv \lambda a\ g.\ f\ `\ (g\ ``\ a)$

relativize functional *trans_apply_image trans_apply_image_rel*
relationalize *trans_apply_image is_trans_apply_image*

schematic_goal *arity_is_recfun_fm*[*arity*]:
$p \in formula \Longrightarrow a \in \omega \Longrightarrow z \in \omega \Longrightarrow r \in \omega \Longrightarrow arity(is\_recfun\_fm(p,\ a,\ z\ ,r)) = ?ar$
unfolding *is_recfun_fm_def*
by (*simp add*:*arity*)

schematic_goal *arity_is_wfrec_fm*[*arity*]:
$p \in formula \Longrightarrow a \in \omega \Longrightarrow z \in \omega \Longrightarrow r \in \omega \Longrightarrow arity(is\_wfrec\_fm(p,\ a,\ z\ ,r)) = ?ar$
unfolding *is_wfrec_fm_def*
by (*simp add*:*arity*)
schematic_goal *arity_is_transrec_fm*[*arity*]:
$p \in formula \Longrightarrow a \in \omega \Longrightarrow z \in \omega \Longrightarrow arity(is\_transrec\_fm(p,\ a,\ z)) = ?ar$
unfolding *is_transrec_fm_def*
by (*simp add*:*arity*)

synthesize *is_trans_apply_image* **from_definition assuming** *nonempty*
arity_theorem for *is_trans_apply_image_fm*

definition *transrec_apply_image_body* **where**
$transrec\_apply\_image\_body(M,f,mesa,x,z) \equiv \exists\, y[M].\ pair(M,\ x,\ y,\ z) \wedge$
$(\exists\, fa[M].$
$(\forall\, z[M].$
$z \in fa \longleftrightarrow$
$(\exists\, xa[M].$
$\exists\, y[M].$

$\exists\, xaa[M].$
$\exists\, sx[M].$
$\exists\, r\_sx[M].$
$\exists\, f\_r\_sx[M].$
$pair(M,\ xa,\ y,\ z)\ \wedge$
$pair(M,\ xa,\ x,\ xaa)\ \wedge$
$upair(M,\ xa,\ xa,\ sx)\ \wedge$
$pre\_image(M,\ mesa,\ sx,\ r\_sx)\ \wedge$
$restriction(M,\ fa,\ r\_sx,\ f\_r\_sx)\ \wedge$
$xaa \in mesa \wedge is\_trans\_apply\_image(M,$
$f,\ xa,\ f\_r\_sx,\ y))) \wedge$
$is\_trans\_apply\_image(M,\ f,\ x,\ fa,\ y))$

synthesize *transrec_apply_image_body* **from_definition assuming** *nonempty*
arity_theorem for *transrec_apply_image_body_fm*

definition *is_trans_apply_image_body* **where**
$is\_trans\_apply\_image\_body(M,f,\beta,a,w) \equiv \exists\, z[M].\ pair(M,a,z,w) \wedge a\in\beta \wedge (\exists\, sa[M].$
$\exists\, esa[M].$
$\exists\, mesa[M].$
$upair(M,\ a,\ a,\ sa)\ \wedge$
$is\_eclose(M,\ sa,\ esa)\ \wedge$
$membership(M,\ esa,\ mesa)\ \wedge$
$(\exists\, fa[M].$
$(\forall\, z[M].$
$z \in fa \longleftrightarrow$
$(\exists\, x[M].$
$\exists\, y[M].$
$\exists\, xa[M].$
$\exists\, sx[M].$
$\exists\, r\_sx[M].$
$\exists\, f\_r\_sx[M].$
$pair(M,\ x,\ y,\ z)\ \wedge$
$pair(M,\ x,\ a,\ xa)\ \wedge$
$upair(M,\ x,\ x,\ sx)\ \wedge$
$pre\_image(M,\ mesa,\ sx,\ r\_sx)\ \wedge$
$restriction(M,\ fa,\ r\_sx,\ f\_r\_sx)\ \wedge$
$xa \in mesa \wedge is\_trans\_apply\_image(M,\ f,$
$x,\ f\_r\_sx,\ y))) \wedge$
$is\_trans\_apply\_image(M,\ f,\ a,\ fa,\ z)))$

synthesize *is_trans_apply_image_body* **from_definition assuming** *nonempty*
arity_theorem for *is_trans_apply_image_body_fm*

definition *replacement_is_omega_funspace_fm* **where** *replacement_is_omega_funspace_fm* $\equiv$ *omega_funspace_fm*(2,0,1)
definition *wfrec_Aleph_fm* **where** *wfrec_Aleph_fm* $\equiv$ *HAleph_wfrec_repl_body_fm*(2,0,1)
definition *replacement_is_fst2_snd2_fm* **where** *replacement_is_fst2_snd2_fm*

≡ *is_fst2_snd2_fm*(*0*,*1*)
definition *replacement_is_sndfst_fst2_snd2_fm* **where** *replacement_is_sndfst_fst2_snd2_fm*
≡ *is_sndfst_fst2_snd2_fm*(*0*,*1*)
definition *omap_replacement_fm* **where** *omap_replacement_fm* ≡ *order_eq_map_fm*(*2*,*3*,*0*,*1*)
definition *rec_constr_abs_fm* **where** *rec_constr_abs_fm* ≡ *transrec_apply_image_body_fm*(*3*,*2*,*0*,*1*)
definition *banach_replacement_iterates_fm* **where** *banach_replacement_iterates_fm*
≡ *banach_is_iterates_body_fm*(*6*,*5*,*4*,*3*,*2*,*0*,*1*)
definition *rec_constr_fm* **where** *rec_constr_fm* ≡ *is_trans_apply_image_body_fm*(*3*,*2*,*0*,*1*)

definition *dc_abs_fm* **where** *dc_abs_fm* ≡ *dcwit_repl_body_fm*(*6*,*5*,*4*,*3*,*2*,*0*,*1*)
definition *lam_replacement_check_fm* **where** *lam_replacement_check_fm* ≡ *Lambda_in_M_fm*(*check_fm*(

The following instances are needed only on the ground model. The first one corresponds to the recursive definition of forces for atomic formulas; the next two corresponds to *PHcheck*; the following is used to get a generic filter using some form of choice.

locale *M_ZF_ground* = *M_ZF1* +
  assumes
    ZF_ground_replacements:
    replacement_assm(*M*,*env*,*wfrec_Hfrc_at_fm*)
    replacement_assm(*M*,*env*,*wfrec_Hcheck_fm*)
    replacement_assm(*M*,*env*,*lam_replacement_check_fm*)

locale *M_ZF_ground_trans* = *M_ZF1_trans* + *M_ZF_ground*

definition *instances_ground_fms* **where** *instances_ground_fms* ≡
  { *wfrec_Hfrc_at_fm*,
    wfrec_Hcheck_fm,
    lam_replacement_check_fm }

lemmas *replacement_instances_ground_defs* =
  wfrec_Hfrc_at_fm_def wfrec_Hcheck_fm_def lam_replacement_check_fm_def

declare (**in** *M_ZF_ground*) *replacement_instances_ground_defs* [*simp*]

lemma *instances_ground_fms_type*[*TC*]: *instances_ground_fms* ⊆ *formula*
  using *Lambda_in_M_fm_type*
  unfolding *instances_ground_fms_def replacement_instances_ground_defs*
  by *simp*

locale *M_ZF_ground_notCH* = *M_ZF_ground* +
  assumes
    ZF_ground_notCH_replacements:
    replacement_assm(*M*,*env*,*rec_constr_abs_fm*)
    replacement_assm(*M*,*env*,*rec_constr_fm*)

definition *instances_ground_notCH_fms* **where** *instances_ground_notCH_fms*
≡
  { *rec_constr_abs_fm*,

rec_constr_fm }

lemma *instances_ground_notCH_fms_type*[*TC*]: *instances_ground_notCH_fms* ⊆ *formula*
unfolding *instances_ground_notCH_fms_def rec_constr_abs_fm_def rec_constr_fm_def*
by *simp*

declare (**in** *M_ZF_ground_notCH*) *rec_constr_abs_fm_def*[*simp*] *rec_constr_fm_def*[*simp*]

locale *M_ZF_ground_notCH_trans* = *M_ZF_ground_trans* + *M_ZF_ground_notCH*

locale *M_ZF_ground_CH* = *M_ZF_ground_notCH* +
assumes
dcwit_replacement: *replacement_assm*(*M*,*env*,*dc_abs_fm*)

declare (**in** *M_ZF_ground_CH*) *dc_abs_fm_def* [*simp*]

locale *M_ZF_ground_CH_trans* = *M_ZF_ground_notCH_trans* + *M_ZF_ground_CH*

locale *M_ctm1* = *M_ZF1_trans* + *M_ZF_ground_trans* +
fixes *enum*
assumes *M_countable*: *enum*∈*bij*(*nat*,*M*)

locale *M_ctm1_AC* = *M_ctm1* + *M_ZFC1_trans*

context *M_ZF_ground_CH_trans*
begin

lemma *replacement_dcwit_repl_body*:
(##*M*)(*mesa*) ⟹ (##*M*)(*A*) ⟹ (##*M*)(*a*) ⟹ (##*M*)(*s*) ⟹ (##*M*)(*R*) ⟹
strong_replacement(##*M*, *dcwit_repl_body*(##*M*,*mesa*,*A*,*a*,*s*,*R*))
using *strong_replacement_rel_in_ctm*[**where** φ=*dcwit_repl_body_fm*(*6*,*5*,*4*,*3*,*2*,*0*,*1*)
and *env*=[*R*,*s*,*a*,*A*,*mesa*] **and** *f*=*dcwit_repl_body*(##*M*,*mesa*,*A*,*a*,*s*,*R*)]
zero_in_M arity_dcwit_repl_body dcwit_replacement
unfolding *dc_abs_fm_def*
by *simp*

lemma *dcwit_repl*:
(##*M*)(*sa*) ⟹
(##*M*)(*esa*) ⟹
(##*M*)(*mesa*) ⟹ (##*M*)(*A*) ⟹ (##*M*)(*a*) ⟹ (##*M*)(*s*) ⟹ (##*M*)(*R*) ⟹
strong_replacement
((##*M*), λ*x z*. ∃*y*[(##*M*)]. *pair*((##*M*), *x*, *y*, *z*) ∧
is_wfrec
((##*M*), λ*n f*. *is_nat_case*

$$\begin{array}{l}((\#\#M),\ a,\\ \lambda m\ bmfm.\\ \exists fm[(\#\#M)].\\ \exists cp[(\#\#M)].\\ is\_apply((\#\#M),\ f,\ m,\ fm)\ \wedge\\ is\_Collect((\#\#M),\ A,\ \lambda x.\ \exists fmx[(\#\#M)].\\ ((\#\#M)(x)\ \wedge\ fmx \in R)\ \wedge\ pair((\#\#M),\ fm,\ x,\ fmx),\ cp)\ \wedge\\ is\_apply((\#\#M),\ s,\ cp,\ bmfm),\\ n),\\ mesa,\ x,\ y))\end{array}$$

using *replacement_dcwit_repl_body* **unfolding** *dcwit_repl_body_def* **by** *simp*

end — *M_ZF_ground_CH_trans*

context *M_ZF1_trans*
begin

lemmas *M_replacement_ZF_instances = lam_replacement_fst lam_replacement_snd*
lam_replacement_Union lam_replacement_Image
lam_replacement_middle_del lam_replacement_prodRepl

lemmas *M_separation_ZF_instances = separation_fstsnd_in_sndsnd separation_sndfst_eq_fstsnd*

lemma *separation_is_dcwit_body*:
assumes $(\#\#M)(A)$ $(\#\#M)(a)$ $(\#\#M)(g)$ $(\#\#M)(R)$
shows *separation*$(\#\#M, is\_dcwit\_body(\#\#M,\ A,\ a,\ g,\ R))$
using *assms separation_in_ctm*[**where** *env*=[*A*,*a*,*g*,*R*] **and** φ=*is_dcwit_body_fm*(*1*,*2*,*3*,*4*,*0*),
OF _ _ _ is_dcwit_body_iff_sats[*symmetric*],
of $\lambda\_.A$ $\lambda\_.a$ $\lambda\_.g$ $\lambda\_.R$ $\lambda x.\ x$]
nonempty arity_is_dcwit_body_fm is_dcwit_body_fm_type
by (*simp add*:*ord_simp_union*)

end — *M_ZF1_trans*

sublocale *M_ZF1_trans* $\subseteq$ *M_replacement* $\#\#M$
using *M_replacement_ZF_instances M_separation_ZF_instances*
by *unfold_locales simp*

context *M_ZF1_trans*
begin

lemma *separation_Pow_rel*: $A \in M \Longrightarrow$
separation$(\#\#M,\ \lambda y.\ \exists x \in M\ .\ x \in A\ \wedge\ y = \langle x,\ Pow^{\#\#M}(x)\rangle)$
using *separation_assm_sats*[*of is_Pow_fm*(*0*,*1*)] *arity_is_Pow_fm ord_simp_union*
Pow_rel_closed nonempty Pow_rel_iff
by *simp*

lemma *strong_replacement_Powapply_rel*:
$f \in M \Longrightarrow$ *strong_replacement*$(\#\#M,\ \lambda x\ y.\ y = Powapply^{\#\#M}(f,x))$

using *Powapply_rel_replacement separation_Pow_rel transM*
by *simp*

end — *M_ZF1_trans*

sublocale *M_ZF1_trans* ⊆ *M_Vfrom ##M*
using *power_ax strong_replacement_Powapply_rel phrank_repl trans_repl_HVFrom wfrec_rank*
by *unfold_locales auto*

sublocale *M_ZF1_trans* ⊆ *M_Perm ##M*
using *separation_PiP_rel separation_injP_rel separation_surjP_rel*
lam_replacement_imp_strong_replacement[*OF*
lam_replacement_Sigfun[*OF lam_replacement_constant*]]
Pi_replacement1 **unfolding** *Sigfun_def*
by *unfold_locales simp_all*

sublocale *M_ZF1_trans* ⊆ *M_pre_seqspace ##M*
by *unfold_locales*

context *M_ZF1_trans*
begin

lemma *separation_inj_rel*: *A*∈*M* ⟹
separation(*##M*, λ*y*. ∃ *x*∈*M*. *x* ∈ *A* ∧ *y* = ⟨*x*, *inj_rel*(*##M*,*fst*(*x*), *snd*(*x*))⟩)
using *arity_is_inj_fm ord_simp_union*
nonempty inj_rel_closed[*simplified*] *inj_rel_iff*[*simplified*]
by (*rule_tac separation_assm_bin_sats*[*of is_inj_fm*(*0*,*1*,*2*)])
(*simp_all add*:*setclass_def*)

lemma *lam_replacement_inj_rel*: *lam_replacement*(*##M*, λ*x* . *inj_rel*(*##M*,*fst*(*x*),*snd*(*x*)))
using *lam_replacement_inj_rel′ separation_inj_rel*
by *simp*

end — *M_ZF1_trans*

lemma (**in** *M_basic*) *rel2_trans_apply*:
M(*f*) ⟹ *relation2*(*M*,*is_trans_apply_image*(*M*,*f*),*trans_apply_image*(*f*))
unfolding *is_trans_apply_image_def trans_apply_image_def relation2_def*
by *auto*

lemma (**in** *M_basic*) *apply_image_closed*:
shows *M*(*f*) ⟹ ∀ *x*[*M*]. ∀ *g*[*M*]. *M*(*trans_apply_image*(*f*, *x*, *g*))
unfolding *trans_apply_image_def* **by** *simp*

context *M_ZF_ground_notCH_trans*
begin

lemma *replacement_transrec_apply_image_body* :
(##M)(f) ⟹ (##M)(mesa) ⟹ *strong_replacement*(##M,*transrec_apply_image_body*(##M,f,mesa))
using *strong_replacement_rel_in_ctm*[**where** φ=*transrec_apply_image_body_fm*(3,2,0,1)
and *env*=[*mesa*,*f*]]
zero_in_M arity_transrec_apply_image_body_fm ord_simp_union
ZF_ground_notCH_replacements(1)
by *simp*

lemma *transrec_replacement_apply_image*:
assumes (##M)(f) (##M)(α)
shows *transrec_replacement*(##M, *is_trans_apply_image*(##M, f), α)
using *replacement_transrec_apply_image_body*[*unfolded transrec_apply_image_body_def*]
assms
Memrel_closed singleton_closed eclose_closed
unfolding *transrec_replacement_def wfrec_replacement_def is_wfrec_def M_is_recfun_def*
by *simp*

lemma *rec_trans_apply_image_abs*:
assumes (##M)(f) (##M)(x) (##M)(y) *Ord*(x)
shows *is_transrec*(##M,*is_trans_apply_image*(##M, f),x,y) ⟷ y = *transrec*(x,*trans_apply_image*(f))
using *transrec_abs*[*OF transrec_replacement_apply_image rel2_trans_apply*]
assms apply_image_closed
by *simp*

lemma *replacement_is_trans_apply_image*:
(##M)(f) ⟹ (##M)(β) ⟹ *strong_replacement*(##M, λ x z .
∃ y[##M]. *pair*(##M,x,y,z) ∧ x∈β ∧ (*is_transrec*(##M,*is_trans_apply_image*(##M, f),x,y)))
unfolding *is_transrec_def is_wfrec_def M_is_recfun_def*
apply(*rule_tac strong_replacement_cong*[
where P=λ x z. M,[x,z,β,f] ⊨ *is_trans_apply_image_body_fm*(3,2,0,1),*THEN iffD1*])
apply(*rule_tac is_trans_apply_image_body_iff_sats*[*symmetric*,*unfolded is_trans_apply_image_body_def*
env=[_,_,β,f]])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac ZF_ground_notCH_replacements*(2)[*unfolded replacement_assm_def*, *rule_format*, **where** *env*=[β,f],*simplified*])
apply(*simp_all add*: *arity_is_trans_apply_image_body_fm is_trans_apply_image_body_fm_type ord_simp_union*)
done

lemma *trans_apply_abs*:
(##M)(f) ⟹ (##M)(β) ⟹ *Ord*(β) ⟹ (##M)(x) ⟹ (##M)(z) ⟹
(x∈β ∧ z = ⟨x, *transrec*(x, λa g. f ‘ (g `` a)) ⟩) ⟷
(∃ y[##M]. *pair*(##M,x,y,z) ∧ x∈β ∧ (*is_transrec*(##M,*is_trans_apply_image*(##M, f),x,y)))
using *rec_trans_apply_image_abs Ord_in_Ord*

transrec_closed[*OF transrec_replacement_apply_image rel2_trans_apply,of f,simplified*]
apply_image_closed
unfolding *trans_apply_image_def*
by *auto*

lemma *replacement_trans_apply_image*:
$(\#\#M)(f) \Longrightarrow (\#\#M)(\beta) \Longrightarrow Ord(\beta) \Longrightarrow$
strong_replacement$(\#\#M, \lambda x\ y.\ x{\in}\beta \wedge y = \langle x, transrec(x, \lambda a\ g.\ f\ `\ (g\ ``\ a))\rangle)$
using *strong_replacement_cong*[*THEN iffD1,OF _ replacement_is_trans_apply_image,simplified*]
trans_apply_abs Ord_in_Ord
by *simp*

end — *M_ZF_ground_notCH_trans*

definition *ifrFb_body* **where**
ifrFb_body$(M,b,f,x,i) \equiv x \in$
(*if* $b = 0$ *then if* $i \in range(f)$ *then*
if $M(converse(f)\ `\ i)$ *then* $converse(f)\ `\ i$ *else 0 else 0 else if* $M(i)$ *then* i *else 0*)

relativize functional *ifrFb_body ifrFb_body_rel*
relationalize *ifrFb_body_rel is_ifrFb_body*

synthesize *is_ifrFb_body* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrFb_body_fm*

definition *ifrangeF_body* :: $[i{\Rightarrow}o,i,i,i,i] \Rightarrow o$ **where**
ifrangeF_body$(M,A,b,f) \equiv \lambda y.\ \exists\, x{\in}A.\ y = \langle x,\mu\ i.\ ifrFb\_body(M,b,f,x,i)\rangle$

relativize functional *ifrangeF_body ifrangeF_body_rel*
relationalize *ifrangeF_body_rel is_ifrangeF_body*

synthesize *is_ifrangeF_body* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrangeF_body_fm*

lemma (**in** *M_Z_trans*) *separation_is_ifrangeF_body*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(r) \Longrightarrow (\#\#M)(s) \Longrightarrow separation(\#\#M, is\_ifrangeF\_body(\#\#M,A,r,s))$
using *separation_in_ctm*[**where** φ=*is_ifrangeF_body_fm*(*1,2,3,0*) **and** *env*=[*A,r,s*]]
zero_in_M arity_is_ifrangeF_body_fm ord_simp_union is_ifrangeF_body_fm_type
by *simp*

lemma (**in** *M_basic*) *is_ifrFb_body_closed*: $M(r) \Longrightarrow M(s) \Longrightarrow is\_ifrFb\_body(M, r, s, x, i) \Longrightarrow M(i)$
unfolding *ifrangeF_body_def is_ifrangeF_body_def is_ifrFb_body_def If_abs*
by (*cases* $i{\in}range(s)$; *cases* $r{=}0$; *auto dest*:*transM*)

lemma (**in** *M_ZF1_trans*) *ifrangeF_body_abs*:
assumes $(\#\#M)(A)$ $(\#\#M)(r)$ $(\#\#M)(s)$ $(\#\#M)(x)$
shows $is\_ifrangeF\_body(\#\#M,A,r,s,x) \longleftrightarrow ifrangeF\_body(\#\#M,A,r,s,x)$

proof -
  {
    fix *a*
    assume *a∈M*
    with *assms*
   have (*μ i. i∈ M ∧ is_ifrFb_body(##M, r, s, z, i))= (μ i. is_ifrFb_body(##M, r, s, z, i)*) **for** *z*
      using *is_ifrFb_body_closed*[*of r s z*]
      by (*rule_tac Least_cong*[*of λi. i∈M ∧ is_ifrFb_body(##M,r,s,z,i)*]) *auto*
    moreover
    have (*μ i. is_ifrFb_body(##M, r, s, z, i))= (μ i. ifrFb_body(##M, r, s, z, i)*) **for** *z*
   proof (*rule_tac Least_cong*[*of λi. is_ifrFb_body(##M,r,s,z,i) λi. ifrFb_body(##M,r,s,z,i)*])
      fix *y*
      from *assms ‹a∈M›*
      show *is_ifrFb_body(##M, r, s, z, y) ⟷ ifrFb_body(##M, r, s, z, y)*
        using *If_abs apply_0*
        unfolding *ifrFb_body_def is_ifrFb_body_def*
        by (*cases y∈M*; *cases y∈range(s)*; *cases converse(s)‘y ∈ M*;
            auto dest:*transM split del*: *split_if del*:*iffI*)
          (*auto simp flip*:*setclass_iff*; (*force simp only*:*setclass_iff*))+
    qed
    moreover from *‹a∈M›*
    have *least(##M, λi. i ∈ M ∧ is_ifrFb_body(##M, r, s, z, i), a)*
      ⟷ a = (μ i. i∈ M ∧ is_ifrFb_body(##M, r, s, z,i)) **for** *z*
      using *If_abs least_abs'*[*of λi. (##M)(i) ∧ is_ifrFb_body(##M,r,s,z,i) a*]
      by *simp*
    ultimately
    have *least(##M, λi. i ∈ M ∧ is_ifrFb_body(##M, r, s, z, i), a)*
      ⟷ a = (μ i. ifrFb_body(##M, r, s, z,i)) **for** *z*
      by *simp*
  }
  with *assms*
  show *?thesis*
    using *pair_in_M_iff apply_closed zero_in_M transitivity*[*of _ A*]
    unfolding *ifrangeF_body_def is_ifrangeF_body_def*
    by (*auto dest*:*transM*)
qed

lemma (**in** *M_ZF1_trans*) *separation_ifrangeF_body*:
  (##M)(A) ⟹ (##M)(b) ⟹ (##M)(f) ⟹ separation
       (##M, λy. ∃x∈A. y = ⟨x, μ i. x ∈ if_range_F_else_F(λx. if (##M)(x)
then x else 0, b, f, i)⟩)
  using *separation_is_ifrangeF_body ifrangeF_body_abs*
   separation_cong[**where** *P=is_ifrangeF_body(##M,A,b,f)* **and** *M=##M*,*THEN*
iffD1]
 unfolding *ifrangeF_body_def if_range_F_def if_range_F_else_F_def ifrFb_body_def*
  by *simp*

definition *ifrFb_body2* **where**
ifrFb_body2(*M*,*G*,*b*,*f*,*x*,*i*) ≡ *x* ∈
(*if b = 0 then if i* ∈ *range*(*f*) *then*
if M(*converse*(*f*) ‘ *i*) *then G*‘(*converse*(*f*) ‘ *i*) *else 0 else 0 else if M*(*i*) *then G*‘*i*
else 0)

relativize functional *ifrFb_body2 ifrFb_body2_rel*
relationalize *ifrFb_body2_rel is_ifrFb_body2*

synthesize *is_ifrFb_body2* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrFb_body2_fm*

definition *ifrangeF_body2* :: [*i*⇒*o*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
ifrangeF_body2(*M*,*A*,*G*,*b*,*f*) ≡ λ*y*. ∃ *x*∈*A*. *y* = ⟨*x*,μ *i*. *ifrFb_body2*(*M*,*G*,*b*,*f*,*x*,*i*)⟩

relativize functional *ifrangeF_body2 ifrangeF_body2_rel*
relationalize *ifrangeF_body2_rel is_ifrangeF_body2*

synthesize *is_ifrangeF_body2* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrangeF_body2_fm*

lemma (**in** *M_Z_trans*) *separation_is_ifrangeF_body2*:
(##*M*)(*A*) ⟹ (##*M*)(*G*) ⟹ (##*M*)(*r*) ⟹ (##*M*)(*s*) ⟹ *separation*(##*M*,
is_ifrangeF_body2(##*M*,*A*,*G*,*r*,*s*))
using *separation_in_ctm*[**where** φ=*is_ifrangeF_body2_fm*(*1*,*2*,*3*,*4*,*0*) **and** *env*=[*A*,*G*,*r*,*s*]]
zero_in_M arity_is_ifrangeF_body2_fm ord_simp_union is_ifrangeF_body2_fm_type
by *simp*

lemma (**in** *M_basic*) *is_ifrFb_body2_closed*: *M*(*G*) ⟹ *M*(*r*) ⟹ *M*(*s*) ⟹
is_ifrFb_body2(*M*, *G*, *r*, *s*, *x*, *i*) ⟹ *M*(*i*)
unfolding *ifrangeF_body2_def is_ifrangeF_body2_def is_ifrFb_body2_def If_abs*
by (*cases i*∈*range*(*s*); *cases r=0*; *auto dest*:*transM*)

lemma (**in** *M_ZF1_trans*) *ifrangeF_body2_abs*:
assumes (##*M*)(*A*) (##*M*)(*G*) (##*M*)(*r*) (##*M*)(*s*) (##*M*)(*x*)
shows *is_ifrangeF_body2*(##*M*,*A*,*G*,*r*,*s*,*x*) ⟷ *ifrangeF_body2*(##*M*,*A*,*G*,*r*,*s*,*x*)
proof -
{
fix *a*
assume *a*∈*M*
with *assms*
have (μ *i*. *i*∈ *M* ∧ *is_ifrFb_body2*(##*M*, *G*, *r*, *s*, *z*, *i*))= (μ *i*. *is_ifrFb_body2*(##*M*,
G, *r*, *s*, *z*, *i*)) **for** *z*
using *is_ifrFb_body2_closed*[*of G r s z*]
by (*rule_tac Least_cong*[*of* λ*i*. *i*∈*M* ∧ *is_ifrFb_body2*(##*M*,*G*,*r*,*s*,*z*,*i*)])
auto
moreover

```
    have (μ i. is_ifrFb_body2(##M, G, r, s, z, i))= (μ i. ifrFb_body2(##M, G,
r, s, z, i)) for z
   proof (rule_tac Least_cong[of λi. is_ifrFb_body2(##M,G,r,s,z,i) λi. ifrFb_body2(##M,G,r,s,z,i)])
      fix y
      from assms ‹a∈M›
      show is_ifrFb_body2(##M, G, r, s, z, y) ⟷ ifrFb_body2(##M, G, r, s,
z, y)
        using If_abs apply_0
        unfolding ifrFb_body2_def is_ifrFb_body2_def
        by (cases y∈M; cases y∈range(s); cases converse(s)‘y ∈ M;
            auto dest:transM split del: split_if del:iffI)
          (auto simp flip:setclass_iff; (force simp only:setclass_iff))+
    qed
    moreover from ‹a∈M›
    have least(##M, λi. i ∈ M ∧ is_ifrFb_body2(##M, G, r, s, z, i), a)
      ⟷ a = (μ i.  i∈ M ∧ is_ifrFb_body2(##M, G, r, s, z,i)) for z
      using If_abs least_abs'[of λi. (##M)(i) ∧ is_ifrFb_body2(##M,G,r,s,z,i)
a]
      by simp
    ultimately
    have least(##M, λi. i ∈ M ∧ is_ifrFb_body2(##M, G, r, s, z, i), a)
      ⟷ a = (μ i. ifrFb_body2(##M, G, r, s, z,i)) for z
      by simp
  }
  with assms
  show ?thesis
    using  pair_in_M_iff apply_closed zero_in_M transitivity[of _ A]
    unfolding ifrangeF_body2_def is_ifrangeF_body2_def
    by (auto dest:transM)
qed

lemma (in M_ZF1_trans) separation_ifrangeF_body2:
  (##M)(A) ⟹ (##M)(G) ⟹ (##M)(b) ⟹ (##M)(f) ⟹
       separation
        (##M,
         λy. ∃x∈A.
               y =
               ⟨x, μ i. x ∈
                       if_range_F_else_F(λa. if (##M)(a) then G ‘ a else 0, b, f,
i)⟩)
  using separation_is_ifrangeF_body2 ifrangeF_body2_abs
   separation_cong[where P=is_ifrangeF_body2(##M,A,G,b,f) and M=##M,THEN
iffD1]
  unfolding ifrangeF_body2_def if_range_F_def if_range_F_else_F_def ifrFb_body2_def
  by simp



definition ifrFb_body3 where
```

ifrFb_body3(M,G,b,f,x,i) $\equiv$ $x \in$
(*if* $b = 0$ *then if* $i \in$ *range*(f) *then*
if M(*converse*(f) ‘ i) *then* G-‘‘{*converse*(f) ‘ i} *else 0 else 0 else if* $M(i)$ *then*
G-‘‘{i} *else 0*)

relativize functional *ifrFb_body3 ifrFb_body3_rel*
relationalize *ifrFb_body3_rel is_ifrFb_body3*

synthesize *is_ifrFb_body3* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrFb_body3_fm*

definition *ifrangeF_body3* :: $[i \Rightarrow o,i,i,i,i] \Rightarrow o$ **where**
ifrangeF_body3(M,A,G,b,f) $\equiv \lambda y.\ \exists\, x{\in}A.\ y = \langle x,\mu\ i.\ ifrFb\_body3(M,G,b,f,x,i)\rangle$

relativize functional *ifrangeF_body3 ifrangeF_body3_rel*
relationalize *ifrangeF_body3_rel is_ifrangeF_body3*

synthesize *is_ifrangeF_body3* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrangeF_body3_fm*

lemma (**in** *M_Z_trans*) *separation_is_ifrangeF_body3*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(G) \Longrightarrow (\#\#M)(r) \Longrightarrow (\#\#M)(s) \Longrightarrow$ *separation*($\#\#M$, *is_ifrangeF_body3*($\#\#M$,A,G,r,s))
using *separation_in_ctm*[**where** φ=*is_ifrangeF_body3_fm*(*1*,*2*,*3*,*4*,*0*) **and** *env*=[A,G,r,s]]
zero_in_M arity_is_ifrangeF_body3_fm ord_simp_union is_ifrangeF_body3_fm_type
by *simp*

lemma (**in** *M_basic*) *is_ifrFb_body3_closed*: $M(G) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow$ *is_ifrFb_body3*(M, G, r, s, x, i) $\Longrightarrow M(i)$
unfolding *ifrangeF_body3_def is_ifrangeF_body3_def is_ifrFb_body3_def If_abs*
by (*cases* $i{\in}$*range*(s); *cases* r=*0*; *auto dest*:*transM*)

lemma (**in** *M_ZF1_trans*) *ifrangeF_body3_abs*:
assumes $(\#\#M)(A)$ $(\#\#M)(G)$ $(\#\#M)(r)$ $(\#\#M)(s)$ $(\#\#M)(x)$
shows *is_ifrangeF_body3*($\#\#M$,A,G,r,s,x) $\longleftrightarrow$ *ifrangeF_body3*($\#\#M$,A,G,r,s,x)
proof -
{
fix a
assume $a{\in}M$
with *assms*
have (μ i. $i{\in} M \wedge$ *is_ifrFb_body3*($\#\#M$, G, r, s, z, i))= (μ i. *is_ifrFb_body3*($\#\#M$, G, r, s, z, i)) **for** z
using *is_ifrFb_body3_closed*[*of* G r s z]
by (*rule_tac Least_cong*[*of* $\lambda i.\ i{\in}M\ \wedge$ *is_ifrFb_body3*($\#\#M$,G,r,s,z,i)]) *auto*
moreover
have (μ i. *is_ifrFb_body3*($\#\#M$, G, r, s, z, i))= (μ i. *ifrFb_body3*($\#\#M$, G, r, s, z, i)) **for** z
proof (*rule_tac Least_cong*[*of* $\lambda i.$ *is_ifrFb_body3*($\#\#M$,G,r,s,z,i) $\lambda i.$ *ifrFb_body3*($\#\#M$,G,r,s,z,i)])

fix y
from *assms* ‹$a{\in}M$›
show *is_ifrFb_body3*($\#\#M$, G, r, s, z, y) $\longleftrightarrow$ *ifrFb_body3*($\#\#M$, G, r, s, z, y)
using *If_abs apply_0*
unfolding *ifrFb_body3_def is_ifrFb_body3_def*
by (*cases* $y{\in}M$; *cases* $y{\in}$*range*(s); *cases converse*(s)‘$y \in M$;
auto dest:*transM split del*: *split_if del*:*iffI*)
(*auto simp flip*:*setclass_iff*; (*force simp only*:*setclass_iff*))+
qed
moreover from ‹$a{\in}M$›
have *least*($\#\#M$, $\lambda i.\ i \in M \wedge$ *is_ifrFb_body3*($\#\#M$, G, r, s, z, i), a)
$\longleftrightarrow a = (\mu\ i.\ \ i{\in}\ M \wedge$ *is_ifrFb_body3*($\#\#M$, G, r, s, z,i)) **for** z
using *If_abs least_abs'*[*of* $\lambda i.$ ($\#\#M$)(i) $\wedge$ *is_ifrFb_body3*($\#\#M$,G,r,s,z,i) a]
by *simp*
ultimately
have *least*($\#\#M$, $\lambda i.\ i \in M \wedge$ *is_ifrFb_body3*($\#\#M$, G, r, s, z, i), a)
$\longleftrightarrow a = (\mu\ i.$ *ifrFb_body3*($\#\#M$, G, r, s, z,i)) **for** z
by *simp*
}
with *assms*
show *?thesis*
using *pair_in_M_iff apply_closed zero_in_M transitivity*[*of* _ A]
unfolding *ifrangeF_body3_def is_ifrangeF_body3_def*
by (*auto dest*:*transM*)
qed

lemma (**in** *M_ZF1_trans*) *separation_ifrangeF_body3*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(G) \Longrightarrow (\#\#M)(b) \Longrightarrow (\#\#M)(f) \Longrightarrow$
separation
($\#\#M$,
$\lambda y.\ \exists x{\in}A.$
$y =$
$\langle x,\ \mu\ i.\ x \in$
if_range_F_else_F($\lambda a.$ *if* ($\#\#M$)(a) *then* G-‘‘{a} *else* 0, b, f, i)$\rangle$)
using *separation_is_ifrangeF_body3 ifrangeF_body3_abs*
separation_cong[**where** P=*is_ifrangeF_body3*($\#\#M$,A,G,b,f) **and** M=$\#\#M$,*THEN iffD1*]
unfolding *ifrangeF_body3_def if_range_F_def if_range_F_else_F_def ifrFb_body3_def*
by *simp*

definition *ifrFb_body4* **where**
ifrFb_body4(G,b,f,x,i) $\equiv x \in$
(*if* b = 0 *then if* $i \in$ *range*(f) *then* G‘(*converse*(f) ‘ i) *else* 0 *else* G‘i)

relativize functional *ifrFb_body4 ifrFb_body4_rel*
relationalize *ifrFb_body4_rel is_ifrFb_body4*

synthesize *is_ifrFb_body4* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrFb_body4_fm*

definition *ifrangeF_body4* :: $[i \Rightarrow o,i,i,i,i] \Rightarrow o$ **where**
ifrangeF_body4$(M,A,G,b,f) \equiv \lambda y.\ \exists\, x{\in}A.\ y = \langle x,\mu\ i.\ ifrFb\_body4(G,b,f,x,i)\rangle$

relativize functional *ifrangeF_body4 ifrangeF_body4_rel*
relationalize *ifrangeF_body4_rel is_ifrangeF_body4*

synthesize *is_ifrangeF_body4* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrangeF_body4_fm*

lemma (**in** *M_Z_trans*) *separation_is_ifrangeF_body4*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(G) \Longrightarrow (\#\#M)(r) \Longrightarrow (\#\#M)(s) \Longrightarrow$ *separation*$(\#\#M,$ *is_ifrangeF_body4*$(\#\#M,A,G,r,s))$
using *separation_in_ctm*[**where** φ=*is_ifrangeF_body4_fm*(*1,2,3,4,0*) **and** *env*=$[A,G,r,s]$]
zero_in_M arity_is_ifrangeF_body4_fm ord_simp_union is_ifrangeF_body4_fm_type
by *simp*

lemma (**in** *M_basic*) *is_ifrFb_body4_closed*: $M(G) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow$ *is_ifrFb_body4*$(M,\ G,\ r,\ s,\ x,\ i) \Longrightarrow M(i)$
using *If_abs*
unfolding *ifrangeF_body4_def is_ifrangeF_body4_def is_ifrFb_body4_def fun_apply_def*
by (*cases i*∈*range*(*s*); *cases r=0*; *auto dest*:*transM*)

lemma (**in** *M_ZF1_trans*) *ifrangeF_body4_abs*:
assumes $(\#\#M)(A)$ $(\#\#M)(G)$ $(\#\#M)(r)$ $(\#\#M)(s)$ $(\#\#M)(x)$
shows *is_ifrangeF_body4*$(\#\#M,A,G,r,s,x) \longleftrightarrow$ *ifrangeF_body4*$(\#\#M,A,G,r,s,x)$
proof -
{
fix *a*
assume *a*∈*M*
with *assms*
have (μ *i*. *i*∈ *M* ∧ *is_ifrFb_body4*$(\#\#M,\ G,\ r,\ s,\ z,\ i)) = (\mu\ i.$ *is_ifrFb_body4*$(\#\#M,$ $G,\ r,\ s,\ z,\ i))$ **for** *z*
using *is_ifrFb_body4_closed*[*of G r s z*]
by (*rule_tac Least_cong*[*of* $\lambda i.$ *i*∈*M* ∧ *is_ifrFb_body4*$(\#\#M,G,r,s,z,i)$])
auto
moreover
have (μ *i*. *is_ifrFb_body4*$(\#\#M,\ G,\ r,\ s,\ z,\ i)) = (\mu\ i.$ *ifrFb_body4*$(G,\ r,\ s,\ z,$ $i))$ **if** *z*∈*M* **for** *z*
proof (*rule_tac Least_cong*[*of* $\lambda i.$ *is_ifrFb_body4*$(\#\#M,G,r,s,z,i)$ $\lambda i.$ *ifrFb_body4*(G,r,s,z,i)])
fix *y*
from *assms* ‹*a*∈*M*› ‹*z*∈*M*›
show *is_ifrFb_body4*$(\#\#M,\ G,\ r,\ s,\ z,\ y) \longleftrightarrow$ *ifrFb_body4*$(G,\ r,\ s,\ z,\ y)$
using *If_abs apply_0*

unfolding *ifrFb_body4_def is_ifrFb_body4_def*
apply (*cases* $y{\in}M$; *cases* $y{\in}range(s)$; *cases* $r{=}0$; *cases* $y{\in}domain(G)$;
auto dest:*transM split del*: *split_if del*:*iffI*)
by (*auto simp flip*:*setclass_iff*; (*force simp only*: *fun_apply_def setclass_iff*))
(*auto simp flip*:*setclass_iff simp*: *fun_apply_def*)
qed
moreover from ‹$a{\in}M$›
have $least(\#\#M,\ \lambda i.\ i \in M \wedge is\_ifrFb\_body4(\#\#M,\ G,\ r,\ s,\ z,\ i),\ a)$
$\longleftrightarrow a = (\mu\ i.\ \ i{\in}\ M \wedge is\_ifrFb\_body4(\#\#M,\ G,\ r,\ s,\ z,i))$ **for** z
using *If_abs least_abs'*[*of* $\lambda i.\ (\#\#M)(i) \wedge is\_ifrFb\_body4(\#\#M,G,r,s,z,i)$ a]
by *simp*
ultimately
have $z{\in}M \Longrightarrow least(\#\#M,\ \lambda i.\ i \in M \wedge is\_ifrFb\_body4(\#\#M,\ G,\ r,\ s,\ z,\ i),$ $a)$
$\longleftrightarrow a = (\mu\ i.\ ifrFb\_body4(G,\ r,\ s,\ z,i))$ **for** z
by *simp*
}
with *assms*
show *?thesis*
using *pair_in_M_iff apply_closed zero_in_M transitivity*[*of* _ A]
unfolding *ifrangeF_body4_def is_ifrangeF_body4_def*
by (*auto dest*:*transM*)
qed

lemma (**in** *M_ZF1_trans*) *separation_ifrangeF_body4*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(G) \Longrightarrow (\#\#M)(b) \Longrightarrow (\#\#M)(f) \Longrightarrow$
$separation(\#\#M,\ \lambda y.\ \exists x{\in}A.\ y = \langle x,\ \mu\ i.\ x \in if\_range\_F\_else\_F((`)(G),$ $b,\ f,\ i)\rangle)$
using *separation_is_ifrangeF_body4 ifrangeF_body4_abs*
separation_cong[**where** $P{=}is\_ifrangeF\_body4(\#\#M,A,G,b,f)$ **and** $M{=}\#\#M$,*THEN iffD1*]
unfolding *ifrangeF_body4_def if_range_F_def if_range_F_else_F_def ifrFb_body4_def*
by *simp*

definition *ifrFb_body5* **where**
$ifrFb\_body5(G,b,f,x,i) \equiv x \in$
(*if* $b = 0$ *then if* $i \in range(f)$ *then* $\{xa \in G\ .\ converse(f)\ `\ i \in xa\}$ *else* 0 *else* $\{xa \in G\ .\ i \in xa\}$)

relativize functional *ifrFb_body5 ifrFb_body5_rel*
relationalize *ifrFb_body5_rel is_ifrFb_body5*

synthesize *is_ifrFb_body5* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrFb_body5_fm*

definition *ifrangeF_body5* :: $[i{\Rightarrow}o,i,i,i,i,i] \Rightarrow o$ **where**

ifrangeF_body5$(M,A,G,b,f) \equiv \lambda y.\ \exists\, x{\in}A.\ y = \langle x,\mu\ i.\ \mathit{ifrFb\_body5}(G,b,f,x,i)\rangle$

relativize functional *ifrangeF_body5 ifrangeF_body5_rel*
relationalize *ifrangeF_body5_rel is_ifrangeF_body5*

synthesize *is_ifrangeF_body5* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrangeF_body5_fm*

lemma (**in** *M_Z_trans*) *separation_is_ifrangeF_body5*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(G) \Longrightarrow (\#\#M)(r) \Longrightarrow (\#\#M)(s) \Longrightarrow$ *separation*$(\#\#M,$ *is_ifrangeF_body5*$(\#\#M,A,G,r,s))$
using *separation_in_ctm*[**where** φ=*is_ifrangeF_body5_fm*(*1,2,3,4,0*) **and** *env*=$[A,G,r,s]$]
zero_in_M arity_is_ifrangeF_body5_fm ord_simp_union is_ifrangeF_body5_fm_type
by *simp*

lemma (**in** *M_basic*) *is_ifrFb_body5_closed*: $M(G) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow$ *is_ifrFb_body5*$(M,\ G,\ r,\ s,\ x,\ i) \Longrightarrow M(i)$
using *If_abs*
unfolding *ifrangeF_body5_def is_ifrangeF_body5_def is_ifrFb_body5_def fun_apply_def*
by (*cases i*∈*range*(*s*); *cases r=0*; *auto dest*:*transM*)

lemma (**in** *M_ZF1_trans*) *ifrangeF_body5_abs*:
assumes $(\#\#M)(A)$ $(\#\#M)(G)$ $(\#\#M)(r)$ $(\#\#M)(s)$ $(\#\#M)(x)$
shows *is_ifrangeF_body5*$(\#\#M,A,G,r,s,x) \longleftrightarrow$ *ifrangeF_body5*$(\#\#M,A,G,r,s,x)$
proof -
{
fix *a*
assume *a*∈*M*
with *assms*
have (μ *i*. *i*∈ $M \wedge$ *is_ifrFb_body5*$(\#\#M,\ G,\ r,\ s,\ z,\ i)) = (\mu\ i.$ *is_ifrFb_body5*$(\#\#M,$ $G,\ r,\ s,\ z,\ i))$ **for** *z*
using *is_ifrFb_body5_closed*[*of G r s z*]
by (*rule_tac Least_cong*[*of* $\lambda i.\ i{\in}M\ \wedge$ *is_ifrFb_body5*$(\#\#M,G,r,s,z,i)$])
auto
moreover
have (μ *i*. *is_ifrFb_body5*$(\#\#M,\ G,\ r,\ s,\ z,\ i)) = (\mu\ i.$ *ifrFb_body5*$(G,\ r,\ s,$ $z,\ i))$ **if** *z*∈*M* **for** *z*
proof (*rule_tac Least_cong*[*of* $\lambda i.$ *is_ifrFb_body5*$(\#\#M,G,r,s,z,i)$ $\lambda i.$ *ifrFb_body5*(G,r,s,z,i)])
fix *y*
from *assms* ‹*a*∈*M*› ‹*z*∈*M*›
show *is_ifrFb_body5*$(\#\#M,\ G,\ r,\ s,\ z,\ y) \longleftrightarrow$ *ifrFb_body5*$(G,\ r,\ s,\ z,\ y)$
using *If_abs apply_0 separation_in_constant separation_in_rev*
unfolding *ifrFb_body5_def is_ifrFb_body5_def*
apply (*cases y*∈*M*; *cases y*∈*range*(*s*); *cases r=0*; *cases y*∈*domain*(*G*);
auto dest:*transM split del*: *split_if del*:*iffI*)
apply (*auto simp flip*:*setclass_iff*; (*force simp only*: *fun_apply_def setclass_iff*))
apply (*auto simp flip*:*setclass_iff simp*: *fun_apply_def*)
apply (*auto dest*:*transM*)

```
      done
    qed
    moreover from ‹a∈M›
    have least(##M, λi. i ∈ M ∧ is_ifrFb_body5(##M, G, r, s, z, i), a)
      ⟷ a = (μ i.  i∈ M ∧ is_ifrFb_body5(##M, G, r, s, z,i)) for z
      using If_abs least_abs'[of λi. (##M)(i) ∧ is_ifrFb_body5(##M,G,r,s,z,i)
a]
      by simp
    ultimately
    have z∈M ⟹ least(##M, λi. i ∈ M ∧ is_ifrFb_body5(##M, G, r, s, z, i),
a)
      ⟷ a = (μ i. ifrFb_body5(G, r, s, z,i)) for z
      by simp
  }
  with assms
  show ?thesis
    using  pair_in_M_iff apply_closed zero_in_M transitivity[of _ A]
    unfolding ifrangeF_body5_def is_ifrangeF_body5_def
    by (auto dest:transM)
qed

lemma (in M_ZF1_trans) separation_ifrangeF_body5:
  (##M)(A) ⟹ (##M)(G) ⟹ (##M)(b) ⟹ (##M)(f) ⟹
       separation(##M, λy. ∃x∈A. y = ⟨x, μ i. x ∈ if_range_F_else_F(λx. {xa
∈ G . x ∈ xa}, b, f, i)⟩)
  using separation_is_ifrangeF_body5 ifrangeF_body5_abs
   separation_cong[where P=is_ifrangeF_body5(##M,A,G,b,f) and M=##M,THEN
iffD1]
 unfolding ifrangeF_body5_def if_range_F_def if_range_F_else_F_def ifrFb_body5_def
  by simp



definition ifrFb_body6 where
  ifrFb_body6(G,b,f,x,i) ≡ x ∈
  (if b = 0 then if i ∈ range(f) then {p∈G . domain(p) = converse(f) ‘ i} else 0
else {p∈G . domain(p) = i})

relativize functional ifrFb_body6 ifrFb_body6_rel
relationalize ifrFb_body6_rel is_ifrFb_body6

synthesize is_ifrFb_body6 from_definition assuming nonempty
arity_theorem for is_ifrFb_body6_fm

definition ifrangeF_body6 :: [i⇒o,i,i,i,i,i] ⇒ o where
  ifrangeF_body6(M,A,G,b,f) ≡ λy. ∃x∈A. y = ⟨x,μ i. ifrFb_body6(G,b,f,x,i)⟩

relativize functional ifrangeF_body6 ifrangeF_body6_rel
relationalize ifrangeF_body6_rel is_ifrangeF_body6
```

synthesize *is_ifrangeF_body6* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrangeF_body6_fm*

lemma (**in** *M_Z_trans*) *separation_is_ifrangeF_body6*:
 $(\#\#M)(A) \Longrightarrow (\#\#M)(G) \Longrightarrow (\#\#M)(r) \Longrightarrow (\#\#M)(s) \Longrightarrow$ *separation*$(\#\#M,$
is_ifrangeF_body6$(\#\#M,A,G,r,s))$
 using *separation_in_ctm*[**where** φ=*is_ifrangeF_body6_fm*(*1*,*2*,*3*,*4*,*0*) **and** *env*=[*A*,*G*,*r*,*s*]]
  zero_in_M arity_is_ifrangeF_body6_fm ord_simp_union is_ifrangeF_body6_fm_type
  by *simp*

lemma (**in** *M_basic*) *ifrFb_body6_closed*: $M(G) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow$ *ifrFb_body6*$(G,$
$r,\ s,\ x,\ i) \longleftrightarrow\ M(i) \wedge$ *ifrFb_body6*$(G,\ r,\ s,\ x,\ i)$
  using *If_abs*
 unfolding *ifrangeF_body6_def is_ifrangeF_body6_def ifrFb_body6_def fun_apply_def*
  by (*cases i*∈*range*(*s*); *cases r=0*; *auto dest*:*transM*)

lemma (**in** *M_basic*) *is_ifrFb_body6_closed*: $M(G) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow$
is_ifrFb_body6$(M,\ G,\ r,\ s,\ x,\ i) \Longrightarrow M(i)$
  using *If_abs*
 unfolding *ifrangeF_body6_def is_ifrangeF_body6_def is_ifrFb_body6_def fun_apply_def*
  by (*cases i*∈*range*(*s*); *cases r=0*; *auto dest*:*transM*)

lemma (**in** *M_ZF1_trans*) *ifrangeF_body6_abs*:
  assumes $(\#\#M)(A)$ $(\#\#M)(G)$ $(\#\#M)(r)$ $(\#\#M)(s)$ $(\#\#M)(x)$
 shows *is_ifrangeF_body6*$(\#\#M,A,G,r,s,x) \longleftrightarrow$ *ifrangeF_body6*$(\#\#M,A,G,r,s,x)$
proof -
  {
    fix *a*
    assume *a*∈*M*
    with *assms*
   have $(\mu\ i.\ i{\in}\ M \wedge$ *is_ifrFb_body6*$(\#\#M,\ G,\ r,\ s,\ z,\ i)) = (\mu\ i.$ *is_ifrFb_body6*$(\#\#M,$
$G,\ r,\ s,\ z,\ i))$ **for** *z*
      using *is_ifrFb_body6_closed*[*of G r s z*]
       by (*rule_tac Least_cong*[*of* $\lambda i.\ i{\in}M\ \wedge$ *is_ifrFb_body6*$(\#\#M,G,r,s,z,i)$])
auto
    moreover
      have $(\mu\ i.\ i{\in}M\ \wedge$ *is_ifrFb_body6*$(\#\#M,\ G,\ r,\ s,\ z,\ i)) = (\mu\ i.\ i{\in}M\ \wedge$
ifrFb_body6$(G,\ r,\ s,\ z,\ i))$ **if** $z{\in}M$ **for** *z*
   proof (*rule_tac Least_cong*[*of* $\lambda i.\ i{\in}M \wedge$ *is_ifrFb_body6*$(\#\#M,G,r,s,z,i)$ $\lambda i.$
$i{\in}M \wedge$ *ifrFb_body6*(G,r,s,z,i)])
      fix *y*
      from *assms* ‹*a*∈*M*› ‹*z*∈*M*›
    show $y{\in}M \wedge$ *is_ifrFb_body6*$(\#\#M,\ G,\ r,\ s,\ z,\ y) \longleftrightarrow y{\in}M \wedge$ *ifrFb_body6*$(G,$
$r,\ s,\ z,\ y)$
       using *If_abs apply_0 separation_in_constant transitivity*[*of _ G*]
         separation_closed converse_closed apply_closed range_closed zero_in_M
        separation_cong[*OF eq_commute*,*THEN iffD1*,*OF domain_eq_separation*]
       unfolding *ifrFb_body6_def is_ifrFb_body6_def*

```
      by auto
    qed
    moreover from ‹a∈M›
    have least(##M, λi. i ∈ M ∧ is_ifrFb_body6(##M, G, r, s, z, i), a)
      ⟷ a = (μ i.  i∈ M ∧ is_ifrFb_body6(##M, G, r, s, z,i)) for z
      using If_abs least_abs'[of λi. (##M)(i) ∧ is_ifrFb_body6(##M,G,r,s,z,i)
a]
      by simp
    ultimately
    have z∈M ⟹ least(##M, λi. i ∈ M ∧ is_ifrFb_body6(##M, G, r, s, z, i),
a)
      ⟷ a = (μ i. ifrFb_body6(G, r, s, z,i)) for z
      using Least_cong[OF ifrFb_body6_closed[of G r s]] assms
      by simp
  }
  with assms
  show ?thesis
    using  pair_in_M_iff apply_closed zero_in_M transitivity[of _ A]
    unfolding ifrangeF_body6_def is_ifrangeF_body6_def
    by (auto dest:transM)
qed

lemma (in M_ZF1_trans) separation_ifrangeF_body6:
  (##M)(A) ⟹ (##M)(G) ⟹ (##M)(b) ⟹ (##M)(f) ⟹
       separation(##M,
      λy. ∃x∈A. y = ⟨x, μ i. x ∈ if_range_F_else_F(λa. {p ∈ G . domain(p) =
a}, b, f, i)⟩)
  using separation_is_ifrangeF_body6 ifrangeF_body6_abs
   separation_cong[where P=is_ifrangeF_body6(##M,A,G,b,f) and M=##M,THEN
iffD1]
  unfolding ifrangeF_body6_def if_range_F_def if_range_F_else_F_def ifrFb_body6_def
  by simp




definition ifrFb_body7 where
  ifrFb_body7(B,D,A,b,f,x,i) ≡ x ∈
    (if b = 0 then if i ∈ range(f) then
        {d ∈ D . ∃r∈A. restrict(r, B) = converse(f) ‘ i ∧ d = domain(r)} else 0
       else {d ∈ D . ∃r∈A. restrict(r, B) = i ∧ d = domain(r)})

relativize functional ifrFb_body7 ifrFb_body7_rel
relationalize ifrFb_body7_rel is_ifrFb_body7

synthesize is_ifrFb_body7 from_definition assuming nonempty
arity_theorem for is_ifrFb_body7_fm
```

definition *ifrangeF_body7* :: $[i \Rightarrow o,i,i,i,i,i,i,i] \Rightarrow o$ **where**
ifrangeF_body7$(M,A,B,D,G,b,f) \equiv \lambda y.\ \exists\, x{\in}A.\ y = \langle x,\mu\ i.\ \mathit{ifrFb\_body7}(B,D,G,b,f,x,i)\rangle$

relativize functional *ifrangeF_body7* *ifrangeF_body7_rel*
relationalize *ifrangeF_body7_rel* *is_ifrangeF_body7*

synthesize *is_ifrangeF_body7* **from_definition assuming** *nonempty*
arity_theorem for *is_ifrangeF_body7_fm*

lemma (**in** *M_Z_trans*) *separation_is_ifrangeF_body7*:
$(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow (\#\#M)(D) \Longrightarrow (\#\#M)(G) \Longrightarrow (\#\#M)(r)$
$\Longrightarrow (\#\#M)(s) \Longrightarrow$ *separation*$(\#\#M,\ \mathit{is\_ifrangeF\_body7}(\#\#M,A,B,D,G,r,s))$
using *separation_in_ctm*[**where** φ=*is_ifrangeF_body7_fm*(*1*,*2*,*3*,*4*,*5*,*6*,*0*) **and** *env*=[*A*,*B*,*D*,*G*,*r*,*s*]]
zero_in_M arity_is_ifrangeF_body7_fm ord_simp_union is_ifrangeF_body7_fm_type
by *simp*

lemma (**in** *M_basic*) *ifrFb_body7_closed*: $M(B) \Longrightarrow M(D) \Longrightarrow M(G) \Longrightarrow M(r)$
$\Longrightarrow M(s) \Longrightarrow$
ifrFb_body7$(B,D,G,\ r,\ s,\ x,\ i) \longleftrightarrow\ M(i) \wedge$ *ifrFb_body7*$(B,D,G,\ r,\ s,\ x,\ i)$
using *If_abs*
unfolding *ifrangeF_body7_def is_ifrangeF_body7_def ifrFb_body7_def fun_apply_def*
by (*cases* $i{\in}$*range*(s); *cases r=0*; *auto dest*:*transM*)

lemma (**in** *M_basic*) *is_ifrFb_body7_closed*: $M(B) \Longrightarrow M(D) \Longrightarrow M(G) \Longrightarrow$
$M(r) \Longrightarrow M(s) \Longrightarrow$
is_ifrFb_body7$(M,\ B,D,G,\ r,\ s,\ x,\ i) \Longrightarrow M(i)$
using *If_abs*
unfolding *ifrangeF_body7_def is_ifrangeF_body7_def is_ifrFb_body7_def fun_apply_def*
by (*cases* $i{\in}$*range*(s); *cases r=0*; *auto dest*:*transM*)

lemma (**in** *M_ZF1_trans*) *ifrangeF_body7_abs*:
assumes $(\#\#M)(A)$ $(\#\#M)(B)$ $(\#\#M)(D)$ $(\#\#M)(G)$ $(\#\#M)(r)$ $(\#\#M)(s)$
$(\#\#M)(x)$
shows *is_ifrangeF_body7*$(\#\#M,A,B,D,G,r,s,x) \longleftrightarrow$ *ifrangeF_body7*$(\#\#M,A,B,D,G,r,s,x)$
proof -
from *assms*
have *sep_dr*: $y{\in}M \Longrightarrow$ *separation*$(\#\#M,\ \lambda d\ .\ \exists\, r{\in}M\ .\ r{\in}G \wedge y = \mathit{restrict}(r,$
$B) \wedge d = \mathit{domain}(r))$ **for** *y*
by(*rule_tac separation_cong*[**where** $P'{=}\lambda d\ .\ \exists\, r{\in}\ M\ .\ r{\in}G \wedge y = \mathit{restrict}(r,$
$B) \wedge d = \mathit{domain}(r)$,*THEN iffD1*,*OF* _
separation_restrict_eq_dom_eq[*rule_format*,*of G B y*]],*auto simp*:*transitivity*[*of*
_ *G*])
from *assms*
have *sep_dr''*: $y{\in}M \Longrightarrow$ *separation*$(\#\#M,\ \lambda d\ .\ \exists\, r{\in}M.\ r \in G \wedge d = \mathit{domain}(r)$
$\wedge$ *converse*(s) ‘ $y = \mathit{restrict}(r,\ B))$ **for** *y*
by(*rule_tac separation_cong*[*THEN iffD1*,*OF* _ *separation_restrict_eq_dom_eq*[*rule_format*,*of*
G B converse(s) ‘ *y*]],
auto simp:*transitivity*[*of* _ *G*] *apply_closed*[*simplified*] *converse_closed*[*simplified*])

```
 {
  fix a
  assume a∈M
  with assms
 have (μ i. i∈ M ∧ is_ifrFb_body7(##M, B,D,G, r, s, z, i))= (μ i. is_ifrFb_body7(##M,B,D,
G, r, s, z, i)) for z
    using is_ifrFb_body7_closed[of B D G r s z]
   by (rule_tac Least_cong[of λi. i∈M ∧ is_ifrFb_body7(##M,B,D,G,r,s,z,i)])
auto
  moreover from this
   have (μ i. i∈M ∧ is_ifrFb_body7(##M, B,D,G, r, s, z, i))= (μ i. i∈M ∧
ifrFb_body7(B,D,G, r, s, z, i)) if z∈M for z
 proof (rule_tac Least_cong[of λi. i∈M ∧ is_ifrFb_body7(##M,B,D,G,r,s,z,i)
λi. i∈M ∧ ifrFb_body7(B,D,G,r,s,z,i)])
    from assms ‹a∈M› ‹z∈M›
    have is_ifrFb_body7(##M, B,D,G, r, s, z, y) ⟷ ifrFb_body7(B,D,G, r,
s, z, y) if y∈M for y
      using If_abs apply_0
        separation_closed converse_closed apply_closed range_closed zero_in_M
        transitivity[of _ D] transitivity[of _ G] that sep_dr sep_dr''
      unfolding ifrFb_body7_def is_ifrFb_body7_def
      by auto
    then
    show  y ∈ M ∧ is_ifrFb_body7(##M, B, D, G, r, s, z, y) ⟷ y ∈ M ∧
ifrFb_body7(B, D, G, r, s, z, y) for y
      using conj_cong
      by simp
  qed
  moreover from ‹a∈M›
  have least(##M, λi. i ∈ M ∧ is_ifrFb_body7(##M, B,D,G, r, s, z, i), a)
    ⟷ a = (μ i.  i∈ M ∧ is_ifrFb_body7(##M,B,D,G, r, s, z,i)) for z
  using If_abs least_abs'[of λi. (##M)(i) ∧ is_ifrFb_body7(##M,B,D,G,r,s,z,i)
a]
    by simp
  ultimately
   have z∈M ⟹ least(##M, λi. i ∈ M ∧ is_ifrFb_body7(##M,B,D,G, r, s,
z, i), a)
    ⟷ a = (μ i. ifrFb_body7(B,D,G, r, s, z,i)) for z
    using Least_cong[OF ifrFb_body7_closed[of B D G r s]] assms
    by simp
 }
 with assms
 show ?thesis
  using  pair_in_M_iff apply_closed zero_in_M transitivity[of _ A]
  unfolding ifrangeF_body7_def is_ifrangeF_body7_def
  by (auto dest:transM)
qed

lemma (in M_ZF1_trans) separation_ifrangeF_body7:
```

$(\#\#M)(A) \Longrightarrow (\#\#M)(B) \Longrightarrow (\#\#M)(D) \Longrightarrow (\#\#M)(G) \Longrightarrow (\#\#M)(b) \Longrightarrow$
$(\#\#M)(f) \Longrightarrow$
separation$(\#\#M,$
$\lambda y.\ \exists\, x{\in}A.\ y = \langle x,\ \mu\ i.\ x \in$ *if_range_F_else_F*$(drSR\_Y(B,\ D,\ G),\ b,\ f,\ i)\rangle)$
using *separation_is_ifrangeF_body7 ifrangeF_body7_abs drSR_Y_equality*
separation_cong[**where** P=*is_ifrangeF_body7*$(\#\#M,A,B,D,G,b,f)$ **and** M=$\#\#M$,*THEN iffD1*]
unfolding *ifrangeF_body7_def if_range_F_def if_range_F_else_F_def ifrFb_body7_def*
by *simp*

definition *omfunspace* :: $[i,i] \Rightarrow o$ **where**
omfunspace$(B) \equiv \lambda z.\ \exists\, x.\ \exists\, n{\in}\omega.\ z{\in}x \wedge x = n{\rightarrow}B$
relativize functional *omfunspace omfunspace_rel*
relationalize *omfunspace_rel is_omfunspace*
synthesize *is_omfunspace* **from_definition assuming** *nonempty*
arity_theorem for *is_omfunspace_fm*

context *M_pre_seqspace*
begin

is_iff_rel for *omfunspace*
using *is_function_space_iff*
unfolding *omfunspace_rel_def is_omfunspace_def*
by (*simp add*:*absolut*)

end — *M_pre_seqspace*

context *M_ZF1_trans*
begin

lemma *separation_omfunspace*:
assumes $(\#\#M)(B)$
shows *separation*$(\#\#M,\ \lambda z.\ \exists\, x[\#\#M].\ \exists\, n[\#\#M].\ n \in \omega \wedge z \in x \wedge x = n \rightarrow^M B)$
using *assms separation_in_ctm*[**where** *env*=$[B]$ **and** φ=*is_omfunspace_fm*$(1,0)$
and Q=*is_omfunspace*$(\#\#M,B)$]
nonempty is_omfunspace_iff[*of* B, *THEN separation_cong*, *of* $\#\#M$]
arity_is_omfunspace_fm is_omfunspace_fm_type
unfolding *omfunspace_rel_def*
by (*auto simp add*:*ord_simp_union*)

end — *M_ZF1_trans*

sublocale *M_ZF1_trans* $\subseteq$ *M_seqspace* $\#\#M$
using *separation_omfunspace* **by** *unfold_locales*

definition *cdltgamma* :: $[i,i] \Rightarrow o$ **where**
cdltgamma$(\gamma) \equiv \lambda Z\ .\ |Z| < \gamma$
relativize functional *cdltgamma cdltgamma_rel*

relationalize *cdltgamma_rel is_cdltgamma*
synthesize *is_cdltgamma* **from_definition assuming** *nonempty*
arity_theorem for *is_cdltgamma_fm*

definition *cdeqgamma* :: $[i] \Rightarrow o$ **where**
  $cdeqgamma \equiv \lambda Z\ .\ |fst(Z)| = snd(Z)$
relativize functional *cdeqgamma cdeqgamma_rel*
relationalize *cdeqgamma_rel is_cdeqgamma*
synthesize *is_cdeqgamma* **from_definition assuming** *nonempty*
arity_theorem for *is_cdeqgamma_fm*

context *M_Perm*
begin

is_iff_rel for *cdltgamma*
  using *is_cardinal_iff*
  unfolding *cdltgamma_rel_def is_cdltgamma_def*
  by (*simp add*:*absolut*)

is_iff_rel for *cdeqgamma*
  using *is_cardinal_iff fst_rel_abs snd_rel_abs*
  unfolding *cdeqgamma_rel_def is_cdeqgamma_def*
  by (*auto simp add*:*absolut*)

lemma *is_cdeqgamma_iff_split*: $M(Z) \Longrightarrow cdeqgamma\_rel(M,\ Z) \longleftrightarrow (\lambda\langle x,y\rangle.\ |x|^M = y)(Z)$
  using *fst_rel_abs snd_rel_abs*
  unfolding *cdeqgamma_rel_def split_def*
  by *simp*

end

context *M_ZF1_trans*
begin

lemma *separation_cdltgamma*:
  assumes $(\#\#M)(\gamma)$
  shows $separation(\#\#M,\ \lambda Z\ .\ cardinal\_rel(\#\#M,Z) < \gamma)$
  using *assms separation_in_ctm*[**where** *env*=[γ] **and** φ=$is\_cdltgamma\_fm(1,0)$
     and Q=$cdltgamma\_rel(\#\#M,\gamma)$]
   nonempty is_cdltgamma_iff[*of* γ] *arity_is_cdltgamma_fm is_cdltgamma_fm_type*
  unfolding *cdltgamma_rel_def*
  by (*auto simp add*:*ord_simp_union*)

lemma *separation_cdeqgamma*:
  shows $separation(\#\#M,\ \lambda Z.\ (\lambda\langle x,y\rangle\ .\ cardinal\_rel(\#\#M,x) = y)(Z))$
  using *separation_in_ctm*[**where** *env*=[] **and** φ=$is\_cdeqgamma\_fm(0)$
     and Q=$cdeqgamma\_rel(\#\#M)$] *is_cdeqgamma_iff_split*
    nonempty is_cdeqgamma_iff arity_is_cdeqgamma_fm is_cdeqgamma_fm_type

separation_cong[*OF is_cdeqgamma_iff_split*, *of* ##*M*]
unfolding *cdeqgamma_rel_def*
by (*simp add*:*ord_simp_union*)

end — *M_ZF1_trans*

end

9 Further instances of axiom-schemes

theory *ZF_Trans_Interpretations*
imports
Internal_ZFC_Axioms
Replacement_Instances

begin

locale *M_ZF2* = *M_ZF1* +
assumes
replacement_ax2:
replacement_assm(*M*,*env*,*ordtype_replacement_fm*)
replacement_assm(*M*,*env*,*wfrec_ordertype_fm*)
replacement_assm(*M*,*env*,*wfrec_Aleph_fm*)
replacement_assm(*M*,*env*,*omap_replacement_fm*)

definition *instances2_fms* **where** *instances2_fms* ≡
{ *ordtype_replacement_fm*,
wfrec_ordertype_fm,
wfrec_Aleph_fm,
omap_replacement_fm }

lemmas *replacement_instances2_defs* =
ordtype_replacement_fm_def wfrec_ordertype_fm_def
wfrec_Aleph_fm_def omap_replacement_fm_def

declare (**in** *M_ZF2*) *replacement_instances2_defs* [*simp*]

locale *M_ZF2_trans* = *M_ZF1_trans* + *M_ZF2*

locale *M_ZFC2* = *M_ZFC1* + *M_ZF2*

locale *M_ZFC2_trans* = *M_ZFC1_trans* + *M_ZF2_trans* + *M_ZFC2*

locale *M_ZF2_ground_notCH* = *M_ZF2* + *M_ZF_ground_notCH*

locale *M_ZF2_ground_notCH_trans* = *M_ZF2_trans* + *M_ZF2_ground_notCH*
+ *M_ZF_ground_notCH_trans*

locale *M_ZFC2_ground_notCH* = *M_ZFC2* + *M_ZF2_ground_notCH*

locale *M_ZFC2_ground_notCH_trans* = *M_ZFC2_trans* + *M_ZFC2_ground_notCH*
+ *M_ZF2_ground_notCH_trans*

locale *M_ZFC2_ground_CH_trans* = *M_ZFC2_ground_notCH_trans* + *M_ZF_ground_CH_trans*

locale *M_ctm2* = *M_ctm1* + *M_ZF2_ground_notCH_trans*

locale *M_ctm2_AC* = *M_ctm2* + *M_ctm1_AC* + *M_ZFC2_ground_notCH_trans*

locale *M_ctm2_AC_CH* = *M_ctm2_AC* + *M_ZFC2_ground_CH_trans*

lemmas (**in** *M_ZF1_trans*) *separation_instances* =
separation_well_ord_iso
separation_obase_equals separation_is_obase
separation_PiP_rel separation_surjP_rel
separation_radd_body separation_rmult_body

context *M_ZF2_trans*
begin

lemma *replacement_HAleph_wfrec_repl_body*:
B∈*M* ⟹ *strong_replacement*(##*M*, *HAleph_wfrec_repl_body*(##*M*,*B*))
using *strong_replacement_rel_in_ctm*[**where** φ=*HAleph_wfrec_repl_body_fm*(*2*,*0*,*1*)
and *env*=[*B*]]
zero_in_M arity_HAleph_wfrec_repl_body_fm replacement_ax2(*3*) *ord_simp_union*
by *simp*

lemma *HAleph_wfrec_repl*:
(##*M*)(*sa*) ⟹
(##*M*)(*esa*) ⟹
(##*M*)(*mesa*) ⟹
strong_replacement
(##*M*,
λ*x z*. ∃ *y*[##*M*].
pair(##*M*, *x*, *y*, *z*) ∧
(∃ *f*[##*M*].
(∀ *z*[##*M*].
z ∈ *f* ⟷
(∃ *xa*[##*M*].
∃ *y*[##*M*].
∃ *xaa*[##*M*].
∃ *sx*[##*M*].
∃ *r_sx*[##*M*].
∃ *f_r_sx*[##*M*].
pair(##*M*, *xa*, *y*, *z*) ∧
pair(##*M*, *xa*, *x*, *xaa*) ∧
upair(##*M*, *xa*, *xa*, *sx*) ∧
pre_image(##*M*, *mesa*, *sx*, *r_sx*) ∧

restriction(##*M*, *f*, *r_sx*, *f_r_sx*) ∧ *xaa* ∈ *mesa* ∧ *is_HAleph*(##*M*, *xa*, *f_r_sx*, *y*))) ∧
is_HAleph(##*M*, *x*, *f*, *y*)))
using *replacement_HAleph_wfrec_repl_body* **unfolding** *HAleph_wfrec_repl_body_def*
by *simp*

lemma *replacement_is_order_eq_map*:
A∈*M* ⟹ *r*∈*M* ⟹ *strong_replacement*(##*M*, *order_eq_map*(##*M*,*A*,*r*))
using *strong_replacement_rel_in_ctm*[**where** *φ*=*order_eq_map_fm*(*2*,*3*,*0*,*1*) **and** *env*=[*A*,*r*] **and** *f*=*order_eq_map*(##*M*,*A*,*r*)]
order_eq_map_iff_sats[**where** *env*=[_,_,*A*,*r*]] *zero_in_M fst_snd_closed pair_in_M_iff*
arity_order_eq_map_fm ord_simp_union replacement_ax2(*4*)
by *simp*

end — *M_ZF2_trans*

definition *omap_wfrec_body* **where**
omap_wfrec_body(*A*,*r*) ≡ (·∃·*image_fm*(*2*, *0*, *1*) ∧ *pred_set_fm*(*A* #+ *9*, *3*, *r* #+ *9*, *0*) ··)

lemma *type_omap_wfrec_body_fm* :*A*∈*nat* ⟹ *r*∈*nat* ⟹ *omap_wfrec_body*(*A*,*r*)∈*formula*
unfolding *omap_wfrec_body_def* **by** *simp*

lemma *arity_aux* : *A*∈*nat* ⟹ *r*∈*nat* ⟹ *arity*(*omap_wfrec_body*(*A*,*r*)) = (*9*+$_\omega$*A*) ∪ (*9*+$_\omega$*r*)
unfolding *omap_wfrec_body_def*
using *arity_image_fm arity_pred_set_fm pred_Un_distrib union_abs2*[*of 3*] *union_abs1*
by (*simp add*:*FOL_arities*, *auto simp add*:*Un_assoc*[*symmetric*] *union_abs1*)

lemma *arity_omap_wfrec*: *A*∈*nat* ⟹ *r*∈*nat* ⟹
arity(*is_wfrec_fm*(*omap_wfrec_body*(*A*,*r*),*succ*(*succ*(*succ*(*r*))), *1*, *0*)) =
(*4*+$_\omega$*A*) ∪ (*4*+$_\omega$*r*)
using *Arities.arity_is_wfrec_fm*[*OF* _ _ _ _ _ *arity_aux*,*of A r* 3+$_\omega$*r 1 0*] *pred_Un_distrib*
union_abs1 union_abs2 type_omap_wfrec_body_fm
by *auto*

lemma *arity_isordermap*: *A*∈*nat* ⟹ *r*∈*nat* ⟹*d*∈*nat*⟹
arity(*is_ordermap_fm*(*A*,*r*,*d*)) = *succ*(*d*) ∪ (*succ*(*A*) ∪ *succ*(*r*))
unfolding *is_ordermap_fm_def*
using *arity_lambda_fm*[**where** *i*=(*4*+$_\omega$*A*) ∪ (*4*+$_\omega$*r*),*OF* _ _ _ _ *arity_omap_wfrec*, *unfolded omap_wfrec_body_def*] *pred_Un_distrib union_abs1*
by *auto*

lemma *arity_is_ordertype*: *A*∈*nat* ⟹ *r*∈*nat* ⟹*d*∈*nat*⟹
arity(*is_ordertype_fm*(*A*,*r*,*d*)) = *succ*(*d*) ∪ (*succ*(*A*) ∪ *succ*(*r*))
unfolding *is_ordertype_fm_def*

using *arity_isordermap arity_image_fm pred_Un_distrib FOL_arities*
by *auto*

lemma *arity_is_order_body*: *arity*(*is_order_body_fm*(*1*,*0*)) = *2*
using *arity_is_order_body_fm arity_is_ordertype ord_simp_union*
by (*simp add*:*FOL_arities*)

lemma (**in** *M_ZF2_trans*) *replacement_is_order_body*:
strong_replacement(*##M*, λ*x z* . ∃ *y*[*##M*]. *is_order_body*(*##M*,*x*,*y*) ∧ *z* = ⟨*x*,*y*⟩)
apply(*rule_tac strong_replacement_cong*[
where *P*=λ *x f*. *M*,[*x*,*f*] ⊨ (·∃ ·*is_order_body_fm*(*1*,*0*) ∧ *pair_fm*(*1*,*0*,*2*) ··),*THEN iffD1*])
apply(*simp add*: *is_order_body_iff_sats*[**where** *env*=[_,_],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac replacement_ax2*(*1*)[*unfolded replacement_assm_def*, *rule_format*, **where** *env*=[],*simplified*])
apply(*simp_all add*:*arity_is_order_body arity pred_Un_distrib ord_simp_union*)
done

definition *H_order_pred* **where**
H_order_pred(*A*,*r*) ≡ λ*x f* . *f* `` *Order.pred*(*A*, *x*, *r*)

relationalize *H_order_pred is_H_order_pred*

lemma (**in** *M_basic*) *H_order_pred_abs* :
M(*A*) ⟹ *M*(*r*) ⟹ *M*(*x*) ⟹ *M*(*f*) ⟹ *M*(*z*) ⟹
is_H_order_pred(*M*,*A*,*r*,*x*,*f*,*z*) ⟷ *z* = *H_order_pred*(*A*,*r*,*x*,*f*)
unfolding *is_H_order_pred_def H_order_pred_def*
by *simp*

synthesize *is_H_order_pred* **from_definition assuming** *nonempty*

lemma (**in** *M_ZF2_trans*) *wfrec_replacement_order_pred*:
A∈*M* ⟹ *r*∈*M* ⟹ *wfrec_replacement*(*##M*, λ*x g z*. *is_H_order_pred*(*##M*,*A*,*r*,*x*,*g*,*z*) , *r*)
unfolding *wfrec_replacement_def is_wfrec_def M_is_recfun_def is_H_order_pred_def*
apply(*rule_tac strong_replacement_cong*[
where *P*=λ *x f*. *M*,[*x*,*f*,*r*,*A*] ⊨ *order_pred_wfrec_body_fm*(*3*,*2*,*1*,*0*),*THEN iffD1*])
apply(*subst order_pred_wfrec_body_def*[*symmetric*])
apply(*rule_tac order_pred_wfrec_body_iff_sats*[**where** *env*=[_,_,*r*,*A*],*symmetric*])
apply(*simp_all add*:*zero_in_M*)
apply(*rule_tac replacement_ax2*(*2*)[*unfolded replacement_assm_def*, *rule_format*, **where** *env*=[*r*,*A*],*simplified*])
apply(*simp_all add*: *arity_order_pred_wfrec_body_fm ord_simp_union*)
done

lemma (**in** *M_ZF2_trans*) *wfrec_replacement_order_pred'*:
*A∈M ⟹ r∈M ⟹ wfrec_replacement(##M, λx g z. z = H_order_pred(A,r,x,g)
, r)*
using *wfrec_replacement_cong*[*OF H_order_pred_abs*[*of A r*,*rule_format*] *refl*,*THEN
iffD1*,
OF _ _ _ _ _ wfrec_replacement_order_pred[*of A r*]]
by *simp*

sublocale *M_ZF2_trans ⊆ M_pre_cardinal_arith ##M*
using *separation_instances wfrec_replacement_order_pred'*[*unfolded H_order_pred_def*]
replacement_is_order_eq_map[*unfolded order_eq_map_def*]
by *unfold_locales simp_all*

definition *is_well_ord_fst_snd* **where**
*is_well_ord_fst_snd(A,x) ≡ (∃ a[A]. ∃ b[A]. is_well_ord(A,a,b) ∧ is_snd(A, x,
b) ∧ is_fst(A, x, a))*

synthesize *is_well_ord_fst_snd* **from_definition assuming** *nonempty*
arity_theorem for *is_well_ord_fst_snd_fm*

lemma (**in** *M_ZF2_trans*) *separation_well_ord*: *separation(##M, λx. is_well_ord(##M,fst(x),
snd(x)))*
using *arity_is_well_ord_fst_snd_fm is_well_ord_iff_sats*[*symmetric*] *nonempty*
fst_closed snd_closed fst_abs snd_abs
separation_in_ctm[**where** *env=[]* **and** *φ=is_well_ord_fst_snd_fm(0)*]
by(*simp_all add*: *is_well_ord_fst_snd_def*)

sublocale *M_ZF2_trans ⊆ M_pre_aleph ##M*
using *HAleph_wfrec_repl replacement_is_order_body*
separation_well_ord separation_Pow_rel
by *unfold_locales* (*simp_all add*: *transrec_replacement_def*
wfrec_replacement_def is_wfrec_def M_is_recfun_def flip:*setclass_iff*)

arity_theorem intermediate for *is_HAleph_fm*
lemma *arity_is_HAleph_fm*: *arity(is_HAleph_fm(2, 1, 0)) = 3*
using *arity_fun_apply_fm*[*of 11 0 1*,*simplified*]
arity_is_HAleph_fm' arity_ordinal_fm arity_is_If_fm
arity_empty_fm arity_is_Limit_fm
arity_is_If_fm
arity_is_Limit_fm arity_empty_fm
arity_Replace_fm[**where** *i=12* **and** *v=10* **and** *n=3*]
pred_Un_distrib ord_simp_union
by (*simp add*:*FOL_arities*)

lemma *arity_is_Aleph*[*arity*]: *arity(is_Aleph_fm(0, 1)) = 2*
unfolding *is_Aleph_fm_def*
using *arity_transrec_fm*[*OF _ _ _ _ arity_is_HAleph_fm*] *ord_simp_union*

by *simp*

definition *bex_Aleph_rel* :: $[i⇒o,i,i] ⇒ o$ **where**
bex_Aleph_rel$(M,x) ≡ λy.\ ∃ z∈x.\ y = ℵ_z^M$

relationalize *bex_Aleph_rel is_bex_Aleph*

schematic_goal *sats_is_bex_Aleph_fm_auto*:
$a ∈ nat ⟹ c ∈ nat ⟹ env ∈ list(A) ⟹$
$a < length(env) ⟹ c < length(env) ⟹ 0 ∈ A ⟹$
is_bex_Aleph$(##A,\ nth(a,\ env),\ nth(c,\ env)) ⟷ A,\ env ⊨ ?fm(a,\ c)$
unfolding *is_bex_Aleph_def*
by (*rule iff_sats* | *simp*)+

synthesize_notc *is_bex_Aleph* **from_schematic**

lemma *is_bex_Aleph_fm_type* [*TC*]:
$x ∈ ω ⟹ z ∈ ω ⟹$ *is_bex_Aleph_fm*$(x,\ z) ∈ formula$
unfolding *is_bex_Aleph_fm_def* **by** *simp*

lemma *sats_is_bex_Aleph_fm*:
$x ∈ ω ⟹$
$z ∈ ω ⟹ x < length(env) ⟹ z < length(env) ⟹$
$env ∈ list(Aa) ⟹$
$0 ∈ Aa ⟹$
$(Aa,\ env ⊨ is\_bex\_Aleph\_fm(x,\ z)) ⟷$
is_bex_Aleph$(##Aa,nth(x,\ env),\ nth(z,\ env))$
using *sats_is_bex_Aleph_fm_auto* **unfolding** *is_bex_Aleph_def is_bex_Aleph_fm_def*
by *simp*

lemma *is_bex_Aleph_iff_sats* [*iff_sats*]:
$nth(x,\ env) = xa ⟹$
$nth(z,\ env) = za ⟹$
$x ∈ ω ⟹$
$z ∈ ω ⟹ x < length(env) ⟹ z < length(env) ⟹$
$env ∈ list(Aa) ⟹$
$0 ∈ Aa ⟹$
is_bex_Aleph$(##Aa,\ xa,\ za) ⟷$
$Aa,\ env ⊨ is\_bex\_Aleph\_fm(x,\ z)$
using *sats_is_bex_Aleph_fm* **by** *simp*

arity_theorem for *is_bex_Aleph_fm*

lemma (**in** *M_ZF1_trans*) *separation_is_bex_Aleph*:
assumes $(##M)(A)$
shows *separation*$(##M,$*is_bex_Aleph*$(##M,\ A))$
using *assms separation_in_ctm*[**where** $env=[A]$ **and** $\varphi=$*is_bex_Aleph_fm*$(1,0)$,
OF _ _ _ is_bex_Aleph_iff_sats[*symmetric*],
of $λ\_.A$]

nonempty arity_is_bex_Aleph_fm is_bex_Aleph_fm_type
by (*simp add*:*ord_simp_union*)

lemma (**in** *M_pre_aleph*) *bex_Aleph_rel_abs*:
assumes *Ord*(*u*) *M*(*u*) *M*(*v*)
shows *is_bex_Aleph*(*M*, *u*, *v*) ⟷ *bex_Aleph_rel*(*M*,*u*,*v*)
unfolding *is_bex_Aleph_def bex_Aleph_rel_def*
using *assms is_Aleph_iff transM*[*of* _ *u*] *Ord_in_Ord*
by *simp*

lemma (**in** *M_ZF2_trans*) *separation_bex_Aleph_rel*:
Ord(*x*) ⟹ (##*M*)(*x*) ⟹ *separation*(##*M*, *bex_Aleph_rel*(##*M*,*x*))
using *separation_is_bex_Aleph bex_Aleph_rel_abs*
separation_cong[**where** *P*=*is_bex_Aleph*(##*M*,*x*) **and** *M*=##*M*,*THEN iffD1*]
unfolding *bex_Aleph_rel_def*
by *simp*

sublocale *M_ZF2_trans* ⊆ *M_aleph* ##*M*
using *separation_bex_Aleph_rel*[*unfolded bex_Aleph_rel_def*]
by *unfold_locales*

sublocale *M_ZF1_trans* ⊆ *M_FiniteFun* ##*M*
using *separation_is_function separation_omfunspace*
by *unfold_locales simp*

sublocale *M_ZFC2_trans* ⊆ *M_cardinal_AC* ##*M*
using *lam_replacement_minimum*
by *unfold_locales simp*

lemma (**in** *M_ZF1_trans*) *separation_cardinal_rel_lesspoll_rel*:
(##*M*)(*κ*) ⟹ *separation*(##*M*, λ*x*. *x* ≺^*M* *κ*)
using *separation_in_ctm*[**where** *φ*=(·*0* ≺ *1*·) **and** *env*=[*κ*]]
is_lesspoll_iff nonempty
arity_is_cardinal_fm arity_is_lesspoll_fm arity_is_bij_fm ord_simp_union
by (*simp add*:*FOL_arities*)

sublocale *M_ZFC2_trans* ⊆ *M_library* ##*M*
using *separation_cardinal_rel_lesspoll_rel lam_replacement_minimum*
by *unfold_locales simp_all*

locale *M_ZF3* = *M_ZF2* +
assumes
ground_replacements3:
ground_replacement_assm(*M*,*env*,*ordtype_replacement_fm*)
ground_replacement_assm(*M*,*env*,*wfrec_ordertype_fm*)
ground_replacement_assm(*M*,*env*,*eclose_abs_fm*)
ground_replacement_assm(*M*,*env*,*wfrec_rank_fm*)

ground_replacement_assm(*M*,*env*,*transrec_VFrom_fm*)
ground_replacement_assm(*M*,*env*,*eclose_closed_fm*)
ground_replacement_assm(*M*,*env*,*wfrec_Aleph_fm*)
ground_replacement_assm(*M*,*env*,*omap_replacement_fm*)

definition *instances3_fms* **where** *instances3_fms* ≡
{ *ground_repl_fm*(*ordtype_replacement_fm*),
ground_repl_fm(*wfrec_ordertype_fm*),
ground_repl_fm(*eclose_abs_fm*),
ground_repl_fm(*wfrec_rank_fm*),
ground_repl_fm(*transrec_VFrom_fm*),
ground_repl_fm(*eclose_closed_fm*),
ground_repl_fm(*wfrec_Aleph_fm*),
ground_repl_fm(*omap_replacement_fm*) }

This set has 8 internalized formulas, corresponding to the total count of previous replacement instances (apart from those 5 in *instances_ground_fms* and *instances_ground_notCH_fms*, and *dc_abs_fm*).

definition *overhead* **where**
overhead ≡ *instances1_fms* ∪ *instances_ground_fms*

definition *overhead_notCH* **where**
overhead_notCH ≡ *overhead* ∪ *instances2_fms* ∪
instances3_fms ∪ *instances_ground_notCH_fms*

definition *overhead_CH* **where**
overhead_CH ≡ *overhead_notCH* ∪ { *dc_abs_fm* }

Hence, the "overhead" to create a proper extension of a ctm by forcing consists of 7 replacement instances. To force $\neg CH$, 21 instances are need, and one further instance is required to force CH.

lemma *instances2_fms_type*[*TC*] : *instances2_fms* ⊆ *formula*
unfolding *instances2_fms_def replacement_instances2_defs*
by (*auto simp del*: *Lambda_in_M_fm_def*)

lemma *overhead_type*: *overhead* ⊆ *formula*
using *instances1_fms_type instances_ground_fms_type*
unfolding *overhead_def replacement_instances1_defs*
by *simp*

lemma *overhead_notCH_type*: *overhead_notCH* ⊆ *formula*
using *overhead_type*
unfolding *overhead_notCH_def rec_constr_abs_fm_def*
rec_constr_fm_def instances_ground_notCH_fms_def
instances2_fms_def instances3_fms_def
by (*auto simp*: *replacement_instances1_defs*
replacement_instances2_defs simp del: *Lambda_in_M_fm_def*)

lemma *overhead_CH_type*: *overhead_CH* ⊆ *formula*

using *overhead_notCH_type* **unfolding** *overhead_CH_def dc_abs_fm_def*
by *auto*

locale *M_ZF3_trans* = *M_ZF2_trans* + *M_ZF3*

locale *M_ZFC3* = *M_ZFC2* + *M_ZF3*

locale *M_ZFC3_trans* = *M_ZFC2_trans* + *M_ZF3_trans* + *M_ZFC3*

locale *M_ctm3* = *M_ctm2* + *M_ZF3_trans*

locale *M_ctm3_AC* = *M_ctm3* + *M_ctm1_AC* + *M_ZFC3_trans*

lemma *M_satT_imp_M_ZF2*: $(M \models ZF) \Longrightarrow M\_ZF1(M)$
proof -
assume $M \models ZF$
then
have *fin*: *upair_ax*(##*M*) *Union_ax*(##*M*) *power_ax*(##*M*)
extensionality(##*M*) *foundation_ax*(##*M*) *infinity_ax*(##*M*)
unfolding *ZF_def ZF_fin_def ZFC_fm_defs satT_def*
using *ZFC_fm_sats*[*of M*] **by** *simp_all*
{
fix φ *env*
assume $\varphi \in formula$ *env*∈*list*(*M*)
moreover from ‹$M \models ZF$›
have ∀ *p*∈*formula*. (*M*, [] ⊨ (*ZF_separation_fm*(*p*)))
∀ *p*∈*formula*. (*M*, [] ⊨ (*ZF_replacement_fm*(*p*)))
unfolding *ZF_def ZF_schemes_def* **by** *auto*
moreover from *calculation*
have *arity*(φ) ≤ *succ*(*length*(*env*)) ⟹ *separation*(##*M*, λ*x*. (*M*, *Cons*(*x*, *env*)
⊨ φ))
arity(φ) ≤ *succ*(*succ*(*length*(*env*))) ⟹ *strong_replacement*(##*M*,λ*x y*.
sats(*M*,φ,*Cons*(*x*,*Cons*(*y*, *env*))))
using *sats_ZF_separation_fm_iff sats_ZF_replacement_fm_iff*
unfolding *replacement_assm_def* **by** *simp_all*
}
with *fin*
show *M_ZF1*(*M*)
by *unfold_locales* (*simp_all add*:*replacement_assm_def ground_replacement_assm_def*)
qed

lemma *M_satT_imp_M_ZFC1*:
shows $(M \models ZFC) \longrightarrow M\_ZFC1(M)$
proof -
have $(M \models ZF) \wedge$ *choice_ax*(##*M*) ⟶ *M_ZFC1*(*M*)
using *M_satT_imp_M_ZF2*[*of M*]
unfolding *M_ZFC1_def M_ZC_basic_def M_ZF1_def M_AC_def*
by *auto*
then

```
  show ?thesis
    unfolding ZFC_def by auto
qed

lemma M_satT_instances1_imp_M_ZF1:
  assumes (M ⊨ ·Z· ∪ {·Replacement(p)· . p ∈ instances1_fms })
  shows M_ZF1(M)
proof -
  from assms
  have fin: upair_ax(##M) Union_ax(##M) power_ax(##M)
    extensionality(##M) foundation_ax(##M) infinity_ax(##M)
    unfolding ZF_fin_def Zermelo_fms_def ZFC_fm_defs satT_def
    using ZFC_fm_sats[of M] by simp_all
  moreover
  {
    fix φ env
    from ‹M ⊨ ·Z· ∪ {·Replacement(p)· . p ∈ instances1_fms }›
    have ∀ p∈formula. (M, [] ⊨ (ZF_separation_fm(p)))
      unfolding Zermelo_fms_def ZF_def instances1_fms_def
      by auto
    moreover
    assume φ ∈ formula env∈list(M)
    ultimately
   have arity(φ) ≤ succ(length(env)) ⟹ separation(##M, λx. (M, Cons(x, env)
⊨ φ))
      using sats_ZF_separation_fm_iff by simp_all
  }
  moreover
  {
    fix φ env
    assume φ ∈ instances1_fms env∈list(M)
   moreover from this and ‹M ⊨ ·Z· ∪ {·Replacement(p)· . p ∈ instances1_fms
}›
    have M, [] ⊨ ·Replacement(φ)· by auto
    ultimately
    have arity(φ) ≤ succ(succ(length(env))) ⟹ strong_replacement(##M,λx y.
sats(M,φ,Cons(x,Cons(y, env))))
      using sats_ZF_replacement_fm_iff[of φ] instances1_fms_type
      unfolding replacement_assm_def by auto
  }
  ultimately
  show ?thesis
    unfolding instances1_fms_def
  by unfold_locales (simp_all add:replacement_assm_def ground_replacement_assm_def)
qed

theorem M_satT_imp_M_ZF_ground_trans:
  assumes Transset(M) M ⊨ ·Z· ∪ {·Replacement(p)· . p ∈ overhead}
  shows M_ZF_ground_trans(M)
```

proof -
from ‹$M \models \cdot Z \cdot \cup \_$›
have $M \models \cdot Z \cdot \cup \{\cdot Replacement(p) \cdot . p \in instances1\_fms\}$
$M \models \{\cdot Replacement(p) \cdot . p \in instances\_ground\_fms\}$
unfolding *overhead_def* **by** *auto*
then
interpret *M_ZF1 M*
using *M_satT_instances1_imp_M_ZF1*
by *simp*
from ‹*Transset*(*M*)›
interpret *M_ZF1_trans M*
using *M_satT_imp_M_ZF2*
by *unfold_locales*
{
fix φ *env*
assume $\varphi \in instances\_ground\_fms$ *env*∈*list*(*M*)
moreover from *this* **and** ‹$M \models \{\cdot Replacement(p) \cdot . p \in instances\_ground\_fms\}$›
have $M, [] \models \cdot Replacement(\varphi) \cdot$ **by** *auto*
ultimately
have $arity(\varphi) \leq succ(succ(length(env))) \Longrightarrow strong\_replacement(\#\#M, \lambda x\ y.$
$sats(M, \varphi, Cons(x, Cons(y,\ env))))$
using *sats_ZF_replacement_fm_iff*[*of* φ] *instances_ground_fms_type*
unfolding *replacement_assm_def* **by** *auto*
}
then
show *?thesis*
unfolding *instances_ground_fms_def*
by *unfold_locales* (*simp_all add*:*replacement_assm_def*)
qed

theorem *M_satT_imp_M_ZF_ground_notCH_trans*:
assumes
Transset(*M*)
$M \models \cdot Z \cdot \cup \{\cdot Replacement(p) \cdot . p \in overhead\_notCH\}$
shows *M_ZF_ground_notCH_trans*(*M*)
proof -
from *assms*
interpret *M_ZF_ground_trans M*
using *M_satT_imp_M_ZF_ground_trans* **unfolding** *overhead_notCH_def*
by *force*
{
fix φ *env*
assume $\varphi \in instances\_ground\_notCH\_fms$ *env*∈*list*(*M*)
moreover from *this* **and** *assms*
have $M, [] \models \cdot Replacement(\varphi) \cdot$
unfolding *overhead_notCH_def* **by** *auto*
ultimately
have $arity(\varphi) \leq succ(succ(length(env))) \Longrightarrow strong\_replacement(\#\#M, \lambda x\ y.$
$sats(M, \varphi, Cons(x, Cons(y,\ env))))$

```
    using sats_ZF_replacement_fm_iff[of φ] instances_ground_notCH_fms_type
      unfolding replacement_assm_def by auto
  }
  then
  show ?thesis
   by unfold_locales (simp_all add:replacement_assm_def instances_ground_notCH_fms_def)
qed

theorem M_satT_imp_M_ZF_ground_CH_trans:
  assumes
    Transset(M)
    M ⊨ ·Z· ∪ {·Replacement(p)· . p ∈ overhead_CH }
  shows M_ZF_ground_CH_trans(M)
proof -
  from assms
  interpret M_ZF_ground_notCH_trans M
   using M_satT_imp_M_ZF_ground_notCH_trans unfolding overhead_CH_def
by auto
  {
    fix env
    assume env ∈ list(M)
    moreover from assms
    have M, [] ⊨ ·Replacement(dc_abs_fm)·
      unfolding overhead_CH_def by auto
    ultimately
    have arity(dc_abs_fm) ≤ succ(succ(length(env)))
        ⟹ strong_replacement(##M,λx y. sats(M,dc_abs_fm,Cons(x,Cons(y,
env))))
      using sats_ZF_replacement_fm_iff[of dc_abs_fm]
      unfolding replacement_assm_def
      by (auto simp:dc_abs_fm_def)
  }
  then
  show ?thesis
    by unfold_locales (simp_all add:replacement_assm_def)
qed

lemma (in M_Z_basic) M_satT_Zermelo_fms: M ⊨ ·Z·
  using upair_ax Union_ax power_ax extensionality foundation_ax
    infinity_ax separation_ax sats_ZF_separation_fm_iff
  unfolding Zermelo_fms_def ZF_fin_def
  by auto

lemma (in M_ZFC1) M_satT_ZC: M ⊨ ZC
  using upair_ax Union_ax power_ax extensionality foundation_ax
    infinity_ax separation_ax sats_ZF_separation_fm_iff choice_ax
  unfolding ZC_def Zermelo_fms_def ZF_fin_def
  by auto
```

locale *M_ZF = M_Z_basic +*
  assumes
    replacement_ax:replacement_assm(M,env,φ)

sublocale *M_ZF ⊆ M_ZF3*
  using *replacement_ax*
  by *unfold_locales (simp_all add:ground_replacement_assm_def)*

lemma *M_satT_imp_M_ZF: M ⊨ ZF ⟹ M_ZF(M)*
proof -
  assume *M ⊨ ZF*
  then
  have *fin: upair_ax(##M) Union_ax(##M) power_ax(##M)*
    extensionality(##M) foundation_ax(##M) infinity_ax(##M)
    unfolding *ZF_def ZF_fin_def ZFC_fm_defs satT_def*
    using *ZFC_fm_sats[of M]* **by** *simp_all*
  {
    fix *φ env*
    assume *φ ∈ formula env∈list(M)*
    moreover from *‹M ⊨ ZF›*
    have *∀ p∈formula. (M, [] ⊨ (ZF_separation_fm(p)))*
      ∀ p∈formula. (M, [] ⊨ (ZF_replacement_fm(p)))
      unfolding *ZF_def ZF_schemes_def* **by** *auto*
    moreover from *calculation*
   have *arity(φ) ≤ succ(length(env)) ⟹ separation(##M, λx. (M, Cons(x, env)*
⊨ φ))
        arity(φ) ≤ succ(succ(length(env))) ⟹ strong_replacement(##M,λx y.
sats(M,φ,Cons(x,Cons(y, env))))
      using *sats_ZF_separation_fm_iff sats_ZF_replacement_fm_iff*
      unfolding *replacement_assm_def* **by** *simp_all*
  }
  with *fin*
  show *M_ZF(M)*
   unfolding *M_ZF_def M_Z_basic_def M_ZF_axioms_def replacement_assm_def*
by *simp*
qed

lemma (**in** *M_ZF*) *M_satT_ZF: M ⊨ ZF*
  using *upair_ax Union_ax power_ax extensionality foundation_ax*
    infinity_ax separation_ax sats_ZF_separation_fm_iff
    replacement_ax sats_ZF_replacement_fm_iff
  unfolding *ZF_def ZF_schemes_def ZF_fin_def replacement_assm_def*
  by *auto*

lemma *M_ZF_iff_M_satT: M_ZF(M) ⟷ (M ⊨ ZF)*
  using *M_ZF.M_satT_ZF M_satT_imp_M_ZF*
  by *auto*

locale *M_ZFC = M_ZF + M_ZC_basic*

sublocale *M_ZFC* ⊆ *M_ZFC3*
by *unfold_locales*

lemma *M_ZFC_iff_M_satT*:
notes *iff_trans*[*trans*]
shows *M_ZFC*(*M*) ⟷ (*M* ⊨ *ZFC*)
proof -
have *M_ZFC*(*M*) ⟷ (*M* ⊨ *ZF*) ∧ *choice_ax*(##*M*)
using *M_ZF_iff_M_satT*
unfolding *M_ZFC_def M_ZC_basic_def M_AC_def M_ZF_def* **by** *auto*
also
have ... ⟷ *M* ⊨ *ZFC*
unfolding *ZFC_def* **by** *auto*
ultimately
show *?thesis* **by** *simp*
qed

lemma *M_satT_imp_M_ZF3*: (*M* ⊨ *ZF*) ⟶ *M_ZF3*(*M*)
proof
assume *M* ⊨ *ZF*
then
interpret *M_ZF M*
using *M_satT_imp_M_ZF* **by** *simp*
show *M_ZF3*(*M*)
by *unfold_locales*
qed

lemma *M_satT_imp_M_ZFC3*:
shows (*M* ⊨ *ZFC*) ⟶ *M_ZFC3*(*M*)
proof
assume *M* ⊨ *ZFC*
then
interpret *M_ZFC M*
using *M_ZFC_iff_M_satT* **by** *simp*
show *M_ZFC3*(*M*)
by *unfold_locales*
qed

lemma *M_satT_overhead_imp_M_ZF3*:
(*M* ⊨ *ZC* ∪ {·*Replacement*(*p*)· . *p* ∈ *overhead_notCH*}) ⟶ *M_ZFC3*(*M*)
proof
assume *M* ⊨ *ZC* ∪ {·*Replacement*(*p*)· . *p* ∈ *overhead_notCH*}
then
have *fin*: *upair_ax*(##*M*) *Union_ax*(##*M*) *power_ax*(##*M*) *choice_ax*(##*M*)
extensionality(##*M*) *foundation_ax*(##*M*) *infinity_ax*(##*M*)
unfolding *ZC_def ZF_fin_def Zermelo_fms_def ZFC_fm_defs satT_def*
using *ZFC_fm_sats*[*of M*] **by** *simp_all*
moreover

{
fix φ *env*
from ‹$M \models ZC \cup \{\cdot Replacement(p)\cdot \,.\, p \in overhead\_notCH\}$›
have $\forall\, p{\in}formula.\ (M,\, [] \models (ZF\_separation\_fm(p)))$
unfolding *ZC_def Zermelo_fms_def ZF_def* **by** *auto*
moreover
assume $\varphi \in formula\ env{\in}list(M)$
ultimately
have $arity(\varphi) \le succ(length(env)) \Longrightarrow separation(\#\#M,\, \lambda x.\ (M,\, Cons(x,\, env) \models \varphi))$
using *sats_ZF_separation_fm_iff* **by** *simp_all*
}
moreover
{
fix φ *env*
assume $\varphi \in overhead\_notCH\ env{\in}list(M)$
moreover from *this* **and** ‹$M \models ZC \cup \{\cdot Replacement(p)\cdot \,.\, p \in overhead\_notCH\}$›
have $M,\, [] \models \cdot Replacement(\varphi)\cdot$ **by** *auto*
ultimately
have $arity(\varphi) \le succ(succ(length(env))) \Longrightarrow strong\_replacement(\#\#M, \lambda x\ y.\ sats(M,\varphi,Cons(x,Cons(y,\, env))))$
using *sats_ZF_replacement_fm_iff*[*of* φ] *overhead_notCH_type*
unfolding *replacement_assm_def* **by** *auto*
}
ultimately
show *M_ZFC3*(M)
unfolding *overhead_def overhead_notCH_def instances1_fms_def instances2_fms_def instances3_fms_def*
by *unfold_locales* (*simp_all add*:*replacement_assm_def ground_replacement_assm_def*)
qed

end

10 Transitive set models of ZF

This theory defines locales for countable transitive models of *ZF*, and on top of that, one that includes a forcing notion. Weakened versions of both locales are included, that only assume finitely many replacement instances.

theory *Forcing_Data*
imports
Forcing_Notions
Cohen_Posets_Relative
ZF_Trans_Interpretations
begin

no_notation *Aleph* (‹ℵ_› [*90*] *90*)

10.1 A forcing locale and generic filters

Ideally, countability should be separated from the assumption of this locale. The fact is that our present proofs of the "definition of forces" (and many consequences) and of the lemma for "forcing a value" of function unnecessarily depend on the countability of the ground model.

locale *forcing_data1* = *forcing_notion* + *M_ctm1* +
  assumes *P_in_M*: $\mathbb{P} \in M$
    and *leq_in_M*: $leq \in M$

locale *forcing_data2* = *forcing_data1* + *M_ctm2_AC*

locale *forcing_data3* = *forcing_data2* + *M_ctm3_AC*

context *forcing_data1*
begin

lemma *P_sub_M* : $\mathbb{P} \subseteq M$
  using *transitivity P_in_M* **by** *auto*

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

declare *iff_trans* [*trans*]

lemma *M_generic_imp_filter*[*dest*]: $M\_generic(G) \Longrightarrow filter(G)$
  unfolding *M_generic_def* **by** *blast*

lemma *generic_filter_existence*:
  $p{\in}\mathbb{P} \Longrightarrow \exists\, G.\ p{\in}G \wedge M\_generic(G)$
proof -
  assume $p{\in}\mathbb{P}$
  let *?D*=$\lambda n{\in}nat.$ (*if* $(enum`n{\subseteq}\mathbb{P} \wedge dense(enum`n))$ *then* $enum`n$ *else* $\mathbb{P}$)
  have $\forall\, n{\in}nat.\ ?D`n \in Pow(\mathbb{P})$
    by *auto*
  then
  have *?D*:$nat \to Pow(\mathbb{P})$
    using *lam_type* **by** *auto*
  have $\forall\, n{\in}nat.\ dense(?D`n)$
  proof(*intro ballI*)
    fix *n*
    assume $n{\in}nat$
    then
     have $dense(?D`n) \longleftrightarrow dense(\textit{if } enum`n \subseteq \mathbb{P} \wedge dense(enum`n) \textit{ then } enum`n \textit{ else } \mathbb{P})$
      by *simp*
    also
    have ... $\longleftrightarrow$ $(\neg(enum`n \subseteq \mathbb{P} \wedge dense(enum`n)) \longrightarrow dense(\mathbb{P}))$

```
    using split_if by simp
  finally
  show dense(?D‘n)
    using P_dense ‹n∈nat› by auto
 qed
 with ‹?D∈_›
 interpret cg: countable_generic ℙ leq 𝟭 ?D
  by (unfold_locales, auto)
 from ‹p∈ℙ›
 obtain G where 1: p∈G ∧ filter(G) ∧ (∀ n∈nat.(?D‘n)∩G≠0)
  using cg.countable_rasiowa_sikorski[where M=λ_. M]  P_sub_M
      M_countable[THEN bij_is_fun]  M_countable[THEN bij_is_surj,  THEN
surj_range]
  unfolding cg.D_generic_def by blast
 then
 have (∀ D∈M. D⊆ℙ ∧ dense(D)⟶D∩G≠0)
 proof (intro ballI impI)
  fix D
  assume D∈M and 2: D ⊆ ℙ ∧ dense(D)
  moreover
  have ∀ y∈M. ∃ x∈nat. enum‘x= y
   using M_countable and  bij_is_surj unfolding surj_def by (simp)
  moreover from calculation
  obtain n where Eq10: n∈nat ∧ enum‘n = D
   by auto
  moreover from calculation if_P
  have ?D‘n = D
   by simp
  moreover
  note 1
  ultimately
  show D∩G≠0
   by auto
 qed
 with 1
 show ?thesis
  unfolding M_generic_def by auto
qed

lemma one_in_M: 𝟭 ∈ M
 using one_in_P P_in_M transitivity
 by simp

declare P_in_M [simp,intro]
declare one_in_M [simp,intro]
declare leq_in_M [simp,intro]
declare one_in_P [intro]

end — forcing_data1
```

locale *G_generic1 = forcing_data1 +*
  fixes *G :: i*
  assumes *generic : M_generic(G)*
begin

lemma *G_nonempty: G≠0*
  using *generic subset_refl[of ℙ] P_dense*
  unfolding *M_generic_def*
  by *auto*

lemma *M_genericD [dest]: x∈G ⟹ x∈ℙ*
  using *generic*
  by (*blast dest:filterD*)

lemma *M_generic_leqD [dest]: p∈G ⟹ q∈ℙ ⟹ p≼q ⟹ q∈G*
  using *generic*
  by (*blast dest:filter_leqD*)

lemma *M_generic_compatD [dest]: p∈G ⟹ r∈G ⟹ ∃ q∈G. q≼p ∧ q≼r*
  using *generic*
  by (*blast dest:low_bound_filter*)

lemma *M_generic_denseD [dest]: dense(D) ⟹ D⊆ℙ ⟹ D∈M ⟹ ∃ q∈G. q∈D*
  using *generic*
  unfolding *M_generic_def* **by** *blast*

lemma *G_subset_P: G⊆ℙ*
  using *generic* **by** *auto*

lemma *one_in_G : 1 ∈ G*
proof -
  have *increasing(G)*
    using *generic*
    unfolding *M_generic_def filter_def* **by** *simp*
  then
  show *?thesis*
    using *G_nonempty one_max*
    unfolding *increasing_def* **by** *blast*
qed

lemma *G_subset_M: G ⊆ M*
  using *generic transitivity[OF _ P_in_M]* **by** *auto*

end — *G_generic1*

locale *G_generic1_AC = G_generic1 + M_ctm1_AC*

end

11 The definition of *forces*

theory *Forces_Definition*
imports
Forcing_Data
begin

This is the core of our development.

11.1 The relation *frecrel*

lemma *names_belowsD*:
assumes $x \in names\_below(P,z)$
obtains *f n1 n2 p* **where**
$x = \langle f,n1,n2,p\rangle$ $f{\in}2$ $n1{\in}ecloseN(z)$ $n2{\in}ecloseN(z)$ $p{\in}P$
using *assms* **unfolding** *names_below_def* **by** *auto*

context *forcing_data1*
begin

lemma *ftype_abs*:
$[\![x{\in}M;\ y{\in}M\ ]\!] \Longrightarrow is\_ftype(\#\#M,x,y) \longleftrightarrow y = ftype(x)$
unfolding *ftype_def is_ftype_def* **by** (*simp add*:*absolut*)

lemma *name1_abs*:
$[\![x{\in}M;\ y{\in}M\ ]\!] \Longrightarrow is\_name1(\#\#M,x,y) \longleftrightarrow y = name1(x)$
unfolding *name1_def is_name1_def*
by (*rule is_hcomp_abs*[*OF fst_abs*],*simp_all add*: *fst_snd_closed*[*simplified*] *absolut*)

lemma *snd_snd_abs*:
$[\![x{\in}M;\ y{\in}M\ ]\!] \Longrightarrow is\_snd\_snd(\#\#M,x,y) \longleftrightarrow y = snd(snd(x))$
unfolding *is_snd_snd_def*
by (*rule is_hcomp_abs*[*OF snd_abs*],
simp_all add: *conjunct2*[*OF fst_snd_closed*,*simplified*] *absolut*)

lemma *name2_abs*:
$[\![x{\in}M;\ y{\in}M\ ]\!] \Longrightarrow is\_name2(\#\#M,x,y) \longleftrightarrow y = name2(x)$
unfolding *name2_def is_name2_def*
by (*rule is_hcomp_abs*[*OF fst_abs snd_snd_abs*],*simp_all add*:*absolut conjunct2*[*OF fst_snd_closed*,*simplified*])

lemma *cond_of_abs*:
$[\![x{\in}M;\ y{\in}M\ ]\!] \Longrightarrow is\_cond\_of(\#\#M,x,y) \longleftrightarrow y = cond\_of(x)$
unfolding *cond_of_def is_cond_of_def*
by (*rule is_hcomp_abs*[*OF snd_abs snd_snd_abs*];*simp_all add*:*fst_snd_closed*[*simplified*])

lemma *tuple_abs*:
$[\![z{\in}M;t1{\in}M;t2{\in}M;p{\in}M;t{\in}M]\!] \Longrightarrow$

is_tuple(##M,z,t1,t2,p,t) $\longleftrightarrow$ $t = \langle z,t1,t2,p\rangle$
unfolding *is_tuple_def* **using** *pair_in_M_iff* **by** *simp*

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

lemma *comp_in_M*:
$p \preceq q \Longrightarrow p{\in}M$
$p \preceq q \Longrightarrow q{\in}M$
using *transitivity*[*of _ leq*] *pair_in_M_iff* **by** *auto*

lemma *eq_case_abs* [*simp*]:
assumes *t1*$\in$*M t2*$\in$*M p*$\in$*M f*$\in$*M*
shows *is_eq_case(##M,t1,t2,p,*$\mathbb{P}$*,leq,f)* $\longleftrightarrow$ *eq_case(t1,t2,p,*$\mathbb{P}$*,leq,f)*
proof -
have $q \preceq p \Longrightarrow q{\in}M$ **for** *q*
using *comp_in_M* **by** *simp*
moreover
have $\langle s,y\rangle{\in}t \Longrightarrow s{\in}domain(t)$ **if** $t{\in}M$ **for** *s y t*
using *that* **unfolding** *domain_def* **by** *auto*
ultimately
have
$(\forall\, s{\in}M.\ s \in domain(t1) \vee s \in domain(t2) \longrightarrow (\forall\, q{\in}M.\ q{\in}\mathbb{P} \wedge q \preceq p \longrightarrow$
$(f\ `\ \langle 1,\ s,\ t1,\ q\rangle = 1 \longleftrightarrow f\ `\ \langle 1,\ s,\ t2,\ q\rangle = 1))) \longleftrightarrow$
$(\forall\, s.\ s \in domain(t1) \vee s \in domain(t2) \longrightarrow (\forall\, q.\ q{\in}\mathbb{P} \wedge q \preceq p \longrightarrow$
$(f\ `\ \langle 1,\ s,\ t1,\ q\rangle = 1 \longleftrightarrow f\ `\ \langle 1,\ s,\ t2,\ q\rangle = 1)))$
using *assms domain_trans*[*OF trans_M,of t1*] *domain_trans*[*OF trans_M,of t2*]
by *auto*
then
show *?thesis*
unfolding *eq_case_def is_eq_case_def*
using *assms pair_in_M_iff nat_into_M domain_closed apply_closed zero_in_M Un_closed*
by (*simp add*:*components_abs*)
qed

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,*$\mathbb{P}$*,leq,f)* $\longleftrightarrow$ *mem_case(t1,t2,p,*$\mathbb{P}$*,leq,f)*
proof
{
fix *v*
assume $v{\in}\mathbb{P}$ $v \preceq p$ *is_mem_case(##M,t1,t2,p,*$\mathbb{P}$*,leq,f)*
moreover
from *this*
have $v{\in}M$ $\langle v,p\rangle \in M$ $(\#\#M)(v)$

```
    using transitivity[OF _ P_in_M,of v] transitivity[OF _ leq_in_M]
    by simp_all
  moreover
  from calculation assms
  obtain q r s where
    r ∈ ℙ ∧ q ∈ ℙ ∧ ⟨q, v⟩ ∈ M ∧ ⟨s, r⟩ ∈ M ∧ ⟨q, r⟩ ∈ M ∧ 0 ∈ M ∧
     ⟨0, t1, s, q⟩ ∈ M ∧ q ≼ v ∧ ⟨s, r⟩ ∈ t2 ∧ q ≼ r ∧ f ‘ ⟨0, t1, s, q⟩ = 1
    unfolding is_mem_case_def by (auto simp add:components_abs)
  then
  have ∃ q s r. r ∈ ℙ ∧ q ∈ ℙ ∧ q ≼ v ∧ ⟨s, r⟩ ∈ t2 ∧ q ≼ r ∧ f ‘ ⟨0, t1, s, q⟩ = 1
    by auto
 }
 then
 show mem_case(t1, t2, p, ℙ, leq, f) if is_mem_case(##M, t1, t2, p, ℙ, leq, f)
  unfolding mem_case_def using that assms by auto
next
 { fix v
  assume v ∈ M v ∈ ℙ ⟨v, p⟩ ∈ M v ≼ p mem_case(t1, t2, p, ℙ, leq, f)
  moreover
  from this
  obtain q s r where r ∈ ℙ ∧ q ∈ ℙ ∧ q ≼ v ∧ ⟨s, r⟩ ∈ t2 ∧ q ≼ r ∧ f ‘ ⟨0, t1,
s, q⟩ = 1
    unfolding mem_case_def by auto
  moreover
  from this ‹t2∈M›
  have r∈M q∈M s∈M r ∈ ℙ ∧ q ∈ ℙ ∧ q ≼ v ∧ ⟨s, r⟩ ∈ t2 ∧ q ≼ r ∧ f ‘ ⟨0,
t1, s, q⟩ = 1
    using transitivity domainI[of s r] domain_closed
    by auto
  moreover
  note ‹t1∈M›
  ultimately
  have ∃ q∈M . ∃ s∈M. ∃ r∈M.
       r ∈ ℙ ∧ q ∈ ℙ ∧ ⟨q, v⟩ ∈ M ∧ ⟨s, r⟩ ∈ M ∧ ⟨q, r⟩ ∈ M ∧ 0 ∈ M ∧
       ⟨0, t1, s, q⟩ ∈ M ∧ q ≼ v ∧ ⟨s, r⟩ ∈ t2 ∧ q ≼ r ∧ f ‘ ⟨0, t1, s, q⟩ = 1
    using pair_in_M_iff zero_in_M by auto
 }
 then
 show is_mem_case(##M, t1, t2, p, ℙ, leq, f) if mem_case(t1, t2, p, ℙ, leq, f)
  unfolding is_mem_case_def
  using assms that zero_in_M pair_in_M_iff apply_closed nat_into_M
  by (auto simp add:components_abs)
qed

lemma Hfrc_abs:
 ⟦fnnc∈M; f∈M⟧ ⟹
  is_Hfrc(##M,ℙ,leq,fnnc,f) ⟷ Hfrc(ℙ,leq,fnnc,f)
 unfolding is_Hfrc_def Hfrc_def using pair_in_M_iff zero_in_M
 by (auto simp add:components_abs)
```

lemma *Hfrc_at_abs*:
 ⟦*fnnc*∈*M*; *f*∈*M* ; *z*∈*M*⟧ ⟹
  is_Hfrc_at(##*M*,ℙ,*leq*,*fnnc*,*f*,*z*) ⟷ *z* = *bool_of_o*(*Hfrc*(ℙ,*leq*,*fnnc*,*f*))
 unfolding *is_Hfrc_at_def* **using** *Hfrc_abs*
 by *auto*

lemma *components_closed* :
 x∈*M* ⟹ (##*M*)(*ftype*(*x*))
 x∈*M* ⟹ (##*M*)(*name1*(*x*))
 x∈*M* ⟹ (##*M*)(*name2*(*x*))
 x∈*M* ⟹ (##*M*)(*cond_of*(*x*))
 unfolding *ftype_def name1_def name2_def cond_of_def* **using** *fst_snd_closed*
by *simp_all*

lemma *ecloseN_closed*:
 (##*M*)(*A*) ⟹ (##*M*)(*ecloseN*(*A*))
 (##*M*)(*A*) ⟹ (##*M*)(*eclose_n*(*name1*,*A*))
 (##*M*)(*A*) ⟹ (##*M*)(*eclose_n*(*name2*,*A*))
 unfolding *ecloseN_def eclose_n_def*
 using *components_closed eclose_closed singleton_closed Un_closed* **by** *auto*

lemma *eclose_n_abs* :
 assumes *x*∈*M ec*∈*M*
 shows *is_eclose_n*(##*M*,*is_name1*,*ec*,*x*) ⟷ *ec* = *eclose_n*(*name1*,*x*)
  is_eclose_n(##*M*,*is_name2*,*ec*,*x*) ⟷ *ec* = *eclose_n*(*name2*,*x*)
 unfolding *is_eclose_n_def eclose_n_def*
using *assms name1_abs name2_abs eclose_abs singleton_closed components_closed*
 by *auto*

lemma *ecloseN_abs* :
 ⟦*x*∈*M*;*ec*∈*M*⟧ ⟹ *is_ecloseN*(##*M*,*x*,*ec*) ⟷ *ec* = *ecloseN*(*x*)
 unfolding *is_ecloseN_def ecloseN_def*
 using *eclose_n_abs Un_closed union_abs ecloseN_closed*
 by *auto*

lemma *frecR_abs* :
 x∈*M* ⟹ *y*∈*M* ⟹ *frecR*(*x*,*y*) ⟷ *is_frecR*(##*M*,*x*,*y*)
 unfolding *frecR_def is_frecR_def*
 using *zero_in_M domain_closed Un_closed components_closed nat_into_M*
 by (*auto simp add*: *components_abs*)

lemma *frecrelP_abs* :
 z∈*M* ⟹ *frecrelP*(##*M*,*z*) ⟷ (∃ *x y*. *z* = ⟨*x*,*y*⟩ ∧ *frecR*(*x*,*y*))
 using *pair_in_M_iff frecR_abs* **unfolding** *frecrelP_def* **by** *auto*

lemma *frecrel_abs*:
 assumes *A*∈*M r*∈*M*

shows *is_frecrel*(*##M*,*A*,*r*) $\longleftrightarrow$ $r = frecrel(A)$
proof -
from ‹$A \in M$›
have $z \in M$ **if** $z \in A \times A$ **for** z
using *cartprod_closed transitivity that* **by** *simp*
then
have $Collect(A \times A, frecrelP(\#\#M)) = Collect(A \times A, \lambda z.\ (\exists x\ y.\ z = \langle x,y \rangle \wedge frecR(x,y)))$
using *Collect_cong*[*of* $A \times A$ $A \times A$ *frecrelP*(*##M*)] *assms frecrelP_abs* **by** *simp*
with *assms*
show *?thesis*
unfolding *is_frecrel_def def_frecrel* **using** *cartprod_closed*
by *simp*
qed

lemma *frecrel_closed*:
assumes $x \in M$
shows $frecrel(x) \in M$
proof -
have $Collect(x \times x, \lambda z.\ (\exists x\ y.\ z = \langle x,y \rangle \wedge frecR(x,y))) \in M$
using *Collect_in_M*[*of frecrelP_fm*(*0*) []] *arity_frecrelP_fm sats_frecrelP_fm frecrelP_abs* ‹$x \in M$› *cartprod_closed*
by *simp*
then
show *?thesis*
unfolding *frecrel_def Rrel_def frecrelP_def* **by** *simp*
qed

lemma *field_frecrel* : $field(frecrel(names\_below(\mathbb{P},x))) \subseteq names\_below(\mathbb{P},x)$
unfolding *frecrel_def*
using *field_Rrel* **by** *simp*

lemma *forcerelD* : $uv \in forcerel(\mathbb{P},x) \Longrightarrow uv \in names\_below(\mathbb{P},x) \times names\_below(\mathbb{P},x)$
unfolding *forcerel_def*
using *trancl_type field_frecrel* **by** *blast*

lemma *wf_forcerel* :
$wf(forcerel(\mathbb{P},x))$
unfolding *forcerel_def* **using** *wf_trancl wf_frecrel* **.**

lemma *restrict_trancl_forcerel*:
assumes $frecR(w,y)$
shows $restrict(f, frecrel(names\_below(\mathbb{P},x)) \text{-`` } \{y\}) `w$
$= restrict(f, forcerel(\mathbb{P},x) \text{-`` } \{y\}) `w$
unfolding *forcerel_def frecrel_def* **using** *assms restrict_trancl_Rrel*[*of frecR*]
by *simp*

lemma *names_belowI* :
assumes $frecR(\langle ft,n1,n2,p \rangle, \langle a,b,c,d \rangle)$ $p \in \mathbb{P}$

```
  shows ⟨ft,n1,n2,p⟩ ∈ names_below(ℙ,⟨a,b,c,d⟩) (is ?x ∈ names_below(_,?y))
proof -
  from assms
  have ft ∈ 2 a ∈ 2
    unfolding frecR_def by (auto simp add:components_simp)
  from assms
  consider (eq) n1 ∈ domain(b) ∪ domain(c) ∧ (n2 = b ∨ n2 =c)
    | (mem) n1 = b ∧ n2 ∈ domain(c)
    unfolding frecR_def by (auto simp add:components_simp)
  then show ?thesis
  proof cases
    case eq
    then
    have n1 ∈ eclose(b) ∨ n1 ∈ eclose(c)
      using Un_iff in_dom_in_eclose by auto
    with eq
    have n1 ∈ ecloseN(?y) n2 ∈ ecloseN(?y)
      using ecloseNI components_in_eclose by auto
    with ‹ft∈2› ‹p∈ℙ›
    show ?thesis
      unfolding names_below_def by  auto
  next
    case mem
    then
    have n1 ∈ ecloseN(?y) n2 ∈ ecloseN(?y)
     using mem_eclose_trans ecloseNI in_dom_in_eclose components_in_eclose
      by auto
    with ‹ft∈2› ‹p∈ℙ›
    show ?thesis
      unfolding names_below_def
      by auto
  qed
qed

lemma names_below_tr :
  assumes x∈ names_below(ℙ,y) y∈ names_below(ℙ,z)
  shows x∈ names_below(ℙ,z)
proof -
  let ?A=λy . names_below(ℙ,y)
  note assms
  moreover from this
  obtain fx x1 x2 px where x = ⟨fx,x1,x2,px⟩ fx∈2 x1∈ecloseN(y) x2∈ecloseN(y)
px∈ℙ
    unfolding names_below_def by auto
  moreover from calculation
  obtain fy y1 y2 py where y = ⟨fy,y1,y2,py⟩ fy∈2 y1∈ecloseN(z) y2∈ecloseN(z)
py∈ℙ
    unfolding names_below_def by auto
  moreover from calculation
```

```
  have x1∈ecloseN(z) x2∈ecloseN(z)
    using ecloseN_mono names_simp by auto
  ultimately
  have x∈?A(z)
    unfolding names_below_def by simp
  then
  show ?thesis using subsetI by simp
qed

lemma arg_into_names_below2 :
  assumes ⟨x,y⟩ ∈ frecrel(names_below(ℙ,z))
  shows  x ∈ names_below(ℙ,y)
proof -
  from assms
  have x∈names_below(ℙ,z) y∈names_below(ℙ,z) frecR(x,y)
    unfolding frecrel_def Rrel_def
    by auto
  obtain f n1 n2 p where x = ⟨f,n1,n2,p⟩ f∈2 n1∈ecloseN(z) n2∈ecloseN(z) p∈ℙ
    using ‹x∈names_below(ℙ,z)›
    unfolding names_below_def by auto
  moreover
  obtain fy m1 m2 q where q∈ℙ y = ⟨fy,m1,m2,q⟩
    using ‹y∈names_below(ℙ,z)›
    unfolding names_below_def by auto
  moreover
  note ‹frecR(x,y)›
  ultimately
  show ?thesis
    using names_belowI by simp
qed

lemma arg_into_names_below :
  assumes ⟨x,y⟩ ∈ frecrel(names_below(ℙ,z))
  shows  x ∈ names_below(ℙ,x)
proof -
  from assms
  have x∈names_below(ℙ,z)
    unfolding frecrel_def Rrel_def
    by auto
  from ‹x∈names_below(ℙ,z)›
  obtain f n1 n2 p where
    x = ⟨f,n1,n2,p⟩ f∈2 n1∈ecloseN(z) n2∈ecloseN(z) p∈ℙ
    unfolding names_below_def by auto
  then
  have n1∈ecloseN(x) n2∈ecloseN(x)
    using components_in_eclose by simp_all
  with ‹f∈2› ‹p∈ℙ› ‹x = ⟨f,n1,n2,p⟩›
  show ?thesis
    unfolding names_below_def by simp
```

qed

lemma *forcerel_arg_into_names_below* :
assumes $\langle x,y\rangle \in forcerel(\mathbb{P},z)$
shows $x \in names\_below(\mathbb{P},x)$
using *assms*
unfolding *forcerel_def*
by(*rule trancl_induct*;*auto simp add*: *arg_into_names_below*)

lemma *names_below_mono* :
assumes $\langle x,y\rangle \in frecrel(names\_below(\mathbb{P},z))$
shows $names\_below(\mathbb{P},x) \subseteq names\_below(\mathbb{P},y)$
proof -
from *assms*
have $x \in names\_below(\mathbb{P},y)$
using *arg_into_names_below2* **by** *simp*
then
show *?thesis*
using *names_below_tr subsetI* **by** *simp*
qed

lemma *frecrel_mono* :
assumes $\langle x,y\rangle \in frecrel(names\_below(\mathbb{P},z))$
shows $frecrel(names\_below(\mathbb{P},x)) \subseteq frecrel(names\_below(\mathbb{P},y))$
unfolding *frecrel_def*
using *Rrel_mono names_below_mono assms* **by** *simp*

lemma *forcerel_mono2* :
assumes $\langle x,y\rangle \in frecrel(names\_below(\mathbb{P},z))$
shows $forcerel(\mathbb{P},x) \subseteq forcerel(\mathbb{P},y)$
unfolding *forcerel_def*
using *trancl_mono frecrel_mono assms* **by** *simp*

lemma *forcerel_mono_aux* :
assumes $\langle x,y\rangle \in frecrel(names\_below(\mathbb{P},\ w))^+$
shows $forcerel(\mathbb{P},x) \subseteq forcerel(\mathbb{P},y)$
using *assms*
by (*rule trancl_induct*,*simp_all add*: *subset_trans forcerel_mono2*)

lemma *forcerel_mono* :
assumes $\langle x,y\rangle \in forcerel(\mathbb{P},z)$
shows $forcerel(\mathbb{P},x) \subseteq forcerel(\mathbb{P},y)$
using *forcerel_mono_aux assms* **unfolding** *forcerel_def* **by** *simp*

lemma *forcerel_eq_aux*: $x \in names\_below(\mathbb{P},\ w) \Longrightarrow \langle x,y\rangle \in forcerel(\mathbb{P},z) \Longrightarrow$
$(y \in names\_below(\mathbb{P},\ w) \longrightarrow \langle x,y\rangle \in forcerel(\mathbb{P},w))$
unfolding *forcerel_def*
proof (*rule_tac* $a{=}x$ **and** $b{=}y$ **and**
$P{=}\lambda\ y\ .\ y \in names\_below(\mathbb{P},\ w) \longrightarrow \langle x,y\rangle \in frecrel(names\_below(\mathbb{P},w))^+$ **in**

```
trancl_induct,simp)
  let ?A=λ a . names_below(ℙ, a)
  let ?R=λ a . frecrel(?A(a))
  let ?fR=λ a .forcerel(a)
   show u∈?A(w) ⟶ ⟨x,u⟩∈?R(w)^+ if x∈?A(w) ⟨x,y⟩∈?R(z)^+ ⟨x,u⟩∈?R(z)
for  u
    using that frecrelD frecrelI r_into_trancl
    unfolding names_below_def by simp
  {
    fix u v
    assume x ∈ ?A(w)
      ⟨x, y⟩ ∈ ?R(z)^+
      ⟨x, u⟩ ∈ ?R(z)^+
      ⟨u, v⟩ ∈ ?R(z)
      u ∈ ?A(w) ⟹ ⟨x, u⟩ ∈ ?R(w)^+
    then
    have v ∈ ?A(w) ⟹ ⟨x, v⟩ ∈ ?R(w)^+
    proof -
      assume v ∈?A(w)
      from ‹⟨u,v⟩∈_›
      have u∈?A(v)
        using arg_into_names_below2 by simp
      with ‹v ∈?A(w)›
      have u∈?A(w)
        using names_below_tr by simp
      with ‹v∈_› ‹⟨u,v⟩∈_›
      have ⟨u,v⟩∈ ?R(w)
       using frecrelD frecrelI r_into_trancl unfolding names_below_def by simp
      with ‹u ∈ ?A(w) ⟹ ⟨x, u⟩ ∈ ?R(w)^+› ‹u∈?A(w)›
      have ⟨x, u⟩ ∈ ?R(w)^+
        by simp
      with ‹⟨u,v⟩∈ ?R(w)›
      show ⟨x,v⟩∈ ?R(w)^+ using trancl_trans r_into_trancl
        by simp
    qed
  }
  then
  show v ∈ ?A(w) ⟶ ⟨x, v⟩ ∈ ?R(w)^+
    if x ∈ ?A(w)
      ⟨x, y⟩ ∈ ?R(z)^+
      ⟨x, u⟩ ∈ ?R(z)^+
      ⟨u, v⟩ ∈ ?R(z)
      u ∈ ?A(w) ⟶ ⟨x, u⟩ ∈ ?R(w)^+ for u v
    using that
    by simp
qed

lemma forcerel_eq :
  assumes ⟨z,x⟩ ∈ forcerel(ℙ,x)
```

shows *forcerel*(ℙ,*z*) = *forcerel*(ℙ,*x*) ∩ *names_below*(ℙ,*z*)×*names_below*(ℙ,*z*)
using *assms forcerel_eq_aux forcerelD forcerel_mono*[*of z x x*] *subsetI*
by *auto*

lemma *forcerel_below_aux* :
assumes ⟨*z*,*x*⟩ ∈ *forcerel*(ℙ,*x*) ⟨*u*,*z*⟩ ∈ *forcerel*(ℙ,*x*)
shows *u* ∈ *names_below*(ℙ,*z*)
using *assms*(*2*)
unfolding *forcerel_def*
proof(*rule trancl_induct*)
show *u* ∈ *names_below*(ℙ,*y*) **if** ⟨*u*, *y*⟩ ∈ *frecrel*(*names_below*(ℙ, *x*)) **for** *y*
using *that vimage_singleton_iff arg_into_names_below2* **by** *simp*
next
show *u* ∈ *names_below*(ℙ,*z*)
if ⟨*u*, *y*⟩ ∈ *frecrel*(*names_below*(ℙ, *x*))^+
⟨*y*, *z*⟩ ∈ *frecrel*(*names_below*(ℙ, *x*))
u ∈ *names_below*(ℙ, *y*)
for *y z*
using *that arg_into_names_below2*[*of y z x*] *names_below_tr* **by** *simp*
qed

lemma *forcerel_below* :
assumes ⟨*z*,*x*⟩ ∈ *forcerel*(ℙ,*x*)
shows *forcerel*(ℙ,*x*) -`` {*z*} ⊆ *names_below*(ℙ,*z*)
using *vimage_singleton_iff assms forcerel_below_aux* **by** *auto*

lemma *relation_forcerel* :
shows *relation*(*forcerel*(ℙ,*z*)) *trans*(*forcerel*(ℙ,*z*))
unfolding *forcerel_def* **using** *relation_trancl trans_trancl* **by** *simp_all*

lemma *Hfrc_restrict_trancl*: *bool_of_o*(*Hfrc*(ℙ, *leq*, *y*, *restrict*(*f*,*frecrel*(*names_below*(ℙ,*x*))-``{*y*})))
= *bool_of_o*(*Hfrc*(ℙ, *leq*, *y*, *restrict*(*f*,(*frecrel*(*names_below*(ℙ,*x*))^+)-``{*y*})))
unfolding *Hfrc_def bool_of_o_def eq_case_def mem_case_def*
using *restrict_trancl_forcerel frecRI1 frecRI2 frecRI3*
unfolding *forcerel_def*
by *simp*

lemma *frc_at_trancl*: *frc_at*(ℙ,*leq*,*z*) = *wfrec*(*forcerel*(ℙ,*z*),*z*,λ*x f*. *bool_of_o*(*Hfrc*(ℙ,*leq*,*x*,*f*)))
unfolding *frc_at_def forcerel_def* **using** *wf_eq_trancl Hfrc_restrict_trancl* **by**
simp

lemma *forcerelI1* :
assumes *n1* ∈ *domain*(*b*) ∨ *n1* ∈ *domain*(*c*) *p*∈ℙ *d*∈ℙ
shows ⟨⟨*1*, *n1*, *b*, *p*⟩, ⟨*0*,*b*,*c*,*d*⟩⟩∈ *forcerel*(ℙ,⟨*0*,*b*,*c*,*d*⟩)
proof -
let *?x*=⟨*1*, *n1*, *b*, *p*⟩
let *?y*=⟨*0*,*b*,*c*,*d*⟩
from *assms*

```
  have frecR(?x,?y)
    using frecRI1 by simp
  then
  have ?x∈names_below(ℙ,?y) ?y ∈ names_below(ℙ,?y)
    using names_belowI  assms components_in_eclose
    unfolding names_below_def by auto
  with ‹frecR(?x,?y)›
  show ?thesis
    unfolding forcerel_def frecrel_def
    using subsetD[OF r_subset_trancl[OF relation_Rrel]] RrelI
    by auto
qed

lemma forcerelI2 :
  assumes n1 ∈ domain(b) ∨ n1 ∈ domain(c) p∈ℙ d∈ℙ
  shows ⟨⟨1, n1, c, p⟩, ⟨0,b,c,d⟩⟩∈ forcerel(ℙ,⟨0,b,c,d⟩)
proof -
  let ?x=⟨1, n1, c, p⟩
  let ?y=⟨0,b,c,d⟩
  note assms
  moreover from this
  have frecR(?x,?y)
    using frecRI2 by simp
  moreover from calculation
  have ?x∈names_below(ℙ,?y) ?y ∈ names_below(ℙ,?y)
    using names_belowI components_in_eclose
    unfolding names_below_def by auto
  ultimately
  show ?thesis
    unfolding forcerel_def frecrel_def
    using subsetD[OF r_subset_trancl[OF relation_Rrel]] RrelI
    by auto
qed

lemma forcerelI3 :
  assumes ⟨n2, r⟩ ∈ c p∈ℙ d∈ℙ r ∈ ℙ
  shows ⟨⟨0, b, n2, p⟩,⟨1, b, c, d⟩⟩ ∈ forcerel(ℙ,⟨1,b,c,d⟩)
proof -
  let ?x=⟨0, b, n2, p⟩
  let ?y=⟨1, b, c, d⟩
  note assms
  moreover from this
  have frecR(?x,?y)
    using frecRI3 by simp
  moreover from calculation
  have ?x∈names_below(ℙ,?y)  ?y ∈ names_below(ℙ,?y)
    using names_belowI components_in_eclose
    unfolding names_below_def by auto
  ultimately
```

show *?thesis*
unfolding *forcerel_def frecrel_def*
using *subsetD*[*OF r_subset_trancl*[*OF relation_Rrel*]] *RrelI*
by *auto*
qed

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

lemma *aux_def_frc_at*:
assumes $z \in$ *forcerel*($\mathbb{P}$,x) -`` $\{x\}$
shows *wfrec*(*forcerel*($\mathbb{P}$,x), z, H) = *wfrec*(*forcerel*($\mathbb{P}$,z), z, H)
proof -
let *?A*=*names_below*($\mathbb{P}$,z)
from *assms*
have $\langle z,x\rangle \in$ *forcerel*($\mathbb{P}$,x)
using *vimage_singleton_iff* **by** *simp*
moreover from *this*
have $z \in$ *?A*
using *forcerel_arg_into_names_below* **by** *simp*
moreover from *calculation*
have *forcerel*($\mathbb{P}$,z) = *forcerel*($\mathbb{P}$,x) $\cap$ (*?A*$\times$*?A*)
forcerel($\mathbb{P}$,x) -`` $\{z\} \subseteq$ *?A*
using *forcerel_eq forcerel_below*
by *auto*
moreover from *calculation*
have *wfrec*(*forcerel*($\mathbb{P}$,x), z, H) = *wfrec*[*?A*](*forcerel*($\mathbb{P}$,x), z, H)
using *wfrec_trans_restr*[*OF relation_forcerel*(*1*) *wf_forcerel relation_forcerel*(*2*),
of x z ?A]
by *simp*
ultimately
show *?thesis*
using *wfrec_restr_eq* **by** *simp*
qed

11.2 Recursive expression of *frc_at*

lemma *def_frc_at* :
assumes $p \in \mathbb{P}$
shows
frc_at($\mathbb{P}$,*leq*,$\langle$*ft*,*n1*,*n2*,$p\rangle$) =
bool_of_o($p \in \mathbb{P} \wedge$
(*ft* = *0* $\wedge$ ($\forall$ *s*. *s*$\in$*domain*(*n1*) $\cup$ *domain*(*n2*) $\longrightarrow$
($\forall$ *q*. *q*$\in\mathbb{P} \wedge$ *q* $\preceq$ *p* $\longrightarrow$ (*frc_at*($\mathbb{P}$,*leq*,$\langle$*1*,*s*,*n1*,*q*$\rangle$) =*1* $\longleftrightarrow$ *frc_at*($\mathbb{P}$,*leq*,$\langle$*1*,*s*,*n2*,*q*$\rangle$)
=*1*)))
$\vee$ *ft* = *1* $\wedge$ ($\forall$ *v*$\in\mathbb{P}$. *v* $\preceq$ *p* $\longrightarrow$
($\exists$ *q*. $\exists$ *s*. $\exists$ *r*. *r*$\in\mathbb{P} \wedge$ *q*$\in\mathbb{P} \wedge$ *q* $\preceq$ *v* $\wedge$ $\langle$*s*,*r*$\rangle \in$ *n2* $\wedge$ *q* $\preceq$ *r* $\wedge$ *frc_at*($\mathbb{P}$,*leq*,$\langle$*0*,*n1*,*s*,*q*$\rangle$)
= *1*))))

proof -
let *?r*=λy. *forcerel*($\mathbb{P}$,*y*) **and** *?Hf*=$\lambda x\ f$. *bool_of_o*(*Hfrc*($\mathbb{P}$,*leq*,*x*,*f*))
let *?t*=λy. *?r*(*y*) -`` {*y*}
let *?arg*=$\langle$*ft*,*n1*,*n2*,*p*$\rangle$
from *wf_forcerel*
have *wfr*: $\forall\, w$. *wf*(*?r*(*w*)) **..**
with *wfrec* [*of ?r*(*?arg*) *?arg ?Hf*]
have *frc_at*($\mathbb{P}$,*leq*,*?arg*) = *?Hf*(*?arg*, $\lambda x{\in}$*?r*(*?arg*) -`` {*?arg*}. *wfrec*(*?r*(*?arg*), *x*, *?Hf*))
using *frc_at_trancl* **by** *simp*
also
have ... = *?Hf*(*?arg*, $\lambda x{\in}$*?r*(*?arg*) -`` {*?arg*}. *frc_at*($\mathbb{P}$,*leq*,*x*))
using *aux_def_frc_at frc_at_trancl* **by** *simp*
finally
show *?thesis*
unfolding *Hfrc_def mem_case_def eq_case_def*
using *forcerelI assms*
by *auto*
qed

11.3 Absoluteness of *frc_at*

lemma *forcerel_in_M* :
assumes $x{\in}M$
shows *forcerel*($\mathbb{P}$,*x*)$\in M$
unfolding *forcerel_def def_frecrel names_below_def*
proof -
let *?Q* = *2* $\times$ *ecloseN*(*x*) $\times$ *ecloseN*(*x*) $\times$ $\mathbb{P}$
have *?Q* $\times$ *?Q* $\in M$
using ‹*x*$\in$*M*› *nat_into_M ecloseN_closed cartprod_closed* **by** *simp*
moreover
have *separation*(*##M*,λz. *frecrelP*(*##M*,*z*))
using *separation_in_ctm*[*of frecrelP_fm*(*0*),*OF* _ _ _ *sats_frecrelP_fm*]
arity_frecrelP_fm frecrelP_fm_type
by *auto*
moreover from *this*
have *separation*(*##M*,λz. $\exists\, x\ y$. $z = \langle x,\ y\rangle \wedge$ *frecR*(x, y))
using *separation_cong*[*OF frecrelP_abs*]
by *force*
ultimately
show {$z \in$ *?Q* $\times$ *?Q* . $\exists\, x\ y$. $z = \langle x,\ y\rangle \wedge$ *frecR*(x, y)}$\widehat{\ }+ \in M$
using *separation_closed frecrelP_abs trancl_closed*
by *simp*
qed

lemma *relation2_Hfrc_at_abs*:
relation2(*##M*,*is_Hfrc_at*(*##M*,$\mathbb{P}$,*leq*),$\lambda x\ f$. *bool_of_o*(*Hfrc*($\mathbb{P}$,*leq*,*x*,*f*)))
unfolding *relation2_def* **using** *Hfrc_at_abs*
by *simp*

lemma *Hfrc_at_closed* :
 $\forall x{\in}M.\ \forall g{\in}M.\ function(g) \longrightarrow bool\_of\_o(Hfrc(\mathbb{P},leq,x,g)){\in}M$
 unfolding *bool_of_o_def* **using** *zero_in_M nat_into_M*[*of 1*] **by** *simp*

lemma *wfrec_Hfrc_at* :
 assumes $X{\in}M$
 shows *wfrec_replacement*(##*M*,*is_Hfrc_at*(##*M*,$\mathbb{P}$,*leq*),*forcerel*($\mathbb{P}$,*X*))
proof -
 have *0*:*is_Hfrc_at*(##*M*,$\mathbb{P}$,*leq*,*a*,*b*,*c*) $\longleftrightarrow$
 sats(*M*,*Hfrc_at_fm*(*8*,*9*,*2*,*1*,*0*),[*c*,*b*,*a*,*d*,*e*,*y*,*x*,*z*,$\mathbb{P}$,*leq*,*forcerel*($\mathbb{P}$,*X*)])
 if $a{\in}M$ $b{\in}M$ $c{\in}M$ $d{\in}M$ $e{\in}M$ $y{\in}M$ $x{\in}M$ $z{\in}M$
 for *a b c d e y x z*
 using *that* ‹$X{\in}M$› *forcerel_in_M*
 Hfrc_at_iff_sats[*of concl*:*M* $\mathbb{P}$ *leq a b c 8 9 2 1 0*]
 by *simp*
 have *1*:*sats*(*M*,*is_wfrec_fm*(*Hfrc_at_fm*(*8*,*9*,*2*,*1*,*0*),*5*,*1*,*0*),[*y*,*x*,*z*,$\mathbb{P}$,*leq*,*forcerel*($\mathbb{P}$,*X*)])
$\longleftrightarrow$
 is_wfrec(##*M*, *is_Hfrc_at*(##*M*,$\mathbb{P}$,*leq*),*forcerel*($\mathbb{P}$,*X*), *x*, *y*)
 if $x{\in}M$ $y{\in}M$ $z{\in}M$ **for** *x y z*
 using *that* ‹$X{\in}M$› *forcerel_in_M sats_is_wfrec_fm*[*OF 0*]
 by *simp*
 let
 ?f=*Exists*(*And*(*pair_fm*(*1*,*0*,*2*),*is_wfrec_fm*(*Hfrc_at_fm*(*8*,*9*,*2*,*1*,*0*),*5*,*1*,*0*)))
 have *satsf*:*sats*(*M*, *?f*, [*x*,*z*,$\mathbb{P}$,*leq*,*forcerel*($\mathbb{P}$,*X*)]) $\longleftrightarrow$
 ($\exists\, y{\in}M.$ *pair*(##*M*,*x*,*y*,*z*) & *is_wfrec*(##*M*, *is_Hfrc_at*(##*M*,$\mathbb{P}$,*leq*),*forcerel*($\mathbb{P}$,*X*),
x, *y*))
 if $x{\in}M$ $z{\in}M$ **for** *x z*
 using *that 1* ‹$X{\in}M$› *forcerel_in_M* **by** (*simp del*:*pair_abs*)
 have *artyf*:*arity*(*?f*) = *5*
 using *arity_wfrec_replacement_fm*[**where** *p*=*Hfrc_at_fm*(*8*,*9*,*2*,*1*,*0*) **and**
i=*10*]
 arity_Hfrc_at_fm ord_simp_union
 by *simp*
 moreover
 have *?f*∈*formula* **by** *simp*
 ultimately
 have *strong_replacement*(##*M*,$\lambda x\ z.$ *sats*(*M*,*?f*,[*x*,*z*,$\mathbb{P}$,*leq*,*forcerel*($\mathbb{P}$,*X*)]))
 using *ZF_ground_replacements*(*1*) *1 artyf* ‹$X{\in}M$› *forcerel_in_M*
 unfolding *replacement_assm_def wfrec_Hfrc_at_fm_def* **by** *simp*
 then
 have *strong_replacement*(##*M*,$\lambda x\ z.$
 $\exists\, y{\in}M.$ *pair*(##*M*,*x*,*y*,*z*) & *is_wfrec*(##*M*, *is_Hfrc_at*(##*M*,$\mathbb{P}$,*leq*),*forcerel*($\mathbb{P}$,*X*),
x, *y*))
 using *repl_sats*[*of M ?f* [$\mathbb{P}$,*leq*,*forcerel*($\mathbb{P}$,*X*)]] *satsf* **by** (*simp del*:*pair_abs*)
 then
 show *?thesis* **unfolding** *wfrec_replacement_def* **by** *simp*
qed

lemma *names_below_abs* :
 ⟦*Q*∈*M*;*x*∈*M*;*nb*∈*M*⟧ ⟹ *is_names_below*(##*M*,*Q*,*x*,*nb*) ⟷ *nb* = *names_below*(*Q*,*x*)
 unfolding *is_names_below_def names_below_def*
 using *succ_in_M_iff zero_in_M cartprod_closed ecloseN_abs ecloseN_closed*
 by *auto*

lemma *names_below_closed*:
 ⟦*Q*∈*M*;*x*∈*M*⟧ ⟹ *names_below*(*Q*,*x*) ∈ *M*
 unfolding *names_below_def*
 using *zero_in_M cartprod_closed ecloseN_closed succ_in_M_iff*
 by *simp*

lemma *names_below_productE* :
 assumes *Q* ∈ *M x* ∈ *M*
 ⋀*A1 A2 A3 A4*. *A1* ∈ *M* ⟹ *A2* ∈ *M* ⟹ *A3* ∈ *M* ⟹ *A4* ∈ *M* ⟹ *R*(*A1* × *A2* × *A3* × *A4*)
 shows *R*(*names_below*(*Q*,*x*))
 unfolding *names_below_def* **using** *assms nat_into_M ecloseN_closed*[*of x*] **by** *auto*

lemma *forcerel_abs* :
 ⟦*x*∈*M*;*z*∈*M*⟧ ⟹ *is_forcerel*(##*M*,ℙ,*x*,*z*) ⟷ *z* = *forcerel*(ℙ,*x*)
 unfolding *is_forcerel_def forcerel_def*
 using *frecrel_abs names_below_abs trancl_abs ecloseN_closed names_below_closed*
 names_below_productE[*of concl*:λ*p*. *is_frecrel*(##*M*,*p*,_) ⟷ _ = *frecrel*(*p*)] *frecrel_closed*
 by *simp*

lemma *frc_at_abs*:
 assumes *fnnc*∈*M z*∈*M*
 shows *is_frc_at*(##*M*,ℙ,*leq*,*fnnc*,*z*) ⟷ *z* = *frc_at*(ℙ,*leq*,*fnnc*)
proof -
 from *assms*
 have (∃ *r*∈*M*. *is_forcerel*(##*M*,ℙ,*fnnc*, *r*) ∧ *is_wfrec*(##*M*, *is_Hfrc_at*(##*M*, ℙ, *leq*), *r*, *fnnc*, *z*))
 ⟷ *is_wfrec*(##*M*, *is_Hfrc_at*(##*M*, ℙ, *leq*), *forcerel*(ℙ,*fnnc*), *fnnc*, *z*)
 using *forcerel_abs forcerel_in_M* **by** *simp*
 then
 show *?thesis*
 unfolding *frc_at_trancl is_frc_at_def*
 using *assms wfrec_Hfrc_at*[*of fnnc*] *wf_forcerel relation_forcerel forcerel_in_M*
 Hfrc_at_closed relation2_Hfrc_at_abs
 trans_wfrec_abs[*of forcerel*(ℙ,*fnnc*) *fnnc z is_Hfrc_at*(##*M*,ℙ,*leq*) λ*x f*. *bool_of_o*(*Hfrc*(ℙ,*leq*,*x*,*f*))]
 by (*simp flip*:*setclass_iff*)
qed

lemma *forces_eq′_abs* :
 ⟦*p*∈*M* ; *t1*∈*M* ; *t2*∈*M*⟧ ⟹ *is_forces_eq′*(##*M*,ℙ,*leq*,*p*,*t1*,*t2*) ⟷ *forces_eq′*(ℙ,*leq*,*p*,*t1*,*t2*)

unfolding *is_forces_eq'_def forces_eq'_def*
using *frc_at_abs nat_into_M pair_in_M_iff* **by** (*auto simp add:components_abs*)

lemma *forces_mem'_abs* :
⟦*p∈M* ; *t1∈M* ; *t2∈M*⟧ ⟹ *is_forces_mem'*(*##M*,ℙ,*leq*,*p*,*t1*,*t2*) ⟷ *forces_mem'*(ℙ,*leq*,*p*,*t1*,*t2*)
unfolding *is_forces_mem'_def forces_mem'_def*
using *frc_at_abs nat_into_M pair_in_M_iff* **by** (*auto simp add:components_abs*)

lemma *forces_neq'_abs* :
assumes *p∈M t1∈M t2∈M*
shows *is_forces_neq'*(*##M*,ℙ,*leq*,*p*,*t1*,*t2*) ⟷ *forces_neq'*(ℙ,*leq*,*p*,*t1*,*t2*)
proof -
have *q∈M* **if** *q∈ℙ* **for** *q*
using *that transitivity* **by** *simp*
with *assms*
show *?thesis*
unfolding *is_forces_neq'_def forces_neq'_def*
using *forces_eq'_abs pair_in_M_iff*
by (*auto simp add:components_abs*,*blast*)
qed

lemma *forces_nmem'_abs* :
assumes *p∈M t1∈M t2∈M*
shows *is_forces_nmem'*(*##M*,ℙ,*leq*,*p*,*t1*,*t2*) ⟷ *forces_nmem'*(ℙ,*leq*,*p*,*t1*,*t2*)
proof -
have *q∈M* **if** *q∈ℙ* **for** *q*
using *that transitivity* **by** *simp*
with *assms*
show *?thesis*
unfolding *is_forces_nmem'_def forces_nmem'_def*
using *forces_mem'_abs pair_in_M_iff*
by (*auto simp add:components_abs*,*blast*)
qed

lemma *leq_abs*:
⟦ *l∈M* ; *q∈M* ; *p∈M* ⟧ ⟹ *is_leq*(*##M*,*l*,*q*,*p*) ⟷ ⟨*q*,*p*⟩∈*l*
unfolding *is_leq_def* **using** *pair_in_M_iff* **by** *simp*

11.4 Forcing for atomic formulas in context

definition
forces_eq :: [*i*,*i*,*i*] ⇒ *o* (‹_ *forces*$_a$ '(_ = _')› [*36*,*1*,*1*] *60*) **where**
forces_eq ≡ *forces_eq'*(ℙ,*leq*)

definition
forces_mem :: [*i*,*i*,*i*] ⇒ *o* (‹_ *forces*$_a$ '(_ ∈ _')› [*36*,*1*,*1*] *60*) **where**
forces_mem ≡ *forces_mem'*(ℙ,*leq*)

abbreviation *is_forces_eq*
where $is\_forces\_eq \equiv is\_forces\_eq'(\#\#M,\mathbb{P},leq)$

abbreviation
$is\_forces\_mem :: [i,i,i] \Rightarrow o$ **where**
$is\_forces\_mem \equiv is\_forces\_mem'(\#\#M,\mathbb{P},leq)$

lemma *def_forces_eq*: $p{\in}\mathbb{P} \Longrightarrow p\ forces_a\ (t1 = t2) \longleftrightarrow$
$(\forall\, s{\in}domain(t1) \cup domain(t2).\ \forall\, q.\ q{\in}\mathbb{P} \wedge q \preceq p \longrightarrow$
$(q\ forces_a\ (s \in t1) \longleftrightarrow q\ forces_a\ (s \in t2)))$
unfolding *forces_eq_def forces_mem_def forces_eq'_def forces_mem'_def*
using *def_frc_at[of p 0 t1 t2]*
unfolding *bool_of_o_def*
by *auto*

lemma *def_forces_mem*: $p{\in}\mathbb{P} \Longrightarrow p\ forces_a\ (t1 \in t2) \longleftrightarrow$
$(\forall\, v{\in}\mathbb{P}.\ v \preceq p \longrightarrow$
$(\exists\, q.\ \exists\, s.\ \exists\, r.\ r{\in}\mathbb{P} \wedge q{\in}\mathbb{P} \wedge q \preceq v \wedge \langle s,r\rangle \in t2 \wedge q \preceq r \wedge q\ forces_a\ (t1 = s)))$
unfolding *forces_eq'_def forces_mem'_def forces_eq_def forces_mem_def*
using *def_frc_at[of p 1 t1 t2]*
unfolding *bool_of_o_def*
by *auto*

lemma *forces_eq_abs* :
$[\![p{\in}M\ ;\ t1{\in}M\ ;\ t2{\in}M]\!] \Longrightarrow is\_forces\_eq(p,t1,t2) \longleftrightarrow p\ forces_a\ (t1 = t2)$
unfolding *forces_eq_def*
using *forces_eq'_abs* **by** *simp*

lemma *forces_mem_abs* :
$[\![p{\in}M\ ;\ t1{\in}M\ ;\ t2{\in}M]\!] \Longrightarrow is\_forces\_mem(p,t1,t2) \longleftrightarrow p\ forces_a\ (t1 \in t2)$
unfolding *forces_mem_def*
using *forces_mem'_abs*
by *simp*

definition
$forces\_neq :: [i,i,i] \Rightarrow o\ (\langle\_\ forces_a\ '(\_ \neq \_')\rangle\ [36,1,1]\ 60)$ **where**
$p\ forces_a\ (t1 \neq t2) \equiv \neg\ (\exists\, q{\in}\mathbb{P}.\ q{\preceq}p \wedge q\ forces_a\ (t1 = t2))$

definition
$forces\_nmem :: [i,i,i] \Rightarrow o\ (\langle\_\ forces_a\ '(\_ \notin \_')\rangle\ [36,1,1]\ 60)$ **where**
$p\ forces_a\ (t1 \notin t2) \equiv \neg\ (\exists\, q{\in}\mathbb{P}.\ q{\preceq}p \wedge q\ forces_a\ (t1 \in t2))$

lemma *forces_neq* :
$p\ forces_a\ (t1 \neq t2) \longleftrightarrow forces\_neq'(\mathbb{P},leq,p,t1,t2)$
unfolding *forces_neq_def forces_neq'_def forces_eq_def* **by** *simp*

lemma *forces_nmem* :
$p\ forces_a\ (t1 \notin t2) \longleftrightarrow forces\_nmem'(\mathbb{P},leq,p,t1,t2)$

unfolding *forces_nmem_def forces_nmem'_def forces_mem_def* **by** *simp*

abbreviation *Forces* :: $[i, i, i] \Rightarrow o$ $(\langle \_ \Vdash \_ \_ \rangle\ [36,36,36]\ 60)$ **where**
$p \Vdash \varphi\ env \equiv M, ([p,\mathbb{P},leq,\mathbf{1}]\ @\ env) \models forces(\varphi)$

lemma *sats_forces_Member* :
assumes $x{\in}nat\ y{\in}nat\ env{\in}list(M)$
$nth(x,env){=}xx\ nth(y,env){=}yy\ q{\in}M$
shows $q \Vdash \cdot x \in y \cdot\ env \longleftrightarrow q \in \mathbb{P} \wedge is\_forces\_mem(q, xx, yy)$
unfolding *forces_def*
using *assms*
by *simp*

lemma *sats_forces_Equal* :
assumes $a{\in}nat\ b{\in}nat\ env{\in}list(M)\ nth(a,env){=}x\ nth(b,env){=}y\ q{\in}M$
shows $q \Vdash \cdot a = b \cdot\ env \longleftrightarrow q \in \mathbb{P} \wedge is\_forces\_eq(q, x, y)$
unfolding *forces_def*
using *assms*
by *simp*

lemma *sats_forces_Nand* :
assumes $\varphi{\in}formula\ \psi{\in}formula\ env{\in}list(M)\ p{\in}M$
shows $p \Vdash \cdot\neg(\varphi \wedge \psi)\cdot\ env \longleftrightarrow$
$p{\in}\mathbb{P} \wedge \neg(\exists\, q{\in}M.\ q{\in}\mathbb{P} \wedge is\_leq(\#\#M,leq,q,p) \wedge (q \Vdash \varphi\ env) \wedge (q \Vdash \psi\ env))$
unfolding *forces_def*
using *sats_is_leq_fm_auto assms sats_ren_forces_nand zero_in_M*
by *simp*

lemma *sats_forces_Neg* :
assumes $\varphi{\in}formula\ env{\in}list(M)\ p{\in}M$
shows $p \Vdash \cdot\neg\varphi\cdot\ env \longleftrightarrow$
$(p{\in}\mathbb{P} \wedge \neg(\exists\, q{\in}M.\ q{\in}\mathbb{P} \wedge is\_leq(\#\#M,leq,q,p) \wedge (q \Vdash \varphi\ env)))$
unfolding *Neg_def* **using** *assms sats_forces_Nand*
by *simp*

lemma *sats_forces_Forall* :
assumes $\varphi{\in}formula\ env{\in}list(M)\ p{\in}M$
shows $p \Vdash (\cdot\forall\,\varphi\cdot)\ env \longleftrightarrow p \in \mathbb{P} \wedge (\forall\, x{\in}M.\ p \Vdash \varphi\ ([x]\ @\ env))$
unfolding *forces_def* **using** *assms sats_ren_forces_forall*
by *simp*

end — *forcing_data1*

end

12 Names and generic extensions

theory *Names*
imports

Forcing_Data
FrecR_Arities
ZF_Trans_Interpretations
begin

definition
Hv :: [*i*,*i*,*i*]⇒*i* **where**
Hv(*G*,*x*,*f*) ≡ { *z* . *y*∈ *domain*(*x*), (∃ *p*∈*G*. ⟨*y*,*p*⟩ ∈ *x*) ∧ *z*=*f*‘*y*}

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

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

definition
GenExt :: [*i*,*i*]⇒*i* (‹_[_]› [71,1])
where *M*[*G*] ≡ {*val*(*G*,*τ*). *τ* ∈ *M*}

lemma *map_val_in_MG*:
assumes
env∈*list*(*M*)
shows
map(*val*(*G*),*env*)∈*list*(*M*[*G*])
unfolding *GenExt_def* **using** *assms map_type2* **by** *simp*

12.1 Values and check-names

context *forcing_data1*
begin

lemma *name_components_in_M*:
assumes ⟨*σ*,*p*⟩∈*ϑ* *ϑ* ∈ *M*
shows *σ*∈*M* *p*∈*M*
using *assms transitivity pair_in_M_iff*
by *auto*

definition
Hcheck :: [*i*,*i*] ⇒ *i* **where**
Hcheck(*z*,*f*) ≡ { ⟨*f*‘*y*,**1**⟩ . *y* ∈ *z*}

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

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

lemma *Hcheck_trancl*:*Hcheck*(*y*, *restrict*(*f*,*Memrel*(*eclose*({*x*}))-‘‘{*y*}))
= *Hcheck*(*y*, *restrict*(*f*,(*Memrel*(*eclose*({*x*}))^+)-‘‘{*y*}))
unfolding *Hcheck_def*
using *restrict_trans_eq* **by** *simp*

lemma *check_trancl*: *check*(*x*) = *wfrec*(*rcheck*(*x*), *x*, *Hcheck*)
using *checkD wf_eq_trancl Hcheck_trancl* **unfolding** *rcheck_def* **by** *simp*

lemma *rcheck_in_M* : *x* ∈ *M* ⟹ *rcheck*(*x*) ∈ *M*
unfolding *rcheck_def* **by** (*simp flip*: *setclass_iff*)

lemma *rcheck_subset_M* : *x* ∈ *M* ⟹ *field*(*rcheck*(*x*)) ⊆ *eclose*({*x*})
unfolding *rcheck_def* **using** *field_Memrel field_trancl* **by** *auto*

lemma *aux_def_check*: *x* ∈ *y* ⟹
wfrec(*Memrel*(*eclose*({*y*})), *x*, *Hcheck*) =
wfrec(*Memrel*(*eclose*({*x*})), *x*, *Hcheck*)
by (*rule wfrec_eclose_eq*,*auto simp add*: *arg_into_eclose eclose_sing*)

lemma *def_check* : *check*(*y*) = { ⟨*check*(*w*),**1**⟩ . *w* ∈ *y*}
proof -
let
?r=λ*y*. *Memrel*(*eclose*({*y*}))
have *wfr*: ∀ *w* . *wf*(*?r*(*w*))
using *wf_Memrel* **..**
then
have *check*(*y*)= *Hcheck*(*y*, λ*x*∈*?r*(*y*) -‘‘ {*y*}. *wfrec*(*?r*(*y*), *x*, *Hcheck*))
using *wfrec*[*of ?r*(*y*) *y Hcheck*] *checkD* **by** *simp*
also
have ... = *Hcheck*(*y*, λ*x*∈*y*. *wfrec*(*?r*(*y*), *x*, *Hcheck*))
using *under_Memrel_eclose arg_into_eclose* **by** *simp*
also
have ... = *Hcheck*(*y*, λ*x*∈*y*. *check*(*x*))
using *aux_def_check checkD* **by** *simp*
finally
show *?thesis*
using *Hcheck_def* **by** *simp*
qed

lemma *def_checkS* :
fixes *n*
assumes *n* ∈ *nat*
shows *check*(*succ*(*n*)) = *check*(*n*) ∪ {⟨*check*(*n*),**1**⟩}
proof -
have *check*(*succ*(*n*)) = {⟨*check*(*i*),**1**⟩ . *i* ∈ *succ*(*n*)}
using *def_check* **by** *blast*
also
have ... = {⟨*check*(*i*),**1**⟩ . *i* ∈ *n*} ∪ {⟨*check*(*n*),**1**⟩}
by *blast*

also
have ... = $check(n) \cup \{\langle check(n),\mathbf{1}\rangle\}$
using *def_check*[*of n*,*symmetric*] **by** *simp*
finally
show *?thesis* **.**
qed

lemma *field_Memrel2* :
assumes $x \in M$
shows $field(Memrel(eclose(\{x\}))) \subseteq M$
proof -
have $field(Memrel(eclose(\{x\}))) \subseteq eclose(\{x\})$ $eclose(\{x\}) \subseteq M$
using *Ordinal.Memrel_type field_rel_subset assms eclose_least*[*OF trans_M*]
by *auto*
then
show *?thesis*
using *subset_trans* **by** *simp*
qed

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

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

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

```
  let
    ?r=λτ . edrel(eclose({τ}))
  let
    ?f=λz∈?r(x)-‘‘{x}. wfrec(?r(x),z,Hv(G))
  have ∀τ. wf(?r(τ))
    using wf_edrel by simp
  with wfrec [of _ x]
  have val(G,x) = Hv(G,x,?f)
    using val_def by simp
  also
  have  ... = Hv(G,x,λz∈domain(x). wfrec(?r(x),z,Hv(G)))
    using dom_under_edrel_eclose by simp
  also
  have  ... = Hv(G,x,λz∈domain(x). val(G,z))
    using aux_def_val val_def by simp
  finally
  show ?thesis
    using Hv_def by simp
qed

lemma val_mono : x⊆y ⟹ val(G,x) ⊆ val(G,y)
  by (subst (1 2) def_val, force)
```

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

```
lemma val_check : 1 ∈ G ⟹  1 ∈ ℙ ⟹ val(G,check(y))  = y
proof (induct rule:eps_induct)
  case (1 y)
  then show ?case
  proof -
    have check(y) = { ⟨check(w), 1⟩ . w ∈ y}  (is _ = ?C)
      using def_check .
    then
    have val(G,check(y)) = val(G, {⟨check(w), 1⟩ . w ∈ y})
      by simp
    also
    have  ...  = {z . t∈domain(?C) , (∃ p∈G .  ⟨t, p⟩∈?C ) ∧ z=val(G,t) }
      using def_val by blast
    also
    have  ... =  {z . t∈domain(?C) , (∃ w∈y. t=check(w)) ∧ z=val(G,t) }
      using 1 by simp
    also
    have  ... = {val(G,check(w)) . w∈y }
      by force
    finally
    show val(G,check(y)) = y
      using 1 by simp
  qed
qed
```

lemma *val_of_name* :
val(*G*,{*x*∈*A*×ℙ. *Q*(*x*)}) = {*z* . *t*∈*A* , (∃ *p*∈ℙ . *Q*(⟨*t*,*p*⟩) ∧ *p* ∈ *G*) ∧ *z*=*val*(*G*,*t*)}
proof -
let
?n={*x*∈*A*×ℙ. *Q*(*x*)} **and**
?r=λτ . *edrel*(*eclose*({τ}))
let
?f=λ*z*∈*?r*(*?n*)-‘‘{*?n*}. *val*(*G*,*z*)
have
wfR : *wf*(*?r*(τ)) **for** τ
by (*simp add*: *wf_edrel*)
have *domain*(*?n*) ⊆ *A* **by** *auto*
{ fix *t*
assume *H*:*t* ∈ *domain*({*x* ∈ *A* × ℙ . *Q*(*x*)})
then have *?f* ‘ *t* = (*if t* ∈ *?r*(*?n*)-‘‘{*?n*} *then val*(*G*,*t*) *else 0*)
by *simp*
moreover have ... = *val*(*G*,*t*)
using *dom_under_edrel_eclose H if_P* **by** *auto*
}
then
have *Eq1*: *t* ∈ *domain*({*x* ∈ *A* × ℙ . *Q*(*x*)}) ⟹ *val*(*G*,*t*) = *?f*‘ *t* **for** *t*
by *simp*
have *val*(*G*,*?n*) = {*z* . *t*∈*domain*(*?n*), (∃ *p* ∈ *G* . ⟨*t*,*p*⟩ ∈ *?n*) ∧ *z*=*val*(*G*,*t*)}
by (*subst def_val*,*simp*)
also
have ... = {*z* . *t*∈*domain*(*?n*), (∃ *p*∈ℙ . ⟨*t*,*p*⟩∈*?n* ∧ *p*∈*G*) ∧ *z*=*?f*‘*t*}
unfolding *Hv_def*
by (*auto simp add*:*Eq1*)
also
have ... = {*z* . *t*∈*domain*(*?n*), (∃ *p*∈ℙ . ⟨*t*,*p*⟩∈*?n* ∧ *p*∈*G*) ∧ *z*=(*if t*∈*?r*(*?n*)-‘‘{*?n*}
then val(*G*,*t*) *else 0*)}
by (*simp*)
also
have ... = { *z* . *t*∈*domain*(*?n*), (∃ *p*∈ℙ . ⟨*t*,*p*⟩∈*?n* ∧ *p*∈*G*) ∧ *z*=*val*(*G*,*t*)}
proof -
have *domain*(*?n*) ⊆ *?r*(*?n*)-‘‘{*?n*}
using *dom_under_edrel_eclose* **by** *simp*
then
have ∀ *t*∈*domain*(*?n*). (*if t*∈*?r*(*?n*)-‘‘{*?n*} *then val*(*G*,*t*) *else 0*) = *val*(*G*,*t*)
by *auto*
then
show { *z* . *t*∈*domain*(*?n*), (∃ *p*∈ℙ . ⟨*t*,*p*⟩∈*?n* ∧ *p*∈*G*) ∧ *z*=(*if t*∈*?r*(*?n*)-‘‘{*?n*}
then val(*G*,*t*) *else 0*)} =
{ *z* . *t*∈*domain*(*?n*), (∃ *p*∈ℙ . ⟨*t*,*p*⟩∈*?n* ∧ *p*∈*G*) ∧ *z*=*val*(*G*,*t*)}
by *auto*
qed
also
have ... = { *z* . *t*∈*A*, (∃ *p*∈ℙ . ⟨*t*,*p*⟩∈*?n* ∧ *p*∈*G*) ∧ *z*=*val*(*G*,*t*)}

by *force*
finally
show $val(G,?n) = \{ z . t{\in}A, (\exists\, p{\in}\mathbb{P} . Q(\langle t,p\rangle) \wedge p{\in}G) \wedge z{=}val(G,t)\}$
by *auto*
qed

lemma *val_of_name_alt* :
$val(G,\{x{\in}A{\times}\mathbb{P}.\ Q(x)\}) = \{z . t{\in}A , (\exists\, p{\in}\mathbb{P}{\cap}G .\ \ Q(\langle t,p\rangle)) \wedge z{=}val(G,t) \}$
using *val_of_name* **by** *force*

lemma *val_only_names*: $val(F,\tau) = val(F,\{x{\in}\tau.\ \exists\, t{\in}domain(\tau).\ \exists\, p{\in}F.\ x{=}\langle t,p\rangle\})$
(**is** $\_ = val(F,?name)$)
proof -
have $val(F,?name) = \{z . t{\in}domain(?name), (\exists\, p{\in}F.\ \langle t,\ p\rangle \in ?name) \wedge z{=}val(F,$
$t)\}$
using *def_val* **by** *blast*
also
have $... = \{val(F,\ t).\ t{\in}\{y{\in}domain(\tau).\ \exists\, p{\in}F.\ \langle y,\ p\rangle \in \tau\ \}\}$
by *blast*
also
have $... = \{z . t{\in}domain(\tau), (\exists\, p{\in}F.\ \langle t,\ p\rangle \in \tau) \wedge z{=}val(F,\ t)\}$
by *blast*
also
have $... = val(F,\ \tau)$
using *def_val*[*symmetric*] **by** *blast*
finally
show *?thesis* **..**
qed

lemma *val_only_pairs*: $val(F,\tau) = val(F,\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\})$
proof
have $val(F,\tau) = val(F,\{x{\in}\tau.\ \exists\, t{\in}domain(\tau).\ \exists\, p{\in}F.\ x{=}\langle t,p\rangle\})$ (**is** $\_ = val(F,?name)$)
using *val_only_names* **.**
also
have $... \subseteq val(F,\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\})$
using *val_mono*[*of ?name* $\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\}$] **by** *auto*
finally
show $val(F,\tau) \subseteq val(F,\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\})$ **by** *simp*
next
show $val(F,\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\}) \subseteq val(F,\tau)$
using *val_mono*[*of* $\{x{\in}\tau.\ \exists\, t\ p.\ x{=}\langle t,p\rangle\}$] **by** *auto*
qed

lemma *val_subset_domain_times_range*: $val(F,\tau) \subseteq val(F,domain(\tau){\times}range(\tau))$
using *val_only_pairs*[*THEN equalityD1*]
val_mono[*of* $\{x \in \tau . \exists\, t\ p.\ x = \langle t,\ p\rangle\}$ $domain(\tau){\times}range(\tau)$] **by** *blast*

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

```
  assume ⟨ϑ,p⟩ ∈ π
  then
  have ϑ∈domain(π)
    by auto
  assume p∈G
  with ‹ϑ∈domain(π)› ‹⟨ϑ,p⟩ ∈ π›
  have val(G,ϑ) ∈ {z . t∈domain(π) , (∃ p∈G .  ⟨t, p⟩∈π) ∧ z=val(G,t) }
    by auto
  then
  show ?thesis
    by (subst def_val)
qed

lemma elem_of_val: x∈val(G,π) ⟹ ∃ ϑ∈domain(π). val(G,ϑ) = x
  by (subst (asm) def_val,auto)

lemma elem_of_val_pair: x∈val(G,π) ⟹ ∃ ϑ. ∃ p∈G.  ⟨ϑ,p⟩∈π ∧ val(G,ϑ) = x
  by (subst (asm) def_val,auto)

lemma elem_of_val_pair':
  assumes π∈M x∈val(G,π)
  shows ∃ ϑ∈M. ∃ p∈G.  ⟨ϑ,p⟩∈π ∧ val(G,ϑ) = x
proof -
  from assms
  obtain ϑ p where p∈G ⟨ϑ,p⟩∈π val(G,ϑ) = x
    using elem_of_val_pair by blast
  moreover from this ‹π∈M›
  have ϑ∈M
    using pair_in_M_iff[THEN iffD1, THEN conjunct1, simplified]
      transitivity by blast
  ultimately
  show ?thesis
    by blast
qed

lemma GenExtD: x ∈ M[G] ⟹ ∃ τ∈M. x = val(G,τ)
  by (simp add:GenExt_def)

lemma GenExtI: x ∈ M ⟹ val(G,x) ∈ M[G]
  by (auto simp add: GenExt_def)

lemma Transset_MG : Transset(M[G])
proof -
  { fix vc y
    assume vc ∈ M[G] and y ∈ vc
    then
    obtain c where c∈M val(G,c)∈M[G] y ∈ val(G,c)
      using GenExtD by auto
    from ‹y ∈ val(G,c)›
```

```
    obtain ϑ where ϑ∈domain(c) val(G,ϑ) = y
      using elem_of_val by blast
    with trans_M ‹c∈M›
    have y ∈ M[G]
      using domain_trans GenExtI by blast
  }
  then
  show ?thesis
    using Transset_def by auto
qed
```

lemmas *transitivity_MG = Transset_intf[OF Transset_MG]*

This lemma can be proved before having *check_in_M*. At some point Miguel naïvely thought that the *check_in_M* could be proved using this argument.

```
lemma check_nat_M :
  assumes n ∈ nat
  shows check(n) ∈ M
  using assms
proof (induct n)
  case 0
  then
  show ?case
    using zero_in_M by (subst def_check,simp)
next
  case (succ x)
  have 1 ∈ M
    using one_in_P P_sub_M subsetD by simp
  with ‹check(x)∈M›
  have ⟨check(x),1⟩ ∈ M
    using pair_in_M_iff by simp
  then
  have {⟨check(x),1⟩} ∈ M
    using singleton_closed by simp
  with ‹check(x)∈M›
  have check(x) ∪ {⟨check(x),1⟩} ∈ M
    using Un_closed by simp
  then
  show ?case
    using ‹x∈nat› def_checkS by simp
qed

lemma def_PHcheck:
  assumes
    z∈M f∈M
  shows
    Hcheck(z,f) = Replace(z,PHcheck(##M,1,f))
proof -
  from assms
```

```
  have ⟨f‘x,1⟩ ∈ M f‘x∈M if x∈z for x
    using pair_in_M_iff transitivity that apply_closed by simp_all
  then
  have {y . x ∈ z, y = ⟨f ‘ x, 1⟩} =  {y . x ∈ z, y = ⟨f ‘ x, 1⟩ ∧ y∈M ∧ f‘x∈M}
    by simp
  then
  show ?thesis
    using ‹z∈M› ‹f∈M› transitivity
    unfolding Hcheck_def PHcheck_def RepFun_def
    by auto
qed


lemma wfrec_Hcheck :
  assumes X∈M
  shows wfrec_replacement(##M,is_Hcheck(##M,1),rcheck(X))
proof -
  let ?f=Exists(And(pair_fm(1,0,2),
              is_wfrec_fm(is_Hcheck_fm(8,2,1,0),4,1,0)))
  have is_Hcheck(##M,1,a,b,c) ⟷
        sats(M,is_Hcheck_fm(8,2,1,0),[c,b,a,d,e,y,x,z,1,rcheck(x)])
    if a∈M b∈M c∈M d∈M e∈M y∈M x∈M z∈M
    for a b c d e y x z
    using that ‹X∈M› rcheck_in_M is_Hcheck_iff_sats zero_in_M
    by simp
  then
  have sats(M,is_wfrec_fm(is_Hcheck_fm(8,2,1,0),4,1,0), [y,x,z,1,rcheck(X)])
        ⟷ is_wfrec(##M, is_Hcheck(##M,1),rcheck(X), x, y)
    if x∈M y∈M z∈M for x y z
    using that sats_is_wfrec_fm ‹X∈M› rcheck_in_M zero_in_M
    by simp
  moreover from this
  have satsf:sats(M, ?f, [x,z,1,rcheck(X)]) ⟷
         (∃ y∈M. pair(##M,x,y,z) & is_wfrec(##M, is_Hcheck(##M,1),rcheck(X),
x, y))
    if x∈M z∈M for x z
    using that ‹X∈M› rcheck_in_M
    by (simp del:pair_abs)
  moreover
  have artyf:arity(?f) = 4
    using arity_wfrec_replacement_fm[where p=is_Hcheck_fm(8, 2, 1, 0) and
i=9]
      arity_is_Hcheck_fm ord_simp_union
    by simp
  ultimately
  have strong_replacement(##M,λx z. sats(M,?f,[x,z,1,rcheck(X)]))
    using ZF_ground_replacements(2) artyf ‹X∈M› rcheck_in_M
    unfolding replacement_assm_def wfrec_Hcheck_fm_def by simp
  then
```

have *strong_replacement*(*##M*,$\lambda x\ z$.
$\exists\, y{\in}M.\ pair(\#\#M,x,y,z)\ \&\ is\_wfrec(\#\#M,\ is\_Hcheck(\#\#M,\mathbf{1}),rcheck(X),$
$x,\ y)$)
using *repl_sats*[*of M ?f* [**1**,*rcheck*(*X*)]] *satsf* **by** (*simp del*:*pair_abs*)
then
show *?thesis*
unfolding *wfrec_replacement_def* **by** *simp*
qed

lemma *Hcheck_closed'* : $f{\in}M \Longrightarrow z{\in}M \Longrightarrow \{f\ `\ x\ .\ x \in z\} \in M$
using *RepFun_closed*[*OF lam_replacement_imp_strong_replacement*]
lam_replacement_apply apply_closed transM[*of _ z*]
by *simp*

lemma *repl_PHcheck* :
assumes $f{\in}M$
shows *lam_replacement*(*##M*,$\lambda x.$ *Hcheck*(*x*,*f*))
proof -
have *Hcheck*(*x*,*f*) = $\{f`y\ .\ y{\in}x\}{\times}\{\mathbf{1}\}$ **for** *x*
unfolding *Hcheck_def* **by** *auto*
moreover
note *assms*
moreover from *this*
have *1*:*lam_replacement*(*##M*, $\lambda x\ .\ \{f`y\ .\ y{\in}x\}{\times}\{\mathbf{1}\}$)
using *lam_replacement_RepFun_apply*
lam_replacement_constant lam_replacement_fst lam_replacement_snd
singleton_closed cartprod_closed fst_snd_closed Hcheck_closed'
by (*rule_tac lam_replacement_CartProd*[*THEN* [*5*] *lam_replacement_hcomp2*],*simp_all*)
ultimately
show *?thesis*
using *singleton_closed cartprod_closed Hcheck_closed'*
by(*rule_tac lam_replacement_cong*[*OF 1*],*auto*)
qed

lemma *univ_PHcheck* : $[\![\ z{\in}M\ ;\ f{\in}M\ ]\!] \Longrightarrow univalent(\#\#M,z,PHcheck(\#\#M,\mathbf{1},f))$
unfolding *univalent_def PHcheck_def*
by *simp*

lemma *PHcheck_closed* : $[\![z{\in}M\ ;\ f{\in}M\ ;\ x{\in}z;\ PHcheck(\#\#M,\mathbf{1},f,x,y)\ ]\!] \Longrightarrow$
$(\#\#M)(y)$
unfolding *PHcheck_def* **by** *simp*

lemma *relation2_Hcheck* : *relation2*(*##M*,*is_Hcheck*(*##M*,**1**),*Hcheck*)
proof -
have *is_Replace*(*##M*,*z*,*PHcheck*(*##M*,**1**,*f*),*hc*) $\longleftrightarrow$ $hc = Replace(z,PHcheck(\#\#M,\mathbf{1},f))$
if $z{\in}M\ f{\in}M\ hc{\in}M$ **for** *z f hc*
using *that Replace_abs*[*OF _ _ univ_PHcheck*] *PHcheck_closed*[*of z f*]
by *simp*
with *def_PHcheck*

show *?thesis*
unfolding *relation2_def is_Hcheck_def Hcheck_def*
by *simp*
qed

lemma *Hcheck_closed* : $\forall\, y{\in}M.\ \forall\, g{\in}M.\ \ Hcheck(y,g){\in}M$
proof -
have *eq*:$Hcheck(x,f) = \{f`y\ .\ y{\in}x\}{\times}\{\mathbf{1}\}$ **for** $f\ x$
unfolding *Hcheck_def* **by** *auto*
then
have $Hcheck(y,g){\in}M$ **if** $y{\in}M\ g{\in}M$ **for** $y\ g$
using *eq that Hcheck_closed' cartprod_closed singleton_closed*
by *simp*
then
show *?thesis*
by *auto*
qed

lemma *wf_rcheck* : $x{\in}M \Longrightarrow wf(rcheck(x))$
unfolding *rcheck_def* **using** *wf_trancl*[*OF wf_Memrel*] **.**

lemma *trans_rcheck* : $x{\in}M \Longrightarrow trans(rcheck(x))$
unfolding *rcheck_def* **using** *trans_trancl* **.**

lemma *relation_rcheck* : $x{\in}M \Longrightarrow relation(rcheck(x))$
unfolding *rcheck_def* **using** *relation_trancl* **.**

lemma *check_in_M* : $x{\in}M \Longrightarrow check(x) \in M$
using *wfrec_Hcheck*[*of x*] *check_trancl wf_rcheck trans_rcheck relation_rcheck rcheck_in_M*
Hcheck_closed relation2_Hcheck trans_wfrec_closed[*of rcheck*(x)]
by *simp*

lemma *rcheck_abs*[*Rel*] : $[\![\ x{\in}M\ ;\ r{\in}M\ ]\!] \Longrightarrow is\_rcheck(\#\#M,x,r) \longleftrightarrow r = rcheck(x)$
unfolding *rcheck_def is_rcheck_def*
using *singleton_closed trancl_closed Memrel_closed eclose_closed zero_in_M*
by *simp*

lemma *check_abs*[*Rel*] :
assumes $x{\in}M\ z{\in}M$
shows $is\_check(\#\#M,\mathbf{1},x,z) \longleftrightarrow z = check(x)$
proof -
have $is\_check(\#\#M,\mathbf{1},x,z) \longleftrightarrow is\_wfrec(\#\#M,is\_Hcheck(\#\#M,\mathbf{1}),rcheck(x),x,z)$
unfolding *is_check_def*
using *assms rcheck_abs rcheck_in_M zero_in_M*
unfolding *check_trancl is_check_def*
by *simp*

then
show *?thesis*
unfolding *check_trancl*
using *assms wfrec_Hcheck*[*of x*] *wf_rcheck trans_rcheck relation_rcheck rcheck_in_M*
Hcheck_closed relation2_Hcheck trans_wfrec_abs[*of rcheck*(*x*) *x z is_Hcheck*(*##M*,**1**)
Hcheck]
by (*simp flip*: *setclass_iff*)
qed

lemma *check_lam_replacement*: *lam_replacement*(*##M*,*check*)
proof -
have *arity*(*check_fm*(*2*,*0*,*1*)) = *3*
by (*simp add*:*ord_simp_union arity*)
then
have *Lambda*(*A*, *check*) ∈ *M* **if** *A*∈*M* **for** *A*
using *that check_in_M transitivity*[*of _ A*]
sats_check_fm check_abs zero_in_M
check_fm_type ZF_ground_replacements(*3*)
by(*rule_tac Lambda_in_M* [*of check_fm*(*2*,*0*,*1*) [**1**]],*simp_all*)
then
show *?thesis*
using *check_in_M lam_replacement_iff_lam_closed*[*THEN iffD2*]
by *simp*
qed

lemma *check_replacement*: {*check*(*x*). *x*∈ℙ} ∈ *M*
using *lam_replacement_imp_strong_replacement_aux*[*OF check_lam_replacement*]
transitivity check_in_M RepFun_closed
by *simp_all*

lemma *M_subset_MG* : **1** ∈ *G* ⟹ *M* ⊆ *M*[*G*]
using *check_in_M GenExtI*
by (*intro subsetI*, *subst val_check* [*of G*,*symmetric*], *auto*)

The name for the generic filter

definition
G_dot :: *i* **where**
G_dot ≡ {⟨*check*(*p*),*p*⟩ . *p*∈ℙ}

lemma *G_dot_in_M* : *G_dot* ∈ *M*
using *lam_replacement_Pair*[*THEN* [*5*] *lam_replacement_hcomp2*,*OF*
check_lam_replacement lam_replacement_identity]
check_in_M lam_replacement_imp_strong_replacement_aux
transitivity check_in_M RepFun_closed pair_in_M_iff
unfolding *G_dot_def*
by *simp*

lemma *zero_in_MG* : *0* ∈ *M*[*G*]
proof -

```
  have 0 = val(G,0)
    using zero_in_M elem_of_val by auto
  also
  have ... ∈ M[G]
    using GenExtI zero_in_M by simp
  finally
  show ?thesis .
qed

declare check_in_M [simp,intro]

end — forcing_data1

context G_generic1
begin

lemma val_G_dot : val(G,G_dot) = G
proof (intro equalityI subsetI)
  fix x
  assume x∈val(G,G_dot)
  then obtain ϑ p where p∈G ⟨ϑ,p⟩ ∈ G_dot val(G,ϑ) = x ϑ = check(p)
    unfolding G_dot_def using elem_of_val_pair G_dot_in_M
    by force
  then
  show x ∈ G
    using G_subset_P one_in_G val_check P_sub_M by auto
next
  fix p
  assume p∈G
  have ⟨check(q),q⟩ ∈ G_dot if q∈ℙ for q
    unfolding G_dot_def using that by simp
  with ‹p∈G›
  have val(G,check(p)) ∈ val(G,G_dot)
    using val_of_elem G_dot_in_M by blast
  with ‹p∈G›
  show p ∈ val(G,G_dot)
    using one_in_G G_subset_P P_sub_M val_check by auto
qed

lemma G_in_Gen_Ext : G ∈ M[G]
  using G_subset_P one_in_G val_G_dot GenExtI[of _ G] G_dot_in_M
  by force

lemmas generic_simps = val_check[OF one_in_G one_in_P]
  M_subset_MG[OF one_in_G, THEN subsetD]
  GenExtI P_in_M

lemmas generic_dests = M_genericD M_generic_compatD
```

bundle *G_generic1_lemmas* = *generic_simps*[*simp*] *generic_dests*[*dest*]

end — *G_generic1*

sublocale *G_generic1* ⊆ *ext*: *M_trans* ##*M*[*G*]
  using *generic transitivity_MG zero_in_MG*
  by *unfold_locales force+*

end

13 The Forcing Theorems

theory *Forcing_Theorems*
  imports
    Cohen_Posets_Relative
    Forces_Definition
    Names

begin

context *forcing_data1*
begin

13.1 The forcing relation in context

lemma *separation_forces* :
  assumes
    fty: φ∈*formula* **and**
    far: *arity*(φ)≤*length*(*env*) **and**
    envty: *env*∈*list*(*M*)
  shows
    separation(##*M*,λ*p*. (*p* ⊩ φ *env*))
  using *separation_ax arity_forces far fty envty arity_forces_le*
    transitivity[*of* _ ℙ]
  by *simp*

lemma *Collect_forces* :
  assumes
    φ∈*formula* **and**
    arity(φ)≤*length*(*env*) **and**
    env∈*list*(*M*)
  shows
    {*p*∈ℙ . *p* ⊩ φ *env*} ∈ *M*
  using *assms separation_forces separation_closed*
  by *simp*

lemma *forces_mem_iff_dense_below*: *p*∈ℙ ⟹ *p forces*$_a$ (*t1* ∈ *t2*) ⟷ *dense_below*(
    {*q*∈ℙ. ∃ *s*. ∃ *r*. *r*∈ℙ ∧ ⟨*s*,*r*⟩ ∈ *t2* ∧ *q*≼*r* ∧ *q forces*$_a$ (*t1* = *s*)}
    ,*p*)

using *def_forces_mem*[*of p t1 t2*] **by** *blast*

13.2 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_eq*:
assumes $p{\in}\mathbb{P}$ $r{\in}\mathbb{P}$ $r{\preceq}p$ p *forces*$_a$ $(t1 = t2)$
shows r *forces*$_a$ $(t1 = t2)$
using *assms def_forces_eq*[*of _ t1 t2*] *leq_transD* **by** *blast*

13.3 Kunen 2013, Lemma IV.2.37(a)

lemma *strengthening_mem*:
assumes $p{\in}\mathbb{P}$ $r{\in}\mathbb{P}$ $r{\preceq}p$ p *forces*$_a$ $(t1 \in t2)$
shows r *forces*$_a$ $(t1 \in t2)$
using *assms forces_mem_iff_dense_below dense_below_under* **by** *auto*

13.4 Kunen 2013, Lemma IV.2.37(b)

lemma *density_mem*:
assumes $p{\in}\mathbb{P}$
shows p *forces*$_a$ $(t1 \in t2)$ $\longleftrightarrow$ *dense_below*($\{q{\in}\mathbb{P}.\ q$ *forces*$_a$ $(t1 \in t2)\}$,p)
proof
assume p *forces*$_a$ $(t1 \in t2)$
with *assms*
show *dense_below*($\{q{\in}\mathbb{P}.\ q$ *forces*$_a$ $(t1 \in t2)\}$,p)
using *forces_mem_iff_dense_below strengthening_mem*[*of p*] *ideal_dense_below*
by *auto*
next
assume *dense_below*($\{q \in \mathbb{P}\ .\ q$ *forces*$_a$ $(\ t1 \in\ t2)\}$, p)
with *assms*
have *dense_below*($\{q{\in}\mathbb{P}.$
dense_below($\{q'{\in}\mathbb{P}.\ \exists\, s\ r.\ r \in \mathbb{P} \wedge \langle s,r\rangle{\in}t2 \wedge q'{\preceq}r \wedge q'$ *forces*$_a$ $(t1 = s)\}$,q)
$\}$,p)
using *forces_mem_iff_dense_below* **by** *simp*
with *assms*
show p *forces*$_a$ $(t1 \in t2)$
using *dense_below_dense_below forces_mem_iff_dense_below*[*of p t1 t2*] **by** *blast*
qed

lemma *aux_density_eq*:
assumes
dense_below(
$\{q'{\in}\mathbb{P}.\ \forall\, q.\ q{\in}\mathbb{P} \wedge q{\preceq}q' \longrightarrow q$ *forces*$_a$ $(s \in t1) \longleftrightarrow q$ *forces*$_a$ $(s \in t2)\}$
,p)
q *forces*$_a$ $(s \in t1)$ $q{\in}\mathbb{P}$ $p{\in}\mathbb{P}$ $q{\preceq}p$
shows
dense_below($\{r{\in}\mathbb{P}.\ r$ *forces*$_a$ $(s \in t2)\}$,q)
proof
fix r

assume $r{\in}\mathbb{P}$ $r{\preceq}q$
moreover from *this* **and** ‹$p{\in}\mathbb{P}$› ‹$q{\preceq}p$› ‹$q{\in}\mathbb{P}$›
have $r{\preceq}p$
using *leq_transD* **by** *simp*
moreover
note ‹q *forces*$_a$ $(s \in t1)$› ‹*dense_below*(_,p)› ‹$q{\in}\mathbb{P}$›
ultimately
obtain *q1* **where** $q1{\preceq}r$ $q1{\in}\mathbb{P}$ *q1 forces*$_a$ $(s \in t2)$
using *strengthening_mem*[*of* q _ s *t1*] *refl_leq leq_transD*[*of* _ r q] **by** *blast*
then
show $\exists\, d{\in}\{r \in \mathbb{P} \,.\, r$ *forces*$_a$ $(\ s \in\ t2)\}.\ d \in \mathbb{P} \wedge d{\preceq}\ r$
by *blast*
qed

lemma *density_eq*:
assumes $p{\in}\mathbb{P}$
shows p *forces*$_a$ $(t1 = t2)$ $\longleftrightarrow$ *dense_below*($\{q{\in}\mathbb{P}.\ q$ *forces*$_a$ $(t1 = t2)\}$,p)
proof
assume p *forces*$_a$ $(t1 = t2)$
with ‹$p{\in}\mathbb{P}$›
show *dense_below*($\{q{\in}\mathbb{P}.\ q$ *forces*$_a$ $(t1 = t2)\}$,p)
using *strengthening_eq ideal_dense_below* **by** *auto*
next
assume *dense_below*($\{q{\in}\mathbb{P}.\ q$ *forces*$_a$ $(t1 = t2)\}$,p)
{
fix s q
let *?D1*=$\{q'{\in}\mathbb{P}.\ \forall\, s{\in}$*domain*$(t1) \cup$ *domain*$(t2).\ \forall\, q.\ q \in \mathbb{P} \wedge q{\preceq}q' \longrightarrow$
q *forces*$_a$ $(s \in t1){\longleftrightarrow}q$ *forces*$_a$ $(s \in t2)\}$
let *?D2*=$\{q'{\in}\mathbb{P}.\ \forall\, q.\ q{\in}\mathbb{P} \wedge q{\preceq}q' \longrightarrow q$ *forces*$_a$ $(s \in t1) \longleftrightarrow q$ *forces*$_a$ $(s \in t2)\}$
assume $s{\in}$*domain*$(t1) \cup$ *domain*$(t2)$
then
have *?D1*$\subseteq$*?D2* **by** *blast*
with ‹*dense_below*(_,p)›
have *dense_below*($\{q'{\in}\mathbb{P}.\ \forall\, s{\in}$*domain*$(t1) \cup$ *domain*$(t2).\ \forall\, q.\ q \in \mathbb{P} \wedge q{\preceq}q' \longrightarrow$
q *forces*$_a$ $(s \in t1){\longleftrightarrow}q$ *forces*$_a$ $(s \in t2)\}$,p)
using *dense_below_cong'*[*OF* ‹$p{\in}\mathbb{P}$› *def_forces_eq*[*of* _ *t1 t2*]] **by** *simp*
with ‹$p{\in}\mathbb{P}$› ‹*?D1*$\subseteq$*?D2*›
have *dense_below*($\{q'{\in}\mathbb{P}.\ \forall\, q.\ q{\in}\mathbb{P} \wedge q{\preceq}q' \longrightarrow$
q *forces*$_a$ $(s \in t1) \longleftrightarrow q$ *forces*$_a$ $(s \in t2)\}$,p)
using *dense_below_mono* **by** *simp*
moreover from *this*
have *dense_below*($\{q'{\in}\mathbb{P}.\ \forall\, q.\ q{\in}\mathbb{P} \wedge q{\preceq}q' \longrightarrow$
q *forces*$_a$ $(s \in t2) \longleftrightarrow q$ *forces*$_a$ $(s \in t1)\}$,p)
by *blast*
moreover
assume $q \in \mathbb{P}$ $q{\preceq}p$
moreover
note ‹$p{\in}\mathbb{P}$›

ultimately
have $q\ forces_a\ (s \in t1) \Longrightarrow dense\_below(\{r{\in}\mathbb{P}.\ r\ forces_a\ (s \in t2)\},q)$
$q\ forces_a\ (s \in t2) \Longrightarrow dense\_below(\{r{\in}\mathbb{P}.\ r\ forces_a\ (s \in t1)\},q)$
using *aux_density_eq* **by** *simp_all*
then
have $q\ forces_a\ (\ s \in\ t1) \longleftrightarrow q\ forces_a\ (\ s \in\ t2)$
using *density_mem*[*OF* ‹$q{\in}\mathbb{P}$›] **by** *blast*
}
with ‹$p{\in}\mathbb{P}$›
show $p\ forces_a\ (t1 = t2)$ **using** *def_forces_eq* **by** *blast*
qed

13.5 Kunen 2013, Lemma IV.2.38

lemma *not_forces_neq*:
assumes $p{\in}\mathbb{P}$
shows $p\ forces_a\ (t1 = t2) \longleftrightarrow \neg\ (\exists\, q{\in}\mathbb{P}.\ q{\preceq}p \wedge q\ forces_a\ (t1 \neq t2))$
using *assms density_eq* **unfolding** *forces_neq_def* **by** *blast*

lemma *not_forces_nmem*:
assumes $p{\in}\mathbb{P}$
shows $p\ forces_a\ (t1 \in t2) \longleftrightarrow \neg\ (\exists\, q{\in}\mathbb{P}.\ q{\preceq}p \wedge q\ forces_a\ (t1 \notin t2))$
using *assms density_mem* **unfolding** *forces_nmem_def* **by** *blast*

13.6 The relation of forcing and atomic formulas

lemma *Forces_Equal*:
assumes
$p{\in}\mathbb{P}\ t1{\in}M\ t2{\in}M\ env{\in}list(M)\ nth(n,env) = t1\ nth(m,env) = t2\ n{\in}nat\ m{\in}nat$
shows
$(p \Vdash Equal(n,m)\ env) \longleftrightarrow p\ forces_a\ (t1 = t2)$
using *assms sats_forces_Equal forces_eq_abs transitivity*
by *simp*

lemma *Forces_Member*:
assumes
$p{\in}\mathbb{P}\ t1{\in}M\ t2{\in}M\ env{\in}list(M)\ nth(n,env) = t1\ nth(m,env) = t2\ n{\in}nat\ m{\in}nat$
shows
$(p \Vdash Member(n,m)\ env) \longleftrightarrow p\ forces_a\ (t1 \in t2)$
using *assms sats_forces_Member forces_mem_abs transitivity*
by *simp*

lemma *Forces_Neg*:
assumes
$p{\in}\mathbb{P}\ env \in list(M)\ \varphi{\in}formula$
shows
$(p \Vdash Neg(\varphi)\ env) \longleftrightarrow \neg(\exists\, q{\in}M.\ q{\in}\mathbb{P} \wedge q{\preceq}p \wedge (q \Vdash \varphi\ env))$
using *assms sats_forces_Neg transitivity pair_in_M_iff leq_abs*
by *simp*

13.7 The relation of forcing and connectives

lemma *Forces_Nand*:
assumes
$p\in\mathbb{P}$ *env* $\in$ *list*(M) $\varphi\in$*formula* $\psi\in$*formula*
shows
($p \Vdash$ *Nand*(φ,ψ) *env*) $\longleftrightarrow$ $\neg(\exists\, q\in M.\ q\in\mathbb{P} \wedge q\preceq p \wedge (q \Vdash \varphi\ env) \wedge (q \Vdash \psi\ env))$
using *assms sats_forces_Nand transitivity pair_in_M_iff leq_abs* **by** *simp*

lemma *Forces_And_aux*:
assumes
$p\in\mathbb{P}$ *env* $\in$ *list*(M) $\varphi\in$*formula* $\psi\in$*formula*
shows
$p \Vdash$ *And*(φ,ψ) *env* $\longleftrightarrow$
$(\forall\, q\in M.\ q\in\mathbb{P} \wedge q\preceq p \longrightarrow (\exists\, r\in M.\ r\in\mathbb{P} \wedge r\preceq q \wedge (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)))$
unfolding *And_def* **using** *assms Forces_Neg Forces_Nand* **by** (*auto simp only*:)

lemma *Forces_And_iff_dense_below*:
assumes
$p\in\mathbb{P}$ *env* $\in$ *list*(M) $\varphi\in$*formula* $\psi\in$*formula*
shows
($p \Vdash$ *And*(φ,ψ) *env*) $\longleftrightarrow$ *dense_below*($\{r\in\mathbb{P}.\ (r \Vdash \varphi\ env) \wedge (r \Vdash \psi\ env)\ \}$,$p$)
unfolding *dense_below_def* **using** *Forces_And_aux assms*
by (*auto dest*:*transitivity*[*OF* _ *P_in_M*]; *rename_tac q*; *drule_tac x*=*q* **in** *bspec*)+

lemma *Forces_Forall*:
assumes
$p\in\mathbb{P}$ *env* $\in$ *list*(M) $\varphi\in$*formula*
shows
($p \Vdash$ *Forall*(φ) *env*) $\longleftrightarrow$ $(\forall\, x\in M.\ (p \Vdash \varphi\ ([x]\ @\ env)))$
using *sats_forces_Forall assms transitivity*[*OF* _ *P_in_M*]
by *simp*

bundle *some_rules* = *elem_of_val_pair* [*dest*]

context
includes *some_rules*
begin

lemma *elem_of_valI*: $\exists\,\vartheta.\ \exists\, p\in\mathbb{P}.\ p\in G \wedge \langle\vartheta,p\rangle\in\pi \wedge val(G,\vartheta) = x \Longrightarrow x\in val(G,\pi)$
by (*subst def_val*, *auto*)

lemma *GenExt_iff*: $x\in M[G] \longleftrightarrow (\exists\,\tau\in M.\ x = val(G,\tau))$
unfolding *GenExt_def* **by** *simp*
end

end
context *G_generic1*

begin

13.8 Kunen 2013, Lemma IV.2.29

lemma *generic_inter_dense_below*:
assumes $D{\in}M$ *dense_below*(D,p) $p{\in}G$
shows $D \cap G \neq 0$
proof -
let *?D*=$\{q{\in}\mathbb{P}.\ p{\perp}q \vee q{\in}D\}$
have *dense*(*?D*)
proof
fix r
assume $r{\in}\mathbb{P}$
show $\exists\, d{\in}\{q \in \mathbb{P}\ .\ p \perp q \vee q \in D\}.\ d \preceq r$
proof (*cases* $p \perp r$)
case *True*
with ‹$r{\in}\mathbb{P}$›

show *?thesis* **using** *refl_leq*[*of r*] **by** (*intro bexI*) (*blast*+)
next
case *False*
then
obtain s **where** $s{\in}\mathbb{P}$ $s{\preceq}p$ $s{\preceq}r$ **by** *blast*
with *assms* ‹$r{\in}\mathbb{P}$›
show *?thesis*
using *dense_belowD*[*OF assms*(*2*), *of s*] *leq_transD*[*of _ s r*]
by *blast*
qed
qed
have *?D*$\subseteq\mathbb{P}$ **by** *auto*
let *?d_fm*=$\cdot\cdot\neg$*compat_in_fm*(*1*, *2*, *3*, *0*) $\cdot \vee \cdot 0 \in 4\cdot\cdot$
from ‹$p{\in}G$›
have $p{\in}M$
using *G_subset_M subsetD* **by** *simp*
moreover
have *?d_fm*$\in$*formula* **by** *simp*
moreover
have *arity*(*?d_fm*) = *5*
by (*auto simp add*: *arity*)
moreover from ‹$D{\in}M$› ‹$p{\in}M$›
have (M, [q,$\mathbb{P}$,*leq*,p,D] $\models$ *?d_fm*) $\longleftrightarrow$ ($\neg$ *is_compat_in*(\#\#M,$\mathbb{P}$,*leq*,p,q) $\vee$ $q{\in}D$)
if $q{\in}M$ **for** q
using *that sats_compat_in_fm zero_in_M*
by *simp*
moreover from ‹$p{\in}M$›
have ($\neg$ *is_compat_in*(\#\#M,$\mathbb{P}$,*leq*,p,q) $\vee$ $q{\in}D$) $\longleftrightarrow$ $p{\perp}q \vee q{\in}D$ **if** $q{\in}M$ **for** q
unfolding *compat_def*
using *that compat_in_abs*

by *simp*
ultimately
have *?D∈M*
using *Collect_in_M*[*of ?d_fm* [ℙ,*leq*,*p*,*D*]] ‹*D∈M*›
by *simp*
note *asm* = ‹*dense*(*?D*)› ‹*?D*⊆ℙ› ‹*?D∈M*›
obtain *x* **where** *x∈G x∈?D*
using *M_generic_denseD*[*OF asm*]
by *force*
moreover from *this*
have *x∈D*
using *M_generic_compatD*[*OF* _ ‹*p∈G*›, *of x*] *refl_leq compatI*[*of* _ *p x*]
by *force*
ultimately
show *?thesis* **by** *auto*
qed

13.9 Auxiliary results for Lemma IV.2.40(a)

lemma (**in** *forcing_data1*) *IV240a_mem_Collect*:
assumes
$\pi{\in}M$ $\tau{\in}M$
shows
$\{q{\in}\mathbb{P}.\ \exists\,\sigma.\ \exists\,r.\ r{\in}\mathbb{P} \wedge \langle\sigma,r\rangle \in \tau \wedge q{\preceq}r \wedge q\ forces_a\ (\pi = \sigma)\}{\in}M$
proof -
let *?rel_pred*= $\lambda M\ x\ a1\ a2\ a3\ a4.\ \exists\,\sigma[M].\ \exists\,r[M].\ \exists\,\sigma r[M].$
$r{\in}a1 \wedge pair(M,\sigma,r,\sigma r) \wedge \sigma r{\in}a4 \wedge is\_leq(M,a2,x,r) \wedge is\_forces\_eq'(M,a1,a2,x,a3,\sigma)$
let *?φ*=*Exists*(*Exists*(*Exists*(*And*(*Member*(*1*,*4*),*And*(*pair_fm*(*2*,*1*,*0*),
And(*Member*(*0*,*7*),*And*(*is_leq_fm*(*5*,*3*,*1*),*forces_eq_fm*(*4*,*5*,*3*,*6*,*2*)))))))))
have $\sigma{\in}M \wedge r{\in}M$ **if** $\langle\sigma,\ r\rangle \in \tau$ **for** σ r
using *that* ‹$\tau{\in}M$› *pair_in_M_iff transitivity*[*of* $\langle\sigma,r\rangle$ τ] **by** *simp*
then
have *?rel_pred*(*##M*,*q*,ℙ,*leq*,π,τ) ⟷ ($\exists\,\sigma.\ \exists\,r.\ r{\in}\mathbb{P} \wedge \langle\sigma,r\rangle \in \tau \wedge q{\preceq}r \wedge q$
$forces_a\ (\pi = \sigma)$)
if $q{\in}M$ **for** q
unfolding *forces_eq_def*
using *assms that leq_abs forces_eq'_abs pair_in_M_iff*
by *auto*
moreover
have (M, [*q*,ℙ,*leq*,π,τ] ⊨ *?φ*) ⟷ *?rel_pred*(*##M*,*q*,ℙ,*leq*,π,τ) **if** $q{\in}M$ **for** q
using *assms that sats_forces_eq_fm sats_is_leq_fm zero_in_M*
by *simp*
moreover
have *?φ∈formula* **by** *simp*
moreover
have *arity*(*?φ*)=*5*
using *arity_forces_eq_fm*
by (*simp add*:*ord_simp_union arity*)
ultimately

show *?thesis*
unfolding *forces_eq_def* **using** *assms Collect_in_M*[*of* ?φ [$\mathbb{P}$,*leq*,π,τ]]
by *simp*
qed

lemma *IV240a_mem*:
assumes
$p{\in}G\ \pi{\in}M\ \tau{\in}M\ p\ forces_a\ (\pi \in \tau)$
$\bigwedge q\ \sigma.\ q{\in}\mathbb{P} \Longrightarrow q{\in}G \Longrightarrow \sigma{\in}domain(\tau) \Longrightarrow q\ forces_a\ (\pi = \sigma) \Longrightarrow$
$val(G,\pi) = val(G,\sigma)$
shows
$val(G,\pi){\in}val(G,\tau)$
proof (*intro elem_of_valI*)
let *?D*=$\{q{\in}\mathbb{P}.\ \exists\sigma.\ \exists r.\ r{\in}\mathbb{P} \wedge \langle\sigma,r\rangle \in \tau \wedge q{\preceq}r \wedge q\ forces_a\ (\pi = \sigma)\}$
from ‹$p{\in}G$›
have $p{\in}\mathbb{P}$ **by** *blast*
moreover
note ‹$\pi{\in}M$› ‹$\tau{\in}M$›
ultimately
have *?D* $\in M$ **using** *IV240a_mem_Collect* **by** *simp*
moreover from *assms* ‹$p{\in}\mathbb{P}$›
have *dense_below*(*?D*,*p*)
using *forces_mem_iff_dense_below* **by** *simp*
moreover
note ‹$p{\in}G$›
ultimately
obtain q **where** $q{\in}G\ q{\in}$*?D*
using *generic_inter_dense_below*[*of ?D p*] **by** *blast*
then
obtain $\sigma\ r$ **where** $r{\in}\mathbb{P}\ \langle\sigma,r\rangle \in \tau\ q{\preceq}r\ q\ forces_a\ (\pi = \sigma)$ **by** *blast*
moreover from *this* **and** ‹$q{\in}G$› *assms*
have $r \in G\ val(G,\pi) = val(G,\sigma)$ **by** *blast*+
ultimately
show $\exists\ \sigma.\ \exists p{\in}\mathbb{P}.\ p \in G \wedge \langle\sigma,\ p\rangle \in \tau \wedge val(G,\ \sigma) = val(G,\ \pi)$ **by** *auto*
qed

lemma *refl_forces_eq*:$p{\in}\mathbb{P} \Longrightarrow p\ forces_a\ (x = x)$
using *def_forces_eq* **by** *simp*

lemma *forces_memI*: $\langle\sigma,r\rangle{\in}\tau \Longrightarrow p{\in}\mathbb{P} \Longrightarrow r{\in}\mathbb{P} \Longrightarrow p{\preceq}r \Longrightarrow p\ forces_a\ (\sigma \in \tau)$
using *refl_forces_eq*[*of* _ σ] *leq_transD refl_leq*
by (*blast intro*:*forces_mem_iff_dense_below*[*THEN iffD2*])

lemma *IV240a_eq_1st_incl*:
includes *some_rules*
assumes

$p \in G\ p\ forces_a\ (\tau = \vartheta)$
and
$IH: \bigwedge q\ \sigma.\ q \in \mathbb{P} \Longrightarrow q \in G \Longrightarrow \sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(q\ forces_a\ (\sigma \in \tau) \longrightarrow val(G,\sigma) \in val(G,\tau)) \wedge$
$(q\ forces_a\ (\sigma \in \vartheta) \longrightarrow val(G,\sigma) \in val(G,\vartheta))$

shows
$val(G,\tau) \subseteq val(G,\vartheta)$
proof
fix x
assume $x \in val(G,\tau)$
then
obtain $\sigma\ r$ **where** $\langle \sigma,r \rangle \in \tau\ r \in G\ val(G,\sigma) = x$ **by** *blast*
moreover from *this* **and** ‹$p \in G$›
obtain q **where** $q \in G\ q \preceq p\ q \preceq r$ **by** *force*
moreover from *this* **and** ‹$p \in G$›
have $q \in \mathbb{P}\ p \in \mathbb{P}$ **by** *blast+*
moreover from *calculation*
have $q\ forces_a\ (\sigma \in \tau)$
using *forces_memI* **by** *auto*
moreover
note ‹$p\ forces_a\ (\tau = \vartheta)$›
ultimately
have $q\ forces_a\ (\sigma \in \vartheta)$
using *def_forces_eq* **by** *auto*
with ‹$q \in \mathbb{P}$› ‹$q \in G$› *IH*[*of* $q\ \sigma$] ‹$\langle \sigma,r \rangle \in \tau$› ‹$val(G,\sigma) = x$›
show $x \in val(G,\vartheta)$ **by** *blast*
qed

lemma *IV240a_eq_2nd_incl*:
includes *some_rules*
assumes
$p \in G\ p\ forces_a\ (\tau = \vartheta)$
and
$IH: \bigwedge q\ \sigma.\ q \in \mathbb{P} \Longrightarrow q \in G \Longrightarrow \sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(q\ forces_a\ (\sigma \in \tau) \longrightarrow val(G,\sigma) \in val(G,\tau)) \wedge$
$(q\ forces_a\ (\sigma \in \vartheta) \longrightarrow val(G,\sigma) \in val(G,\vartheta))$
shows
$val(G,\vartheta) \subseteq val(G,\tau)$
proof
fix x
assume $x \in val(G,\vartheta)$
then
obtain $\sigma\ r$ **where** $\langle \sigma,r \rangle \in \vartheta\ r \in G\ val(G,\sigma) = x$ **by** *blast*
moreover from *this* **and** ‹$p \in G$›
obtain q **where** $q \in G\ q \preceq p\ q \preceq r$ **by** *force*
moreover from *this* **and** ‹$p \in G$›

have $q{\in}\mathbb{P}$ $p{\in}\mathbb{P}$ **by** *blast+*
moreover from *calculation*
have q $forces_a$ $(\sigma \in \vartheta)$
using *forces_memI* **by** *auto*
moreover
note ‹p $forces_a$ $(\tau = \vartheta)$›
ultimately
have q $forces_a$ $(\sigma \in \tau)$
using *def_forces_eq* **by** *auto*
with ‹$q{\in}\mathbb{P}$› ‹$q{\in}G$› *IH*[*of* q σ] ‹$\langle\sigma,r\rangle{\in}\vartheta$› ‹$val(G,\sigma) = x$›
show $x{\in}val(G,\tau)$ **by** *blast*
qed

lemma *IV240a_eq*:
includes *some_rules*
assumes
$p{\in}G$ p $forces_a$ $(\tau = \vartheta)$
and
IH:$\bigwedge q\ \sigma.\ q{\in}\mathbb{P} \Longrightarrow q{\in}G \Longrightarrow \sigma{\in}domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(q\ forces_a\ (\sigma \in \tau) \longrightarrow val(G,\sigma) \in val(G,\tau)) \wedge$
$(q\ forces_a\ (\sigma \in \vartheta) \longrightarrow val(G,\sigma) \in val(G,\vartheta))$
shows
$val(G,\tau) = val(G,\vartheta)$
using *IV240a_eq_1st_incl*[*OF assms*] *IV240a_eq_2nd_incl*[*OF assms*] *IH* **by** *blast*

13.10 Induction on names

lemma (**in** *forcing_data1*) *core_induction*:
assumes
$\bigwedge\tau\ \vartheta\ p.\ p \in \mathbb{P} \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![q{\in}\mathbb{P}\ ;\ \sigma{\in}domain(\vartheta)]\!] \Longrightarrow Q(0,\tau,\sigma,q)]\!] \Longrightarrow Q(1,\tau,\vartheta,p)$
$\bigwedge\tau\ \vartheta\ p.\ p \in \mathbb{P} \Longrightarrow [\![\bigwedge q\ \sigma.\ [\![q{\in}\mathbb{P}\ ;\ \sigma{\in}domain(\tau) \cup domain(\vartheta)]\!] \Longrightarrow Q(1,\sigma,\tau,q) \wedge Q(1,\sigma,\vartheta,q)]\!] \Longrightarrow Q(0,\tau,\vartheta,p)$
$ft \in 2\ p \in \mathbb{P}$
shows
$Q(ft,\tau,\vartheta,p)$
proof -
{
fix $ft\ p\ \tau\ \vartheta$
have $Transset(eclose(\{\tau,\vartheta\}))$ (**is** $Transset(?e)$)
using *Transset_eclose* **by** *simp*
have $\tau \in ?e\ \vartheta \in ?e$
using *arg_into_eclose* **by** *simp_all*
moreover
assume $ft \in 2\ p \in \mathbb{P}$
ultimately
have $\langle ft,\tau,\vartheta,p\rangle{\in}\ 2{\times}?e{\times}?e{\times}\mathbb{P}$ (**is** $?a{\in}2{\times}?e{\times}?e{\times}\mathbb{P}$) **by** *simp*
then

```
    have Q(ftype(?a), name1(?a), name2(?a), cond_of(?a))
    using core_induction_aux[of ?e ℙ Q ?a,OF ‹Transset(?e)› assms(1,2) ‹?a∈_›]
      by (clarify) (blast)
    then have Q(ft,τ,ϑ,p) by (simp add:components_simp)
  }
  then show ?thesis using assms by simp
qed

lemma (in forcing_data1) forces_induction_with_conds:
  assumes
    ⋀τ ϑ p. p ∈ ℙ ⟹ ⟦⋀q σ. ⟦q∈ℙ ; σ∈domain(ϑ)⟧ ⟹ Q(q,τ,σ)⟧ ⟹ R(p,τ,ϑ)
    ⋀τ ϑ p. p ∈ ℙ ⟹ ⟦⋀q σ. ⟦q∈ℙ ; σ∈domain(τ) ∪ domain(ϑ)⟧ ⟹ R(q,σ,τ) ∧
R(q,σ,ϑ)⟧ ⟹ Q(p,τ,ϑ)
    p ∈ ℙ
  shows
    Q(p,τ,ϑ) ∧ R(p,τ,ϑ)
proof -
  let ?Q=λft τ ϑ p. (ft = 0 ⟶ Q(p,τ,ϑ)) ∧ (ft = 1 ⟶ R(p,τ,ϑ))
  from assms(1)
  have ⋀τ ϑ p. p ∈ ℙ ⟹ ⟦⋀q σ. ⟦q∈ℙ ; σ∈domain(ϑ)⟧ ⟹ ?Q(0,τ,σ,q)⟧ ⟹
?Q(1,τ,ϑ,p)
    by simp
  moreover from assms(2)
  have ⋀τ ϑ p. p ∈ ℙ ⟹ ⟦⋀q σ. ⟦q∈ℙ ; σ∈domain(τ) ∪ domain(ϑ)⟧ ⟹
?Q(1,σ,τ,q) ∧ ?Q(1,σ,ϑ,q)⟧ ⟹ ?Q(0,τ,ϑ,p)
    by simp
  moreover
  note ‹p∈ℙ›
  ultimately
  have ?Q(ft,τ,ϑ,p) if ft∈2 for ft
    by (rule core_induction[OF _ _ that, of ?Q])
  then
  show ?thesis by auto
qed

lemma (in forcing_data1) forces_induction:
  assumes
    ⋀τ ϑ. ⟦⋀σ. σ∈domain(ϑ) ⟹ Q(τ,σ)⟧ ⟹ R(τ,ϑ)
    ⋀τ ϑ. ⟦⋀σ. σ∈domain(τ) ∪ domain(ϑ) ⟹ R(σ,τ) ∧ R(σ,ϑ)⟧ ⟹ Q(τ,ϑ)
  shows
    Q(τ,ϑ) ∧ R(τ,ϑ)
proof (intro forces_induction_with_conds[OF _ _ one_in_P ])
  fix τ ϑ p
  assume q ∈ ℙ ⟹ σ ∈ domain(ϑ) ⟹ Q(τ, σ) for q σ
  with assms(1)
  show R(τ,ϑ)
    using one_in_P by simp
next
  fix τ ϑ p
```

assume $q \in \mathbb{P} \Longrightarrow \sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow R(\sigma,\tau) \wedge R(\sigma,\vartheta)$ **for** $q\ \sigma$
with *assms*(2)
show $Q(\tau,\vartheta)$
using *one_in_P* **by** *simp*
qed

13.11 Lemma IV.2.40(a), in full

lemma *IV240a*:
shows
$(\tau{\in}M \longrightarrow \vartheta{\in}M \longrightarrow (\forall\, p{\in}G.\ p\ forces_a\ (\tau = \vartheta) \longrightarrow val(G,\tau) = val(G,\vartheta))) \wedge$
$(\tau{\in}M \longrightarrow \vartheta{\in}M \longrightarrow (\forall\, p{\in}G.\ p\ forces_a\ (\tau \in \vartheta) \longrightarrow val(G,\tau) \in val(G,\vartheta)))$
(**is** *?Q*$(\tau,\vartheta) \wedge$ *?R*(τ,ϑ))
proof (*intro forces_induction*[*of ?Q ?R*] *impI*)
fix $\tau\ \vartheta$
assume $\tau{\in}M\ \vartheta{\in}M\ \ \sigma{\in}domain(\vartheta) \Longrightarrow$ *?Q*(τ,σ) **for** σ
moreover from *this*
have $\sigma{\in}domain(\vartheta) \Longrightarrow q\ forces_a\ (\tau = \sigma) \Longrightarrow val(G,\tau) = val(G,\sigma)$
if $q{\in}\mathbb{P}\ q{\in}G$ **for** $q\ \sigma$
using *that domain_closed*[*of* ϑ] *transitivity* **by** *auto*
ultimately
show $\forall\, p{\in}G.\ p\ forces_a\ (\tau \in \vartheta) \longrightarrow val(G,\tau) \in val(G,\vartheta)$
using *IV240a_mem domain_closed transitivity* **by** *simp*
next
fix $\tau\ \vartheta$
assume $\tau{\in}M\ \vartheta{\in}M$ **and** $d{:}\sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow$ *?R*$(\sigma,\tau) \wedge$ *?R*(σ,ϑ)
for σ
moreover from *this*
have *IH'*:$(q\ forces_a\ (\sigma \in \tau) \longrightarrow val(G,\sigma) \in val(G,\tau)) \wedge$
$(q\ forces_a\ (\sigma \in \vartheta) \longrightarrow val(G,\sigma) \in val(G,\vartheta))$
if $\sigma \in domain(\tau) \cup domain(\vartheta)\ q{\in}G$ **for** $q\ \sigma$
proof -
from *d that*
have *A*:*?R*(σ,τ) *?R*(σ,ϑ)
by *auto*
from ‹$\tau{\in}\_$› ‹$\vartheta{\in}M$› ‹$q{\in}G$› ‹$\sigma{\in}\_$›
show *?thesis*
using *transitivity*[*of* σ] *domain_closed A*[*rule_format,of q*]
by *auto*
qed
show $\forall\, p{\in}G.\ p\ forces_a\ (\tau = \vartheta) \longrightarrow val(G,\tau) = val(G,\vartheta)$
using *IV240a_eq*[*OF _ _ IH'*] **by** *simp*
qed

13.12 Lemma IV.2.40(b)

lemma *IV240b_mem*:
includes *some_rules*
assumes
$val(G,\pi){\in}val(G,\tau)\ \pi{\in}M\ \tau{\in}M$

and
IH:$\bigwedge\sigma$. $\sigma\in domain(\tau) \Longrightarrow val(G,\pi) = val(G,\sigma) \Longrightarrow$
$\exists\, p\in G.\ p\ forces_a\ (\pi = \sigma)$
shows
$\exists\, p\in G.\ p\ forces_a\ (\pi \in \tau)$
proof -
from ‹$val(G,\pi)\in val(G,\tau)$›
obtain $\sigma\ r$ **where** $r\in G\ \langle\sigma,r\rangle\in\tau\ val(G,\pi) = val(G,\sigma)$ **by** *auto*
moreover from *this* **and** *IH*
obtain p' **where** $p'\in G\ p'\ forces_a\ (\pi = \sigma)$ **by** *blast*
ultimately
obtain p **where** $p\preceq r\ p\preceq p'\ p\in G\ p\ forces_a\ (\pi = \sigma)$
using *M_generic_compatD strengthening_eq*[*of* p'] *M_genericD* **by** *auto*
moreover from *calculation*
have $q\ forces_a\ (\pi = \sigma)$ **if** $q\in\mathbb{P}\ q\preceq p$ **for** q
using *that strengthening_eq* **by** *blast*
moreover
note ‹$\langle\sigma,r\rangle\in\tau$› ‹$r\in G$›
ultimately
have $r\in\mathbb{P} \wedge \langle\sigma,r\rangle \in \tau \wedge q\preceq r \wedge q\ forces_a\ (\pi = \sigma)$ **if** $q\in\mathbb{P}\ q\preceq p$ **for** q
using *that leq_transD*[*of* _ p r] **by** *blast*
then
have *dense_below*($\{q\in\mathbb{P}.\ \exists\, s\ r.\ r\in\mathbb{P} \wedge \langle s,r\rangle \in \tau \wedge q\preceq r \wedge q\ forces_a\ (\pi = s)\}$,$p$)
using *refl_leq* **by** *blast*
moreover
note ‹$p\in G$›
moreover from *calculation*
have $p\ forces_a\ (\pi \in \tau)$
using *forces_mem_iff_dense_below* **by** *blast*
ultimately
show *?thesis* **by** *blast*
qed

end — *G_generic1*

context *forcing_data1*
begin

lemma *Collect_forces_eq_in_M*:
assumes $\tau \in M\ \vartheta \in M$
shows $\{p\in\mathbb{P}.\ p\ forces_a\ (\tau = \vartheta)\} \in M$
using *assms Collect_in_M*[*of forces_eq_fm*(*1*,*2*,*0*,*3*,*4*) [$\mathbb{P}$,*leq*,τ,ϑ]]
arity_forces_eq_fm sats_forces_eq_fm forces_eq_abs forces_eq_fm_type
by (*simp add*: *union_abs1 Un_commute*)

lemma *IV240b_eq_Collects*:
assumes $\tau \in M\ \vartheta \in M$
shows $\{p\in\mathbb{P}.\ \exists\sigma\in domain(\tau) \cup domain(\vartheta).\ p\ forces_a\ (\sigma \in \tau) \wedge p\ forces_a\ (\sigma \notin \vartheta)\}\in M$ **and**

$\{p{\in}\mathbb{P}.\ \exists\sigma{\in}domain(\tau)\cup domain(\vartheta).\ p\ forces_a\ (\sigma\notin\tau)\wedge p\ forces_a\ (\sigma\in\vartheta)\}{\in}M$
proof -
let *?rel_pred*=$\lambda M\ x\ a1\ a2\ a3\ a4.$
$\exists\sigma[M].\ \exists u[M].\ \exists da3[M].\ \exists da4[M].\ is\_domain(M,a3,da3)\wedge is\_domain(M,a4,da4)\wedge$
$union(M,da3,da4,u)\wedge\sigma{\in}u\wedge is\_forces\_mem'(M,a1,a2,x,\sigma,a3)\wedge$
$is\_forces\_nmem'(M,a1,a2,x,\sigma,a4)$
let *?φ*=$Exists(Exists(Exists(Exists(And(domain\_fm(7,1),And(domain\_fm(8,0),$
$And(union\_fm(1,0,2),And(Member(3,2),And(forces\_mem\_fm(5,6,4,3,7),$
$forces\_nmem\_fm(5,6,4,3,8)))))))))$
have *1*:$\sigma{\in}M$ **if** $\langle\sigma,y\rangle{\in}\delta\ \delta{\in}M$ **for** $\sigma\ \delta\ y$
using *that pair_in_M_iff transitivity*[*of* $\langle\sigma,y\rangle\ \delta$] **by** *simp*
have *abs1*:*?rel_pred*$(\#\#M,p,\mathbb{P},leq,\tau,\vartheta)\longleftrightarrow$
$(\exists\sigma{\in}domain(\tau)\cup domain(\vartheta).\ forces\_mem'(\mathbb{P},leq,p,\sigma,\tau)\wedge forces\_nmem'(\mathbb{P},leq,p,\sigma,\vartheta))$
if $p{\in}M$ **for** p
unfolding *forces_mem_def forces_nmem_def*
using *assms that forces_mem'_abs forces_nmem'_abs*
domain_closed Un_closed
by (*auto simp add*:*1*[*of* _ _ τ] *1*[*of* _ _ ϑ])
have *abs2*:*?rel_pred*$(\#\#M,p,\mathbb{P},leq,\vartheta,\tau)\longleftrightarrow(\exists\sigma{\in}domain(\tau)\cup domain(\vartheta).$
$forces\_nmem'(\mathbb{P},leq,p,\sigma,\tau)\wedge forces\_mem'(\mathbb{P},leq,p,\sigma,\vartheta))$ **if** $p{\in}M$ **for** p
unfolding *forces_mem_def forces_nmem_def*
using *assms that forces_mem'_abs forces_nmem'_abs*
domain_closed Un_closed
by (*auto simp add*:*1*[*of* _ _ τ] *1*[*of* _ _ ϑ])
have *fsats1*:$(M,[p,\mathbb{P},leq,\tau,\vartheta]\models$ *?φ*$)\longleftrightarrow$ *?rel_pred*$(\#\#M,p,\mathbb{P},leq,\tau,\vartheta)$ **if** $p{\in}M$ **for** p
using *that assms sats_forces_mem_fm sats_forces_nmem_fm zero_in_M*
domain_closed Un_closed **by** *simp*
have *fsats2*:$(M,[p,\mathbb{P},leq,\vartheta,\tau]\models$ *?φ*$)\longleftrightarrow$ *?rel_pred*$(\#\#M,p,\mathbb{P},leq,\vartheta,\tau)$ **if** $p{\in}M$ **for** p
using *that assms sats_forces_mem_fm sats_forces_nmem_fm zero_in_M*
domain_closed Un_closed **by** *simp*
have *fty*:*?φ*$\in formula$ **by** *simp*
have *farit*:$arity($*?φ*$)=5$
by (*simp add*:*ord_simp_union arity*)
show
$\{p\in\mathbb{P}\ .\ \exists\sigma{\in}domain(\tau)\cup domain(\vartheta).\ p\ forces_a\ (\sigma\in\tau)\wedge p\ forces_a\ (\sigma\notin\vartheta)\}{\in}M$
and $\{p\in\mathbb{P}\ .\ \exists\sigma{\in}domain(\tau)\cup domain(\vartheta).\ p\ forces_a\ (\sigma\notin\tau)\wedge p\ forces_a\ (\sigma\in\vartheta)\}\in M$
unfolding *forces_mem_def*
using *abs1 fty fsats1 farit assms forces_nmem*
Collect_in_M[*of* *?φ* [$\mathbb{P},leq,\tau,\vartheta$]]
using *abs2 fty fsats2 farit assms forces_nmem domain_closed Un_closed*
Collect_in_M[*of* *?φ* [$\mathbb{P},leq,\vartheta,\tau$]]
by *simp_all*
qed

end — *forcing_data1*

context *G_generic1*
begin

lemma *IV240b_eq*:
includes *some_rules*
assumes
$val(G,\tau) = val(G,\vartheta)$ $\tau \in M$ $\vartheta \in M$
and
$IH: \bigwedge \sigma.\ \sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(val(G,\sigma) \in val(G,\tau) \longrightarrow (\exists\, q \in G.\ q\ forces_a\ (\sigma \in \tau))) \wedge$
$(val(G,\sigma) \in val(G,\vartheta) \longrightarrow (\exists\, q \in G.\ q\ forces_a\ (\sigma \in \vartheta)))$

shows
$\exists\, p \in G.\ p\ forces_a\ (\tau = \vartheta)$
proof -
let *?D1*=$\{p \in \mathbb{P}.\ p\ forces_a\ (\tau = \vartheta)\}$
let *?D2*=$\{p \in \mathbb{P}.\ \exists\, \sigma \in domain(\tau) \cup domain(\vartheta).\ p\ forces_a\ (\sigma \in \tau) \wedge p\ forces_a\ (\sigma \notin \vartheta)\}$
let *?D3*=$\{p \in \mathbb{P}.\ \exists\, \sigma \in domain(\tau) \cup domain(\vartheta).\ p\ forces_a\ (\sigma \notin \tau) \wedge p\ forces_a\ (\sigma \in \vartheta)\}$
let *?D*=*?D1* $\cup$ *?D2* $\cup$ *?D3*
note *assms*
moreover from *this*
have $domain(\tau) \cup domain(\vartheta) \in M$ (**is** *?B*$\in M$) **using** *domain_closed Un_closed*
by *auto*
moreover from *calculation*
have *?D2*$\in M$ **and** *?D3*$\in M$ **using** *IV240b_eq_Collects* **by** *simp_all*
ultimately
have *?D*$\in M$ **using** *Collect_forces_eq_in_M Un_closed* **by** *auto*
moreover
have *dense*(*?D*)
proof
fix *p*
assume $p \in \mathbb{P}$
have $\exists\, d \in \mathbb{P}.\ (d\ forces_a\ (\tau = \vartheta) \vee$
$(\exists\, \sigma \in domain(\tau) \cup domain(\vartheta).\ d\ forces_a\ (\sigma \in \tau) \wedge d\ forces_a\ (\sigma \notin \vartheta)) \vee$
$(\exists\, \sigma \in domain(\tau) \cup domain(\vartheta).\ d\ forces_a\ (\sigma \notin \tau) \wedge d\ forces_a\ (\sigma \in \vartheta))) \wedge$
$d \preceq p$
proof (*cases* $p\ forces_a\ (\tau = \vartheta)$)
case *True*
with ‹$p \in \mathbb{P}$›
show *?thesis* **using** *refl_leq* **by** *blast*
next
case *False*
moreover note ‹$p \in \mathbb{P}$›
moreover from *calculation*

```
    obtain σ q where σ∈domain(τ)∪domain(ϑ) q∈ℙ q≼p
      (q forces_a (σ ∈ τ) ∧ ¬ q forces_a (σ ∈ ϑ)) ∨
       (¬ q forces_a (σ ∈ τ) ∧ q forces_a (σ ∈ ϑ))
      using def_forces_eq by blast
    moreover from this
    obtain r where r≼q r∈ℙ
      (r forces_a (σ ∈ τ) ∧ r forces_a (σ ∉ ϑ)) ∨
       (r forces_a (σ ∉ τ) ∧ r forces_a (σ ∈ ϑ))
      using not_forces_nmem strengthening_mem by blast
    ultimately
    show ?thesis using leq_transD by blast
  qed
  then
  show ∃ d∈?D . d ≼ p by blast
qed
moreover
have ?D ⊆ ℙ
  by auto
ultimately
obtain p where p∈G p∈?D
  using M_generic_denseD[of ?D] by blast
then
consider
  (1) p forces_a (τ = ϑ) |
  (2) ∃σ∈domain(τ) ∪ domain(ϑ). p forces_a (σ ∈ τ) ∧ p forces_a (σ ∉ ϑ) |
  (3) ∃σ∈domain(τ) ∪ domain(ϑ). p forces_a (σ ∉ τ) ∧ p forces_a (σ ∈ ϑ)
  by blast
then
show ?thesis
proof (cases)
  case 1
  with ‹p∈G›
  show ?thesis by blast
next
  case 2
  then
  obtain σ where σ∈domain(τ) ∪ domain(ϑ) p forces_a (σ ∈ τ) p forces_a (σ ∉
ϑ)
    by blast
  moreover from this and ‹p∈G› and assms
  have val(G,σ)∈val(G,τ)
    using IV240a[of σ τ] transitivity[OF _ domain_closed[simplified]] by force
  moreover note ‹val(G,τ) = _›
  ultimately
  obtain q where q∈G q forces_a (σ ∈ ϑ)
    using IH[OF ‹σ∈_›]
    by auto
  moreover from this and ‹p∈G›
  obtain r where r∈ℙ r≼p r≼q
```

by *blast*
ultimately
have r *forces*$_a$ $(\sigma \in \vartheta)$
using *strengthening_mem* **by** *blast*
with ‹$r \preceq p$› ‹p *forces*$_a$ $(\sigma \notin \vartheta)$› ‹$r \in \mathbb{P}$›
have *False*
unfolding *forces_nmem_def* **by** *blast*
then
show *?thesis* **by** *simp*
next
case *3*
then
obtain σ **where** $\sigma \in domain(\tau) \cup domain(\vartheta)$ p *forces*$_a$ $(\sigma \in \vartheta)$ p *forces*$_a$ $(\sigma \notin \tau)$
by *blast*
moreover from *this* **and** ‹$p \in G$› **and** *assms*
have $val(G,\sigma) \in val(G,\vartheta)$
using *IV240a*[*of* σ ϑ] *transitivity*[*OF* _ *domain_closed*[*simplified*]] **by** *force*
moreover note ‹$val(G,\tau) = \_$›
ultimately
obtain q **where** $q \in G$ q *forces*$_a$ $(\sigma \in \tau)$
using *IH*[*OF* ‹$\sigma \in \_$›]
by *auto*
moreover from *this* **and** ‹$p \in G$›
obtain r **where** $r \in \mathbb{P}$ $r \preceq p$ $r \preceq q$
by *blast*
ultimately
have r *forces*$_a$ $(\sigma \in \tau)$
using *strengthening_mem* **by** *blast*
with ‹$r \preceq p$› ‹p *forces*$_a$ $(\sigma \notin \tau)$› ‹$r \in \mathbb{P}$›
have *False*
unfolding *forces_nmem_def* **by** *blast*
then
show *?thesis* **by** *simp*
qed
qed

lemma *IV240b*:
$(\tau \in M \longrightarrow \vartheta \in M \longrightarrow val(G,\tau) = val(G,\vartheta) \longrightarrow (\exists\, p \in G.\ p$ *forces*$_a$ $(\tau = \vartheta))) \wedge$
$(\tau \in M \longrightarrow \vartheta \in M \longrightarrow val(G,\tau) \in val(G,\vartheta) \longrightarrow (\exists\, p \in G.\ p$ *forces*$_a$ $(\tau \in \vartheta)))$
(**is** $?Q(\tau,\vartheta) \wedge ?R(\tau,\vartheta)$)
proof (*intro forces_induction*)
fix τ ϑ p
assume $\sigma \in domain(\vartheta) \Longrightarrow ?Q(\tau, \sigma)$ **for** σ
then show $?R(\tau, \vartheta)$
using *IV240b_mem domain_closed transitivity* **by** *simp*
next
fix τ ϑ p

assume $\sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow ?R(\sigma,\tau) \wedge ?R(\sigma,\vartheta)$ **for** σ
moreover from *this*
have $IH'{:}\tau{\in}M \Longrightarrow \vartheta{\in}M \Longrightarrow \sigma \in domain(\tau) \cup domain(\vartheta) \Longrightarrow$
$(val(G,\sigma) \in val(G,\tau) \longrightarrow (\exists\, q{\in}G.\ q\ forces_a\ (\sigma \in \tau))) \wedge$
$(val(G,\sigma) \in val(G,\vartheta) \longrightarrow (\exists\, q{\in}G.\ q\ forces_a\ (\sigma \in \vartheta)))$ **for** σ
using *domain_trans*[*OF trans_M*]
by *blast*
ultimately
show $?Q(\tau,\vartheta)$
using *IV240b_eq* **by** *auto*
qed

lemma *truth_lemma_mem*:
assumes
$env{\in}list(M)$
$n{\in}nat\ m{\in}nat\ n{<}length(env)\ m{<}length(env)$
shows
$(\exists\, p{\in}G.\ p \Vdash Member(n,m)\ env) \longleftrightarrow M[G],\ map(val(G),env) \models Member(n,m)$
using *assms IV240a*[*of* $nth(n,env)\ nth(m,env)$]
IV240b[*of* $nth(n,env)\ nth(m,env)$]
M_genericD
Forces_Member[*of* _ $nth(n,env)\ nth(m,env)\ env\ n\ m$] *map_val_in_MG*
by *auto*

lemma *truth_lemma_eq*:
assumes
$env{\in}list(M)$
$n{\in}nat\ m{\in}nat\ n{<}length(env)\ m{<}length(env)$
shows
$(\exists\, p{\in}G.\ p \Vdash Equal(n,m)\ env) \longleftrightarrow M[G],\ map(val(G),env) \models Equal(n,m)$
using *assms IV240a*(*1*)[*of* $nth(n,env)\ nth(m,env)$]
IV240b(*1*)[*of* $nth(n,env)\ nth(m,env)$]
M_genericD
Forces_Equal[*of* _ $nth(n,env)\ nth(m,env)\ env\ n\ m$] *map_val_in_MG*
by *auto*

end — *G_generic1*

lemma *arities_at_aux*:
assumes
$n \in nat\ m \in nat\ env \in list(M)\ succ(n) \cup succ(m) \le length(env)$
shows
$n < length(env)\ m < length(env)$
using *assms succ_leE*[*OF Un_leD1*, *of* $n\ succ(m)\ length(env)$]
succ_leE[*OF Un_leD2*, *of* $succ(n)\ m\ length(env)$] **by** *auto*

13.13 The Strenghtening Lemma

context *forcing_data1*

begin

lemma *strengthening_lemma*:
assumes
$p{\in}\mathbb{P}$ $\varphi{\in}formula$ $r{\in}\mathbb{P}$ $r{\preceq}p$
$env{\in}list(M)$ $arity(\varphi){\leq}length(env)$
shows
$p \Vdash \varphi\ env \Longrightarrow r \Vdash \varphi\ env$
using *assms*(*2-*)
proof (*induct arbitrary*:*env*)
case (*Member n m*)
then
have $n{<}length(env)$ $m{<}length(env)$
using *arities_at_aux* **by** *simp_all*
moreover
assume $env{\in}list(M)$
moreover
note *assms Member*
ultimately
show *?case*
using *Forces_Member*[*of _ nth*(*n*,*env*) *nth*(*m*,*env*) *env n m*]
strengthening_mem[*of p r nth*(*n*,*env*) *nth*(*m*,*env*)] **by** *simp*
next
case (*Equal n m*)
then
have $n{<}length(env)$ $m{<}length(env)$
using *arities_at_aux* **by** *simp_all*
moreover
assume $env{\in}list(M)$
moreover
note *assms Equal*
ultimately
show *?case*
using *Forces_Equal*[*of _ nth*(*n*,*env*) *nth*(*m*,*env*) *env n m*]
strengthening_eq[*of p r nth*(*n*,*env*) *nth*(*m*,*env*)] **by** *simp*
next
case (*Nand* φ ψ)
with *assms*
show *?case*
using *Forces_Nand transitivity*[*OF _ P_in_M*] *pair_in_M_iff*
transitivity[*OF _ leq_in_M*] *leq_transD* **by** *auto*
next
case (*Forall* φ)
with *assms*
have $p \Vdash \varphi\ ([x]\ @\ env)$ **if** $x{\in}M$ **for** x
using *that Forces_Forall* **by** *simp*
with *Forall*
have $r \Vdash \varphi\ ([x]\ @\ env)$ **if** $x{\in}M$ **for** x
using *that pred_le2* **by** (*simp*)

with *assms Forall*
show *?case*
using *Forces_Forall* **by** *simp*
qed

13.14 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 $arity(\varphi) \leq length(env)$ $arity(\psi) \leq length(env)$
using *assms*
by (*rule_tac Un_leD1*, *rule_tac* [*5*] *Un_leD2*, *auto*)

lemma *dense_below_imp_forces*:
assumes
$p \in \mathbb{P}$ $\varphi \in formula$
$env \in list(M)$ $arity(\varphi) \leq length(env)$
shows
$dense\_below(\{q \in \mathbb{P}.\ (q \Vdash \varphi\ env)\}, p) \Longrightarrow (p \Vdash \varphi\ env)$
using *assms*(*2-*)
proof (*induct arbitrary*:*env*)
case (*Member n m*)
then
have $n < length(env)$ $m < length(env)$
using *arities_at_aux* **by** *simp_all*
moreover
assume $env \in list(M)$
moreover
note *assms Member*
ultimately
show *?case*
using *Forces_Member*[*of _ nth*(*n*,*env*) *nth*(*m*,*env*) *env n m*]
density_mem[*of p nth*(*n*,*env*) *nth*(*m*,*env*)] **by** *simp*
next
case (*Equal n m*)
then
have $n < length(env)$ $m < length(env)$
using *arities_at_aux* **by** *simp_all*
moreover
assume $env \in list(M)$
moreover
note *assms Equal*
ultimately
show *?case*
using *Forces_Equal*[*of _ nth*(*n*,*env*) *nth*(*m*,*env*) *env n m*]
density_eq[*of p nth*(*n*,*env*) *nth*(*m*,*env*)] **by** *simp*
next
case (*Nand* φ ψ)
{

```
    fix q
    assume q∈M q∈ℙ q≼ p q ⊩ φ env
    moreover
    note Nand
    moreover from calculation
    obtain d where d∈ℙ d ⊩ Nand(φ, ψ) env d≼ q
      using dense_belowI by auto
    moreover from calculation
    have ¬(d⊩ ψ env) if d ⊩ φ env
      using that Forces_Nand refl_leq transitivity[OF _ P_in_M, of d] by auto
    moreover
    note arity_Nand_le[of φ ψ]
    moreover from calculation
    have d ⊩ φ env
      using strengthening_lemma[of q φ d env] Un_leD1 by auto
    ultimately
    have ¬ (q ⊩ ψ env)
      using strengthening_lemma[of q ψ d env] by auto
  }
  with ‹p∈ℙ›
  show ?case
    using Forces_Nand[symmetric, OF _ Nand(6,1,3)] by blast
next
  case (Forall φ)
  have dense_below({q∈ℙ. q ⊩ φ ([a]@env)},p) if a∈M for a
  proof
    fix r
    assume r∈ℙ r≼p
    with ‹dense_below(_,p)›
    obtain q where q∈ℙ q≼r q ⊩ Forall(φ) env
      by blast
    moreover
    note Forall ‹a∈M›
    moreover from calculation
    have q ⊩ φ ([a]@env)
      using Forces_Forall by simp
    ultimately
    show ∃ d ∈ {q∈ℙ. q ⊩ φ ([a]@env)}. d ∈ ℙ ∧ d≼r
      by auto
  qed
  moreover
  note Forall(2)[of Cons(_,env)] Forall(1,3-5)
  ultimately
  have p ⊩ φ ([a]@env) if a∈M for a
    using that pred_le2 by simp
  with assms Forall
  show ?case using Forces_Forall by simp
qed
```

lemma *density_lemma*:
assumes
$p{\in}\mathbb{P}$ $\varphi{\in}$*formula* *env*$\in$*list*(M) *arity*(φ)$\leq$*length*(*env*)
shows
$p \Vdash \varphi$ *env* $\longleftrightarrow$ *dense_below*({$q{\in}\mathbb{P}$. ($q \Vdash \varphi$ *env*)},p)
proof
assume *dense_below*({$q{\in}\mathbb{P}$. ($q \Vdash \varphi$ *env*)},p)
with *assms*
show ($p \Vdash \varphi$ *env*)
using *dense_below_imp_forces* **by** *simp*
next
assume $p \Vdash \varphi$ *env*
with *assms*
show *dense_below*({$q{\in}\mathbb{P}$. $q \Vdash \varphi$ *env*},p)
using *strengthening_lemma refl_leq* **by** *auto*
qed

13.15 The Truth Lemma

lemma *Forces_And*:
assumes
$p{\in}\mathbb{P}$ *env* $\in$ *list*(M) $\varphi{\in}$*formula* $\psi{\in}$*formula*
arity(φ) $\leq$ *length*(*env*) *arity*(ψ) $\leq$ *length*(*env*)
shows
$p \Vdash$ *And*(φ,ψ) *env* $\longleftrightarrow$ ($p \Vdash \varphi$ *env*) $\wedge$ ($p \Vdash \psi$ *env*)
proof
assume $p \Vdash$ *And*(φ, ψ) *env*
with *assms*
have *dense_below*({$r \in \mathbb{P}$. ($r \Vdash \varphi$ *env*) $\wedge$ ($r \Vdash \psi$ *env*)}, p)
using *Forces_And_iff_dense_below* **by** *simp*
then
have *dense_below*({$r \in \mathbb{P}$. ($r \Vdash \varphi$ *env*)}, p) *dense_below*({$r \in \mathbb{P}$. ($r \Vdash \psi$ *env*)}, p)
by *blast+*
with *assms*
show ($p \Vdash \varphi$ *env*) $\wedge$ ($p \Vdash \psi$ *env*)
using *density_lemma*[*symmetric*] **by** *simp*
next
assume ($p \Vdash \varphi$ *env*) $\wedge$ ($p \Vdash \psi$ *env*)
have *dense_below*({$r \in \mathbb{P}$. ($r \Vdash \varphi$ *env*) $\wedge$ ($r \Vdash \psi$ *env*)}, p)
proof (*intro dense_belowI bexI conjI*, *assumption*)
fix q
assume $q{\in}\mathbb{P}$ $q \preceq p$
with *assms* ‹($p \Vdash \varphi$ *env*) $\wedge$ ($p \Vdash \psi$ *env*)›
show $q{\in}${$r \in \mathbb{P}$. ($r \Vdash \varphi$ *env*) $\wedge$ ($r \Vdash \psi$ *env*)} $q \preceq q$
using *strengthening_lemma refl_leq* **by** *auto*
qed
with *assms*
show $p \Vdash$ *And*(φ,ψ) *env*

using *Forces_And_iff_dense_below* **by** *simp*
qed

lemma *Forces_Nand_alt*:
assumes
$p\in\mathbb{P}$ $env \in list(M)$ $\varphi\in formula$ $\psi\in formula$
$arity(\varphi) \le length(env)$ $arity(\psi) \le length(env)$
shows
$(p \Vdash Nand(\varphi,\psi)\ env) \longleftrightarrow (p \Vdash Neg(And(\varphi,\psi))\ env)$
using *assms Forces_Nand Forces_And Forces_Neg* **by** *auto*

end

context *G_generic1*
begin

lemma *truth_lemma_Neg*:
assumes
$\varphi\in formula$ $env\in list(M)$ $arity(\varphi)\le length(env)$ **and**
IH: $(\exists\, p\in G.\ p \Vdash \varphi\ env) \longleftrightarrow M[G],\ map(val(G),env) \models \varphi$
shows
$(\exists\, p\in G.\ p \Vdash Neg(\varphi)\ env) \longleftrightarrow M[G],\ map(val(G),env) \models Neg(\varphi)$
proof (*intro iffI*, *elim bexE*, *rule ccontr*)

fix p
assume $p\in G$ $p \Vdash Neg(\varphi)$ env $\neg(M[G],map(val(G),env) \models Neg(\varphi))$
moreover
note *assms*
moreover from *calculation*
have $M[G],\ map(val(G),env) \models \varphi$ $p\in\mathbb{P}$
using *map_val_in_MG* **by** *auto*
with *IH*
obtain r **where** $r \Vdash \varphi$ env $r\in G$ $r\in\mathbb{P}$ **by** *blast*
moreover from *this* **and** ‹$p\in G$›
obtain q **where** $q\preceq p$ $q\preceq r$ $q\in G$ $q\in\mathbb{P}$ $q\in M$
using *transitivity*[*OF _ P_in_M*]
by *blast*
moreover from *calculation*
have $q \Vdash \varphi$ env
using *strengthening_lemma*
by *simp*
with *assms* ‹$p \Vdash \_\ \_$› ‹$q\preceq p$› ‹$q\in M$› ‹$p\in\mathbb{P}$› ‹$q\in\mathbb{P}$›
show *False*
using *Forces_Neg*
by *auto*
next
assume $M[G],\ map(val(G),env) \models Neg(\varphi)$
with *assms*
have $\neg\ (M[G],\ map(val(G),env) \models \varphi)$

```
    using map_val_in_MG by simp
  let ?D={p∈ℙ. (p ⊩ φ env) ∨ (p ⊩ Neg(φ) env)}
  from assms
  have ?D ∈ M
    using separation_disj separation_closed separation_forces by simp
  moreover
  have ?D ⊆ ℙ by auto
  moreover
  have dense(?D)
  proof
    fix q
    assume q∈ℙ
    with assms
    show ∃ d∈{p ∈ ℙ . (p ⊩ φ env) ∨ (p ⊩ Neg(φ) env)}. d≼ q
      using refl_leq Forces_Neg by (cases q ⊩ Neg(φ) env, auto)
  qed
  ultimately
  obtain p where p∈G (p ⊩ φ env) ∨ (p ⊩ Neg(φ) env)
    by blast
  then
  consider (1) p ⊩ φ env | (2) p ⊩ Neg(φ) env by blast
  then
  show ∃ p∈G. (p ⊩ Neg(φ) env)
  proof (cases)
    case 1
    with ‹¬ (M[G],map(val(G),env) ⊨ φ)› ‹p∈G› IH
    show ?thesis
      by blast
  next
    case 2
    with ‹p∈G›
    show ?thesis by blast
  qed
qed

lemma truth_lemma_And:
  assumes
    env∈list(M) φ∈formula ψ∈formula
    arity(φ)≤length(env) arity(ψ) ≤ length(env)
    and
    IH: (∃ p∈G. p ⊩ φ env)  ⟷   M[G], map(val(G),env) ⊨ φ
    (∃ p∈G. p ⊩ ψ env)  ⟷   M[G], map(val(G),env) ⊨ ψ
  shows
    (∃ p∈G. (p ⊩ And(φ,ψ) env)) ⟷ M[G] , map(val(G),env) ⊨ And(φ,ψ)
  using assms map_val_in_MG Forces_And[OF M_genericD assms(1-5)]
proof (intro iffI, elim bexE)
  fix p
  assume p∈G p ⊩ And(φ,ψ) env
  with assms
```

```
  show M[G], map(val(G),env) ⊨ And(φ,ψ)
    using Forces_And[of _ _ φ ψ] map_val_in_MG M_genericD by auto
next
  assume M[G], map(val(G),env) ⊨ And(φ,ψ)
  moreover
  note assms
  moreover from calculation
  obtain q r where q ⊩ φ env r ⊩ ψ env q∈G r∈G r∈ℙ q∈ℙ
   using map_val_in_MG Forces_And[OF M_genericD assms(1-5)] M_genericD
by auto
  moreover from calculation
  obtain p where p≼q p≼r p∈G
    by auto
  moreover from calculation
  have (p ⊩ φ env) ∧ (p ⊩ ψ env)
    using strengthening_lemma[OF M_genericD] by force
  ultimately
  show ∃p∈G. (p ⊩ And(φ,ψ) env)
    using Forces_And[OF M_genericD assms(1-5)] by auto
qed

end

definition
  ren_truth_lemma :: i⇒i where
  ren_truth_lemma(φ) ≡
    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(λp. incr_bv(p)`5 , 6, φ))))))))))

lemma ren_truth_lemma_type[TC] :
  φ∈formula ⟹ ren_truth_lemma(φ) ∈formula
  unfolding ren_truth_lemma_def
  by simp

lemma arity_ren_truth :
  assumes φ∈formula
  shows arity(ren_truth_lemma(φ)) ≤ 6 ∪ succ(arity(φ))
proof -
  consider (lt) 5 <arity(φ) | (ge) ¬ 5 < arity(φ)
    by auto
  then
  show ?thesis
  proof cases
    case lt
    consider (a) 5<arity(φ)+ω5 | (b) arity(φ)+ω5 ≤ 5
      using not_lt_iff_le ‹φ∈_› by force
    then
    show ?thesis
```

```
    proof cases
      case a
      with ‹φ∈_› lt
      have 5 < succ(arity(φ)) 5<arity(φ)+ω2  5<arity(φ)+ω3  5<arity(φ)+ω4
        using succ_ltI by auto
      with ‹φ∈_›
      have c:arity(iterates(λp. incr_bv(p)‘5,5,φ)) = 5+ωarity(φ) (is arity(?φ′) =
_)
        using arity_incr_bv_lemma lt a
        by simp
      with ‹φ∈_›
      have arity(incr_bv(?φ′)‘5) = 6+ωarity(φ)
        using arity_incr_bv_lemma[of ?φ′ 5] a by auto
      with ‹φ∈_›
      show ?thesis
        unfolding ren_truth_lemma_def
        using pred_Un_distrib union_abs1 Un_assoc[symmetric] a c union_abs2
        by (simp add:arity)
    next
      case b
      with ‹φ∈_› lt
       have 5 < succ(arity(φ)) 5<arity(φ)+ω2  5<arity(φ)+ω3  5<arity(φ)+ω4
5<arity(φ)+ω5
        using succ_ltI by auto
      with ‹φ∈_›
     have arity(iterates(λp. incr_bv(p)‘5,6,φ)) = 6+ωarity(φ) (is arity(?φ′) = _)
        using arity_incr_bv_lemma lt
        by simp
      with ‹φ∈_›
      show ?thesis
        unfolding ren_truth_lemma_def
        using pred_Un_distrib union_abs1 Un_assoc[symmetric]  union_abs2
        by (simp add:arity)
    qed
  next
    case ge
    with ‹φ∈_›
    have arity(φ) ≤ 5 pred^5(arity(φ)) ≤ 5
      using not_lt_iff_le le_trans[OF le_pred]
      by auto
    with ‹φ∈_›
   have arity(iterates(λp. incr_bv(p)‘5,6,φ)) = arity(φ) arity(φ)≤6 pred^5(arity(φ))
≤ 6
        using arity_incr_bv_lemma ge le_trans[OF ‹arity(φ)≤5›] le_trans[OF
‹pred^5(arity(φ))≤5›]
      by auto
    with ‹arity(φ) ≤ 5› ‹φ∈_› ‹pred^5(_) ≤ 5›
    show ?thesis
      unfolding ren_truth_lemma_def
```

```
    using pred_Un_distrib union_abs1 Un_assoc[symmetric] union_abs2
    by (simp add:arity)
  qed
qed

lemma sats_ren_truth_lemma:
  [q,b,d,a1,a2,a3] @ env ∈ list(M) ⟹ φ ∈ formula ⟹
   (M, [q,b,d,a1,a2,a3] @ env ⊨ ren_truth_lemma(φ) ) ⟷
   (M, [q,a1,a2,a3,b] @ env ⊨ φ)
  unfolding ren_truth_lemma_def
  by (insert sats_incr_bv_iff [of _ _ M _ [q,a1,a2,a3,b]], simp)

context forcing_data1
begin

lemma truth_lemma′ :
  assumes
    φ∈formula env∈list(M) arity(φ) ≤ succ(length(env))
  shows
    separation(##M,λd. ∃ b∈M. ∀ q∈ℙ. q⪯d ⟶ ¬(q ⊩ φ ([b]@env)))
proof -
  let ?rel_pred=λM x a1 a2 a3. ∃ b∈M. ∀ q∈M. q∈a1 ∧ is_leq(##M,a2,q,x) ⟶
                ¬(M, [q,a1,a2,a3,b] @ env ⊨ forces(φ))
  let ?ψ=Exists(Forall(Implies(And(Member(0,3),is_leq_fm(4,0,2)),
        Neg(ren_truth_lemma(forces(φ))))))
  have q∈M if q∈ℙ for q using that transitivity[OF _ P_in_M] by simp
  then
  have 1:∀ q∈M. q∈ℙ ∧ R(q) ⟶ Q(q) ⟹ (∀ q∈ℙ. R(q) ⟶ Q(q)) for R Q
    by auto
  then
  have ⟦b ∈ M; ∀ q∈M. q ∈ ℙ ∧ q ⪯ d ⟶ ¬(q ⊩ φ ([b]@env))⟧ ⟹
        ∃ c∈M. ∀ q∈ℙ. q ⪯ d ⟶ ¬(q ⊩ φ ([c]@env)) for b d
    by (rule bexI,simp_all)
  then
  have ?rel_pred(M,d,ℙ,leq,1) ⟷ (∃ b∈M. ∀ q∈ℙ. q⪯d ⟶ ¬(q ⊩ φ ([b]@env)))
if d∈M for d
    using that leq_abs assms
    by auto
  moreover
  have ?ψ∈formula using assms by simp
  moreover
  have (M, [d,ℙ,leq,1]@env ⊨ ?ψ) ⟷ ?rel_pred(M,d,ℙ,leq,1) if d∈M for d
    using assms that sats_is_leq_fm sats_ren_truth_lemma zero_in_M
    by simp
  moreover
  have arity(?ψ) ≤ 4+ω length(env)
  proof -
    have eq:arity(is_leq_fm(4, 0, 2)) = 5
      using arity_is_leq_fm succ_Un_distrib ord_simp_union
```

```
    by simp
  with ‹φ∈_›
  have arity(?ψ) = 3 ∪ (pred^2(arity(ren_truth_lemma(forces(φ)))))
    using union_abs1 pred_Un_distrib by (simp add:arity)
  moreover
  have ... ≤ 3 ∪ (pred(pred(6 ∪ succ(arity(forces(φ)))))) (is _ ≤ ?r)
    using ‹φ∈_› Un_le_compat[OF le_refl[of 3]]
      le_imp_subset arity_ren_truth[of forces(φ)]
      pred_mono
    by auto
  finally
  have arity(?ψ) ≤ ?r by simp
  have i:?r ≤ 4 ∪ pred(arity(forces(φ)))
  using pred_Un_distrib pred_succ_eq ‹φ∈_› Un_assoc[symmetric] union_abs1
by simp
  have h:4 ∪ pred(arity(forces(φ))) ≤ 4 ∪ (4+_ω length(env))
    using ‹env∈_› add_commute ‹φ∈_›
        Un_le_compat[of 4 4,OF _ pred_mono[OF _ arity_forces_le[OF _ _
‹arity(φ)≤_›]] ]
      ‹env∈_› by auto
  with ‹φ∈_› ‹env∈_›
  show ?thesis
    using le_trans[OF ‹arity(?ψ) ≤ ?r› le_trans[OF i h]] ord_simp_union by
simp
 qed
 ultimately
 show ?thesis using assms
    separation_ax[of ?ψ [ℙ,leq,1]@env]
    separation_cong[of ##M λy. (M, [y,ℙ,leq,1]@env ⊨ ?ψ)]
  by simp
qed

end

context G_generic1
begin

lemma truth_lemma:
 assumes
  φ∈formula
  env∈list(M) arity(φ)≤length(env)
 shows
  (∃p∈G. p ⊩ φ env)  ⟷  M[G], map(val(G),env) ⊨ φ
 using assms
proof (induct arbitrary:env)
 case (Member x y)
 then
 show ?case
  using truth_lemma_mem[OF ‹env∈list(M)› ‹x∈nat› ‹y∈nat›] arities_at_aux
```

```
    by simp
next
  case (Equal x y)
  then
  show ?case
    using truth_lemma_eq[OF ‹env∈list(M)› ‹x∈nat› ‹y∈nat›] arities_at_aux by
simp
next
  case (Nand φ ψ)
  then
  show ?case
    using truth_lemma_And truth_lemma_Neg[of ·φ ∧ ψ·] Forces_Nand_alt
      M_genericD map_val_in_MG arity_Nand_le[of φ ψ] FOL_arities by auto
next
  case (Forall φ)
  then
  show ?case
  proof (intro iffI)
    assume ∃ p∈G. (p ⊩ Forall(φ) env)
    then
    obtain p where p∈G p∈M p∈ℙ p ⊩ Forall(φ) env
      using transitivity[OF _ P_in_M] by auto
    with ‹env∈list(M)› ‹φ∈formula›
    have p ⊩ φ ([x]@env) if x∈M for x
      using that Forces_Forall by simp
    with ‹p∈G› ‹φ∈formula› ‹env∈_› ‹arity(Forall(φ)) ≤ length(env)›
      Forall(2)[of Cons(_,env)]
    show M[G], map(val(G),env) ⊨ Forall(φ)
      using pred_le2 map_val_in_MG
      by (auto iff:GenExt_iff)
  next
    assume M[G], map(val(G),env) ⊨ Forall(φ)
    let ?D1={d∈ℙ. (d ⊩ Forall(φ) env)}
    let ?D2={d∈ℙ. ∃ b∈M. ∀ q∈ℙ. q≼d ⟶ ¬(q ⊩ φ ([b]@env))}
    define D where D ≡ ?D1 ∪ ?D2
    note ‹arity(Forall(φ)) ≤ length(env)› ‹φ∈formula› ‹env∈list(M)›
    moreover from this
    have arφ:arity(φ)≤succ(length(env))
      using pred_le2 by simp
    moreover from calculation
    have ?D1∈M using Collect_forces by simp
    moreover from ‹env∈list(M)› ‹φ∈formula›
    have ?D2∈M
      using truth_lemma'[of φ] separation_closed arφ
      by simp
    ultimately
    have D∈M unfolding D_def using Un_closed by simp
    moreover
```

have $D \subseteq \mathbb{P}$ **unfolding** *D_def* **by** *auto*
moreover
have *dense*(D)
proof
  fix p
  assume $p{\in}\mathbb{P}$
  show $\exists\, d{\in}D.\ d{\preceq}\ p$
  proof (*cases* $p \Vdash$ *Forall*(φ) *env*)
    case *True*
    with ‹$p{\in}\mathbb{P}$›
    show *?thesis* **unfolding** *D_def* **using** *refl_leq* **by** *blast*
  next
    case *False*
    with *Forall* ‹$p{\in}\mathbb{P}$›
    obtain b **where** $b{\in}M$ $\neg(p \Vdash \varphi\ ([b]@env))$
      using *Forces_Forall* **by** *blast*
    moreover from *this* ‹$p{\in}\mathbb{P}$› *Forall*
    have $\neg$*dense_below*($\{q{\in}\mathbb{P}.\ q \Vdash \varphi\ ([b]@env)\}$,$p$)
      using *density_lemma pred_le2* **by** *auto*
    moreover from *this*
    obtain d **where** $d{\preceq}p\ \forall\, q{\in}\mathbb{P}.\ q{\preceq}d \longrightarrow \neg(q \Vdash \varphi\ ([b]\ @\ env))$
      $d{\in}\mathbb{P}$ **by** *blast*
    ultimately
    show *?thesis* **unfolding** *D_def* **by** *auto*
  qed
qed
moreover
note *generic*
ultimately
obtain d **where** $d \in D\ d \in G$ **by** *blast*
then
consider (*1*) $d{\in}$*?D1* | (*2*) $d{\in}$*?D2* **unfolding** *D_def* **by** *blast*
then
show $\exists\, p{\in}G.\ (p \Vdash$ *Forall*(φ) *env*)
proof (*cases*)
  case *1*
  with ‹$d{\in}G$›
  show *?thesis* **by** *blast*
next
  case *2*
  then
  obtain b **where** $b{\in}M\ \forall\, q{\in}\mathbb{P}.\ q{\preceq}d \longrightarrow\neg(q \Vdash \varphi\ ([b]\ @\ env))$
    by *blast*
  moreover from *this*(*1*) **and** ‹$M[G], \_ \models$ *Forall*(φ)› **and**
    Forall(*2*)[*of Cons*(b,*env*)] *Forall*(*1*,*3*-)
  obtain p **where** $p{\in}G\ p{\in}\mathbb{P}\ p \Vdash \varphi\ ([b]\ @\ env)$
    using *pred_le2 map_val_in_MG M_genericD* **by** (*auto iff*:*GenExt_iff*)
  moreover
  note ‹$d{\in}G$›

```
      ultimately
      obtain q where q∈G q∈ℙ q≼d q≼p
        using M_genericD by force
      moreover from this and  ‹p ⊩ φ ([b] @ env)›
        Forall  ‹b∈M› ‹p∈ℙ›
      have q ⊩ φ ([b] @ env)
        using pred_le2 strengthening_lemma by simp
      moreover
      note ‹∀ q∈ℙ. q≼d ⟶¬(q ⊩ φ ([b] @ env))›
      ultimately
      show ?thesis by simp
    qed
  qed
qed

end

context forcing_data1
begin
```

13.16 The "Definition of forcing"

```
lemma definition_of_forcing:
  assumes
    p∈ℙ φ∈formula env∈list(M) arity(φ)≤length(env)
  shows
    (p ⊩ φ env) ⟷
     (∀ G. M_generic(G) ∧ p∈G ⟶ M[G], map(val(G),env) ⊨ φ)
proof (intro iffI allI impI, elim conjE)
  fix G
  assume (p ⊩ φ env) M_generic(G) p ∈ G
  moreover from this
  interpret G_generic1 ℙ leq 1 M enum G
    by (unfold_locales,simp)
  from calculation assms
  show M[G], map(val(G),env) ⊨ φ
    using truth_lemma[of φ] by auto
next
  assume 1: ∀ G.(M_generic(G)∧ p∈G) ⟶ M[G] , map(val(G),env) ⊨ φ
  {
    fix r
    assume 2: r∈ℙ r≼p
    then
    obtain G where r∈G M_generic(G)
```

Here we're using countability (via the existence of generic filters) of M as a shortcut.

```
      using generic_filter_existence by auto
    moreover from this
```

```
    interpret G_generic1 ℙ leq 1 M enum G
      by (unfold_locales,simp)
    from calculation 2 ‹p∈ℙ›
    have p∈G
      using filter_leqD by auto
    moreover note 1
    ultimately
    have M[G], map(val(G),env) ⊨ φ
      by simp
    moreover
    note assms
    moreover from calculation
    obtain s where s∈G (s ⊩ φ env)
      using truth_lemma[of φ] by blast
    moreover from this ‹r∈G›
    obtain q where q∈G q≼s q≼r s∈ℙ q∈ℙ
      by blast
    ultimately
    have ∃ q∈ℙ. q≼r ∧ (q ⊩ φ env)
      using strengthening_lemma[of s] by auto
  }
  then
  have dense_below({q∈ℙ. (q ⊩ φ env)},p)
    unfolding dense_below_def by blast
  with assms
  show (p ⊩ φ env)
    using density_lemma by blast
qed

lemmas definability = forces_type

end — forcing_data1

end
```

14 Ordinals in generic extensions

```
theory Ordinals_In_MG
  imports
    Forcing_Theorems
begin

context G_generic1
begin

lemma rank_val: rank(val(G,x)) ≤ rank(x) (is ?Q(x))
proof (induct rule:ed_induction[of ?Q])
  case (1 x)
  have val(G,x) = {val(G,u). u∈{t∈domain(x). ∃ p∈G . ⟨t,p⟩∈x }}
```

using *def_val*[*of* G x] **by** *auto*
then
have $rank(val(G,x)) = (\bigcup u{\in}\{t{\in}domain(x).\ \exists\, p{\in}G\ .\ \langle t,p\rangle{\in}x\ \}.\ succ(rank(val(G,u))))$
using *rank*[*of* $val(G,x)$] **by** *simp*
moreover
have $succ(rank(val(G,\ y))) \le rank(x)$ **if** $ed(y,\ x)$ **for** y
using *1*[*OF that*] *rank_ed*[*OF that*] **by** (*auto intro*:*lt_trans1*)
moreover from *this*
have $(\bigcup u{\in}\{t{\in}domain(x).\ \exists\, p{\in}G\ .\ \langle t,p\rangle{\in}x\ \}.\ succ(rank(val(G,u)))) \le rank(x)$
by (*rule_tac UN_least_le*) (*auto*)
ultimately
show *?case*
by *simp*
qed

lemma *Ord_MG_iff*:
assumes $Ord(\alpha)$
shows $\alpha \in M \longleftrightarrow \alpha \in M[G]$
proof
show $\alpha \in M[G]$ **if** $\alpha \in M$
using *M_subset_MG*[*OF one_in_G*] *that* **..**
next
assume $\alpha \in M[G]$
then
obtain x **where** $x{\in}M$ $val(G,x) = \alpha$
using *GenExtD* **by** *auto*
then
have $rank(\alpha) \le rank(x)$
using *rank_val* **by** *blast*
with *assms*
have $\alpha \le rank(x)$
using *rank_of_Ord* **by** *simp*
then
have $\alpha \in succ(rank(x))$
using *ltD* **by** *simp*
with ‹$x{\in}M$›
show $\alpha \in M$
using *cons_closed transitivity*[*of* α $succ(rank(x))$] *rank_closed*
unfolding *succ_def* **by** *simp*
qed

end — *G_generic1*

end

15 Auxiliary renamings for Separation

theory *Separation_Rename*
imports

Interface
begin

no_notation *Aleph* (‹ℵ_› [*90*] *90*)

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

lemma *nth_concat* : [*p*,*t*] ∈ *list*(*A*) ⟹ *env*∈ *list*(*A*) ⟹ *nth*(*1* +$_\omega$ *length*(*env*),[*p*]@ *env* @ [*t*]) = *t*
  by(*auto simp add*:*nth_append*)

lemma *nth_concat2* : *env*∈ *list*(*A*) ⟹ *nth*(*length*(*env*),*env* @ [*p*,*t*]) = *p*
  by(*auto simp add*:*nth_append*)

lemma *nth_concat3* : *env*∈ *list*(*A*) ⟹ *u* = *nth*(*succ*(*length*(*env*)), *env* @ [*pi*, *u*])
  by(*auto simp add*:*nth_append*)

definition
  sep_var :: *i* ⇒ *i* **where**
  sep_var(*n*) ≡ {⟨*0*,*1*⟩,⟨*1*,*3*⟩,⟨*2*,*4*⟩,⟨*3*,*5*⟩,⟨*4*,*0*⟩,⟨*5*+$_\omega$*n*,*6*⟩,⟨*6*+$_\omega$*n*,*2*⟩}

definition
  sep_env :: *i* ⇒ *i* **where**
  sep_env(*n*) ≡ λ *i* ∈ (*5*+$_\omega$*n*)-*5* . *i*+$_\omega$*2*

definition *weak* :: [*i*, *i*] ⇒ *i* **where**
  weak(*n*,*m*) ≡ {*i*+$_\omega$*m* . *i* ∈ *n*}

lemma *weakD* :
  assumes *n* ∈ *nat* *k*∈*nat* *x* ∈ *weak*(*n*,*k*)
  shows ∃ *i* ∈ *n* . *x* = *i*+$_\omega$*k*
  using *assms* **unfolding** *weak_def* **by** *blast*

lemma *weak_equal* :
  assumes *n*∈*nat* *m*∈*nat*
  shows *weak*(*n*,*m*) = (*m*+$_\omega$*n*) - *m*
proof -
  have *weak*(*n*,*m*)⊆(*m*+$_\omega$*n*)-*m*
  proof(*intro subsetI*)
    fix *x*
    assume *x*∈*weak*(*n*,*m*)
    with *assms*
    obtain *i* **where**
      i∈*n* *x*=*i*+$_\omega$*m*
      using *weakD* **by** *blast*
    then
    have *m*≤*i*+$_\omega$*m* *i*<*n*
      using *add_le_self2*[*of m i*] ‹*m*∈*nat*› ‹*n*∈*nat*› *ltI*[*OF* ‹*i*∈*n*›] **by** *simp_all*
    then

```
    have ¬i+ω m<m
      using not_lt_iff_le in_n_in_nat[OF ‹n∈nat› ‹i∈n›] ‹m∈nat› by simp
    with ‹x=i+ω m›
    have x∉m
      using ltI ‹m∈nat› by auto
    moreover
    from assms ‹x=i+ω m› ‹i<n›
    have x<m+ω n
      using add_lt_mono1[OF ‹i<n› ‹n∈nat›] by simp
    ultimately
    show x∈(m+ω n)-m
      using ltD DiffI by simp
  qed
  moreover
  have (m+ω n)-m⊆weak(n,m)
  proof (intro subsetI)
    fix x
    assume x∈(m+ω n)-m
    then
    have x∈m+ω n x∉m
      using DiffD1[of x n+ω m m] DiffD2[of x n+ω m m] by simp_all
    then
    have x<m+ω n x∈nat
      using ltI in_n_in_nat[OF add_type[of m n]] by simp_all
    then
    obtain i where
      m+ω n = succ(x+ω i)
      using less_iff_succ_add[OF ‹x∈nat›,of m+ω n] add_type by auto
    then
    have x+ω i<m+ω n using succ_le_iff by simp
    with ‹x∉m›
    have ¬x<m using ltD by blast
    with ‹m∈nat› ‹x∈nat›
    have m≤x using not_lt_iff_le by simp
    with ‹x<m+ω n›  ‹n∈nat›
    have x-ω m<m+ω n-ω m
      using diff_mono[OF ‹x∈nat› _ ‹m∈nat›] by simp
    have m+ω n-ω m =  n using diff_cancel2 ‹m∈nat› ‹n∈nat› by simp
    with ‹x-ω m<m+ω n-ω m› ‹x∈nat›
    have x-ω m ∈ n x=x-ω m+ω m
      using ltD add_diff_inverse2[OF ‹m≤x›] by simp_all
    then
    show x∈weak(n,m)
      unfolding weak_def by auto
  qed
  ultimately
  show ?thesis by auto
qed
```

```
lemma weak_zero:
  shows weak(0,n) = 0
  unfolding weak_def by simp

lemma weakening_diff :
  assumes n ∈ nat
  shows weak(n,7) - weak(n,5) ⊆ {5+ω n, 6+ω n}
  unfolding weak_def using assms
proof(auto)
  {
    fix i
    assume i∈n succ(succ(natify(i)))≠n ∀ w∈n. succ(succ(natify(i))) ≠ natify(w)
    then
    have i<n
      using ltI ‹n∈nat› by simp
    from ‹n∈nat› ‹i∈n› ‹succ(succ(natify(i)))≠n›
    have i∈nat succ(succ(i))≠n using in_n_in_nat by simp_all
    from ‹i<n›
    have succ(i)≤n using succ_leI by simp
    with ‹n∈nat›
    consider (a) succ(i) = n | (b) succ(i) < n
      using leD by auto
    then have succ(i) = n
    proof cases
      case a
      then show ?thesis .
    next
      case b
      then
      have succ(succ(i))≤n using succ_leI by simp
      with ‹n∈nat›
      consider (a) succ(succ(i)) = n | (b) succ(succ(i)) < n
        using leD by auto
      then have succ(i) = n
      proof cases
        case a
        with ‹succ(succ(i))≠n› show ?thesis by blast
      next
        case b
        then
        have succ(succ(i))∈n using ltD by simp
        with ‹i∈nat›
        have succ(succ(natify(i))) ≠ natify(succ(succ(i)))
          using ‹∀ w∈n. succ(succ(natify(i))) ≠ natify(w)› by auto
        then
        have False using ‹i∈nat› by auto
        then show ?thesis by blast
      qed
      then show ?thesis .
```

```
    qed
    with ‹i∈nat› have succ(natify(i)) = n by simp
  }
  then
  show n ∈ nat ⟹
    succ(succ(natify(y))) ≠ n ⟹
    ∀ x∈n. succ(succ(natify(y))) ≠ natify(x) ⟹
    y ∈ n ⟹ succ(natify(y)) = n for y
    by blast
qed

lemma in_add_del :
  assumes x∈j+ω n n∈nat j∈nat
  shows x < j ∨ x ∈ weak(n,j)
proof (cases x<j)
  case True
  then show ?thesis ..
next
  case False
  have x∈nat j+ω n∈nat
    using in_n_in_nat[OF _ ‹x∈j+ω n›] assms by simp_all
  then
  have j ≤ x x < j+ω n
    using not_lt_iff_le False ‹j∈nat› ‹n∈nat› ltI[OF ‹x∈j+ω n›] by auto
  then
  have x-ω j < (j +ω n) -ω j x = j +ω (x -ω j)
    using diff_mono ‹x∈nat› ‹j+ω n∈nat› ‹j∈nat› ‹n∈nat›
      add_diff_inverse[OF ‹j≤x›] by simp_all
  then
  have x-ω j < n x = (x -ω j ) +ω j
    using diff_add_inverse ‹n∈nat› add_commute by simp_all
  then
  have x-ω j ∈n using ltD by simp
  then
  have x ∈ weak(n,j)
    unfolding weak_def
    using ‹x= (x-ω j) +ω j› RepFunI[OF ‹x-ω j∈n›] add_commute by force
  then show ?thesis ..
qed


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

```
  from assms
  have A: 5+ω length(env)∈nat [p, P, leq, o, t] ∈list(M)
    by simp_all
  let ?f=sep_env(length(env))
  have EQ: weak(length(env),5) = 5+ω length(env) - 5
    using weak_equal length_type[OF ‹env∈list(M)›] by simp
  let ?tgt=[t,p,u,P,leq,o,pi]@env
  let ?src=[p,P,leq,o,t] @ env @ [pi,u]
  have nth(?f‘i,[t,p,u,P,leq,o,pi]@env) = nth(i,[p,P,leq,o,t] @ env @ [pi,u])
    if i ∈ (5+ω length(env)-5) for i
  proof -
    from that
    have 2: i ∈ 5+ω length(env)  i ∉ 5 i ∈ nat i-ω 5∈nat i+ω 2∈nat
      using in_n_in_nat[OF ‹5+ω length(env)∈nat›] by simp_all
    then
    have 3: ¬ i < 5 using ltD by force
    then
    have 5 ≤ i 2 ≤ 5
      using  not_lt_iff_le ‹i∈nat› by simp_all
    then have 2 ≤ i using le_trans[OF ‹2≤5›] by simp
    from A ‹i ∈ 5+ω length(env)›
    have i < 5+ω length(env) using ltI by simp
    with ‹i∈nat› ‹2≤i› A
    have C:i+ω 2 < 7+ω length(env)  by simp
    with that
    have B: ?f‘i = i+ω 2 unfolding sep_env_def by simp
    from 3 assms(1) ‹i∈nat›
    have ¬ i+ω 2 < 7 using not_lt_iff_le add_le_mono by simp
    from ‹i < 5+ω length(env)› 3 ‹i∈nat›
    have i-ω 5 < 5+ω length(env) -ω 5
      using diff_mono[of i 5+ω length(env) 5,OF _ _ _ ‹i < 5+ω length(env)›]
        not_lt_iff_le[THEN iffD1] by force
    with assms(2)
    have i-ω 5 < length(env) using diff_add_inverse length_type by simp
    have nth(i,?src) =nth(i-ω 5,env@[pi,u])
      using nth_append[OF A(2) ‹i∈nat›] 3 by simp
    also
    have ... = nth(i-ω 5, env)
      using nth_append[OF ‹env ∈list(M)› ‹i-ω 5∈nat›] ‹i-ω 5 < length(env)› by
simp
    also
    have ... = nth(i+ω 2, ?tgt)
      using nth_append[OF assms(1) ‹i+ω 2∈nat›] ‹¬ i+ω 2 <7› by simp
    ultimately
    have nth(i,?src) = nth(?f‘i,?tgt)
      using B by simp
    then show ?thesis using that by simp
  qed
  then show ?thesis using EQ by force
```

qed

lemma *sep_env_type* :
assumes $n \in nat$
shows $sep\_env(n) : (5+_\omega n)\text{-}5 \to (7+_\omega n)\text{-}7$
proof -
let *?h*=$sep\_env(n)$
from ‹$n \in nat$›
have $(5+_\omega n)+_\omega 2 = 7+_\omega n$ $7+_\omega n \in nat$ $5+_\omega n \in nat$ **by** *simp_all*
have
D: $sep\_env(n)\text{`}x \in (7+_\omega n)\text{-}7$ **if** $x \in (5+_\omega n)\text{-}5$ **for** x
proof -
from ‹$x \in 5+_\omega n\text{-}5$›
have *?h*`$x = x+_\omega 2$ $x<5+_\omega n$ $x \in nat$
unfolding *sep_env_def* **using** *ltI in_n_in_nat*[*OF* ‹$5+_\omega n \in nat$›] **by** *simp_all*
then
have $x+_\omega 2 < 7+_\omega n$ **by** *simp*
then
have $x+_\omega 2 \in 7+_\omega n$ **using** *ltD* **by** *simp*
from ‹$x \in 5+_\omega n\text{-}5$›
have $x \notin 5$ **by** *simp*
then have $\neg x<5$ **using** *ltD* **by** *blast*
then have $5 \le x$ **using** *not_lt_iff_le* ‹$x \in nat$› **by** *simp*
then have $7 \le x+_\omega 2$ **using** *add_le_mono* ‹$x \in nat$› **by** *simp*
then have $\neg x+_\omega 2<7$ **using** *not_lt_iff_le* ‹$x \in nat$› **by** *simp*
then have $x+_\omega 2 \notin 7$ **using** *ltI* ‹$x \in nat$› **by** *force*
with ‹$x+_\omega 2 \in 7+_\omega n$› **show** *?thesis* **using** ‹*?h*`$x = x+_\omega 2$› *DiffI* **by** *simp*
qed
then show *?thesis* **unfolding** *sep_env_def* **using** *lam_type* **by** *simp*
qed

lemma *sep_var_fin_type* :
assumes $n \in nat$
shows $sep\_var(n) : 7+_\omega n \ \text{-||>}\ 7+_\omega n$
unfolding *sep_var_def*
using *consI ltD emptyI* **by** *force*

lemma *sep_var_domain* :
assumes $n \in nat$
shows $domain(sep\_var(n)) = \ 7+_\omega n$ - $weak(n,5)$
proof -
let *?A*=$weak(n,5)$
have *A*:$domain(sep\_var(n)) \subseteq (7+_\omega n)$
unfolding *sep_var_def*
by(*auto simp add*: *le_natE*)
have *C*: $x=5+_\omega n \vee x=6+_\omega n \vee x \le 4$ **if** $x \in domain(sep\_var(n))$ **for** x
using *that* **unfolding** *sep_var_def* **by** *auto*
have *D* : $x<n+_\omega 7$ **if** $x \in 7+_\omega n$ **for** x
using *that* ‹$n \in nat$› *ltI* **by** *simp*

```
have ¬ 5+ω n < 5+ω n using ‹n∈nat› lt_irrefl[of _ False] by force
have ¬ 6+ω n < 5+ω n using ‹n∈nat› by force
have R: x < 5+ω n if x∈?A for x
proof -
  from that
  obtain i where
    i<n x=5+ω i
    unfolding weak_def
    using ltI ‹n∈nat› RepFun_iff by force
  with ‹n∈nat›
  have 5+ω i < 5+ω n using add_lt_mono2 by simp
  with ‹x=5+ω i›
  show x < 5+ω n by simp
qed
then
have 1:x∉?A if ¬x <5+ω n for x using that by blast
have 5+ω n ∉ ?A 6+ω n∉?A
proof -
  show 5+ω n ∉ ?A using 1 ‹¬5+ω n<5+ω n› by blast
  with 1 show 6+ω n ∉ ?A using ‹¬6+ω n<5+ω n› by blast
qed
then
have E:x∉?A if x∈domain(sep_var(n)) for x
  unfolding weak_def
  using C that by force
then
have F: domain(sep_var(n)) ⊆ 7+ω n - ?A using A by auto
from assms
have x<7 ∨ x∈weak(n,7) if x∈7+ω n for x
  using in_add_del[OF ‹x∈7+ω n›] by simp
moreover
{
  fix x
  assume asm:x∈7+ω n  x∉?A  x∈weak(n,7)
  then
  have x∈domain(sep_var(n))
  proof -
    from ‹n∈nat›
    have weak(n,7)-weak(n,5)⊆{n+ω 5,n+ω 6}
      using weakening_diff by simp
    with ‹x∉?A› asm
    have x∈{n+ω 5,n+ω 6} using subsetD DiffI by blast
    then
    show ?thesis unfolding sep_var_def by simp
  qed
}
moreover
{
  fix x
```

```
    assume asm:x∈7+ω n  x∉?A x<7
    then have x∈domain(sep_var(n))
    proof (cases 2 ≤ n)
      case True
      moreover
      have 0<n using  leD[OF ‹n∈nat› ‹2≤n›] lt_imp_0_lt by auto
      ultimately
      have x<5
        using ‹x<7› ‹x∉?A› ‹n∈nat› in_n_in_nat
        unfolding weak_def
        by (clarsimp simp add:not_lt_iff_le, auto simp add:lt_def)
      then
      show ?thesis unfolding sep_var_def
        by (clarsimp simp add:not_lt_iff_le, auto simp add:lt_def)
    next
      case False
      then
      show ?thesis
      proof (cases n=0)
        case True
        then show ?thesis
          unfolding sep_var_def using ltD asm ‹n∈nat› by auto
      next
        case False
        then
        have n < 2 using  ‹n∈nat› not_lt_iff_le ‹¬ 2 ≤ n›  by force
        then
        have ¬ n <1 using ‹n≠0› by simp
        then
        have n=1 using not_lt_iff_le ‹n<2› le_iff by auto
        then show ?thesis
          using ‹x∉?A›
          unfolding weak_def sep_var_def
          using ltD asm ‹n∈nat› by force
      qed
    qed
  }
  ultimately
  have w∈domain(sep_var(n)) if w∈ 7+ω n - ?A for w
    using that by blast
  then
  have 7+ω n - ?A ⊆ domain(sep_var(n)) by blast
  with F
  show ?thesis by auto
qed

lemma sep_var_type :
  assumes n ∈ nat
  shows sep_var(n) : (7+ω n)-weak(n,5) → 7+ω n
```

using *FiniteFun_is_fun*[*OF sep_var_fin_type*[*OF* ‹*n*∈*nat*›]]
sep_var_domain[*OF* ‹*n*∈*nat*›] **by** *simp*

lemma *sep_var_action* :
assumes
[*t*,*p*,*u*,*P*,*leq*,*o*,*pi*] ∈ *list*(*M*)
env ∈ *list*(*M*)
shows ∀ *i* . *i* ∈ (7+$_\omega$*length*(*env*)) - *weak*(*length*(*env*),5) ⟶
nth(*sep_var*(*length*(*env*))‘*i*,[*t*,*p*,*u*,*P*,*leq*,*o*,*pi*]@*env*) = *nth*(*i*,[*p*,*P*,*leq*,*o*,*t*] @ *env* @ [*pi*,*u*])
using *assms*
proof (*subst sep_var_domain*[*OF length_type*[*OF* ‹*env*∈*list*(*M*)›],*symmetric*],*auto*)
fix *i y*
assume ⟨*i*, *y*⟩ ∈ *sep_var*(*length*(*env*))
with *assms*
show *nth*(*sep_var*(*length*(*env*)) ‘ *i*,
Cons(*t*, *Cons*(*p*, *Cons*(*u*, *Cons*(*P*, *Cons*(*leq*, *Cons*(*o*, *Cons*(*pi*, *env*))))))))
=
nth(*i*, *Cons*(*p*, *Cons*(*P*, *Cons*(*leq*, *Cons*(*o*, *Cons*(*t*, *env* @ [*pi*, *u*]))))))
using *apply_fun*[*OF sep_var_type*] *assms*
unfolding *sep_var_def*
using *nth_concat2*[*OF* ‹*env*∈*list*(*M*)›] *nth_concat3*[*OF* ‹*env*∈*list*(*M*)›,*symmetric*]
by *force*
qed

definition
rensep :: *i* ⇒ *i* **where**
rensep(*n*) ≡ *union_fun*(*sep_var*(*n*),*sep_env*(*n*),7+$_\omega$*n*-*weak*(*n*,5),*weak*(*n*,5))

lemma *rensep_aux* :
assumes *n*∈*nat*
shows (7+$_\omega$*n*-*weak*(*n*,5)) ∪ *weak*(*n*,5) = 7+$_\omega$*n* 7+$_\omega$*n* ∪ (7 +$_\omega$ *n* - 7) = 7+$_\omega$*n*
proof -
from ‹*n*∈*nat*›
have *weak*(*n*,5) = *n*+$_\omega$5-5
using *weak_equal* **by** *simp*
with ‹*n*∈*nat*›
show (7+$_\omega$*n*-*weak*(*n*,5)) ∪ *weak*(*n*,5) = 7+$_\omega$*n* 7+$_\omega$*n* ∪ (7 +$_\omega$ *n* - 7) = 7+$_\omega$*n*
using *Diff_partition le_imp_subset* **by** *auto*
qed

lemma *rensep_type* :
assumes *n*∈*nat*
shows *rensep*(*n*) ∈ 7+$_\omega$*n* → 7+$_\omega$*n*
proof -
from ‹*n*∈*nat*›
have *rensep*(*n*) ∈ (7+$_\omega$*n*-*weak*(*n*,5)) ∪ *weak*(*n*,5) → 7+$_\omega$*n* ∪ (7+$_\omega$*n* - 7)
unfolding *rensep_def*
using *union_fun_type sep_var_type* ‹*n*∈*nat*› *sep_env_type weak_equal*

```
    by force
  then
  show ?thesis using rensep_aux ‹n∈nat› by auto
qed

lemma rensep_action :
  assumes [t,p,u,P,leq,o,pi] @ env ∈ list(M)
 shows ∀ i . i < 7+ω length(env) ⟶ nth(rensep(length(env)) ‘i,[t,p,u,P,leq,o,pi]@env)
= nth(i,[p,P,leq,o,t] @ env @ [pi,u])
proof -
  let ?tgt=[t,p,u,P,leq,o,pi]@env
  let ?src=[p,P,leq,o,t] @ env @ [pi,u]
  let ?m=7 +ω length(env) - weak(length(env),5)
  let ?p=weak(length(env),5)
  let ?f=sep_var(length(env))
  let ?g=sep_env(length(env))
  let ?n=length(env)
  from assms
  have 1 : [t,p,u,P,leq,o,pi] ∈ list(M)  env ∈ list(M)
    ?src ∈ list(M)  ?tgt ∈ list(M)
    7+ω ?n = (7+ω ?n-weak(?n,5)) ∪ weak(?n,5)
     length(?src) = (7+ω ?n-weak(?n,5)) ∪ weak(?n,5)
    using Diff_partition le_imp_subset rensep_aux by auto
  then
 have nth(i, ?src) = nth(union_fun(?f, ?g, ?m, ?p) ‘ i, ?tgt) if i < 7+ω length(env)
for i
  proof -
    from ‹i<7+ω ?n›
    have i ∈ (7+ω ?n-weak(?n,5)) ∪ weak(?n,5)
      using ltD by simp
    then show ?thesis
      unfolding rensep_def using
     union_fun_action[OF ‹?src∈list(M)› ‹?tgt∈list(M)› ‹length(?src) = (7+ω ?n-weak(?n,5))
∪ weak(?n,5)›
          sep_var_action[OF ‹[t,p,u,P,leq,o,pi] ∈ list(M)› ‹env∈list(M)›]
          sep_env_action[OF ‹[t,p,u,P,leq,o,pi] ∈ list(M)› ‹env∈list(M)›]
          ] that
      by simp
  qed
  then show ?thesis unfolding rensep_def by simp
qed

definition sep_ren :: [i,i] ⇒ i where
  sep_ren(n,φ) ≡ ren(φ)‘(7+ω n)‘(7+ω n)‘rensep(n)

lemma arity_rensep: assumes φ∈formula env ∈ list(M)
  arity(φ) ≤ 7+ω length(env)
shows arity(sep_ren(length(env),φ)) ≤ 7+ω length(env)
  unfolding sep_ren_def
```

using *arity_ren rensep_type assms*
by *simp*

lemma *type_rensep* [*TC*]:
assumes φ∈*formula env*∈*list*(*M*)
shows *sep_ren*(*length*(*env*),φ) ∈ *formula*
unfolding *sep_ren_def*
using *ren_tc rensep_type assms*
by *simp*

lemma *sepren_action*:
assumes *arity*(φ) ≤ 7 +$_\omega$ *length*(*env*)
[*t*,*p*,*u*,*P*,*leq*,*o*,*pi*] ∈ *list*(*M*)
env∈*list*(*M*)
φ∈*formula*
shows *sats*(*M*, *sep_ren*(*length*(*env*),φ),[*t*,*p*,*u*,*P*,*leq*,*o*,*pi*] @ *env*) ⟷ *sats*(*M*, φ,[*p*,*P*,*leq*,*o*,*t*] @ *env* @ [*pi*,*u*])
proof -
from *assms*
have *1*: [*t*, *p*, *u*, *P*, *leq*, *o*, *pi*] @ *env* ∈ *list*(*M*)
by *simp_all*
then
have *2*: [*p*,*P*,*leq*,*o*,*t*] @ *env* @ [*pi*,*u*] ∈ *list*(*M*)
using *app_type* **by** *simp*
show *?thesis*
unfolding *sep_ren_def*
using *sats_iff_sats_ren*[*OF* ‹φ∈*formula*›
add_type[*of* 7 *length*(*env*)]
add_type[*of* 7 *length*(*env*)]
2 1
rensep_type[*OF length_type*[*OF* ‹*env*∈*list*(*M*)›]]
‹*arity*(φ) ≤ 7 +$_\omega$ *length*(*env*)›]
rensep_action[*OF 1*,*rule_format*,*symmetric*]
by *simp*
qed

end

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

theory *Separation_Axiom*
imports *Forcing_Theorems Separation_Rename*
begin

context *G_generic1*
begin

lemma *map_val* :
assumes *env*∈*list*($M[G]$)

shows $\exists\, nenv{\in}list(M).\ env = map(val(G),nenv)$
using *assms*
proof(*induct env*)
case *Nil*
have $map(val(G),Nil) = Nil$ **by** *simp*
then show *?case* **by** *force*
next
case (*Cons a l*)
then obtain $a'\ l'$ **where**
$l' \in list(M)\ l{=}map(val(G),l')\ a = val(G,a')$
$Cons(a,l) = map(val(G),Cons(a',l'))\ Cons(a',l') \in list(M)$
using *GenExtD*
by *force*
then show *?case* **by** *force*
qed

lemma *Collect_sats_in_MG* :
assumes
$A{\in}M[G]$
$\varphi \in formula\ env{\in}list(M[G])\ arity(\varphi) \le 1 +_\omega length(env)$
shows
$\{x \in A\ .\ (M[G],\ [x]\ @\ env \models \varphi)\} \in M[G]$
proof -
from ‹$A{\in}M[G]$›
obtain π **where** $\pi \in M\ val(G,\ \pi) = A$
using *GenExt_def* **by** *auto*
then
have $domain(\pi){\in}M\ domain(\pi) \times \mathbb{P} \in M$
using *cartprod_closed*[*of* _ $\mathbb{P}$,*simplified*]
by (*simp_all flip*:*setclass_iff*)
let $?\chi{=}\cdot\cdot\ 0 \in (1 +_\omega length(env)) \cdot \wedge\ \varphi\ \cdot$
let *?new_form*=*sep_ren*(*length*(*env*),*forces*(*?χ*))
let $?\psi{=}(\cdot\exists\,(\cdot\exists\cdot\cdot\langle 0,1\rangle\ is\ 2\ \cdot \wedge\ ?new\_form\ \cdot\ \cdot)\cdot)$
note *phi* = ‹$\varphi{\in}formula$› ‹$arity(\varphi) \le 1 +_\omega length(env)$›
then
have $?\chi{\in}formula\ forces(?\chi) \in formula\ arity(\varphi) \le 2{+_\omega}\ length(env)$
using *definability le_trans*[*OF* ‹$arity(\varphi){\le}\_$›] *add_le_mono*[*of 1 2*,*OF* _ *le_refl*]
by *simp_all*
with ‹$env{\in}\_$› *phi*
have $arity(?\chi) \le 2{+_\omega}length(env)$
using *ord_simp_union leI FOL_arities* **by** *simp*
with ‹$env{\in}list(\_)$› *phi*
have $arity(forces(?\chi)) \le 6 +_\omega length(env)$
using *arity_forces_le* **by** *simp*
then
have $arity(forces(?\chi)) \le 7 +_\omega length(env)$
using *ord_simp_union arity_forces leI* **by** *simp*
with ‹$arity(forces(?\chi)) \le 7 +_\omega \_$› ‹$env \in \_$› ‹$\varphi \in formula$›
have $arity(?new\_form) \le 7 +_\omega length(env)$ *?new_form* $\in$ *formula* $?\psi{\in}formula$

```
    using arity_rensep[OF definability[of ?χ]]
    by auto
  then
  have arity(?ψ) ≤ 5 +ω length(env)
    using ord_simp_union arity_forces pred_mono[OF _ pred_mono[OF _ ‹arity(?new_form) ≤ _›]]
    by (auto simp:arity)
  from ‹env ∈ _›
   obtain nenv where nenv∈list(M) env = map(val(G),nenv) length(nenv) = length(env)
    using map_val by auto
  from phi ‹nenv∈_› ‹env∈_› ‹π∈M› ‹φ∈_› ‹length(nenv) = length(env)›
  have arity(?χ) ≤ length([ϑ] @ nenv @ [π]) for ϑ
    using union_abs2[OF ‹arity(φ) ≤ 2+ω _›] ord_simp_union FOL_arities
    by simp
  note in_M = ‹π∈M› ‹domain(π) × ℙ ∈ M›
  have Equivalence:
      (M, [u,ℙ,leq,1,π] @ nenv ⊨ ?ψ) ⟷
        (∃ϑ∈M. ∃p∈ℙ. u =⟨ϑ,p⟩ ∧
        (∀F. M_generic(F) ∧ p ∈ F ⟶ M[F],  map(val(F), [ϑ] @ nenv @[π]) ⊨ ?χ))
    if u ∈ domain(π) × ℙ
    for u
  proof -
    from ‹u ∈ domain(π) × ℙ› ‹domain(π) × ℙ ∈ M›
    have u∈M by (simp add:transitivity)
    have (M, [ϑ,p,u,ℙ,leq,1,π]@nenv ⊨ ?new_form) ⟷
        (∀F. M_generic(F) ∧ p ∈ F ⟶ (M[F],  map(val(F), [ϑ] @ nenv@[π]) ⊨ ?χ))
      if ϑ∈M p∈ℙ
      for ϑ p
    proof -
      from ‹p∈ℙ›
      have p∈M by (simp add: transitivity)
      let ?env=[p,ℙ,leq,1,ϑ] @ nenv @ [π,u]
      let ?new_env= [ϑ,p,u,ℙ,leq,1,π] @ nenv
     note types = in_M ‹ϑ ∈ M› ‹p∈M› ‹u ∈ domain(π) × ℙ› ‹u ∈ M› ‹nenv∈_›
      then
      have tyenv:?env ∈ list(M) ?new_env ∈ list(M)
        by simp_all
      from types
      have eq_env:[p, ℙ, leq, 1] @ ([ϑ] @ nenv @ [π,u]) =
                ([p, ℙ, leq, 1] @ ([ϑ] @ nenv @ [π])) @ [u]
        using app_assoc by simp
      then
       have (M, [ϑ,p,u,ℙ,leq,1,π] @ nenv ⊨ ?new_form) ⟷ (M, ?new_env ⊨ ?new_form)
        by simp
    from tyenv ‹length(nenv) = length(env)› ‹arity(forces(?χ)) ≤ 7 +ω length(env)›
```

‹*forces*(*?χ*) ∈ *formula*›
have ... ⟷ *p* ⊩ *?χ* ([*ϑ*] @ *nenv* @ [*π*,*u*])
using *sepren_action*[*of forces*(*?χ*) *nenv*,*OF* _ _ ‹*nenv*∈*list*(*M*)›]
by *simp*
also from *types phi* ‹*env*∈_› ‹*length*(*nenv*) = *length*(*env*)› ‹*arity*(*forces*(*?χ*)) ≤ 6 +$_\omega$ *length*(*env*)›
have ... ⟷ *p* ⊩ *?χ* ([*ϑ*] @ *nenv* @ [*π*])
by (*subst eq_env*,*rule_tac arity_sats_iff*,*auto*)
also from *types phi* ‹*p*∈ℙ› ‹*arity*(*forces*(*?χ*)) ≤ 6 +$_\omega$ *length*(*env*)› ‹*arity*(*?χ*) ≤ *length*([*ϑ*] @ *nenv* @ [*π*])›
have ... ⟷ (∀ *F* . *M_generic*(*F*) ∧ *p* ∈ *F* ⟶ *M*[*F*], *map*(*val*(*F*), [*ϑ*] @ *nenv* @ [*π*]) ⊨ *?χ*)
using *definition_of_forcing*[**where** *φ*=·· 0 ∈ (1 +$_\omega$ *length*(*env*)) · ∧ *φ* ·]
by *auto*
finally
show *?thesis*
by *simp*
qed
with *in_M* ‹*?new_form* ∈ *formula*› ‹*?ψ*∈*formula*› ‹*nenv* ∈ _› ‹*u* ∈ *domain*(*π*)×ℙ›
show *?thesis*
by (*auto simp add*: *transitivity*)
qed
moreover from ‹*env* = _› ‹*π*∈*M*› ‹*nenv*∈*list*(*M*)›
have *map_nenv*:*map*(*val*(*G*), *nenv* @ [*π*]) = *env* @ [*val*(*G*,*π*)]
using *map_app_distrib append1_eq_iff* **by** *auto*
ultimately
have *aux*:(∃ *ϑ*∈*M*. ∃ *p*∈ℙ. *u* =⟨*ϑ*,*p*⟩ ∧ (*p*∈*G* ⟶ *M*[*G*], [*val*(*G*,*ϑ*)] @ *env* @ [*val*(*G*,*π*)] ⊨ *?χ*))
(**is** (∃ *ϑ*∈*M*. ∃ *p*∈ℙ. _ (_ ⟶ *M*[*G*] , *?vals*(*ϑ*) ⊨ _)))
if *u* ∈ *domain*(*π*) × ℙ *M*, [*u*,ℙ,*leq*,**1**,*π*] @ *nenv* ⊨ *?ψ* **for** *u*
using *Equivalence*[*THEN iffD1*, *OF that*] *generic* **by** *force*
moreover
have [*val*(*G*, *ϑ*)] @ *env* @ [*val*(*G*, *π*)] ∈ *list*(*M*[*G*]) **if** *ϑ*∈*M* **for** *ϑ*
using ‹*π*∈*M*› ‹*env* ∈ *list*(*M*[*G*])› *GenExtI that* **by** *force*
ultimately
have (∃ *ϑ*∈*M*. ∃ *p*∈ℙ. *u*=⟨*ϑ*,*p*⟩ ∧ (*p*∈*G* ⟶ *val*(*G*,*ϑ*)∈*nth*(1 +$_\omega$ *length*(*env*),[*val*(*G*, *ϑ*)] @ *env* @ [*val*(*G*, *π*)])
∧ (*M*[*G*], *?vals*(*ϑ*) ⊨ *φ*)))
if *u* ∈ *domain*(*π*) × ℙ *M*, [*u*,ℙ,*leq*,**1**,*π*] @ *nenv* ⊨ *?ψ* **for** *u*
using *aux*[*OF that*] **by** *simp*
moreover from ‹*env* ∈ _› ‹*π*∈*M*›
have *nth*:*nth*(1 +$_\omega$ *length*(*env*),[*val*(*G*, *ϑ*)] @ *env* @ [*val*(*G*, *π*)]) = *val*(*G*,*π*)
if *ϑ*∈*M* **for** *ϑ*
using *nth_concat*[*of val*(*G*,*ϑ*) *val*(*G*,*π*) *M*[*G*]] *that GenExtI* **by** *simp*
ultimately
have (∃ *ϑ*∈*M*. ∃ *p*∈ℙ. *u*=⟨*ϑ*,*p*⟩ ∧ (*p*∈*G* ⟶ *val*(*G*,*ϑ*)∈*val*(*G*,*π*) ∧ (*M*[*G*],*?vals*(*ϑ*) ⊨ *φ*)))
if *u* ∈ *domain*(*π*) × ℙ *M*, [*u*,ℙ,*leq*,**1**,*π*] @ *nenv* ⊨ *?ψ* **for** *u*

using *that* ‹$\pi{\in}M$› ‹$env \in \_$› **by** *simp*
with ‹$domain(\pi){\times}\mathbb{P}{\in}M$›
have $\forall\, u{\in}domain(\pi){\times}\mathbb{P}\ .\ (M,\ [u,\mathbb{P},leq,\mathbf{1},\pi]\ @\ nenv \models ?\psi) \longrightarrow (\exists\, \vartheta{\in}M.\ \exists\, p{\in}\mathbb{P}.\ u = \langle\vartheta,p\rangle \wedge (p \in G \longrightarrow val(G,\ \vartheta){\in}val(G,\ \pi) \wedge (M[G],?vals(\vartheta) \models\ \varphi)))$
by (*simp add*:*transitivity*)
then
have $\{u{\in}domain(\pi){\times}\mathbb{P}\ .\ (M,[u,\mathbb{P},leq,\mathbf{1},\pi]\ @\ nenv \models ?\psi)\ \} \subseteq \{u{\in}domain(\pi){\times}\mathbb{P}\ .\ \exists\, \vartheta{\in}M.\ \exists\, p{\in}\mathbb{P}.\ u = \langle\vartheta,p\rangle \wedge (p \in G \longrightarrow val(G,\ \vartheta){\in}val(G,\ \pi) \wedge (M[G],\ ?vals(\vartheta) \models \varphi))\}$
(**is** $?n{\subseteq}?m$)
by *auto*
then
have *first_incl*: $val(G,?n) \subseteq val(G,?m)$
using *val_mono* **by** *simp*
note ‹$val(G,\pi) = A$›
with ‹$?\psi{\in}formula$› ‹$arity(?\psi) \le \_$› *in_M* ‹$nenv \in \_$› ‹$env \in \_$› ‹$length(nenv) = \_$›
have $?n{\in}M$
using *separation_ax leI separation_iff* **by** *auto*
from *generic*
have $filter(G)\ G{\subseteq}\mathbb{P}$
by *auto*
from ‹$val(G,\pi) = A$›
have $val(G,?m) = \{z\ .\ t{\in}domain(\pi)\ ,\ (\exists\, q{\in}\mathbb{P}\ .\ (\exists\, \vartheta{\in}M.\ \exists\, p{\in}\mathbb{P}.\ \langle t,q\rangle = \langle\vartheta,\ p\rangle \wedge (p \in G \longrightarrow val(G,\ \vartheta) \in A \wedge (M[G],\ [val(G,\ \vartheta)]\ @\ env\ @\ [A] \models\ \varphi)) \wedge q \in G)) \wedge z{=}val(G,t)\}$
using *val_of_name* **by** *auto*
also
have $... = \{z\ .\ t{\in}domain(\pi)\ ,\ (\exists\, q{\in}\mathbb{P}.\ val(G,\ t) \in A \wedge (M[G],\ [val(G,\ t)]\ @\ env\ @\ [A] \models\ \varphi) \wedge q \in G) \wedge z{=}val(G,t)\}$
using ‹$domain(\pi){\in}M$› **by** (*auto simp add*:*transitivity*)
also
have $... = \{x{\in}A\ .\ \exists\, q{\in}\mathbb{P}.\ x \in A \wedge (M[G],\ [x]\ @\ env\ @\ [A] \models\ \varphi) \wedge q \in G\}$
proof(*intro equalityI*, *auto*)

{
fix $x\ q$
assume $M[G],\ Cons(x,\ env\ @\ [A]) \models \varphi\ x{\in}A\ q \in \mathbb{P}\ q \in G$
from *this* ‹$val(G,\pi) = A$›
show $x \in \{y\ .\ x \in domain(\pi),\ val(G,\ x) \in A \wedge (M[G],\ Cons(val(G,\ x),\ env\ @\ [A]) \models \varphi) \wedge (\exists\, q{\in}\mathbb{P}.\ q \in G) \wedge y = val(G,\ x)\}$
using *elem_of_val* **by** *force*
}
qed

also
have $\ldots = \{x \in A.\ (M[G],\ [x]\ @\ env\ @\ [A] \models \varphi)\}$
using ‹$G \subseteq \mathbb{P}$› *G_nonempty* **by** *force*
finally
have *val_m*: $val(G,?m) = \{x \in A.\ (M[G],\ [x]\ @\ env\ @\ [A] \models \varphi)\}$ **by** *simp*
have $val(G,?m) \subseteq val(G,?n)$
proof
fix x
assume $x \in val(G,?m)$
with *val_m*
have $x \in \{x \in A.\ (M[G],\ [x]\ @\ env\ @\ [A] \models \varphi)\}$ **by** *simp*
with ‹$val(G,\pi) = A$›
have $x \in val(G,\pi)$ **by** *simp*
then
obtain $\vartheta\ q$ **where** $\langle\vartheta,q\rangle \in \pi\ \ q \in G\ val(G,\vartheta) = x\ \vartheta \in M$
using *elem_of_val_pair domain_trans*[*OF trans_M* ‹$\pi \in \_$›]
by *force*
with ‹$\pi \in M$› ‹$nenv \in \_$› ‹$env = \_$›
have $[val(G,\vartheta),\ val(G,\pi)]\ @\ env \in list(M[G])\ [\vartheta]\ @\ nenv\ @\ [\pi] \in list(M)$
using *GenExt_def* **by** *auto*
with ‹$val(G,\vartheta) = x$› ‹$val(G,\pi) = A$› ‹$x \in val(G,\pi)$› *nth* ‹$\vartheta \in M$› ‹$x \in \{x \in A\ .\ \_\}$›
have $M[G],\ [val(G,\vartheta)]\ @\ env\ @\ [val(G,\pi)] \models \cdot\cdot\ 0 \in (1 +_\omega length(env)) \cdot \wedge\ \varphi\ \cdot$
by *auto*
— Recall $\cdot\cdot 0 \in 1 +_\omega length(env) \cdot \wedge\ \varphi \cdot = \cdot\cdot 0 \in 1 +_\omega length(env) \cdot \wedge\ \varphi \cdot$
with ‹$[\_]\ @\ nenv\ @\ [\_] \in \_$› *map_nenv* ‹$arity(?\chi) \leq length(\_)$› ‹$length(nenv) = \_$›
obtain r **where** $r \in G\ r \Vdash\ ?\chi\ ([\vartheta]\ @\ nenv\ @\ [\pi])$
using *truth_lemma*[*OF* ‹$?\chi \in \_$›,*of* $[\vartheta]\ @\ nenv\ @\ [\pi]$]
by *auto*
with ‹$filter(G)$› **and** ‹$q \in G$›
obtain p **where** $p \in G\ p \preceq q\ p \preceq r$
unfolding *filter_def compat_in_def* **by** *force*
with ‹$r \in G$› ‹$q \in G$› ‹$G \subseteq \mathbb{P}$›
have $p \in \mathbb{P}\ r \in \mathbb{P}\ q \in \mathbb{P}\ p \in M$
using *transitivity*[*OF* _ *P_in_M*] *subsetD*
by *simp_all*
with ‹$\varphi \in formula$› ‹$\vartheta \in M$› ‹$\pi \in M$› ‹$p \preceq r$› ‹$nenv \in \_$› ‹$arity(?\chi) \leq length(\_)$› ‹$r \Vdash\ ?\chi\ \_$› ‹$env \in \_$›
have $p \Vdash\ ?\chi\ ([\vartheta]\ @\ nenv\ @\ [\pi])$
using *strengthening_lemma*
by *simp*
with ‹$p \in \mathbb{P}$› ‹$\varphi \in formula$› ‹$\vartheta \in M$› ‹$\pi \in M$› ‹$nenv \in \_$› ‹$arity(?\chi) \leq length(\_)$›
have $\forall F.\ M\_generic(F) \wedge p \in F \longrightarrow M[F],\quad map(val(F),\ [\vartheta]\ @\ nenv\ @\ [\pi]) \models\ ?\chi$
using *definition_of_forcing*[**where** $\varphi = \cdot\cdot\ 0 \in (1 +_\omega length(env)) \cdot \wedge\ \varphi\ \cdot$]
by *simp*
with ‹$p \in \mathbb{P}$› ‹$\vartheta \in M$›
have *Eq6*: $\exists\,\vartheta' \in M.\ \exists\,p' \in \mathbb{P}.\ \ \langle\vartheta,p\rangle = \langle\vartheta',p'\rangle \wedge (\forall F.\ M\_generic(F) \wedge p' \in F$

```
⟶
                M[F],   map(val(F), ⌜ϑ⌝ @ nenv @ [π]) ⊨  ?χ) by auto
    from ‹π∈M› ‹⟨ϑ,q⟩∈π› ‹ϑ∈M› ‹p∈ℙ› ‹p∈M›
    have ⟨ϑ,q⟩ ∈ M ⟨ϑ,p⟩∈M ⟨ϑ,p⟩∈domain(π)×ℙ
      using pair_in_M_iff transitivity
      by auto
    with ‹ϑ∈M› Eq6 ‹p∈ℙ›
    have M, [⟨ϑ,p⟩,ℙ,leq,1,π] @ nenv ⊨ ?ψ
      using Equivalence by auto
    with ‹⟨ϑ,p⟩∈domain(π)×ℙ›
    have ⟨ϑ,p⟩∈?n by simp
    with ‹p∈G› ‹p∈ℙ›
    have val(G,ϑ)∈val(G,?n)
      using val_of_elem[of ϑ p] by simp
    with ‹val(G,ϑ)=x›
    show x∈val(G,?n) by simp
  qed
  with val_m first_incl
  have val(G,?n) = {x ∈ A. (M[G], [x] @ env @ [A] ⊨ φ)} by auto
  also from ‹A∈_› phi ‹env ∈ _›
  have  ... = {x ∈ A. (M[G], [x] @ env ⊨ φ)}
    using arity_sats_iff[where env=[_]@env] transitivity_MG
    by auto
  finally
  show {x ∈ A. (M[G], [x] @ env ⊨ φ)}∈ M[G]
    using ‹?n∈M› GenExt_def by force
qed

theorem separation_in_MG:
  assumes
    φ∈formula and arity(φ) ≤ 1 +ω length(env) and env∈list(M[G])
  shows
    separation(##M[G],λx. (M[G], [x] @ env ⊨ φ))
proof -
  {
    fix A
    assume A∈M[G]
    moreover from ‹env ∈ _›
      obtain nenv where nenv∈list(M)env = map(val(G),nenv) length(env) =
length(nenv)
      using GenExt_def map_val[of env] by auto
    moreover note ‹φ ∈ _› ‹arity(φ) ≤ _› ‹env ∈ _›
    ultimately
    have {x ∈ A . (M[G], [x] @ env ⊨ φ)} ∈ M[G]
      using Collect_sats_in_MG by auto
  }
  then
  show ?thesis
    using separation_iff rev_bexI unfolding is_Collect_def by force
```

qed

end — *G_generic1*

end

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

theory *Pairing_Axiom*
  imports
    Names
begin

context *G_generic1*
begin

lemma *val_Upair* :
  $\mathbf{1} \in G \Longrightarrow val(G,\{\langle\tau,\mathbf{1}\rangle,\langle\varrho,\mathbf{1}\rangle\}) = \{val(G,\tau),val(G,\varrho)\}$
  by (*rule trans*, *subst def_val*,*auto*)

lemma *pairing_in_MG* : *upair_ax*(##$M[G]$)
proof -
  {
    fix x y
    assume $x \in M[G]$ $y \in M[G]$
    moreover from *this*
    obtain τ ϱ **where** $val(G,\tau) = x$ $val(G,\varrho) = y$ $\varrho \in M$ $\tau \in M$
      using *GenExtD* **by** *blast*
    moreover from *this*
    have $\langle\tau,\mathbf{1}\rangle \in M$ $\langle\varrho,\mathbf{1}\rangle \in M$
      using *pair_in_M_iff* **by** *auto*
    moreover from *this*
    have $\{\langle\tau,\mathbf{1}\rangle,\langle\varrho,\mathbf{1}\rangle\} \in M$ (**is** $?\sigma \in \_$)
      using *upair_in_M_iff* **by** *simp*
    moreover from *this*
    have $val(G,?\sigma) \in M[G]$
      using *GenExtI* **by** *simp*
    moreover from *calculation*
    have $\{val(G,\tau),val(G,\varrho)\} \in M[G]$
      using *val_Upair one_in_G* **by** *simp*
    ultimately
    have $\{x,y\} \in M[G]$
      by *simp*
  }
  then
  show *?thesis*
    unfolding *upair_ax_def upair_def* **by** *auto*
qed

end — *G_generic1*

end

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

theory *Union_Axiom*
  imports *Names*
begin

definition *Union_name_body* :: $[i,i,i,i] \Rightarrow o$ **where**
  $Union\_name\_body(P,leq,\tau,x) \equiv \exists\ \sigma{\in}domain(\tau)\ .\ \exists\ q{\in}P\ .\ \exists\ r{\in}P\ .$
    $\langle\sigma,q\rangle \in \tau \wedge \langle fst(x),r\rangle \in \sigma \wedge \langle snd(x),r\rangle \in leq \wedge \langle snd(x),q\rangle \in leq$

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

context *forcing_data1*
begin

lemma *Union_name_closed* :
  assumes $\tau \in M$
  shows $Union\_name(\mathbb{P},leq,\tau) \in M$
proof -
  let *?Q*=*Union_name_body*($\mathbb{P}$,*leq*,τ)
 note *lr_fst2* = *lam_replacement_hcomp*[*OF lam_replacement_fst lam_replacement_fst*]
  and *lr_fst3* = *lam_replacement_hcomp*[*OF lr_fst2*] *lam_replacement_hcomp*[*OF lr_fst2 lr_fst2*]
  note ‹$\tau{\in}M$›
  moreover from *this*
  have $domain(\bigcup(domain(\tau))){\in}M$ (**is** *?d* ∈ _)
    using *domain_closed Union_closed* **by** *simp*
  moreover from *this*
  have *?d* × $\mathbb{P}$ ∈ *M*
    using *cartprod_closed* **by** *simp*
  note *types* = *assms* ‹*?d*×$\mathbb{P}$ ∈ *M*› ‹*?d*∈*M*›
  ultimately
  show *?thesis*
    using *domain_closed pair_in_M_iff fst_closed snd_closed separation_closed*
      lam_replacement_constant lam_replacement_hcomp
      lam_replacement_fst lam_replacement_snd lam_replacement_product
      separation_bex separation_conj separation_in lr_fst2 lr_fst3
      lam_replacement_hcomp[*OF lr_fst3*(*1*) *lam_replacement_snd*]
    unfolding *Union_name_body_def Union_name_def*
    by *simp*
qed

lemma *Union_MG_Eq* :
  assumes $a \in M[G]$ **and** $a = val(G,\tau)$ **and** *filter*(G) **and** $\tau \in M$

shows $\bigcup\ a = val(G,Union\_name(\mathbb{P},leq,\tau))$
proof (*intro equalityI subsetI*)
fix x
assume $x \in \bigcup\ a$
with ‹a=_›
have $x \in \bigcup\ (val(G,\tau))$
by *simp*
then
obtain i **where** $i \in val(G,\tau)\ x \in i$
by *blast*
with ‹$\tau \in M$›
obtain $\sigma\ q$ **where** $q \in G\ \langle\sigma,q\rangle \in \tau\ val(G,\sigma) = i\ \sigma \in M$
using *elem_of_val_pair domain_trans*[*OF trans_M*] **by** *blast*
moreover from *this* ‹$x \in i$›
obtain $\vartheta\ r$ **where** $r \in G\ \langle\vartheta,r\rangle \in \sigma\ val(G,\vartheta) = x\ \vartheta \in M$
using *elem_of_val_pair domain_trans*[*OF trans_M*] **by** *blast*
moreover from *calculation*
have $\vartheta \in domain(\bigcup(domain(\tau)))$
by *auto*
moreover from *calculation* ‹*filter*(G)›
obtain p **where** $p \in G\ \langle p,r\rangle \in leq\ \langle p,q\rangle \in leq\ p \in \mathbb{P}\ r \in \mathbb{P}\ q \in \mathbb{P}$
using *low_bound_filter filterD* **by** *blast*
moreover from *this*
have $p \in M\ q{\in}M\ r{\in}M$
by (*auto dest*:*transitivity*)
moreover from *calculation*
have $\langle\vartheta,p\rangle \in Union\_name(\mathbb{P},leq,\tau)$
unfolding *Union_name_def Union_name_body_def*
by *auto*
moreover from *this* ‹$p{\in}\mathbb{P}$› ‹$p{\in}G$›
have $val(G,\vartheta) \in val(G,Union\_name(\mathbb{P},leq,\tau))$
using *val_of_elem* **by** *simp*
ultimately
show $x \in val(G,Union\_name(\mathbb{P},leq,\tau))$
by *simp*
next
fix x
assume $x \in (val(G,Union\_name(\mathbb{P},leq,\tau)))$
moreover
note ‹*filter*(G)› ‹a=$val(G,\tau)$›
moreover from *calculation*
obtain $\vartheta\ p$ **where** $p \in G\ \langle\vartheta,p\rangle \in Union\_name(\mathbb{P},leq,\tau)\ val(G,\vartheta) = x$
using *elem_of_val_pair* **by** *blast*
moreover from *calculation*
have $p{\in}\mathbb{P}$
using *filterD* **by** *simp*
moreover from *calculation*
obtain $\sigma\ q\ r$ **where** $\langle\sigma,q\rangle \in \tau\ \langle\vartheta,r\rangle \in \sigma\ \langle p,r\rangle \in leq\ \langle p,q\rangle \in leq\ r{\in}\mathbb{P}\ q{\in}\mathbb{P}$
unfolding *Union_name_def Union_name_body_def*

by *auto*
moreover from *calculation*
have $r \in G\ q \in G$
using *filter_leqD* **by** *auto*
moreover from *this* ‹$\langle\vartheta,r\rangle \in \sigma$› ‹$\langle\sigma,q\rangle\in\tau$› ‹$q\in\mathbb{P}$› ‹$r\in\mathbb{P}$›
have $val(G,\sigma) \in val(G,\tau)\ val(G,\vartheta) \in val(G,\sigma)$
using *val_of_elem* **by** *simp+*
ultimately
show $x \in \bigcup a$
by *blast*
qed

lemma *union_in_MG* :
assumes *filter*(G)
shows *Union_ax*$(\#\#M[G])$
unfolding *Union_ax_def*
proof(*clarsimp*)
fix a
assume $a \in M[G]$
moreover
note ‹*filter*(G)›
moreover from *calculation*
interpret *mgtrans* : *M_trans* $\#\#M[G]$
using *transitivity_MG* **by** (*unfold_locales*; *auto*)
from *calculation*
obtain τ **where** $\tau \in M\ a{=}val(G,\tau)$
using *GenExtD* **by** *blast*
moreover from *this*
have $val(G,Union\_name(\mathbb{P},leq,\tau)) \in M[G]$
using *GenExtI Union_name_closed* **by** *simp*
ultimately
show $\exists\, z{\in}M[G]\ .\ big\_union(\#\#M[G],a,z)$
using *Union_MG_Eq* **by** *auto*
qed

theorem *Union_MG* : *M_generic*$(G) \Longrightarrow$ *Union_ax*$(\#\#M[G])$
by (*auto simp*:*union_in_MG*)

end — *forcing_data1*

end

19 The Powerset Axiom in $M[G]$

theory *Powerset_Axiom*
imports
Separation_Axiom Pairing_Axiom Union_Axiom
begin

simple_rename *perm_pow* **src** [*ss*,*p*,*l*,*o*,*fs*,χ] **tgt** [*fs*,*ss*,*sp*,*p*,*l*,*o*,χ]

context *G_generic1*
begin

lemma *sats_fst_snd_in_M*:
  assumes
    $A{\in}M$ $B{\in}M$ $\varphi \in$ *formula* $p{\in}M$ $l{\in}M$ $o{\in}M$ $\chi{\in}M$ *arity*(φ) $\leq$ 6
  shows $\{\langle s,q\rangle{\in}A{\times}B$. M, [*q*,*p*,*l*,*o*,*s*,χ] $\models \varphi\} \in M$ (**is** *?ϑ* $\in M$)
proof -
  let *?φ′* = *ren*(φ)‘6‘7‘*perm_pow_fn*
  from ‹$A{\in}M$› ‹$B{\in}M$›
  have $A{\times}B \in M$
    using *cartprod_closed* **by** *simp*
  from ‹*arity*(φ) $\leq$ 6› ‹$\varphi{\in}$ *formula*›
  have *?φ′* $\in$ *formula* *arity*(*?φ′*)$\leq$7
    unfolding *perm_pow_fn_def*
    using *perm_pow_thm* *arity_ren* *ren_tc* *Nil_type*
    by *auto*
  with ‹*?φ′* $\in$ *formula*›
  have *arty*: *arity*(*Exists*(*Exists*(*And*(*pair_fm*(0,1,2),*?φ′*))))$\leq$5 (**is** *arity*(*?ψ*)$\leq$5)
    using *ord_simp_union* *pred_le*
    by (*auto* *simp*:*arity*)
  {
    fix *sp*
    note ‹$A{\times}B \in M$› ‹$A{\in}M$› ‹$B{\in}M$›
    moreover
    assume *sp* $\in A{\times}B$
    moreover from *calculation*
    have *fst*(*sp*) $\in A$ *snd*(*sp*) $\in B$
      using *fst_type* *snd_type* **by** *simp_all*
    ultimately
    have *sp* $\in M$ *fst*(*sp*) $\in M$ *snd*(*sp*) $\in M$
      using *transitivity*
      by *simp_all*
    note *inM* = ‹$A{\in}M$› ‹$B{\in}M$› ‹$p{\in}M$› ‹$l{\in}M$› ‹$o{\in}M$› ‹$\chi{\in}M$›
      ‹*sp*$\in M$› ‹*fst*(*sp*)$\in M$› ‹*snd*(*sp*)$\in M$›
    with *arty* ‹*sp* $\in M$› ‹*?φ′* $\in$ *formula*›
   have (M, [*sp*,*p*,*l*,*o*,χ]@[*p*] $\models$ *?ψ*) $\longleftrightarrow$ M,[*sp*,*p*,*l*,*o*,χ] $\models$ *?ψ* (**is** (M,*?env0*@ _$\models$_)
$\longleftrightarrow$ _)
      using *arity_sats_iff*[*of* *?ψ* [*p*] M *?env0*] **by** *auto*
    also from *inM* ‹*sp* $\in A{\times}B$›
    have ... $\longleftrightarrow$ *sats*(M,*?φ′*,[*fst*(*sp*),*snd*(*sp*),*sp*,*p*,*l*,*o*,χ])
      by *auto*
    also from *inM* ‹$\varphi \in$ *formula*› ‹*arity*(φ) $\leq$ 6›
    have ... $\longleftrightarrow$ M, [*snd*(*sp*),*p*,*l*,*o*,*fst*(*sp*),χ] $\models \varphi$
      (**is** *sats*(_,_,*?env1*) $\longleftrightarrow$ *sats*(_,_,*?env2*))
    using *sats_iff_sats_ren*[*of* φ 6 7 *?env2* M *?env1* *perm_pow_fn*] *perm_pow_thm*
      unfolding *perm_pow_fn_def* **by** *simp*

```
    finally
    have (M,[sp,p,l,o,χ,p] ⊨ ?ψ) ⟷ M, [snd(sp),p,l,o,fst(sp),χ] ⊨ φ
      by simp
  }
  then
  have ?ϑ = {sp∈A×B . sats(M,?ψ,[sp,p,l,o,χ,p])}
    by auto
  with assms ‹A×B∈M›
  show ?thesis
    using separation_ax separation_iff arty leI ‹?φ' ∈ formula›
    by simp
qed

declare nat_into_M[rule del, simplified setclass_iff, intro]
lemmas ssimps = domain_closed cartprod_closed cons_closed Pow_rel_closed
declare ssimps [simp del, simplified setclass_iff, simp, intro]

— We keep Pow(a) ∩ M[G] to be consistent with Kunen.
lemma Pow_inter_MG:
  assumes a∈M[G]
  shows Pow(a) ∩ M[G] ∈ M[G]
proof -
  from assms
  obtain τ where τ ∈ M val(G, τ) = a
    using GenExtD by auto
  let ?Q=Pow^M(domain(τ)×ℙ)
  let ?π=?Q×{𝟭}
  let ?b=val(G,?π)
  from ‹τ∈M›
  have domain(τ)×ℙ ∈ M domain(τ) ∈ M
    by simp_all
  then
  have ?b ∈ M[G]
    by (auto intro!:GenExtI)
  have Pow(a) ∩ M[G] ⊆ ?b
  proof
    fix c
    assume c ∈ Pow(a) ∩ M[G]
    then
    obtain χ where c∈M[G] χ ∈ M val(G,χ) = c
      using GenExt_iff by auto
    let ?ϑ={⟨σ,p⟩ ∈domain(τ)×ℙ . p ⊩ ·0 ∈ 1· [σ,χ] }
    have arity(forces( ·0 ∈ 1· )) = 6
      using arity_forces_at by auto
    with ‹domain(τ) ∈ M› ‹χ ∈ M›
    have ?ϑ ∈ M
      using sats_fst_snd_in_M
      by simp
    with ‹domain(τ)×ℙ ∈ M›
```

have *?ϑ ∈ ?Q*
using *Pow_rel_char* **by** *auto*
have *val*(*G*,*?ϑ*) = *c*
proof(*intro equalityI subsetI*)
fix *x*
assume *x* ∈ *val*(*G*,*?ϑ*)
then
obtain *σ p* **where** *1*: ⟨*σ*,*p*⟩∈*?ϑ p*∈*G val*(*G*,*σ*) = *x*
using *elem_of_val_pair*
by *blast*
moreover from ‹⟨*σ*,*p*⟩∈*?ϑ*› ‹*?ϑ* ∈ *M*›
have *σ*∈*M*
using *name_components_in_M*[*of _ _ ?ϑ*] **by** *auto*
moreover from *1*
have *p* ⊩ ⋅*0* ∈ *1*⋅ [*σ*,*χ*] *p*∈ℙ
by *simp_all*
moreover
note ‹*val*(*G*,*χ*) = *c*› ‹*χ* ∈ *M*›
ultimately
have *M*[*G*], [*x*, *c*] ⊨ ⋅*0* ∈ *1*⋅
using *generic definition_of_forcing*[**where** *φ*=⋅*0* ∈ *1*⋅] *ord_simp_union*
by *auto*
moreover from ‹*σ*∈*M*› ‹*χ*∈*M*›
have *x*∈*M*[*G*]
using ‹*val*(*G*,*σ*) = *x*› *GenExtI* **by** *blast*
ultimately
show *x*∈*c*
using ‹*c*∈*M*[*G*]› **by** *simp*
next
fix *x*
assume *x* ∈ *c*
with ‹*c* ∈ *Pow*(*a*) ∩ *M*[*G*]›
have *x* ∈ *a c*∈*M*[*G*] *x*∈*M*[*G*]
using *transitivity_MG* **by** *auto*
with ‹*val*(*G*, *τ*) = *a*›
obtain *σ* **where** *σ*∈*domain*(*τ*) *val*(*G*,*σ*) = *x*
using *elem_of_val* **by** *blast*
moreover
note ‹*x*∈*c*› ‹*val*(*G*,*χ*) = *c*› ‹*c*∈*M*[*G*]› ‹*x*∈*M*[*G*]›
moreover from *calculation*
have *val*(*G*,*σ*) ∈ *val*(*G*,*χ*)
by *simp*
moreover from *calculation*
have *M*[*G*], [*x*, *c*] ⊨ ⋅*0* ∈ *1*⋅
by *simp*
moreover
have *σ*∈*M*
proof -
from ‹*σ*∈*domain*(*τ*)›

```
      obtain p where ⟨σ,p⟩ ∈ τ
        by auto
      with ‹τ∈M›
      show ?thesis
        using name_components_in_M by blast
    qed
    moreover
    note ‹χ ∈ M›
    ultimately
    obtain p where p∈G p ⊩ ·0 ∈ 1· [σ,χ]
      using generic truth_lemma[of ·0 ∈ 1· [σ,χ] ] ord_simp_union
      by auto
    moreover from ‹p∈G›
    have p∈ℙ
      using generic by blast
    ultimately
    have ⟨σ,p⟩∈?ϑ
      using ‹σ∈domain(τ)› by simp
    with ‹val(G,σ) =  x› ‹p∈G›
    show x∈val(G,?ϑ)
      using val_of_elem [of _ _ ?ϑ G] by auto
  qed
  with ‹?ϑ ∈ ?Q›
  show c∈?b
    using one_in_G generic val_of_elem [of ?ϑ 1 ?π G]
    by auto
qed
then
have Pow(a) ∩ M[G] = {x∈?b . x⊆a ∧ x∈M[G]}
  by auto
also from ‹a∈M[G]›
have  ... = {x∈?b . ( M[G], [x,a] ⊨ ·0 ⊆ 1· )} ∩ M[G]
  using Transset_MG by force
also from ‹?b∈M[G]›
have  ... = {x∈?b . ( M[G], [x,a] ⊨ ·0 ⊆ 1· )}
  by (intro equalityI) (auto dest:ext.transM)
also from ‹?b∈M[G]› ‹a∈M[G]›
have  ... ∈ M[G]
  using Collect_sats_in_MG GenExtI ord_simp_union by (simp add:arity)
finally
show ?thesis .
qed

end — G_generic1

sublocale G_generic1 ⊆ ext: M_trivial ##M[G]
  using generic Union_MG pairing_in_MG
  by unfold_locales (simp; blast)
```

context *G_generic1* **begin**

theorem *power_in_MG* : *power_ax*(##(*M*[*G*]))
unfolding *power_ax_def*
proof (*intro rallI*, *simp only*:*setclass_iff rex_setclass_is_bex*)
fix *a*

After simplification, we have to show that for every $a \in M[G]$ there exists some $x \in M[G]$ satisfying *powerset*(##*M*[*G*], *a*, *x*)

assume $a \in M[G]$
have $\{x \in Pow(a) \,.\, x \in M[G]\} = Pow(a) \cap M[G]$
by *auto*
also from ‹$a \in M[G]$›
have $\ldots \in M[G]$
using *Pow_inter_MG* **by** *simp*
finally
have $\{x \in Pow(a) \,.\, x \in M[G]\} \in M[G]$ **.**
moreover from ‹$a \in M[G]$› *this*
have *powerset*(##*M*[*G*], *a*, $\{x \in Pow(a) \,.\, x \in M[G]\}$)
using *ext.powerset_abs*
by *simp*
ultimately
show $\exists\, x \in M[G] \,.\, powerset(\#\#M[G],\, a,\, x)$
by *auto*
qed

end — *G_generic1*

end

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

theory *Extensionality_Axiom*
imports
Names
begin

context *forcing_data1*
begin

lemma *extensionality_in_MG* : *extensionality*(##(*M*[*G*]))
unfolding *extensionality_def*
proof(*clarsimp*)
fix *x y*
assume $x \in M[G]$ $y \in M[G]$ $(\forall\, w \in M[G] \,.\, w \in x \longleftrightarrow w \in y)$
moreover from *this*
have $z \in x \longleftrightarrow z \in M[G] \wedge z \in y$ **for** *z*
using *transitivity_MG* **by** *auto*
moreover from *calculation*

```
  have z∈M[G] ∧ z∈x ⟷ z∈y for z
    using transitivity_MG by auto
  ultimately
  show x=y
    by auto
qed

end — forcing_data1

end
```

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

```
theory Foundation_Axiom
  imports
    Names
begin

context forcing_data1
begin


lemma foundation_in_MG : foundation_ax(##(M[G]))
  unfolding foundation_ax_def
  by (rule rallI, cut_tac A=x in foundation, auto intro: transitivity_MG)


lemma foundation_ax(##(M[G]))
proof -
  {
    fix x
    assume x∈M[G] ∃ y∈M[G] . y∈x
    then
    have ∃ y∈M[G] . y∈x∩M[G]
      by simp
    then
    obtain y where y∈x∩M[G] ∀ z∈y. z ∉ x∩M[G]
      using foundation[of x∩M[G]]  by blast
    then
    have ∃ y∈M[G] . y ∈ x ∧ (∀ z∈M[G] . z ∉ x ∨ z ∉ y)
      by auto
  }
  then
  show ?thesis
    unfolding foundation_ax_def by auto
qed

end — forcing_data1
```

end

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

theory *Replacement_Axiom*
imports
Separation_Axiom
begin

context *forcing_data1*
begin

bundle *sharp_simps1* = *snd_abs*[*simp*] *fst_abs*[*simp*] *fst_closed*[*simp del*, *simplified*, *simp*]
snd_closed[*simp del*, *simplified*, *simp*] *M_inhabited*[*simplified*, *simp*]
pair_in_M_iff[*simp del*, *simplified*, *simp*]

lemma *sats_body_ground_repl_fm*:
includes *sharp_simps1*
assumes
$\exists\, t\ p.\ x{=}\langle t{,}p\rangle\ [x{,}\alpha{,}m{,}\mathbb{P}{,}leq{,}\mathbf{1}]\ @\ nenv \in list(M)$
$\varphi{\in}formula$
shows
$(\exists\,\tau{\in}M.\ \exists\, V{\in}M.\ is\_Vset(\lambda a.\ (\#\#M)(a){,}\alpha{,}V)\ \wedge\ \tau\ \in\ V\ \wedge\ (snd(x)\ \Vdash\ \varphi([fst(x){,}\tau]@nenv)))$
$\longleftrightarrow M,\ [\alpha,\ x,\ m,\ \mathbb{P},\ leq,\ \mathbf{1}]\ @\ nenv \models body\_ground\_repl\_fm(\varphi)$
unfolding *body_ground_repl_fm_def rename_split_fm_def*
by ((*insert assms*,*rule iff_sats* | *simp add*:*nonempty*[*simplified*])+,
insert sats_incr_bv_iff[**where** *bvs*=[_,_,_,_,_,_], *simplified*],*auto del*: *iffI*)

end — *forcing_data1*

context *G_generic1*
begin

lemma *Replace_sats_in_MG*:
assumes
$c{\in}M[G]\ env \in list(M[G])$
$\varphi \in formula\ arity(\varphi) \le 2 +_\omega length(env)$
$univalent(\#\#M[G],\ c,\ \lambda x\ v.\ (M[G]\ ,\ [x{,}v]@env \models \varphi)\ )$
and
ground_replacement:
$\bigwedge nenv.\ ground\_replacement\_assm(M{,}[\mathbb{P}{,}leq{,}\mathbf{1}]\ @\ nenv,\ \varphi)$
shows
$\{v.\ x{\in}c,\ v{\in}M[G] \wedge (M[G]\ ,\ [x{,}v]@env \models \varphi)\} \in M[G]$
proof -
let *?R* = $\lambda\ x\ v\ .\ v{\in}M[G] \wedge (M[G]\ ,\ [x{,}v]@env \models \varphi)$
from ‹$c{\in}M[G]$›
obtain π' **where** $val(G,\ \pi') = c\ \pi' \in M$

```
    using GenExt_def by auto
  then
  have domain(π')×ℙ∈M (is ?π∈M)
    using cartprod_closed domain_closed by simp
  from ‹val(G, π') = c›
  have c ⊆ val(G,?π)
    using def_val[of G ?π] elem_of_val[of _ G π'] one_in_G
      domain_of_prod[OF one_in_P, of domain(π')]
    by (force del:M_genericD)
  from ‹env ∈ _›
  obtain nenv where nenv∈list(M) env = map(val(G),nenv)
    using map_val by auto
  then
  have length(nenv) = length(env) by simp
  with ‹arity(φ) ≤ _›
  have arity(φ) ≤ 2 +ω length(nenv) by simp
  define f where f(ϱp) ≡ μ α. α∈M ∧ (∃τ∈M. τ ∈ Vset(α) ∧
       (snd(ϱp) ⊩ φ ([fst(ϱp),τ] @ nenv))) (is _ ≡ μ α. ?P(ϱp,α)) for ϱp
  have f(ϱp) = (μ α. α∈M ∧ (∃τ∈M. ∃ V∈M. is_Vset(##M,α,V) ∧ τ∈V ∧
       (snd(ϱp) ⊩ φ ([fst(ϱp),τ] @ nenv)))) (is _ = (μ α. α∈M ∧ ?Q(ϱp,α))) for
ϱp
    unfolding f_def using Vset_abs Vset_closed Ord_Least_cong[of ?P(ϱp) λ α.
α∈M ∧ ?Q(ϱp,α)]
    by (simp, simp del:setclass_iff)
  moreover
  note inM = ‹nenv∈list(M)› ‹?π∈M›
  moreover
  have f(ϱp) ∈ M Ord(f(ϱp)) for ϱp
    unfolding f_def using Least_closed'[of ?P(ϱp)] by simp_all
  ultimately
  have 1:least(##M,λα. ?Q(ϱp,α),f(ϱp)) for ϱp
    using least_abs'[of λα. α∈M ∧ ?Q(ϱp,α) f(ϱp)] least_conj
    by (simp flip: setclass_iff)
  define QQ where QQ≡?Q
  from 1
  have least(##M,λα. QQ(ϱp,α),f(ϱp)) for ϱp
    unfolding QQ_def .
  have body:(M, [ϱp,m,ℙ,leq,1] @ nenv ⊨ ground_repl_fm(φ)) ⟷ least(##M,
QQ(ϱp), m)
    if ϱp∈M ϱp∈?π m∈M for ϱp m
  proof -
    note inM that
    moreover from this assms 1
    have (M , [α,ϱp,m,ℙ,leq,1] @ nenv ⊨ body_ground_repl_fm(φ)) ⟷ ?Q(ϱp,α)
if α∈M for α
      using that sats_body_ground_repl_fm[of ϱp α m nenv φ]
      by auto
    moreover from calculation
    have body:⋀α. α ∈ M ⟹ (∃τ∈M. ∃ V∈M. is_Vset(λa. a∈M, α, V) ∧ τ ∈
```

$V \wedge$
(*snd*(ϱp) $\Vdash$ φ ([*fst*(ϱp),τ] @ *nenv*))) $\longleftrightarrow$
M, *Cons*(α, [ϱp, *m*, $\mathbb{P}$, *leq*, **1**] @ *nenv*) $\models$ *body_ground_repl_fm*(φ)
by *simp*
ultimately
show (*M* , [ϱp,*m*,$\mathbb{P}$,*leq*,**1**] @ *nenv* $\models$ *ground_repl_fm*(φ)) $\longleftrightarrow$ *least*(##*M*, *QQ*(ϱp), *m*)
using *sats_least_fm*[*OF body*,*of 1*] **unfolding** *QQ_def ground_repl_fm_def*
by (*simp*, *simp flip*: *setclass_iff*)
qed
then
have *univalent*(##*M*, *?π*, $\lambda \varrho p\ m.\ M$, [ϱp,*m*] @ ([$\mathbb{P}$,*leq*,**1**] @ *nenv*) $\models$ *ground_repl_fm*(φ))
unfolding *univalent_def* **by** (*auto intro*:*unique_least*)
moreover from ‹*length*(_) = _› ‹*env* ∈ _›
have *length*([$\mathbb{P}$,*leq*,**1**] @ *nenv*) = *3* +$_\omega$ *length*(*env*) **by** *simp*
moreover from ‹*arity*(φ) ≤ *2* +$_\omega$ *length*(*nenv*)›
‹*length*(_) = *length*(_)›[*symmetric*] ‹*nenv*∈_› ‹φ∈_›
have *arity*(*ground_repl_fm*(φ)) ≤ *5* +$_\omega$ *length*(*env*)
using *arity_ground_repl_fm*[*of* φ] *le_trans Un_le* **by** *auto*
moreover from ‹φ∈*formula*›
have *ground_repl_fm*(φ)∈*formula* **by** *simp*
moreover
note ‹*length*(*nenv*) = *length*(*env*)› *inM*
ultimately
obtain *Y* **where** *Y*∈*M*
∀ *m*∈*M*. *m* ∈ *Y* $\longleftrightarrow$ (∃ ϱp∈*M*. ϱp ∈ *?π* ∧ (*M*, [ϱp,*m*] @ ([$\mathbb{P}$,*leq*,**1**] @ *nenv*) $\models$ *ground_repl_fm*(φ)))
using *ground_replacement*[*of nenv*]
unfolding *strong_replacement_def ground_replacement_assm_def replacement_assm_def* **by** *auto*
with ‹*least*(_,*QQ*(_),*f*(_))› ‹*f*(_) ∈ *M*› ‹*?π*∈*M*› *body*
have *f*(ϱp)∈*Y* **if** ϱp∈*?π* **for** ϱp
using *that transitivity*[*OF* _ ‹*?π*∈*M*›]
by (*clarsimp*,*rename_tac* ϱ *p* ϱp, *rule_tac x*=⟨ϱ,*p*⟩ **in** *bexI*, *auto*)
from ‹*Y*∈*M*›
have $\bigcup$ {*y*∈*Y*. *Ord*(*y*)} ∈ *M* (**is** *?sup* ∈ *M*)
using *separation_Ord separation_closed Union_closed* **by** *simp*
then
have {*x*∈*Vset*(*?sup*). *x* ∈ *M*} × {**1**} ∈ *M* (**is** *?big_name* ∈ *M*)
using *Vset_closed cartprod_closed singleton_closed* **by** *simp*
then
have *val*(*G*,*?big_name*) ∈ *M*[*G*]
by (*blast intro*:*GenExtI*)
have {*v*. *x*∈*c*, *?R*(*x*,*v*)} ⊆ *val*(*G*,*?big_name*) (**is** *?repl*⊆*?big*)
proof(*intro subsetI*)
fix *v*
assume *v*∈*?repl*
moreover from *this*
obtain *x* **where** *x*∈*c* *M*[*G*], [*x*, *v*] @ *env* $\models$ φ *v*∈*M*[*G*]

by *auto*
moreover
note ‹$val(G,\pi')=c$› ‹$\pi'\in M$›
moreover
from *calculation*
obtain ϱ p **where** $\langle\varrho,p\rangle\in\pi'$ $val(G,\varrho) = x$ $p\in G$ $\varrho\in M$
using *elem_of_val_pair'* **by** *blast*
moreover from *this* ‹$v\in M[G]$›
obtain σ **where** $val(G,\sigma) = v$ $\sigma\in M$
using *GenExtD* **by** (*force del*:*M_genericD*)
moreover
note ‹$\varphi\in\_$› ‹$nenv\in\_$› ‹$env = \_$› ‹$arity(\varphi)\leq 2 +_\omega length(env)$›
ultimately
obtain q **where** $q\in G$ $q \Vdash \varphi$ $([\varrho,\sigma]@nenv)$ $q\in\mathbb{P}$
using *truth_lemma*[*OF* ‹$\varphi\in\_$›,*of* $[\varrho,\sigma]$ @ *nenv*]
by *auto*
with ‹$\langle\varrho,p\rangle\in\pi'$› ‹$\langle\varrho,q\rangle\in ?\pi \Longrightarrow f(\langle\varrho,q\rangle)\in Y$›
have $f(\langle\varrho,q\rangle)\in Y$
using *generic* **by** *blast*
let $?\alpha=succ(rank(\sigma))$
note ‹$\sigma\in M$›
moreover from *this*
have $?\alpha \in M$ $\sigma \in Vset(?\alpha)$
using *rank_closed cons_closed Vset_Ord_rank_iff*
by (*simp_all flip*: *setclass_iff*)
moreover
note ‹$q \Vdash \varphi$ $([\varrho,\sigma]$ @ *nenv*)›
ultimately
have $?P(\langle\varrho,q\rangle,?\alpha)$ **by** (*auto simp del*: *Vset_rank_iff*)
moreover
have $(\mu\ \alpha.\ ?P(\langle\varrho,q\rangle,\alpha)) = f(\langle\varrho,q\rangle)$
unfolding *f_def* **by** *simp*
ultimately
obtain τ **where** $\tau\in M$ $\tau \in Vset(f(\langle\varrho,q\rangle))$ $q \Vdash \varphi$ $([\varrho,\tau]$ @ *nenv*)
using *LeastI*[*of* $\lambda\ \alpha.\ ?P(\langle\varrho,q\rangle,\alpha)$ $?\alpha$] **by** *auto*
with ‹$q\in G$› ‹$\varrho\in M$› ‹$nenv\in\_$› ‹$arity(\varphi)\leq 2 +_\omega length(nenv)$›
have $M[G]$, $map(val(G),[\varrho,\tau]$ @ *nenv*$) \models \varphi$
using *truth_lemma*[*OF* ‹$\varphi\in\_$›, *of* $[\varrho,\tau]$ @ *nenv*] **by** *auto*
moreover from ‹$x\in c$› ‹$c\in M[G]$›
have $x\in M[G]$ **using** *transitivity_MG* **by** *simp*
moreover
note ‹$M[G],[x,v]$ @ $env\models\varphi$› ‹$env = map(val(G),nenv)$› ‹$\tau\in M$› ‹$val(G,\varrho)=x$›
‹*univalent*$(\#\#M[G],\_,\_)$› ‹$x\in c$› ‹$v\in M[G]$›
ultimately
have $v=val(G,\tau)$
using *GenExtI*[*of* τ G] **unfolding** *univalent_def* **by** (*auto*)
from ‹$\tau \in Vset(f(\langle\varrho,q\rangle))$› ‹$Ord(f(\_))$› ‹$f(\langle\varrho,q\rangle)\in Y$›
have $\tau \in Vset(?sup)$
using *Vset_Ord_rank_iff lt_Union_iff*[*of* _ $rank(\tau)$] **by** *auto*

with ‹τ∈M›
have *val*(G,τ) ∈ *val*(G,*?big_name*)
using *domain_of_prod*[*of* **1** {**1**} {x∈*Vset*(*?sup*). x ∈ M}] *def_val*[*of* G *?big_name*]
one_in_G one_in_P **by** (*auto simp del*: *Vset_rank_iff*)
with ‹v=*val*(G,τ)›
show v ∈ *val*(G,*?big_name*)
by *simp*
qed
from ‹*?big_name*∈M›
have *?repl* = {v∈*?big*. ∃ x∈c. M[G], [x,v] @ *env* ⊨ φ} (**is** _ = *?rhs*)
proof(*intro equalityI subsetI*)
fix v
assume v∈*?repl*
with ‹*?repl*⊆*?big*›
obtain x **where** x∈c M[G], [x, v] @ *env* ⊨ φ v∈*?big*
using *subsetD* **by** *auto*
with ‹*univalent*(##M[G],_,_)› ‹c∈M[G]›
show v ∈ *?rhs*
unfolding *univalent_def*
using *transitivity_MG ReplaceI*[*of* λ x v. ∃ x∈c. M[G], [x, v] @ *env* ⊨ φ] **by** *blast*
next
fix v
assume v∈*?rhs*
then
obtain x **where**
v∈*val*(G, *?big_name*) M[G], [x, v] @ *env* ⊨ φ x∈c
by *blast*
moreover from *this* ‹c∈M[G]›
have v∈M[G] x∈M[G]
using *transitivity_MG GenExtI*[*OF* ‹*?big_name*∈_›,*of* G] **by** *auto*
moreover from *calculation* ‹*univalent*(##M[G],_,_)›
have *?R*(x,y) ⟹ y = v **for** y
unfolding *univalent_def* **by** *auto*
ultimately
show v∈*?repl*
using *ReplaceI*[*of ?R x v c*]
by *blast*
qed
moreover
let *?ψ* = (·∃··*0* ∈ *2* +$_\omega$ *length*(*env*) · ∧ φ··)
from ‹φ∈_›
have *?ψ*∈*formula arity*(*?ψ*) ≤ *2* +$_\omega$ *length*(*env*)
using *pred_mono*[*OF* _ ‹*arity*(φ)≤*2*+$_\omega$*length*(*env*)›] *lt_trans*[*OF* _ *le_refl*]
by (*auto simp add*:*ord_simp_union arity*)
moreover
from ‹φ∈_› ‹*arity*(φ)≤*2*+$_\omega$*length*(*env*)› ‹c∈M[G]› ‹*env*∈_›
have (∃ x∈c. M[G], [x,v] @ *env* ⊨ φ) ⟷ M[G], [v] @ *env* @ [c] ⊨ *?ψ* **if** v∈M[G]

for *v*
  using *that nth_concat transitivity_MG*[*OF* _ ‹*c*∈*M*[*G*]›] *arity_sats_iff*[*of* *φ* [*c*] _ [_,*v*]@*env*]
  by *auto*
 moreover from *this*
 have {*v*∈*?big*. ∃ *x*∈*c*. *M*[*G*], [*x*,*v*] @ *env* ⊨ *φ*} = {*v*∈*?big*. *M*[*G*], [*v*] @ *env* @ [*c*] ⊨ *?ψ*}
  using *transitivity_MG*[*OF* _ *GenExtI*, *OF* _ ‹*?big_name*∈*M*›]
  by *simp*
 moreover from *calculation* **and** ‹*env*∈_› ‹*c*∈_› ‹*?big*∈*M*[*G*]›
 have {*v*∈*?big*. *M*[*G*] , [*v*] @ *env* @ [*c*] ⊨ *?ψ*} ∈ *M*[*G*]
  using *Collect_sats_in_MG* **by** *auto*
 ultimately
 show *?thesis* **by** *simp*
qed

theorem *strong_replacement_in_MG*:
 assumes
  φ∈*formula* **and** *arity*(*φ*) ≤ *2* +ω *length*(*env*) *env* ∈ *list*(*M*[*G*])
  and
  ground_replacement:
  ⋀*nenv*. *ground_replacement_assm*(*M*,[ℙ,*leq*,**1**] @ *nenv*, *φ*)
 shows
  strong_replacement(##*M*[*G*],λ*x* *v*. *M*[*G*],[*x*,*v*] @ *env* ⊨ *φ*)
proof -
 let *?R*=λ*x* *y* . *M*[*G*], [*x*, *y*] @ *env* ⊨ *φ*
 {
  fix *A*
  let *?Y*={*v* . *x* ∈ *A*, *v*∈*M*[*G*] ∧ *?R*(*x*,*v*)}
  assume *1*: (##*M*[*G*])(*A*) *univalent*(##*M*[*G*], *A*, *?R*)
  with *assms*
  have (##*M*[*G*])(*?Y*)
   using *Replace_sats_in_MG ground_replacement 1*
   unfolding *ground_replacement_assm_def* **by** *auto*
  have *b* ∈ *?Y* ⟷ (∃ *x*[##*M*[*G*]]. *x* ∈ *A* ∧ *?R*(*x*,*b*)) **if** (##*M*[*G*])(*b*) **for** *b*
  proof(*rule*)
   from ‹(##*M*[*G*])(*A*)›
   show ∃ *x*[##*M*[*G*]]. *x* ∈ *A* ∧ *?R*(*x*,*b*) **if** *b* ∈ *?Y*
    using *that transitivity_MG* **by** *auto*
  next
   show *b* ∈ *?Y* **if** ∃ *x*[##*M*[*G*]]. *x* ∈ *A* ∧ *?R*(*x*,*b*)
   proof -
    from ‹(##*M*[*G*])(*b*)›
    have *b*∈*M*[*G*] **by** *simp*
    with *that*
    obtain *x* **where** (##*M*[*G*])(*x*) *x*∈*A* *b*∈*M*[*G*] ∧ *?R*(*x*,*b*)
     by *blast*
    moreover from *this 1* ‹(##*M*[*G*])(*b*)›
    have *x*∈*M*[*G*] *z*∈*M*[*G*] ∧ *?R*(*x*,*z*) ⟹ *b* = *z* **for** *z*

unfolding *univalent_def*
by *auto*
ultimately
show *?thesis*
using *ReplaceI*[*of* $\lambda\ x\ y.\ y{\in}M[G] \wedge\ ?R(x,y)$] **by** *blast*
qed
qed
then
have $\forall\, b[\#\#M[G]].\ b \in\ ?Y \longleftrightarrow (\exists\, x[\#\#M[G]].\ x \in A \wedge\ ?R(x,b))$
by *simp*
with ‹$(\#\#M[G])(?Y)$›
have $(\exists\, Y[\#\#M[G]].\ \forall\, b[\#\#M[G]].\ b \in\ Y \longleftrightarrow (\exists\, x[\#\#M[G]].\ x \in A \wedge\ ?R(x,b)))$
by *auto*
}
then show *?thesis* **unfolding** *strong_replacement_def*
by *simp*
qed

lemma *replacement_assm_MG*:
assumes
ground_replacement:
$\bigwedge$*nenv. ground_replacement_assm*(M,[$\mathbb{P}$,*leq*,$\mathbf{1}$] @ *nenv*, φ)
shows
replacement_assm($M[G]$,*env*,φ)
using *assms strong_replacement_in_MG*
unfolding *replacement_assm_def* **by** *simp*

end — *G_generic1*

end

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

theory *Infinity_Axiom*
imports *Union_Axiom Pairing_Axiom*
begin

context *G_generic1* **begin**

interpretation *mg_triv*: *M_trivial*$\#\#M[G]$
using *transitivity_MG zero_in_MG*[*of G*] *generic Union_MG pairing_in_MG*
by *unfold_locales auto*

lemma *infinity_in_MG* : *infinity_ax*($\#\#M[G]$)
proof -
have $\omega \in M[G]$
using *M_subset_MG one_in_G nat_in_M* **by** *auto*
moreover from *this*

have $succ(y) \in \omega \cap M[G]$ **if** $y \in \omega$ **for** y
using *that transitivity_MG* **by** *blast*
ultimately
show *?thesis*
using *transitivity_MG*[*of 0* ω]
unfolding *infinity_ax_def*
by *auto*
qed

end — *G_generic1*

end

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

theory *Choice_Axiom*
imports
Powerset_Axiom
Extensionality_Axiom
Foundation_Axiom
Replacement_Axiom
Infinity_Axiom
begin

definition
$upair\_name :: i \Rightarrow i \Rightarrow i \Rightarrow i$ **where**
$upair\_name(\tau,\varrho,on) \equiv Upair(\langle\tau,on\rangle,\langle\varrho,on\rangle)$

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

definition
$induced\_surj :: i \Rightarrow i \Rightarrow i \Rightarrow i$ **where**
$induced\_surj(f,a,e) \equiv f\text{-}``(range(f)\text{-}a)\times\{e\} \cup restrict(f,f\text{-}``a)$

lemma *domain_induced_surj*: $domain(induced\_surj(f,a,e)) = domain(f)$
unfolding *induced_surj_def* **using** *domain_restrict domain_of_prod* **by** *auto*

lemma *range_restrict_vimage*:
assumes $function(f)$
shows $range(restrict(f,f\text{-}``a)) \subseteq a$
proof
from *assms*
have $function(restrict(f,f\text{-}``a))$
using *function_restrictI* **by** *simp*
fix y
assume $y \in range(restrict(f,f\text{-}``a))$
then

```
  obtain x where ⟨x,y⟩ ∈ restrict(f,f-‘‘a)  x ∈ f-‘‘a x∈domain(f)
    using domain_restrict domainI[of _ _ restrict(f,f-‘‘a)] by auto
  moreover
  note ‹function(restrict(f,f-‘‘a))›
  ultimately
  have y = restrict(f,f-‘‘a)‘x
    using function_apply_equality by blast
  also from ‹x ∈ f-‘‘a›
  have restrict(f,f-‘‘a)‘x = f‘x
    by simp
  finally
  have y = f‘x .
  moreover from assms ‹x∈domain(f)›
  have ⟨x,f‘x⟩ ∈ f
    using function_apply_Pair by auto
  moreover
  note assms ‹x ∈ f-‘‘a›
  ultimately
  show y∈a
    using function_image_vimage[of f a] by auto
qed

lemma induced_surj_type:
  assumes function(f)
  shows
    induced_surj(f,a,e): domain(f) → {e} ∪ a
    and
    x ∈ f-‘‘a ⟹ induced_surj(f,a,e)‘x = f‘x
proof -
  let ?f1=f-‘‘(range(f)-a) × {e} and ?f2=restrict(f, f-‘‘a)
  have domain(?f2) = domain(f) ∩ f-‘‘a
    using domain_restrict by simp
  moreover from assms
  have domain(?f1) = f-‘‘(range(f))-f-‘‘a
    using domain_of_prod function_vimage_Diff by simp
  ultimately
  have domain(?f1) ∩ domain(?f2) = 0
    by auto
  moreover
  have function(?f1) relation(?f1) range(?f1) ⊆ {e}
    unfolding function_def relation_def range_def by auto
  moreover from this and assms
  have ?f1: domain(?f1) → range(?f1)
    using function_imp_Pi by simp
  moreover from assms
  have ?f2: domain(?f2) → range(?f2)
    using function_imp_Pi[of restrict(f, f -‘‘ a)] function_restrictI by simp
  moreover from assms
  have range(?f2) ⊆ a
```

```
    using range_restrict_vimage by simp
  ultimately
  have induced_surj(f,a,e): domain(?f1) ∪ domain(?f2) → {e} ∪ a
   unfolding induced_surj_def using fun_is_function fun_disjoint_Un fun_weaken_type
by simp
  moreover
  have domain(?f1) ∪ domain(?f2) = domain(f)
    using domain_restrict domain_of_prod by auto
  ultimately
  show induced_surj(f,a,e): domain(f) → {e} ∪ a
    by simp
  assume x ∈ f-``a
  then
  have ?f2`x = f`x
    using restrict by simp
  moreover from ‹x ∈ f-``a› ‹domain(?f1) = _›
  have x ∉ domain(?f1)
    by simp
  ultimately
  show induced_surj(f,a,e)`x = f`x
    unfolding induced_surj_def using fun_disjoint_apply2[of x ?f1 ?f2] by simp
qed

lemma induced_surj_is_surj :
  assumes
    e∈a function(f) domain(f) = α ⋀y. y ∈ a ⟹ ∃x∈α. f ` x = y
  shows induced_surj(f,a,e) ∈ surj(α,a)
  unfolding surj_def
proof (intro CollectI ballI)
  from assms
  show induced_surj(f,a,e): α → a
    using induced_surj_type[of f a e] cons_eq cons_absorb by simp
  fix y
  assume y ∈ a
  with assms
  have ∃x∈α. f ` x = y
    by simp
  then
  obtain x where x∈α f ` x = y by auto
  with ‹y∈a› assms
  have x∈f-``a
    using vimage_iff function_apply_Pair[of f x] by auto
  with ‹f ` x = y› assms
  have induced_surj(f, a, e) ` x = y
    using induced_surj_type by simp
  with ‹x∈α› show
    ∃x∈α. induced_surj(f, a, e) ` x = y by auto
qed
```

lemma (**in** *M_ZF1_trans*) *upair_name_closed* :
  ⟦ *x*∈*M*; *y*∈*M* ; *o*∈*M*⟧ ⟹ *upair_name*(*x*,*y*,*o*)∈*M*
  unfolding *upair_name_def*
  using *upair_in_M_iff pair_in_M_iff Upair_eq_cons*
  by *simp*

context *G_generic1*
begin

lemma *val_upair_name* : *val*(*G*,*upair_name*(*τ*,*ϱ*,**1**)) = {*val*(*G*,*τ*),*val*(*G*,*ϱ*)}
  unfolding *upair_name_def*
  using *val_Upair Upair_eq_cons generic one_in_G*
  by *simp*

lemma *val_opair_name* : *val*(*G*,*opair_name*(*τ*,*ϱ*,**1**)) = ⟨*val*(*G*,*τ*),*val*(*G*,*ϱ*)⟩
  unfolding *opair_name_def Pair_def*
  using *val_upair_name* **by** *simp*

lemma *val_RepFun_one*: *val*(*G*,{⟨*f*(*x*),**1**⟩ . *x*∈*a*}) = {*val*(*G*,*f*(*x*)) . *x*∈*a*}
proof -
  let *?A* = {*f*(*x*) . *x* ∈ *a*}
  let *?Q* = *λ*⟨*x*,*p*⟩ . *p* = **1**
  have **1** ∈ ℙ∩*G* **using** *generic one_in_G one_in_P* **by** *simp*
  have {⟨*f*(*x*),**1**⟩ . *x* ∈ *a*} = {*t* ∈ *?A* × ℙ . *?Q*(*t*)}
    using *one_in_P* **by** *force*
  then
  have *val*(*G*,{⟨*f*(*x*),**1**⟩ . *x* ∈ *a*}) = *val*(*G*,{*t* ∈ *?A* × ℙ . *?Q*(*t*)})
    by *simp*
  also
  have ... = {*z* . *t* ∈ *?A* , (∃ *p*∈ℙ∩*G* . *?Q*(⟨*t*,*p*⟩)) ∧ *z*= *val*(*G*,*t*)}
    using *val_of_name_alt* **by** *simp*
  also from ‹**1**∈ℙ∩*G*›
  have ... = {*val*(*G*,*t*) . *t* ∈ *?A* }
    by *force*
  also
  have ... = {*val*(*G*,*f*(*x*)) . *x* ∈ *a*}
    by *auto*
  finally
  show *?thesis*
    by *simp*
qed

end— *G_generic1*

24.1 $M[G]$ is a transitive model of ZF

sublocale *G_generic1* ⊆ *ext*:*M_Z_trans M*[*G*]
  using *Transset_MG generic pairing_in_MG Union_MG*
    extensionality_in_MG power_in_MG foundation_in_MG

replacement_assm_MG separation_in_MG infinity_in_MG
replacement_ax1
by *unfold_locales*

lemma (**in** *M_replacement*) *upair_name_lam_replacement* :
M(*z*) ⟹ *lam_replacement*(*M*,λ*x* . *upair_name*(*fst*(*x*),*snd*(*x*),*z*))
using *lam_replacement_Upair*[*THEN* [5] *lam_replacement_hcomp2*]
lam_replacement_product
lam_replacement_fst lam_replacement_snd lam_replacement_constant
unfolding *upair_name_def*
by *simp*

lemma (**in** *forcing_data1*) *repl_opname_check* :
assumes *A*∈*M f*∈*M*
shows {*opair_name*(*check*(*x*),*f*‘*x*,**1**). *x*∈*A*}∈*M*
using *assms lam_replacement_constant check_lam_replacement lam_replacement_identity*
upair_name_lam_replacement[*THEN* [5] *lam_replacement_hcomp2*]
lam_replacement_apply2[*THEN* [5] *lam_replacement_hcomp2*]
lam_replacement_imp_strong_replacement_aux
transitivity RepFun_closed upair_name_closed apply_closed
unfolding *opair_name_def*
by *simp*

theorem (**in** *G_generic1*) *choice_in_MG*:
assumes *choice_ax*(##*M*)
shows *choice_ax*(##*M*[*G*])
proof -
{
fix *a*
assume *a*∈*M*[*G*]
then
obtain τ **where** τ∈*M val*(*G*,τ) = *a*
using *GenExt_def* **by** *auto*
with ‹τ∈*M*›
have *domain*(τ)∈*M*
using *domain_closed* **by** *simp*
then
obtain *s* α **where** *s*∈*surj*(α,*domain*(τ)) *Ord*(α) *s*∈*M* α∈*M*
using *assms choice_ax_abs*
by *auto*
then
have α∈*M*[*G*]
using *M_subset_MG generic one_in_G subsetD*
by *blast*
let *?A*=*domain*(τ)×ℙ
let *?g* = {*opair_name*(*check*(β),*s*‘β,**1**). β∈α}
have *?g* ∈ *M*
using ‹*s*∈*M*› ‹α∈*M*› *repl_opname_check*
by *simp*

let *?f_dot*={⟨*opair_name*(*check*(β),*s*‘β,**1**),**1**⟩. β∈α}
have *?f_dot* = *?g* × {**1**} **by** *blast*
define *f* **where**
  f ≡ *val*(*G*,*?f_dot*)
from ‹*?g*∈*M*› ‹*?f_dot* = *?g*×{**1**}›
have *?f_dot*∈*M*
  using *cartprod_closed singleton_closed*
  by *simp*
then
have *f* ∈ *M*[*G*]
  unfolding *f_def*
  by (*blast intro*:*GenExtI*)
have *f* = {*val*(*G*,*opair_name*(*check*(β),*s*‘β,**1**)) . β∈α}
  unfolding *f_def*
  using *val_RepFun_one*
  by *simp*
also
have ... = {⟨β,*val*(*G*,*s*‘β)⟩ . β∈α}
  using *val_opair_name val_check generic one_in_G one_in_P*
  by *simp*
finally
have *f* = {⟨β,*val*(*G*,*s*‘β)⟩ . β∈α} **.**
then
have *1*: *domain*(*f*) = α *function*(*f*)
  unfolding *function_def* **by** *auto*
have *2*: *y* ∈ *a* ⟹ ∃*x*∈α. *f* ‘ *x* = *y* **for** *y*
proof -
  fix *y*
  assume
    y ∈ *a*
  with ‹*val*(*G*,τ) = *a*›
  obtain σ **where** σ∈*domain*(τ) *val*(*G*,σ) = *y*
    using *elem_of_val*[*of y _ τ*]
    by *blast*
  with ‹*s*∈*surj*(α,*domain*(τ))›
  obtain β **where** β∈α *s*‘β = σ
    unfolding *surj_def*
    by *auto*
  with ‹*val*(*G*,σ) = *y*›
  have *val*(*G*,*s*‘β) = *y*
    by *simp*
  with ‹*f* = {⟨β,*val*(*G*,*s*‘β)⟩ . β∈α}› ‹β∈α›
  have ⟨β,*y*⟩∈*f*
    by *auto*
  with ‹*function*(*f*)›
  have *f*‘β = *y*
    using *function_apply_equality* **by** *simp*
  with ‹β∈α› **show**
    ∃β∈α. *f* ‘ β = *y*

```
      by auto
    qed
    then
    have ∃α∈(M[G]). ∃f'∈(M[G]). Ord(α) ∧ f' ∈ surj(α,a)
    proof (cases a=0)
      case True
      then
      show ?thesis
        unfolding surj_def
        using zero_in_MG
        by auto
    next
      case False
      with ‹a∈M[G]›
      obtain e where e∈a e∈M[G]
        using transitivity_MG
        by blast
      with 1 and 2
      have induced_surj(f,a,e) ∈ surj(α,a)
        using induced_surj_is_surj by simp
      moreover from ‹f∈M[G]› ‹a∈M[G]› ‹e∈M[G]›
      have induced_surj(f,a,e) ∈ M[G]
        unfolding induced_surj_def
        by (simp flip: setclass_iff)
      moreover
      note ‹α∈M[G]› ‹Ord(α)›
      ultimately
      show ?thesis
        by auto
    qed
  }
  then
  show ?thesis
    using ext.choice_ax_abs
    by simp
qed

sublocale G_generic1_AC ⊆ ext:M_ZC_basic M[G]
  using choice_ax choice_in_MG
  by unfold_locales

end
```

25 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*: $p\in\mathbb{P} \Longrightarrow \exists\, q\in\mathbb{P}.\ \exists\, r\in\mathbb{P}.\ q \preceq p \wedge r \preceq p \wedge q \perp r$
begin

For separative preorders, the complement of every filter is dense. Hence an M-generic filter cannot belong to the ground model.

lemma *filter_complement_dense*:
  assumes *filter*(G)
  shows *dense*($\mathbb{P}$ - G)
proof
  fix p
  assume $p\in\mathbb{P}$
  show $\exists\, d\in\mathbb{P}$ - $G.\ d \preceq p$
  proof (*cases* $p\in G$)
    case *True*
    note ‹$p\in\mathbb{P}$› *assms*
    moreover
    obtain $q\ r$ **where** $q \preceq p\ r \preceq p\ q \perp r\ q\in\mathbb{P}\ r\in\mathbb{P}$
      using *separative*[*OF* ‹$p\in\mathbb{P}$›]
      by *force*
    with ‹*filter*(G)›
    obtain s **where** $s \preceq p\ s \notin G\ s \in \mathbb{P}$
      using *filter_imp_compat*[*of* $G\ q\ r$]
      by *auto*
    then
    show *?thesis*
      by *blast*
  next
    case *False*
    with ‹$p\in\mathbb{P}$›
    show *?thesis*
      using *refl_leq* **unfolding** *Diff_def* **by** *auto*
  qed
qed

end — *separative_notion*

locale *ctm_separative* = *forcing_data1* + *separative_notion*
begin

context
  fixes G
  assumes *generic*: *M_generic*(G)
begin

interpretation *G_generic1* $\mathbb{P}$ *leq* **1** *M enum G*
by *unfold_locales* (*simp add*:*generic*)

lemma *generic_not_in_M*:
shows $G \notin M$
proof
assume $G{\in}M$
then
have $\mathbb{P}$ - $G \in M$
using *Diff_closed* **by** *simp*
moreover
have $\neg(\exists\, q{\in}G.\ q \in \mathbb{P}$ - $G)$ $(\mathbb{P}$ - $G) \subseteq \mathbb{P}$
unfolding *Diff_def* **by** *auto*
moreover
note *generic*
ultimately
show *False*
using *filter_complement_dense*[*of G*] *M_generic_denseD*[*of* $\mathbb{P}$-*G*]
by *auto*
qed

theorem *proper_extension*: $M \neq M[G]$
using *generic G_in_Gen_Ext one_in_G generic_not_in_M*
by *force*
end
end — *ctm_separative*

end

26 A poset of successions

theory *Succession_Poset*
imports
ZF_Trans_Interpretations
Proper_Extension
begin

In this theory we define a separative poset. Its underlying set is the set of finite binary sequences (that is, with codomain $2 = 0, 1$); of course, one can see that set as the set ω -||> *2* or equivalently as the set of partial functions *Fn*(ω, ω, *2*), i.e. the set of partial functions bounded by ω.

The order relation of the poset is that of being less defined as functions (cf. *Fnlerel*($A^{<\omega}$)), so it could be surprising that we have not used *Fn*(ω, ω, *2*) for the set. The only reason why we keep this alternative definition is because we can prove $A^{<\omega} \in M$ (and therefore *Fnlerel*($A^{<\omega}$) $\in M$) using only one instance of separation.

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

$seq\_upd(f,a) \equiv \lambda\ j \in succ(domain(f))\ .\ if\ j < domain(f)\ then\ f`j\ else\ a$

lemma *seq_upd_succ_type* :
assumes $n \in nat\ f \in n \to A\ a \in A$
shows $seq\_upd(f,a) \in succ(n) \to A$
proof -
from *assms*
have *equ*: $domain(f) = n$
using *domain_of_fun* **by** *simp*
{
fix j
assume $j \in succ(domain(f))$
with *equ* ‹$n \in \_$›
have $j \leq n$
using *ltI* **by** *auto*
with ‹$n \in \_$›
consider (*lt*) $j < n$ | (*eq*) $j = n$
using *leD* **by** *auto*
then
have $(if\ j < n\ then\ f`j\ else\ a) \in A$
proof *cases*
case *lt*
with ‹$f \in \_$›
show *?thesis*
using *apply_type ltD*[*OF lt*] **by** *simp*
next
case *eq*
with ‹$a \in \_$›
show *?thesis*
by *auto*
qed
}
with *equ*
show *?thesis*
unfolding *seq_upd_def*
using *lam_type*[*of* $succ(domain(f))$]
by *auto*
qed

lemma *seq_upd_type* :
assumes $f \in A^{<\omega}\ a \in A$
shows $seq\_upd(f,a) \in A^{<\omega}$
proof -
from ‹$f \in \_$›
obtain y **where** $y \in nat\ f \in y \to A$
unfolding *seqspace_def* **by** *blast*
with ‹$a \in A$›
have $seq\_upd(f,a) \in succ(y) \to A$
using *seq_upd_succ_type* **by** *simp*

with ‹$y \in \_$›
show *?thesis*
unfolding *seqspace_def* **by** *auto*
qed

lemma *seq_upd_apply_domain* [*simp*]:
assumes $f:n \to A$ $n \in nat$
shows $seq\_upd(f,a)`n = a$
unfolding *seq_upd_def* **using** *assms domain_of_fun* **by** *auto*

lemma *zero_in_seqspace* :
shows $0 \in A^{<\omega}$
unfolding *seqspace_def*
by *force*

definition
$seqlerel :: i \Rightarrow i$ **where**
$seqlerel(A) \equiv Fnlerel(A^{<\omega})$

definition
$seqle :: i$ **where**
$seqle \equiv seqlerel(2)$

lemma *seqleI*[*intro*!]:
$\langle f,g \rangle \in 2^{<\omega} \times 2^{<\omega} \Longrightarrow g \subseteq f \Longrightarrow \langle f,g \rangle \in seqle$
unfolding *seqle_def seqlerel_def seqspace_def Rrel_def Fnlerel_def*
by *blast*

lemma *seqleD*[*dest*!]:
$z \in seqle \Longrightarrow \exists x\ y.\ \langle x,y \rangle \in 2^{<\omega} \times 2^{<\omega} \wedge y \subseteq x \wedge z = \langle x,y \rangle$
unfolding *Rrel_def seqle_def seqlerel_def Fnlerel_def*
by *blast*

lemma *upd_leI* :
assumes $f \in 2^{<\omega}$ $a \in 2$
shows $\langle seq\_upd(f,a),f \rangle \in seqle$ (**is** $\langle ?f,\_ \rangle \in \_$)
proof
show $\langle ?f, f \rangle \in 2^{<\omega} \times 2^{<\omega}$
using *assms seq_upd_type* **by** *auto*
next
show $f \subseteq seq\_upd(f,a)$
proof
fix x
assume $x \in f$
moreover from ‹$f \in 2^{<\omega}$›
obtain n **where** $n \in nat\ f : n \to 2$
by *blast*
moreover from *calculation*
obtain y **where** $y \in n\ x = \langle y, f`y \rangle$

```
    using Pi_memberD[of f n λ_ . 2]
    by blast
  moreover from ‹f:n→2›
  have domain(f) = n
    using domain_of_fun by simp
  ultimately
  show x ∈ seq_upd(f,a)
    unfolding seq_upd_def lam_def
    by (auto intro:ltI)
 qed
qed

lemma preorder_on_seqle: preorder_on(2^<ω,seqle)
 unfolding preorder_on_def refl_def trans_on_def by blast

lemma zero_seqle_max: x∈2^<ω ⟹ ⟨x,0⟩ ∈ seqle
 using zero_in_seqspace
 by auto

interpretation sp:forcing_notion 2^<ω seqle 0
 using preorder_on_seqle zero_seqle_max zero_in_seqspace
 by unfold_locales simp_all

notation sp.Leq (infixl ‹⪯s› 50)
notation sp.Incompatible (infixl ‹⊥s› 50)

lemma seqspace_separative:
 assumes f∈2^<ω
 shows seq_upd(f,0) ⊥s seq_upd(f,1) (is ?f ⊥s ?g)
proof
 assume sp.compat(?f, ?g)
 then
 obtain h where h ∈ 2^<ω ?f ⊆ h ?g ⊆ h
   by blast
 moreover from ‹f∈_›
 obtain y where y∈nat f:y→2
   by blast
 moreover from this
 have ?f: succ(y) → 2 ?g: succ(y) → 2
   using seq_upd_succ_type by blast+
 moreover from this
 have ⟨y,?f‘y⟩ ∈ ?f ⟨y,?g‘y⟩ ∈ ?g
   using apply_Pair by auto
 ultimately
 have ⟨y,0⟩ ∈ h ⟨y,1⟩ ∈ h
   by auto
 moreover from ‹h ∈ 2^<ω›
 obtain n where n∈nat h:n→2
   by blast
```

ultimately
show *False*
using *fun_is_function*[*of h n* λ_. *2*]
unfolding *seqspace_def function_def* **by** *auto*
qed

definition *seqleR_fm* :: *i* ⇒ *i* **where**
seqleR_fm(*fg*) ≡ *Exists*(*Exists*(*And*(*pair_fm*(*0*,*1*,*fg*+ω *2*),*subset_fm*(*1*,*0*))))

lemma *type_seqleR_fm* : *fg* ∈ *nat* ⟹ *seqleR_fm*(*fg*) ∈ *formula*
unfolding *seqleR_fm_def*
by *simp*

arity_theorem for *seqleR_fm*

lemma (**in** *M_ctm1*) *seqleR_fm_sats* :
assumes *fg*∈*nat env*∈*list*(*M*)
shows (*M*, *env* ⊨ *seqleR_fm*(*fg*)) ⟷ (∃ *f*[##*M*]. ∃ *g*[##*M*]. *pair*(##*M*,*f*,*g*,*nth*(*fg*,*env*))
∧ *f* ⊇ *g*)
unfolding *seqleR_fm_def*
using *assms trans_M sats_subset_fm pair_iff_sats*
by *auto*

context *M_ctm1*
begin

lemma *seqle_in_M*: *seqle* ∈ *M*
using *arity_seqleR_fm seqleR_fm_sats type_seqleR_fm*
cartprod_closed seqspace_closed nat_into_M nat_in_M pair_in_M_iff
unfolding *seqle_def seqlerel_def Rrel_def Fnlerel_def*
by (*rule_tac Collect_in_M*[*of seqleR_fm*(*0*) []],*auto*)

26.1 Cohen extension is proper

interpretation *ctm_separative* $2^{<\omega}$ *seqle 0*
proof (*unfold_locales*)
fix *f*
let *?q*=*seq_upd*(*f*,*0*) **and** *?r*=*seq_upd*(*f*,*1*)
assume *f* ∈ $2^{<\omega}$
then
have *?q* ≼*s f* ∧ *?r* ≼*s f* ∧ *?q* ⊥*s ?r*
using *upd_leI seqspace_separative* **by** *auto*
moreover from *calculation*
have *?q* ∈ $2^{<\omega}$ *?r* ∈ $2^{<\omega}$
using *seq_upd_type*[*of f 2*] **by** *auto*
ultimately
show ∃ *q*∈$2^{<\omega}$. ∃ *r*∈$2^{<\omega}$. *q* ≼*s f* ∧ *r* ≼*s f* ∧ *q* ⊥*s r*
by (*rule_tac bexI*)+ — why the heck auto-tools don't solve this?
next

show $2^{<\omega} \in M$
using *nat_into_M seqspace_closed* **by** *simp*
next
show *seqle* $\in M$
using *seqle_in_M* **.**
qed

lemma *cohen_extension_is_proper*: $\exists\, G.\ M\_generic(G) \wedge M \neq M[G]$
using *proper_extension generic_filter_existence zero_in_seqspace*
by *force*

end — *M_ctm1*

end

27 The existence of generic extensions

theory *Forcing_Main*
imports
Ordinals_In_MG
Choice_Axiom
Succession_Poset

begin

27.1 The generic extension is countable

lemma (**in** *forcing_data1*) *surj_nat_MG* : $\exists f.\ f \in surj(\omega, M[G])$
proof -
let *?f*=$\lambda n{\in}\omega.\ val(G, enum`n)$
have $x \in \omega \Longrightarrow val(G,\ enum\ `\ x) \in M[G]$ **for** x
using *GenExtI bij_is_fun*[*OF M_countable*]
by *simp*
then
have *?f*: $\omega \to M[G]$
using *lam_type*[*of* ω $\lambda n.\ val(G, enum`n)$ $\lambda\_.M[G]$] **by** *simp*
moreover
have $\exists\, n{\in}\omega.\ ?f`n = x$ **if** $x{\in}M[G]$ **for** x
using *that GenExt_iff*[*of _ G*] *bij_is_surj*[*OF M_countable*]
unfolding *surj_def* **by** *auto*
ultimately
show *?thesis*
unfolding *surj_def* **by** *blast*
qed

lemma (**in** *G_generic1*) *MG_eqpoll_nat*: $M[G] \approx \omega$
proof -
obtain f **where** $f \in surj(\omega, M[G])$
using *surj_nat_MG* **by** *blast*

then
have $M[G] \lesssim \omega$
using *well_ord_surj_imp_lepoll well_ord_Memrel*[*of* ω] **by** *simp*
moreover
have $\omega \lesssim M[G]$
using *ext.nat_into_M subset_imp_lepoll* **by** (*auto del*:*lepollI*)
ultimately
show *?thesis*
using *eqpollI* **by** *simp*
qed

27.2 Extensions of ctms of fragments of *ZFC*

context *G_generic1*
begin

lemma *sats_ground_repl_fm_imp_sats_ZF_replacement_fm*:
assumes
$\varphi \in$*formula* M, $[] \models \cdot$*Replacement*(*ground_repl_fm*(φ))$\cdot$
shows
$M[G]$, $[] \models \cdot$*Replacement*(φ)$\cdot$
using *assms sats_ZF_replacement_fm_iff*
by (*auto simp*:*replacement_assm_def ground_replacement_assm_def*
intro:*strong_replacement_in_MG*[*simplified*])

lemma *satT_ground_repl_fm_imp_satT_ZF_replacement_fm*:
assumes
$\Phi \subseteq$ *formula* $M \models \{ \cdot$*Replacement*(*ground_repl_fm*(φ))$\cdot$. $\varphi \in \Phi\}$
shows
$M[G] \models \{ \cdot$*Replacement*(φ)$\cdot$. $\varphi \in \Phi\}$
using *assms sats_ground_repl_fm_imp_sats_ZF_replacement_fm*
by *auto*

end — *G_generic1*

theorem *extensions_of_ctms*:
assumes
$M \approx \omega$ *Transset*(M)
$M \models \cdot Z \cdot \cup \{\cdot$*Replacement*($p$)$\cdot$. $p \in$ *overhead*$\}$
$\Phi \subseteq$ *formula* $M \models \{ \cdot$*Replacement*(*ground_repl_fm*(φ))$\cdot$. $\varphi \in \Phi\}$
shows
$\exists N.$
$M \subseteq N \wedge N \approx \omega \wedge$ *Transset*(N) $\wedge$ $M \neq N \wedge$
$(\forall \alpha.$ *Ord*(α) $\longrightarrow (\alpha \in M \longleftrightarrow \alpha \in N)) \wedge$
$((M, [] \models \cdot AC \cdot) \longrightarrow N, [] \models \cdot AC \cdot) \wedge N \models \cdot Z \cdot \cup \{ \cdot$*Replacement*($\varphi$)$\cdot$. $\varphi \in \Phi\}$
proof -
from ‹$M \models \cdot Z \cdot \cup$ _› ‹*Transset*(M)›
interpret *M_ZF_ground_trans M*
using *M_satT_imp_M_ZF_ground_trans*

by *simp*
from ‹$M \approx \omega$›
obtain *enum* **where** *enum* ∈ *bij*(ω,*M*)
using *eqpoll_sym* **unfolding** *eqpoll_def* **by** *blast*
then
interpret *M_ctm1 M enum* **by** *unfold_locales*
interpret *forcing_data1* $2^{<\omega}$ *seqle 0 M enum*
using *nat_into_M seqspace_closed seqle_in_M*
by *unfold_locales simp*
obtain *G* **where** *M_generic*(*G*) *M* ≠ *M*[*G*]
using *cohen_extension_is_proper*
by *blast*

Recall that $M[G]$ denotes the generic extension of M using the poset of sequences $2^{<\omega}$.

then
interpret *G_generic1* $2^{<\omega}$ *seqle 0 _ enum G* **by** *unfold_locales*
interpret *MG*: *M_Z_basic M*[*G*]
using *generic pairing_in_MG*
Union_MG extensionality_in_MG power_in_MG
foundation_in_MG replacement_assm_MG
separation_in_MG infinity_in_MG replacement_ax1
by *unfold_locales simp*
have *M*, []⊨ ·*AC*· ⟹ *M*[*G*], [] ⊨ ·*AC*·
proof -
assume *M*, [] ⊨ ·*AC*·
then
have *choice_ax*(##*M*)
unfolding *ZF_choice_fm_def* **using** *ZF_choice_auto* **by** *simp*
then
have *choice_ax*(##*M*[*G*]) **using** *choice_in_MG* **by** *simp*
then
show *M*[*G*], [] ⊨ ·*AC*·
using *ZF_choice_auto sats_ZFC_iff_sats_ZF_AC*
unfolding *ZF_choice_fm_def* **by** *simp*
qed
moreover
note ‹$M \neq M[G]$› ‹$M \models \{ \cdot Replacement(ground\_repl\_fm(\varphi))\cdot . \varphi \in \Phi\}$› ‹$\Phi \subseteq formula$›
moreover
have *Transset*(*M*[*G*]) **using** *Transset_MG* **.**
moreover
have *M* ⊆ *M*[*G*] **using** *M_subset_MG*[*OF one_in_G*] *generic* **by** *simp*
ultimately
show *?thesis*
using *Ord_MG_iff MG_eqpoll_nat ext.M_satT_Zermelo_fms*
satT_ground_repl_fm_imp_satT_ZF_replacement_fm[*of* Φ]
by (*rule_tac x*=*M*[*G*] **in** *exI*, *auto*)
qed

lemma *ZF_replacement_overhead_sub_ZF*: {·*Replacement*(*p*)· . *p* ∈ *overhead*} ⊆ *ZF*
using *instances1_fms_type instances_ground_fms_type*
unfolding *overhead_def ZF_def ZF_schemes_def* **by** *auto*

theorem *extensions_of_ctms_ZF*:
assumes
M ≈ ω *Transset*(*M*) *M* ⊨ *ZF*
shows
∃ *N*.
M ⊆ *N* ∧ *N* ≈ ω ∧ *Transset*(*N*) ∧ *N* ⊨ *ZF* ∧ *M*≠*N* ∧
(∀ α. *Ord*(α) ⟶ (α ∈ *M* ⟷ α ∈ *N*)) ∧
((*M*, []⊨ ·*AC*·) ⟶ *N* ⊨ *ZFC*)
proof -
from *assms*
have ∃ *N*.
M ⊆ *N* ∧ *N* ≈ ω ∧ *Transset*(*N*) ∧ *M*≠*N* ∧
(∀ α. *Ord*(α) ⟶ (α ∈ *M* ⟷ α ∈ *N*)) ∧
((*M*, []⊨ ·*AC*·) ⟶ *N*, [] ⊨ ·*AC*·) ∧ *N* ⊨ ·*Z*· ∪ { ·*Replacement*(φ)· . φ ∈ *formula*}
using *extensions_of_ctms*[*of M formula*] *satT_ZF_imp_satT_Z*[*of M*]
satT_mono[*OF _ ground_repl_fm_sub_ZF*, *of M*]
satT_mono[*OF _ ZF_replacement_overhead_sub_ZF*, *of M*]
by (*auto simp*: *satT_Un_iff*)
then
obtain *N* **where** *N* ⊨ ·*Z*· ∪ { ·*Replacement*(φ)· . φ ∈ *formula*} *M* ⊆ *N N* ≈ ω *Transset*(*N*)
M ≠ *N* (∀ α. *Ord*(α) ⟶ α ∈ *M* ⟷ α ∈ *N*)
(*M*, []⊨ ·*AC*·) ⟶ *N*, [] ⊨ ·*AC*·
by *blast*
moreover from ‹*N* ⊨ ·*Z*· ∪ { ·*Replacement*(φ)· . φ ∈ *formula*}›
have *N* ⊨ *ZF*
using *satT_Z_ZF_replacement_imp_satT_ZF* **by** *auto*
moreover from *this* **and** ‹(*M*, []⊨ ·*AC*·) ⟶ *N*, [] ⊨ ·*AC*·›
have (*M*, []⊨ ·*AC*·) ⟶ *N* ⊨ *ZFC*
using *sats_ZFC_iff_sats_ZF_AC* **by** *simp*
ultimately
show *?thesis*
by *auto*
qed

end

28 Preservation of cardinals in generic extensions

theory *Cardinal_Preservation*
imports
Forcing_Main

begin

context *forcing_data1*

begin

lemma *antichain_abs'* [*absolut*]:
  ⟦ *A*∈*M* ⟧ ⟹ *antichain*^{M}(ℙ,*leq*,*A*) ⟷ *antichain*(ℙ,*leq*,*A*)
  unfolding *antichain_rel_def antichain_def compat_def*
  using *transitivity*[*of* _ *A*]
  by (*auto simp add*:*absolut*)

lemma *inconsistent_imp_incompatible*:
  assumes *p* ⊩ *φ env q* ⊩ *Neg*(*φ*) *env p*∈ℙ *q*∈ℙ
    arity(*φ*) ≤ *length*(*env*) *φ* ∈ *formula env* ∈ *list*(*M*)
  shows *p* ⊥ *q*
proof
  assume *compat*(*p*,*q*)
  then
  obtain *d* **where** *d* ≼ *p d* ≼ *q d* ∈ ℙ **by** *blast*
  moreover
  note *assms*
  moreover from *calculation*
  have *d* ⊩ *φ env d* ⊩ *Neg*(*φ*) *env*
    using *strengthening_lemma* **by** *simp_all*
  ultimately
  show *False*
    using *Forces_Neg*[*of d env φ*] *refl_leq*
    by (*auto dest*:*transitivity*; *drule_tac bspec*; *auto dest*:*transitivity*)
qed

notation *check* (‹_^{v}› [*101*] *100*)

end — *forcing_data1*

locale *G_generic2* = *G_generic1* + *forcing_data2*
locale *G_generic2_AC* = *G_generic1_AC* + *G_generic2*

locale *G_generic3* = *G_generic2* + *forcing_data3*
locale *G_generic3_AC* = *G_generic2_AC* + *G_generic3*

locale *G_generic3_AC_CH* = *G_generic3_AC* + *M_ZFC2_ground_CH_trans*

sublocale *G_generic3_AC* ⊆ *ext*:*M_ZFC2_trans M*[*G*]
  using *ground_replacements3 replacement_assm_MG*
  by *unfold_locales simp_all*

lemma (**in** *forcing_data1*) *forces_neq_apply_imp_incompatible*:

assumes
$p \Vdash \cdot 0`1 \text{ is } 2\cdot\ [f,a,b^v]$
$q \Vdash \cdot 0`1 \text{ is } 2\cdot\ [f,a,b'^v]$
$b \neq b'$
— More general version: taking general names b^v and b'^v, satisfying $p \Vdash \cdot\neg\cdot 0 = 1\cdot\cdot\ [b^v,\ b'^v]$ and $q \Vdash \cdot\neg\cdot 0 = 1\cdot\cdot\ [b^v,\ b'^v]$.
and
types:$f\in M\ a\in M\ b\in M\ b'\in M\ p\in\mathbb{P}\ q\in\mathbb{P}$
shows
$p \perp q$
proof -
{
fix G
assume *M_generic*(G)
then
interpret *G_generic1* _ _ _ _ _ G **by** *unfold_locales*
include *G_generic1_lemmas*
assume $q\in G$
with *assms* ‹*M_generic*(G)›
have $M[G],\ map(val(G),[f,a,b'^v]) \models \cdot 0`1 \text{ is } 2\cdot$
using *truth_lemma*[*of* $\cdot 0`1 \text{ is } 2\cdot\ [f,a,b'^v]$]
by (*auto simp add*:*ord_simp_union arity_fun_apply_fm fun_apply_type*)
with ‹$b \neq b'$› *types*
have $M[G],\ map(val(G),[f,a,b^v]) \models \cdot\neg\cdot 0`1 \text{ is } 2\cdot\cdot$
using *GenExtI* **by** *auto*
}
with *types*
have $q \Vdash \cdot\neg\cdot 0`1 \text{ is } 2\cdot\cdot\ [f,a,b^v]$
using *definition_of_forcing*[**where** φ=$\cdot\neg\cdot 0`1 \text{ is } 2\cdot\cdot$]
by (*auto simp add*:*ord_simp_union arity_fun_apply_fm*)
with ‹$p \Vdash \cdot 0`1 \text{ is } 2\cdot\ [f,a,b^v]$› **and** *types*
show $p \perp q$
using *inconsistent_imp_incompatible*
by (*simp add*:*ord_simp_union arity_fun_apply_fm fun_apply_type*)
qed

context *M_ctm2_AC*
begin

— Simplifying simp rules (because of the occurrence of *setclass*)
lemmas *sharp_simps* = *Card_rel_Union Card_rel_cardinal_rel Collect_abs Cons_abs Cons_in_M_iff Diff_closed Equal_abs Equal_in_M_iff Finite_abs Forall_abs Forall_in_M_iff Inl_abs Inl_in_M_iff Inr_abs Inr_in_M_iff Int_closed Inter_abs Inter_closed M_nat Member_abs Member_in_M_iff Memrel_closed Nand_abs Nand_in_M_iff Nil_abs Nil_in_M Ord_cardinal_rel Pow_rel_closed Un_closed Union_abs Union_closed and_abs and_closed apply_abs apply_closed bij_rel_closed bijection_abs bool_of_o_abs bool_of_o_closed cadd_rel_0 cadd_rel_closed cardinal_rel_0_iff_0*

cardinal_rel_closed cardinal_rel_idem cartprod_abs cartprod_closed
cmult_rel_0 cmult_rel_1 cmult_rel_closed comp_closed composition_abs
cons_abs cons_closed converse_abs converse_closed csquare_lam_closed
csquare_rel_closed depth_closed domain_abs domain_closed eclose_abs
eclose_closed empty_abs field_abs field_closed finite_funspace_closed
finite_ordinal_abs fst_closed function_abs function_space_rel_closed
hd_abs image_abs image_closed inj_rel_closed injection_abs inter_abs
irreflexive_abs is_eclose_n_abs is_funspace_abs
iterates_closed length_closed lepoll_rel_refl
limit_ordinal_abs linear_rel_abs
mem_bij_abs mem_eclose_abs mem_inj_abs membership_abs
minimum_closed nat_case_abs nat_case_closed nonempty not_abs
not_closed number1_abs number2_abs number3_abs omega_abs
or_abs or_closed order_isomorphism_abs ordermap_closed
ordertype_closed ordinal_abs pair_abs pair_in_M_iff powerset_abs
pred_closed pred_set_abs quasilist_abs quasinat_abs radd_closed
rall_abs range_abs range_closed relation_abs restrict_closed
restriction_abs rex_abs rmult_closed rtrancl_abs rtrancl_closed
rvimage_closed separation_closed setdiff_abs singleton_abs
singleton_in_M_iff snd_closed strong_replacement_closed subset_abs
succ_in_M_iff successor_abs successor_ordinal_abs sum_abs sum_closed
surj_rel_closed surjection_abs tl_abs trancl_abs trancl_closed
transitive_rel_abs transitive_set_abs typed_function_abs union_abs
upair_abs upair_in_M_iff vimage_abs vimage_closed well_ord_abs
nth_closed Aleph_rel_closed csucc_rel_closed
Card_rel_Aleph_rel

declare *sharp_simps*[*simp del*, *simplified setclass_iff*, *simp*]

lemmas *sharp_intros* = *nat_into_M Aleph_rel_closed Card_rel_Aleph_rel*

declare *sharp_intros*[*rule del*, *simplified setclass_iff*, *intro*]

end — *M_ctm2_AC*

context *G_generic3_AC* **begin**

context
  includes *G_generic1_lemmas*
begin

lemmas *mg_sharp_simps* = *ext.Card_rel_Union ext.Card_rel_cardinal_rel*
ext.Collect_abs ext.Cons_abs ext.Cons_in_M_iff ext.Diff_closed
ext.Equal_abs ext.Equal_in_M_iff ext.Finite_abs ext.Forall_abs
ext.Forall_in_M_iff ext.Inl_abs ext.Inl_in_M_iff ext.Inr_abs
ext.Inr_in_M_iff ext.Int_closed ext.Inter_abs ext.Inter_closed
ext.M_nat ext.Member_abs ext.Member_in_M_iff ext.Memrel_closed
ext.Nand_abs ext.Nand_in_M_iff ext.Nil_abs ext.Nil_in_M
ext.Ord_cardinal_rel ext.Pow_rel_closed ext.Un_closed

ext.Union_abs ext.Union_closed ext.and_abs ext.and_closed
ext.apply_abs ext.apply_closed ext.bij_rel_closed
ext.bijection_abs ext.bool_of_o_abs ext.bool_of_o_closed
ext.cadd_rel_0 ext.cadd_rel_closed ext.cardinal_rel_0_iff_0
ext.cardinal_rel_closed ext.cardinal_rel_idem ext.cartprod_abs
ext.cartprod_closed ext.cmult_rel_0 ext.cmult_rel_1
ext.cmult_rel_closed ext.comp_closed ext.composition_abs
ext.cons_abs ext.cons_closed ext.converse_abs ext.converse_closed
ext.csquare_lam_closed ext.csquare_rel_closed ext.depth_closed
ext.domain_abs ext.domain_closed ext.eclose_abs ext.eclose_closed
ext.empty_abs ext.field_abs ext.field_closed
ext.finite_funspace_closed ext.finite_ordinal_abs
ext.fst_closed ext.function_abs ext.function_space_rel_closed
ext.hd_abs ext.image_abs ext.image_closed ext.inj_rel_closed
ext.injection_abs ext.inter_abs ext.irreflexive_abs
ext.is_eclose_n_abs ext.is_funspace_abs
ext.iterates_closed ext.length_closed
ext.lepoll_rel_refl ext.limit_ordinal_abs ext.linear_rel_abs
ext.mem_bij_abs ext.mem_eclose_abs
ext.mem_inj_abs ext.membership_abs
ext.nat_case_abs ext.nat_case_closed
ext.nonempty ext.not_abs ext.not_closed
ext.number1_abs ext.number2_abs ext.number3_abs ext.omega_abs
ext.or_abs ext.or_closed ext.order_isomorphism_abs
ext.ordermap_closed ext.ordertype_closed ext.ordinal_abs
ext.pair_abs ext.pair_in_M_iff ext.powerset_abs ext.pred_closed
ext.pred_set_abs ext.quasilist_abs ext.quasinat_abs
ext.radd_closed ext.rall_abs ext.range_abs ext.range_closed
ext.relation_abs ext.restrict_closed ext.restriction_abs
ext.rex_abs ext.rmult_closed ext.rtrancl_abs ext.rtrancl_closed
ext.rvimage_closed ext.separation_closed ext.setdiff_abs
ext.singleton_abs ext.singleton_in_M_iff ext.snd_closed
ext.strong_replacement_closed ext.subset_abs ext.succ_in_M_iff
ext.successor_abs ext.successor_ordinal_abs ext.sum_abs
ext.sum_closed ext.surj_rel_closed ext.surjection_abs ext.tl_abs
ext.trancl_abs ext.trancl_closed ext.transitive_rel_abs
ext.transitive_set_abs ext.typed_function_abs ext.union_abs
ext.upair_abs ext.upair_in_M_iff ext.vimage_abs ext.vimage_closed
ext.well_ord_abs ext.nth_closed ext.Aleph_rel_closed
ext.csucc_rel_closed ext.Card_rel_Aleph_rel

— The following was motivated by the fact that *ext.apply_closed* did not simplify appropriately.
declare *mg_sharp_simps*[*simp del*, *simplified setclass_iff*, *simp*]

lemmas *mg_sharp_intros* = *ext.nat_into_M ext.Aleph_rel_closed*
ext.Card_rel_Aleph_rel

declare *mg_sharp_intros*[*rule del*, *simplified setclass_iff*, *intro*]

— Kunen IV.2.31
lemma *forces_below_filter*:
assumes $M[G]$, $map(val(G),env) \models \varphi\ p \in G$
$arity(\varphi) \leq length(env)\ \varphi \in formula\ env \in list(M)$
shows $\exists\, q{\in}G.\ q \preceq p \wedge q \Vdash \varphi\ env$
proof -
note *assms*
moreover from *this*
obtain r **where** $r \Vdash \varphi\ env\ r{\in}G$
using *generic truth_lemma*[*of* φ *env*]
by *blast*
moreover from *this* **and** ‹$p{\in}G$›
obtain q **where** $q \preceq p\ q \preceq r\ q \in G$ **by** *auto*
ultimately
show *?thesis*
using *strengthening_lemma*[*of* $r\ \varphi$ _ *env*] **by** *blast*
qed

28.1 Preservation by ccc forcing notions

lemma *ccc_fun_closed_lemma_aux*:
assumes *f_dot*$\in M$ $p{\in}M$ $a{\in}M$ $b{\in}M$
shows $\{q \in \mathbb{P}\ .\ q \preceq p \wedge (M, [q, \mathbb{P}, leq, \mathbf{1}, f\_dot, a^v, b^v] \models forces(\cdot 0\text{`}1 \text{ is } 2\cdot\ ))\} \in M$
using *separation_forces*[**where** *env*=[*f_dot*, a^v, b^v] **and** φ=$\cdot 0\text{`}1 \text{ is } 2\cdot$,*simplified*]
assms G_subset_M[*THEN subsetD*] *generic*
separation_in lam_replacement_constant lam_replacement_identity
lam_replacement_product
separation_conj arity_fun_apply_fm union_abs1
by *simp_all*

lemma *ccc_fun_closed_lemma_aux2*:
assumes $B{\in}M$ *f_dot*$\in M$ $p{\in}M$ $a{\in}M$
shows $(\#\#M)(\lambda b{\in}B.\ \{q \in \mathbb{P}\ .\ q \preceq p \wedge (M, [q, \mathbb{P}, leq, \mathbf{1}, f\_dot, a^v, b^v] \models forces(\cdot 0\text{`}1 \text{ is } 2\cdot\ ))\})$
proof -
have $separation(\#\#M, \lambda z.\ M, [snd(z), \mathbb{P}, leq, \mathbf{1}, f\_dot, \tau, fst(z)^v] \models forces(\cdot 0\text{`}1 \text{ is } 2\cdot\ ))$
if $\tau{\in}M$ **for** τ
proof -
let *?f_fm*=*snd_fm*(1,0)
let *?g_fm*=*hcomp_fm*(*check_fm*(6),*fst_fm*,2,0)
note *assms*
moreover
have $arity(forces(\cdot 0\text{`}1 \text{ is } 2\cdot\ )) \leq 7$
using *arity_fun_apply_fm union_abs1 arity_forces*[*of* $\cdot 0\text{`}1 \text{ is } 2\cdot$]
by *simp*
moreover

```
    have ?f_fm ∈ formula arity(?f_fm) ≤ 7 ?g_fm ∈ formula arity(?g_fm) ≤ 8
      using ord_simp_union
      unfolding hcomp_fm_def
      by (simp_all add:arity)
    ultimately
    show ?thesis
      using separation_sat_after_function assms that sats_fst_fm
        snd_abs sats_snd_fm sats_check_fm check_abs fst_abs
      unfolding hcomp_fm_def
      by simp
  qed
  with assms
  show ?thesis
    using lam_replacement_imp_lam_closed separation_conj separation_in
      lam_replacement_product lam_replacement_constant transitivity[of _ B]
     lam_replacement_snd lam_replacement_Collect' ccc_fun_closed_lemma_aux
    by simp
qed

lemma ccc_fun_closed_lemma:
  assumes A∈M B∈M f_dot∈M p∈M
  shows (λa∈A. {b∈B. ∃ q∈ℙ. q ≼ p ∧ (q ⊩ ·0‘1 is 2· [f_dot, a^v, b^v])}) ∈ M
proof -
  have separation(##M, λz. M, [snd(z), ℙ, leq, 1, f_dot, fst(fst(z))^v, snd(fst(z))^v]
⊨ forces(·0‘1 is 2· ))
  proof -
    let ?f_fm=snd_fm(1,0)
    let ?g=λz . fst(fst(fst(z)))^v
    let ?g_fm=hcomp_fm(check_fm(6),hcomp_fm(fst_fm,fst_fm),2,0)
    let ?h_fm=hcomp_fm(check_fm(7),hcomp_fm(snd_fm,fst_fm),3,0)
    note assms
    moreover
    have arity(forces(·0‘1 is 2· )) ≤ 7
      using arity_fun_apply_fm union_abs1 arity_forces[of ·0‘1 is 2· ]
      by simp
    moreover
    have ?f_fm ∈ formula arity(?f_fm) ≤ 6 ?g_fm ∈ formula arity(?g_fm) ≤ 7
      ?h_fm ∈ formula arity(?h_fm) ≤ 8
      using ord_simp_union
      unfolding hcomp_fm_def
      by (simp_all add:arity)
    ultimately
    show ?thesis
      using separation_sat_after_function3 assms sats_check_fm check_abs
        fst_abs snd_abs
      unfolding hcomp_fm_def
      by simp
  qed
  moreover
```

have *1*:*separation*(*##M*, *λz. M*, [*snd*(*z*), ℙ, *leq*, **1**, *f_dot*, *τ*, *fst*(*z*)^v] ⊨ *forces*(*·0‘1 is 2·*))
if *τ*∈*M* **for** *τ*
proof -
let *?f_fm*=*snd_fm*(*1*,*0*)
let *?g_fm*=*hcomp_fm*(*check_fm*(*6*),*fst_fm*,*2*,*0*)
note *assms*
moreover
have *arity*(*forces*(*·0‘1 is 2·*)) ≤ *7*
using *arity_forces*[*of ·0‘1 is 2·*] *arity_fun_apply_fm union_abs1*
by *simp*
moreover
have *?f_fm* ∈ *formula arity*(*?f_fm*) ≤ *7 ?g_fm* ∈ *formula arity*(*?g_fm*) ≤ *8*
using *ord_simp_union*
unfolding *hcomp_fm_def*
by (*simp_all add*:*arity*)
ultimately
show *?thesis*
using *separation_sat_after_function that fst_abs snd_abs sats_check_fm check_abs*
unfolding *hcomp_fm_def*
by *simp*
qed
moreover note *assms*
ultimately
show *?thesis*
using *lam_replacement_imp_lam_closed lam_replacement_Collect′ transitivity*[*of _ A*]
lam_replacement_constant lam_replacement_identity lam_replacement_snd lam_replacement_product separation_conj separation_in separation_bex separation_iff′
by *simp*
qed

— Kunen IV.3.5
lemma *ccc_fun_approximation_lemma*:
notes *le_trans*[*trans*]
assumes *ccc*^{M}(ℙ,*leq*) *A*∈*M B*∈*M f*∈*M*[*G*] *f* : *A* → *B*
shows
∃ *F*∈*M*. *F* : *A* → *Pow*^{M}(*B*) ∧ (∀ *a*∈*A*. *f‘a* ∈ *F‘a* ∧ |*F‘a*|^M ≤ *ω*)
proof -
from ‹*f*∈*M*[*G*]›
obtain *f_dot* **where** *f* = *val*(*G*,*f_dot*) *f_dot*∈*M* **using** *GenExtD* **by** *force*
with *assms*
obtain *p* **where** *p* ⊩ *·0*:*1*→*2·* [*f_dot*, *A*^{v}, *B*^{v}] *p*∈*G p*∈*M*
using *G_subset_M truth_lemma*[*of ·0*:*1*→*2·* [*f_dot*, *A*^{v}, *B*^{v}]]
by (*auto simp add*:*ord_simp_union arity_typed_function_fm*
— NOTE: type-checking is not performed here by the Simplifier
typed_function_type)

define F **where** $F{\equiv}\lambda a{\in}A.\ \{b{\in}B.\ \exists\, q{\in}\mathbb{P}.\ q \preceq p \wedge (q \Vdash \cdot 0`1 \text{ is } 2{\cdot}\ [f\_dot,\ a^v,\ b^v])\}$
from *assms* ‹$f\_dot{\in}\_$› ‹$p{\in}M$›
have $F \in M$
unfolding *F_def* **using** *ccc_fun_closed_lemma* **by** *simp*
moreover from *calculation*
have $f`a \in F`a$ **if** $a \in A$ **for** a
proof -
note ‹$f\colon A \to B$› ‹$a \in A$›
moreover from *this*
have $f\ `\ a \in B$ **by** *simp*
moreover
note ‹$f{\in}M[G]$› ‹$A{\in}M$›
moreover from *calculation*
have $M[G],\ [f,\ a,\ f`a] \models \cdot 0`1 \text{ is } 2{\cdot}$
by (*auto dest*:*transitivity*)
moreover
note ‹$B{\in}M$› ‹$f = val(G,f\_dot)$›
moreover from *calculation*
have $a{\in}M\ val(G,\ f\_dot)`a{\in}M$
by (*auto dest*:*transitivity*)
moreover
note ‹$f\_dot{\in}M$› ‹$p{\in}G$›
ultimately
obtain q **where** $q \preceq p\ q \Vdash \cdot 0`1 \text{ is } 2{\cdot}\ [f\_dot,\ a^v,\ (f`a)^v]\ q{\in}G$
using *forces_below_filter*[*of* $\cdot 0`1 \text{ is } 2{\cdot}\ [f\_dot,\ a^v,\ (f`a)^v]\ p$]
by (*auto simp add*: *ord_simp_union arity_fun_apply_fm*
fun_apply_type)
with ‹$f`a \in B$›
have $f`a \in \{b{\in}B\ .\ \exists\, q{\in}\mathbb{P}.\ q \preceq p \wedge q \Vdash \cdot 0`1 \text{ is } 2{\cdot}\ [f\_dot,\ a^v,\ b^v]\}$
by *blast*
with ‹$a{\in}A$›
show *?thesis* **unfolding** *F_def* **by** *simp*
qed
moreover
have $|F`a|^M \leq \omega \wedge F`a{\in}M$ **if** $a \in A$ **for** a
proof -
let *?Q*=$\lambda b.\ \{q{\in}\mathbb{P}.\ q \preceq p \wedge (q \Vdash \cdot 0`1 \text{ is } 2{\cdot}\ [f\_dot,\ a^v,\ b^v])\}$
from ‹$F \in M$› ‹$a{\in}A$› ‹$A{\in}M$›
have $F`a \in M\ a{\in}M$
using *transitivity*[*OF* _ ‹$A{\in}M$›] **by** *simp_all*
moreover
have *2*:$\bigwedge x.\ x{\in}F`a \Longrightarrow x{\in}M$
using *transitivity*[*OF* _ ‹$F`a{\in}M$›] **by** *simp*
moreover
have *3*:$\bigwedge x.\ x{\in}F`a \Longrightarrow (\#\#M)(?Q(x))$
using *ccc_fun_closed_lemma_aux*[*OF* ‹$f\_dot{\in}M$› ‹$p{\in}M$› ‹$a{\in}M$› *2*] *transitivity*[*of* _ $F`a$]
by *simp*
moreover

have *4:lam_replacement(##M,λb. {q ∈ ℙ . q ≼ p ∧ (M, [q, ℙ, leq, **1**, f_dot, a^v, b^v] ⊨ forces(·0‘1 is 2·))})*
using *ccc_fun_closed_lemma_aux2[OF _ ‹f_dot∈M› ‹p∈M› ‹a∈M›]*
lam_replacement_iff_lam_closed[THEN iffD2]
ccc_fun_closed_lemma_aux[OF ‹f_dot∈M› ‹p∈M› ‹a∈M›]
by *simp*
ultimately
interpret *M_Pi_assumptions_choice ##M F‘a ?Q*
using *Pi_replacement1[OF _ 3] lam_replacement_Sigfun[OF 4]*
lam_replacement_imp_strong_replacement
ccc_fun_closed_lemma_aux[OF ‹f_dot∈M› ‹p∈M› ‹a∈M›]
lam_replacement_hcomp2[OF lam_replacement_constant 4 _ _
lam_replacement_minimum,unfolded lam_replacement_def]
by *unfold_locales simp_all*
from *‹F‘a ∈ M›*
interpret *M_Pi_assumptions2 ##M F‘a ?Q λ_ . ℙ*
using *lam_replacement_imp_strong_replacement[OF*
lam_replacement_Sigfun[OF lam_replacement_constant]]
Pi_replacement1 transitivity[of _ F‘a]
by *unfold_locales simp_all*
from *‹p ⊩ ·0:1→2· [f_dot, A^v, B^v]› ‹a∈A›*
have *∃ y. y ∈ ?Q(b)* **if** *b ∈ F‘a* **for** *b*
using *that* **unfolding** *F_def* **by** *auto*
then
obtain *q* **where** *q ∈ Pi^M(F‘a,?Q) q∈M* **using** *AC_Pi_rel* **by** *auto*
moreover
note *‹F‘a ∈ M›*
moreover from *calculation*
have *q : F‘a →^M ℙ*
using *Pi_rel_weaken_type def_function_space_rel* **by** *auto*
moreover from *calculation*
have *q : F‘a → range(q) q : F‘a → ℙ q : F‘a →^M range(q)*
using *mem_function_space_rel_abs range_of_fun* **by** *simp_all*
moreover
have *q‘b ⊥ q‘c* **if** *b ∈ F‘a c ∈ F‘a b ≠ c*
— For the next step, if the premise *b ≠ c* is first, the proof breaks down badly
for *b c*
proof -
from *‹b ∈ F‘a› ‹c ∈ F‘a› ‹q ∈ Pi^M(F‘a,?Q)› ‹q∈M›*
have *q‘b ⊩ ·0‘1 is 2· [f_dot, a^v, b^v]*
q‘c ⊩ ·0‘1 is 2· [f_dot, a^v, c^v]
using *mem_Pi_rel_abs[of q] apply_type[of _ _ ?Q]*
by *simp_all*
with *‹b ≠ c› ‹q : F‘a → ℙ› ‹a∈A› ‹b∈_› ‹c∈_›*
‹A∈M› ‹f_dot∈M› ‹F‘a∈M›
show *?thesis*
using *forces_neq_apply_imp_incompatible*
transitivity[of _ A] transitivity[of _ F‘a]
by *auto*

qed
moreover from *calculation*
have *antichain*($\mathbb{P}$,*leq*,*range*(q))
using *Pi_range_eq*[*of* _ _ λ_ . $\mathbb{P}$]
unfolding *antichain_def compat_in_def* **by** *auto*
moreover from *this* **and** ‹q∈M›
have *antichain*^{M}($\mathbb{P}$,*leq*,*range*(q))
by (*simp add*:*absolut del*:*P_in_M*)
moreover from *calculation*
have q‘$b \neq q$‘c **if** $b \neq c$ $b \in F$‘a $c \in F$‘a **for** b c
using *that Incompatible_imp_not_eq apply_type*
mem_function_space_rel_abs **by** *simp*
ultimately
have $q \in inj^M(F‘a,range(q))$
using *def_inj_rel* **by** *auto*
with ‹F‘$a \in M$› ‹q∈M›
have $|F‘a|^M \leq |range(q)|^M$
using *def_lepoll_rel*
by (*rule_tac lepoll_rel_imp_cardinal_rel_le*) *auto*
also from ‹*antichain*^{M}($\mathbb{P}$,*leq*,*range*(q))› ‹*ccc*^{M}($\mathbb{P}$,*leq*)› ‹q∈M›
have $|range(q)|^M \leq \omega$
using *def_ccc_rel* **by** *simp*
finally
show *?thesis* **using** ‹F‘a∈M› **by** *auto*
qed
moreover from *this*
have F‘a∈M **if** a∈A **for** a
using *that* **by** *simp*
moreover from *this* ‹B∈M›
have $F : A \to Pow^M(B)$
using *Pow_rel_char*
unfolding *F_def* **by** (*rule_tac lam_type*) *auto*
ultimately
show *?thesis* **by** *auto*
qed

end — G_generic1_lemmas bundle

end — *G_generic3_AC*

end

29 Model of the negation of the Continuum Hypothesis

theory *Not_CH*
imports
Cardinal_Preservation

begin

We are taking advantage that the poset of finite functions is absolute, and thus we work with the unrelativized *Fn*. But it would have been more appropriate to do the following using the relative *Fn_rel*. As it turns out, the present theory was developed prior to having *Fn* relativized!

We also note that $Fn(\omega,\ \kappa \times \omega,\ 2)$ is separative, i.e. each $X \in Fn(\omega,\ \kappa \times \omega,\ 2)$ has two incompatible extensions; therefore we may recover part of our previous theorem *extensions_of_ctms_ZF*. But that result also included the possibility of not having AC in the ground model, which would not be sensible in a context where the cardinality of the continuum is under discussion. It is also the case that *extensions_of_ctms_ZF* was historically our first formalized result (with a different proof) that showed the forcing machinery had all of its elements in place.

abbreviation
  $Add\_subs :: i \Rightarrow i$ **where**
  $Add\_subs(\kappa) \equiv Fn(\omega,\kappa\times\omega,2)$

abbreviation
  $Add\_le :: i \Rightarrow i$ **where**
  $Add\_le(\kappa) \equiv Fnle(\omega,\kappa \times \omega,2)$

lemma (**in** *M_aleph*) *Aleph_rel2_closed*[*intro*,*simp*]: $M(\aleph_2{}^M)$
  using *nat_into_Ord* **by** *simp*

locale *M_master* = *M_cohen* + *M_library* +
  assumes
    UN_lepoll_assumptions:
    $M(A) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow M(A') \Longrightarrow separation(M,\ \lambda y.\ \exists\, x{\in}A'.\ y = \langle x,\ \mu\ i.\ x{\in}if\_range\_F\_else\_F((`)(A),\ b,\ f,\ i)\rangle)$

29.1 Non-absolute concepts between extensions

sublocale *M_master* $\subseteq$ *M_Pi_replacement*
  by *unfold_locales*

locale *M_master_sub* = *M_master* + *N*:*M_aleph N* **for** *N* +
  assumes
    M_imp_N: $M(x) \Longrightarrow N(x)$ **and**
    Ord_iff: $Ord(x) \Longrightarrow M(x) \longleftrightarrow N(x)$

sublocale *M_master_sub* $\subseteq$ *M_N_Perm*
  using *M_imp_N* **by** *unfold_locales*

context *M_master_sub*
begin

lemma *cardinal_rel_le_cardinal_rel*: $M(X) \Longrightarrow |X|^N \leq |X|^M$

using *M_imp_N N.lepoll_rel_cardinal_rel_le*[*OF lepoll_rel_transfer Card_rel_is_Ord*]
cardinal_rel_eqpoll_rel[*THEN eqpoll_rel_sym*, *THEN eqpoll_rel_imp_lepoll_rel*]
by *simp*

lemma *Aleph_rel_sub_closed*: $Ord(\alpha) \Longrightarrow M(\alpha) \Longrightarrow N(\aleph_\alpha{}^M)$
using *Ord_iff*[*THEN iffD1*, *OF Card_rel_Aleph_rel*[*THEN Card_rel_is_Ord*]]
by *simp*

lemma *Card_rel_imp_Card_rel*: $Card^N(\kappa) \Longrightarrow M(\kappa) \Longrightarrow Card^M(\kappa)$
using *N.Card_rel_is_Ord*[*of* κ] *M_imp_N Ord_cardinal_rel_le*[*of* κ]
cardinal_rel_le_cardinal_rel[*of* κ] *le_anti_sym*
unfolding *Card_rel_def* **by** *auto*

lemma *csucc_rel_le_csucc_rel*:
assumes $Ord(\kappa)$ $M(\kappa)$
shows $(\kappa^+)^M \leq (\kappa^+)^N$
proof -
note *assms*
moreover from *this*
have $N(L) \wedge Card^N(L) \wedge \kappa < L \Longrightarrow M(L) \wedge Card^M(L) \wedge \kappa < L$
(**is** *?P*(*L*) $\Longrightarrow$ *?Q*(*L*)) **for** *L*
using *M_imp_N Ord_iff*[*THEN iffD2*, *of L*] *N.Card_rel_is_Ord lt_Ord*
Card_rel_imp_Card_rel **by** *auto*
moreover from *assms*
have $N((\kappa^+)^N)$ $Card^N((\kappa^+)^N)$ $\kappa < (\kappa^+)^N$
using *N.lt_csucc_rel*[*of* κ] *N.Card_rel_csucc_rel*[*of* κ] *M_imp_N* **by** *simp_all*
ultimately
show *?thesis*
using *M_imp_N Least_antitone*[*of _ ?P ?Q*] **unfolding** *csucc_rel_def* **by**
blast
qed

lemma *Aleph_rel_le_Aleph_rel*: $Ord(\alpha) \Longrightarrow M(\alpha) \Longrightarrow \aleph_\alpha{}^M \leq \aleph_\alpha{}^N$
proof (*induct rule:trans_induct3*)
case *0*
then
show *?case*
using *Aleph_rel_zero N.Aleph_rel_zero* **by** *simp*
next
case (*succ x*)
then
have $\aleph_x{}^M \leq \aleph_x{}^N$ $Ord(x)$ $M(x)$ **by** *simp_all*
moreover from *this*
have $(\aleph_x{}^{M+})^M \leq (\aleph_x{}^{N+})^M$
using *M_imp_N Ord_iff*[*THEN iffD2*, *OF N.Card_rel_is_Ord*]
by (*intro csucc_rel_le_mono*) *simp_all*
moreover from *calculation*
have $(\aleph_x{}^{N+})^M \leq (\aleph_x{}^{N+})^N$
using *M_imp_N N.Card_rel_is_Ord Ord_iff*[*THEN iffD2*, *OF N.Card_rel_is_Ord*]

by (*intro csucc_rel_le_csucc_rel*) *auto*
ultimately
show *?case*
using *M_imp_N Aleph_rel_succ N.Aleph_rel_succ csucc_rel_le_csucc_rel le_trans* **by** *auto*
next
case (*limit x*)
then
show *?case*
using *M_imp_N Aleph_rel_limit N.Aleph_rel_limit*
by *simp* (*blast dest*: *transM intro*!:*le_implies_UN_le_UN*)
qed

end — *M_master_sub*

lemmas (**in** *M_ZF2_trans*) *sep_instances* =
separation_ifrangeF_body separation_ifrangeF_body2 separation_ifrangeF_body3
separation_ifrangeF_body4 separation_ifrangeF_body5 separation_ifrangeF_body6
separation_ifrangeF_body7 separation_cardinal_rel_lesspoll_rel
separation_is_dcwit_body separation_cdltgamma separation_cdeqgamma

lemmas (**in** *M_ZF2_trans*) *repl_instances* = *lam_replacement_inj_rel*

sublocale *M_ZFC2_ground_notCH_trans* ⊆ *M_master* ##*M*
using *replacement_trans_apply_image*
by *unfold_locales* (*simp_all add*:*repl_instances sep_instances del*:*setclass_iff add*: *transrec_replacement_def wfrec_replacement_def*)

sublocale *M_ZFC2_trans* ⊆ *M_Pi_replacement* ##*M*
by *unfold_locales*

29.2 Cohen forcing is ccc

context *M_ctm2_AC*
begin

lemma *ccc_Add_subs_Aleph_2*: $ccc^M(Add\_subs(\aleph_2{}^M), Add\_le(\aleph_2{}^M))$
proof -
interpret *M_add_reals* ##*M* $\aleph_2{}^M \times \omega$
by *unfold_locales blast*
show *?thesis*
using *ccc_rel_Fn_nat* **by** *fast*
qed

end — *M_ctm2_AC*

sublocale *G_generic3_AC* ⊆ *M_master_sub* ##*M* ##(*M*[*G*])
using *M_subset_MG*[*OF one_in_G*] *generic Ord_MG_iff*
by *unfold_locales auto*

lemma (**in** *M_trans*) *mem_F_bound4*:
fixes *F A*
defines $F \equiv (`)$
shows $x{\in}F(A,c) \Longrightarrow c \in (range(f) \cup domain(A))$
using *apply_0* **unfolding** *F_def*
by (*cases* $M(c)$, *auto simp*:*F_def*)

lemma (**in** *M_trans*) *mem_F_bound5*:
fixes *F A*
defines $F \equiv \lambda\_\ x.\ A`x$
shows $x{\in}F(A,c) \Longrightarrow c \in (range(f) \cup domain(A))$
using *apply_0* **unfolding** *F_def*
by (*cases* $M(c)$, *auto simp*:*F_def drSR_Y_def dC_F_def*)

sublocale $M\_ctm2\_AC \subseteq M\_replacement\_lepoll\ \#\#M\ (`)$
using *UN_lepoll_assumptions lam_replacement_apply lam_replacement_inj_rel*
mem_F_bound4 apply_0 lam_replacement_minimum
unfolding *lepoll_assumptions_defs*
proof (*unfold_locales*,
rule_tac [*3*] *lam_Least_assumption_general*[**where** *U*=*domain*, *OF _ mem_F_bound4*],
simp_all)
fix *A i x*
assume $A \in M\ x \in M\ x \in A\ `\ i$
then
show $i \in M$
using *apply_0*[*of i A*] *transM*[*of _ domain*(*A*), *simplified*]
by *force*
qed

context *G_generic3_AC* **begin**

context
includes *G_generic1_lemmas*
begin

lemma *G_in_MG*: $G \in M[G]$
using *G_in_Gen_Ext*
by *blast*

lemma *ccc_preserves_Aleph_succ*:
assumes $ccc^M(\mathbb{P},leq)$ $Ord(z)$ $z \in M$
shows $Card^{M[G]}({\aleph_{succ(z)}}^M)$
proof (*rule ccontr*)
assume $\neg\ Card^{M[G]}({\aleph_{succ(z)}}^M)$
moreover
note $\langle z \in M\rangle$ $\langle Ord(z)\rangle$
moreover from *this*
have $Ord({\aleph_{succ(z)}}^M)$

using *Card_rel_is_Ord* **by** *fastforce*
ultimately
obtain α f **where** $\alpha < \aleph_{succ(z)}{}^M$ $f \in surj^{M[G]}(\alpha, \aleph_{succ(z)}{}^M)$
using *ext.lt_surj_rel_empty_imp_Card_rel M_subset_MG*[*OF one_in_G*]
by *force*
moreover from *this* **and** ‹$z{\in}M$› ‹$Ord(z)$›
have $\alpha \in M$ $f \in M[G]$
using *ext.trans_surj_rel_closed*
by (*auto dest*:*transM ext.transM dest*!:*ltD*)
moreover
note ‹$ccc^M(\mathbb{P}$,*leq*)› ‹$z{\in}M$›
ultimately
obtain F **where** $F{:}\alpha{\to}Pow^M(\aleph_{succ(z)}{}^M)$ $\forall\,\beta{\in}\alpha.\ f`\beta \in F`\beta$ $\forall\,\beta{\in}\alpha.\ |F`\beta|^M \le \omega$ $F \in M$
using *ccc_fun_approximation_lemma*[*of* α $\aleph_{succ(z)}{}^M$ f]
ext.mem_surj_abs[*of* f α $\aleph_{succ(z)}{}^M$] ‹$Ord(z)$›
surj_is_fun[*of* f α $\aleph_{succ(z)}{}^M$] **by** *auto*
then
have $\beta \in \alpha \Longrightarrow |F`\beta|^M \le \aleph_0{}^M$ **for** β
using *Aleph_rel_zero* **by** *simp*
have $w \in F\ `\ x \Longrightarrow x \in M$ **for** w x
proof -
fix w x
assume $w \in F`x$
then
have $x \in domain(F)$
using *apply_0* **by** *auto*
with ‹$F{:}\alpha{\to}Pow^M(\aleph_{succ(z)}{}^M)$› ‹$\alpha \in M$›
show $x \in M$ **using** *domain_of_fun*
by (*auto dest*:*transM*)
qed
with ‹$\alpha \in M$› ‹$F{:}\alpha{\to}Pow^M(\aleph_{succ(z)}{}^M)$› ‹$F{\in}M$›
interpret *M_cardinal_UN_lepoll* ##*M* $\lambda\beta.\ F`\beta\ \alpha$
using *UN_lepoll_assumptions lepoll_assumptions*
lam_replacement_apply lam_replacement_inj_rel lam_replacement_minimum
proof (*unfold_locales*, *auto dest*:*transM simp del*:*if_range_F_else_F_def*)
fix f b
assume $b{\in}M$ $f{\in}M$
with ‹$F{\in}M$›
show *lam_replacement*(##*M*, $\lambda x.\ \mu\ i.\ x \in$ *if_range_F_else_F*((`)(*F*), *b*, *f*, *i*))
using *UN_lepoll_assumptions mem_F_bound5*
by (*rule_tac lam_Least_assumption_general*[**where** *U*=*domain*, *OF* _ *mem_F_bound5*])
simp_all
qed
from ‹$\alpha < \aleph_{succ(z)}{}^M$› ‹$\alpha \in M$› ‹$Ord(z)$› ‹$z{\in}M$›
have $\alpha \lesssim^M \aleph_z{}^M$

using
cardinal_rel_lt_csucc_rel_iff[*of* $\aleph_z{}^M$ α]
le_Card_rel_iff[*of* $\aleph_z{}^M$ α]
Aleph_rel_succ[*of z*] *Card_rel_lt_iff*[*of* α $\aleph_{succ(z)}{}^M$]
lt_Ord[*of* α $\aleph_{succ(z)}{}^M$]
Card_rel_csucc_rel[*of* $\aleph_z{}^M$]
Card_rel_Aleph_rel[*THEN Card_rel_is_Ord*]
by *simp*
with ‹$\alpha < \aleph_{succ(z)}{}^M$› ‹$\forall\beta{\in}\alpha.\ |F`\beta|^M \le \omega$› ‹$\alpha \in M$› *assms*
have $|\bigcup\beta{\in}\alpha.\ F`\beta|^M \le \aleph_z{}^M$
using *InfCard_rel_Aleph_rel*[*of z*] *Aleph_rel_zero*
subset_imp_lepoll_rel[*THEN lepoll_rel_imp_cardinal_rel_le*,
of $\bigcup\beta{\in}\alpha.\ F`\beta$ $\aleph_z{}^M$] *Aleph_rel_succ*
Aleph_rel_increasing[*THEN leI*, *THEN* [*2*] *le_trans*, *of _ 0 z*]
Ord_0_lt_iff[*THEN iffD1*, *of z*]
by (*cases 0<z*; *rule_tac lepoll_rel_imp_cardinal_rel_UN_le*) (*auto*, *force*)
moreover
note ‹$z{\in}M$› ‹$Ord(z)$›
moreover from ‹$\forall\beta{\in}\alpha.\ f`\beta \in F`\beta$› ‹$f \in surj^{M[G]}(\alpha, \aleph_{succ(z)}{}^M)$›
‹$\alpha \in M$› ‹$f \in M[G]$› **and** *this*
have $\aleph_{succ(z)}{}^M \subseteq (\bigcup\beta{\in}\alpha.\ F`\beta)$
using *ext.mem_surj_abs* **by** (*force simp add:surj_def*)
moreover from ‹$F \in M$› ‹$\alpha \in M$›
have $(\bigcup x{\in}\alpha.\ F\ `\ x) \in M$
using *j.B_replacement*
by (*intro Union_closed*[*simplified*] *RepFun_closed*[*simplified*])
(*auto dest:transM*)
ultimately
have $\aleph_{succ(z)}{}^M \le \aleph_z{}^M$
using *subset_imp_le_cardinal_rel*[*of* $\aleph_{succ(z)}{}^M$ $\bigcup\beta{\in}\alpha.\ F`\beta$]
le_trans **by** *auto*
with *assms*
show *False*
using *Aleph_rel_increasing not_le_iff_lt*[*of* $\aleph_{succ(z)}{}^M$ $\aleph_z{}^M$]
Card_rel_Aleph_rel[*THEN Card_rel_is_Ord*]
by *auto*
qed

end — bundle G_generic1_lemmas

end — *G_generic3_AC*

context *M_ctm1*
begin

abbreviation
Add :: *i* **where**
Add ≡ *Fn*(ω, $\aleph_2{}^M \times \omega$, *2*)

end — *M_ctm1*

locale *add_generic3* = *G_generic3_AC Fn*(ω, $\aleph_2^{\#\#M} \times \omega$, *2*) *Fnle*($\omega$, $\aleph_2^{\#\#M} \times \omega$, *2*) *0*

sublocale *add_generic3* $\subseteq$ *cohen_data* ω $\aleph_2{}^M \times \omega$ *2* **by** *unfold_locales auto*

context *add_generic3*
begin

notation *Leq* (**infixl** ‹$\preceq$› *50*)
notation *Incompatible* (**infixl** ‹$\perp$› *50*)

lemma *Add_subs_preserves_Aleph_succ*: *Ord*(z) $\Longrightarrow$ $z{\in}M$ $\Longrightarrow$ *Card*$^{M[G]}$($\aleph_{succ(z)}{}^M$)
  using *ccc_preserves_Aleph_succ ccc_Add_subs_Aleph_2*
  by *auto*

lemma *Aleph_rel_nats_MG_eq_Aleph_rel_nats_M*:
  includes *G_generic1_lemmas*
  assumes $z \in \omega$
  shows $\aleph_z{}^{M[G]} = \aleph_z{}^M$
  using *assms*
proof (*induct*)
  case *0*
  show *?case*
    by(*rule trans*[*OF ext.Aleph_rel_zero Aleph_rel_zero*[*symmetric*]])
next
  case (*succ z*)
  then
  have $\aleph_{succ(z)}{}^M \leq \aleph_{succ(z)}{}^{M[G]}$
    using *Aleph_rel_le_Aleph_rel nat_into_M* **by** *simp*
  moreover from ‹$z \in \omega$›
  have $\aleph_z{}^M \in M[G]$ $\aleph_{succ(z)}{}^M \in M[G]$
    using *nat_into_M* **by** *simp_all*
  moreover from *this* **and** ‹$\aleph_z{}^{M[G]} = \aleph_z{}^M$› ‹$z \in \omega$›
  have $\aleph_{succ(z)}{}^{M[G]} \leq \aleph_{succ(z)}{}^M$
    using *ext.Aleph_rel_succ nat_into_M*
      Add_subs_preserves_Aleph_succ[*THEN ext.csucc_rel_le*, *of z*]
      Aleph_rel_increasing[*of z succ*(z)]
    by *simp*
  ultimately
  show *?case* **using** *le_anti_sym* **by** *blast*
qed

abbreviation
  f_G :: *i* (‹f_G›) **where**
  $f_G \equiv \bigcup G$

abbreviation
dom_dense :: $i \Rightarrow i$ **where**
$dom\_dense(x) \equiv \{p \in Add\ .\ x \in domain(p)\ \}$

declare (**in** *M_ctm2_AC*) *Fn_nat_closed*[*simplified setclass_iff*, *simp*, *intro*]
declare (**in** *M_ctm2_AC*) *Fnle_nat_closed*[*simp del*, *rule del*,
simplified setclass_iff, *simp*, *intro*]
declare (**in** *M_ctm2_AC*) *cexp_rel_closed*[*simplified setclass_iff*, *simp*, *intro*]
declare (**in** *G_generic3_AC*) *ext.cexp_rel_closed*[*simplified setclass_iff*, *simp*, *intro*]

lemma *dom_dense_closed*[*intro*,*simp*]: $x \in \aleph_2{}^M \times \omega \Longrightarrow dom\_dense(x) \in M$
using *separation_in_domain*[*of x*] *nat_into_M*
by (*rule_tac separation_closed*[*simplified*], *blast dest*:*transM*) *simp*

lemma *domain_f_G*: **assumes** $x \in \aleph_2{}^M$ $y \in \omega$
shows $\langle x,\ y\rangle \in domain(f_G)$
proof -
from *assms*
have $Add = Fn^M(\omega,\aleph_2{}^M\times\omega,2)$
using *Fn_nat_abs* **by** *auto*
moreover from *this*
have $Fnle(\omega,\aleph_2{}^M\times\omega,2) = Fnle^M(\omega,\aleph_2{}^M\times\omega,2)$
unfolding *Fnle_rel_def Fnle_def* **by** *auto*
moreover from *calculation assms*
have $dense(dom\_dense(\langle x,\ y\rangle))$
using *dense_dom_dense*[*of* $\langle x,y\rangle$ $\aleph_2{}^M\times\omega$ ω *2*] *InfCard_rel_nat*
unfolding *dense_def* **by** *auto*
with *assms*
obtain *p* **where** $p\in dom\_dense(\langle x,\ y\rangle)$ $p\in G$
using *M_generic_denseD*[*of* $dom\_dense(\langle x,\ y\rangle)$]
by *auto*
then
show $\langle x,\ y\rangle \in domain(f_G)$ **by** *blast*
qed

lemma *f_G_funtype*:
includes *G_generic1_lemmas*
shows $f_G : \aleph_2{}^M \times \omega \to 2$
using *generic domain_f_G Pi_iff Un_filter_is_function generic*
subset_trans[*OF filter_subset_notion Fn_nat_subset_Pow*]
by *force*

lemma *inj_dense_closed*[*intro*,*simp*]:
$w \in \aleph_2{}^M \Longrightarrow x \in \aleph_2{}^M \Longrightarrow inj\_dense(\aleph_2{}^M,2,w,x) \in M$
using *transM*[*OF _ Aleph_rel2_closed*] *separation_conj separation_bex*
lam_replacement_product
separation_in lam_replacement_fst lam_replacement_snd lam_replacement_constant
lam_replacement_hcomp[*OF lam_replacement_snd lam_replacement_restrict'*]

separation_bex separation_conj
by *simp*

lemma *Aleph_rel2_new_reals*:
assumes $w \in \aleph_2^M$ $x \in \aleph_2^M$ $w \neq x$
shows $(\lambda n{\in}\omega.\ f_G\ `\ \langle w,\ n\rangle) \neq (\lambda n{\in}\omega.\ f_G\ `\ \langle x,\ n\rangle)$
proof -
have *0∈2* **by** *auto*
with *assms*
have *dense(inj_dense(*$\aleph_2^M$*,2,w,x))*
unfolding *dense_def* **using** *dense_inj_dense* **by** *auto*
with *assms*
obtain *p* **where** *p∈inj_dense(*$\aleph_2^M$*,2,w,x)* $p{\in}G$
using *M_generic_denseD[of inj_dense(*$\aleph_2^M$*,2,w,x)]*
by *blast*
then
obtain *n* **where** $n \in \omega\ \langle\langle w,\ n\rangle,\ 1\rangle \in p\ \langle\langle x,\ n\rangle,\ 0\rangle \in p$
by *blast*
moreover from *this* **and** ‹$p{\in}G$›
have $\langle\langle w,\ n\rangle,\ 1\rangle \in f_G\ \langle\langle x,\ n\rangle,\ 0\rangle \in f_G$ **by** *auto*
moreover from *calculation*
have $f_G\ `\ \langle w,\ n\rangle = 1\ f_G\ `\ \langle x,\ n\rangle = 0$
using *f_G_funtype apply_equality*
by *auto*
ultimately
have $(\lambda n{\in}\omega.\ f_G\ `\ \langle w,\ n\rangle)\ `\ n \neq (\lambda n{\in}\omega.\ f_G\ `\ \langle x,\ n\rangle)\ `\ n$
by *simp*
then
show *?thesis* **by** *fastforce*
qed

definition
$h\_G :: i\ (\langle h_G\rangle)$ **where**
$h_G \equiv \lambda\alpha{\in}\aleph_2^M.\ \lambda n{\in}\omega.\ f_G`\langle\alpha,n\rangle$

lemma *h_G_in_MG[simp]*:
includes *G_generic1_lemmas*
shows $h_G \in M[G]$
using *ext.curry_closed[unfolded curry_def] G_in_MG*
unfolding *h_G_def*
by *simp*

lemma *h_G_inj_Aleph_rel2_reals*: $h_G \in inj^{M[G]}(\aleph_2^M,\ \omega \to^{M[G]} 2)$
using *Aleph_rel_sub_closed f_G_funtype G_in_MG Aleph_rel_sub_closed*
ext.curry_rel_exp[unfolded curry_def] ext.curry_closed[unfolded curry_def]
ext.mem_function_space_rel_abs
by (*intro ext.mem_inj_abs[THEN iffD2],simp_all*)
(*auto simp*: *inj_def h_G_def dest*:*Aleph_rel2_new_reals*)

lemma *Aleph2_extension_le_continuum_rel*:
includes *G_generic1_lemmas*
shows $\aleph_2^{M[G]} \leq 2^{\uparrow \aleph_0^{M[G]}, M[G]}$
proof -
have $\aleph_2^{M[G]} \lesssim^{M[G]} \omega \to^{M[G]} 2$
using *ext.def_lepoll_rel*[*of* $\aleph_2^M$ $\omega \to^{M[G]} 2$]
h_G_inj_Aleph_rel2_reals Aleph_rel_nats_MG_eq_Aleph_rel_nats_M
by *auto*
moreover from *calculation*
have $\aleph_2^{M[G]} \lesssim^{M[G]} |\omega \to^{M[G]} 2|^{M[G]}$
using *ext.lepoll_rel_imp_lepoll_rel_cardinal_rel* **by** *simp*
ultimately
have $|\aleph_2^{M[G]}|^{M[G]} \leq 2^{\uparrow \aleph_0^{M[G]}, M[G]}$
using *ext.lepoll_rel_imp_cardinal_rel_le*[*of* $\aleph_2^{M[G]}$ $\omega \to^{M[G]} 2$,
OF _ _ ext.function_space_rel_closed]
ext.Aleph_rel_zero
unfolding *cexp_rel_def* **by** *simp*
then
show $\aleph_2^{M[G]} \leq 2^{\uparrow \aleph_0^{M[G]}, M[G]}$
using *ext.Card_rel_Aleph_rel*[*of 2*, *THEN ext.Card_rel_cardinal_rel_eq*]
by *simp*
qed

lemma *Aleph_rel_lt_continuum_rel*: $\aleph_1^{M[G]} < 2^{\uparrow \aleph_0^{M[G]}, M[G]}$
using *Aleph2_extension_le_continuum_rel*
ext.Aleph_rel_increasing[*of 1 2*] *le_trans* **by** *auto*

corollary *not_CH*: $\aleph_1^{M[G]} \neq 2^{\uparrow \aleph_0^{M[G]}, M[G]}$
using *Aleph_rel_lt_continuum_rel* **by** *auto*

end — *add_generic3*

29.3 Models of fragments of $ZFC + \neg CH$

definition
ContHyp :: *o* **where**
ContHyp $\equiv \aleph_1 = 2^{\uparrow \aleph_0}$

relativize functional *ContHyp ContHyp_rel*
notation *ContHyp_rel* (‹*CH*—›)
relationalize *ContHyp_rel is_ContHyp*

context *M_ZF_library*
begin

is_iff_rel for *ContHyp*
using *is_cexp_iff is_Aleph_iff*[*of 0*] *is_Aleph_iff*[*of 1*]
unfolding *is_ContHyp_def ContHyp_rel_def*
by (*auto simp del*:*setclass_iff*) (*rule rexI*[*of _ _ M*, *OF _ nonempty*], *auto*)

end — *M_ZF_library*

synthesize *is_ContHyp* **from_definition assuming** *nonempty*
arity_theorem for *is_ContHyp_fm*

notation *is_ContHyp_fm* (‹·CH·›)

theorem *ctm_of_not_CH*:
  assumes
    $M \approx \omega$ *Transset*(*M*) $M \models ZC \cup \{\cdot Replacement(p)\cdot . p \in overhead\_notCH\}$
    $\Phi \subseteq formula$ $M \models \{ \cdot Replacement(ground\_repl\_fm(\varphi))\cdot . \varphi \in \Phi\}$
  shows
    $\exists N.$
     $M \subseteq N \wedge N \approx \omega \wedge Transset(N) \wedge N \models ZC \cup \{\cdot\neg\cdot CH\cdot\cdot\} \cup \{ \cdot Replacement(\varphi)\cdot . \varphi \in \Phi\} \wedge$
      $(\forall \alpha.\ Ord(\alpha) \longrightarrow (\alpha \in M \longleftrightarrow \alpha \in N))$
proof -
  from ‹$M \models ZC \cup \{\cdot Replacement(p)\cdot . p \in overhead\_notCH\}$›
  interpret *M_ZFC3 M*
    using *M_satT_overhead_imp_M_ZF3* **unfolding** *overhead_notCH_def* **by** *force*
  from ‹$M \models ZC \cup \{\cdot Replacement(p)\cdot . p \in overhead\_notCH\}$› ‹*Transset*(*M*)›
  interpret *M_ZF_ground_notCH_trans M*
    using *M_satT_imp_M_ZF_ground_notCH_trans*
    unfolding *ZC_def* **by** *auto*
  from ‹$M \approx \omega$›
  obtain *enum* **where** $enum \in bij(\omega,M)$
    using *eqpoll_sym* **unfolding** *eqpoll_def* **by** *blast*
  then
  interpret *M_ctm3_AC M enum* **by** *unfold_locales*
  interpret *cohen_data* ω ${\aleph_2}^M \times \omega$ *2* **by** *unfold_locales auto*
  have $Add \in M$ $Add\_le({\aleph_2}^M) \in M$
    using *nat_into_M Aleph_rel_closed M_nat cartprod_closed Fn_nat_closed Fnle_nat_closed*
    by *simp_all*
  then
  interpret *forcing_data1 Add Add_le*($\aleph_2{}^M$) *0 M enum*
    by *unfold_locales simp_all*
  obtain *G* **where** *M_generic*(*G*)
    using *generic_filter_existence*[*OF one_in_P*]
    by *auto*
  moreover from *this*
  interpret *add_generic3 M enum G* **by** *unfold_locales*
  have $\neg$ ($\aleph_1{}^{M[G]} = 2^{\uparrow \aleph_0{}^{M[G]}, M[G]}$)
    using *not_CH* **.**
  then
  have $M[G], [] \models \cdot\neg\cdot CH\cdot\cdot$
    using *ext.is_ContHyp_iff*

by (*simp add*:*ContHyp_rel_def*)
then
have $M[G] \models ZC \cup \{\cdot\neg\cdot CH\cdot\cdot\}$
using *ext.M_satT_ZC* **by** *auto*
moreover
have *Transset*($M[G]$) **using** *Transset_MG* **.**
moreover
have $M \subseteq M[G]$ **using** *M_subset_MG*[*OF one_in_G*] *generic* **by** *simp*
moreover
note ‹$M \models \{ \cdot Replacement(ground\_repl\_fm(\varphi))\cdot \,.\, \varphi \in \Phi\}$› ‹$\Phi \subseteq formula$›
ultimately
show *?thesis*
using *Ord_MG_iff MG_eqpoll_nat satT_ground_repl_fm_imp_satT_ZF_replacement_fm*[*of* Φ]
by (*rule_tac* x=$M[G]$ **in** *exI*, *blast*)
qed

lemma *ZF_replacement_overhead_sub_ZFC*: $\{\cdot Replacement(p)\cdot \,.\, p \in overhead\} \subseteq ZFC$
using *overhead_type* **unfolding** *ZFC_def ZF_def ZF_schemes_def* **by** *auto*

lemma *ZF_replacement_overhead_notCH_sub_ZFC*: $\{\cdot Replacement(p)\cdot \,.\, p \in overhead\_notCH\} \subseteq ZFC$
using *overhead_notCH_type* **unfolding** *ZFC_def ZF_def ZF_schemes_def* **by** *auto*

lemma *ZF_replacement_overhead_CH_sub_ZFC*: $\{\cdot Replacement(p)\cdot \,.\, p \in overhead\_CH\} \subseteq ZFC$
using *overhead_CH_type* **unfolding** *ZFC_def ZF_def ZF_schemes_def* **by** *auto*

corollary *ctm_ZFC_imp_ctm_not_CH*:
assumes
$M \approx \omega$ *Transset*(M) $M \models ZFC$
shows
$\exists\, N.$
$M \subseteq N \wedge N \approx \omega \wedge Transset(N) \wedge N \models ZFC \cup \{\cdot\neg\cdot CH\cdot\cdot\} \wedge$
$(\forall\, \alpha.\ Ord(\alpha) \longrightarrow (\alpha \in M \longleftrightarrow \alpha \in N))$
proof-
from *assms*
have $\exists\, N.$
$M \subseteq N \wedge$
$N \approx \omega \wedge$
Transset(N) $\wedge$
$N \models ZC \wedge N \models \{\cdot\neg\cdot CH\cdot\cdot\} \wedge N \models \{\cdot Replacement(x)\cdot \,.\, x \in formula\} \wedge (\forall\, \alpha.\ Ord(\alpha) \longrightarrow \alpha \in M \longleftrightarrow \alpha \in N)$
using *ctm_of_not_CH*[*of M formula*] *satT_ZFC_imp_satT_ZC*[*of M*]
satT_mono[*OF _ ground_repl_fm_sub_ZFC*, *of M*]
satT_mono[*OF _ ZF_replacement_overhead_notCH_sub_ZFC*, *of M*]

satT_mono[*OF _ ZF_replacement_fms_sub_ZFC*, *of M*]
by (*simp add*: *satT_Un_iff*)
then
obtain *N* **where** $N \models ZC$ $N \models \{\cdot\neg\cdot CH\cdot\cdot\}$ $N \models \{\cdot Replacement(x)\cdot . x \in formula\}$
$M \subseteq N$ $N \approx \omega$ *Transset*(*N*) $(\forall\, \alpha.\ Ord(\alpha) \longrightarrow \alpha \in M \longleftrightarrow \alpha \in N)$
by *auto*
moreover from *this*
have $N \models ZFC$
using *satT_ZC_ZF_replacement_imp_satT_ZFC*
by *auto*
moreover from *this* **and** ‹$N \models \{\cdot\neg\cdot CH\cdot\cdot\}$›
have $N \models ZFC \cup \{\cdot\neg\cdot CH\cdot\cdot\}$
by *auto*
ultimately
show *?thesis* **by** *auto*
qed

end

30 Preservation results for κ-closed forcing notions

theory *Kappa_Closed_Notions*
imports
Not_CH
begin

definition
lerel :: $i \Rightarrow i$ **where**
$lerel(\alpha) \equiv Memrel(\alpha) \cup id(\alpha)$

lemma *lerelI*[*intro*!]: $x \leq y \Longrightarrow y \in \alpha \Longrightarrow Ord(\alpha) \Longrightarrow \langle x,y\rangle \in lerel(\alpha)$
using *Ord_trans*[*of x y* α] *ltD* **unfolding** *lerel_def* **by** *auto*

lemma *lerelD*[*dest*]: $\langle x,y\rangle \in lerel(\alpha) \Longrightarrow Ord(\alpha) \Longrightarrow x \leq y$
using *ltI*[*THEN leI*] *Ord_in_Ord* **unfolding** *lerel_def* **by** *auto*

definition
mono_seqspace :: $[i,i,i] \Rightarrow i$ (‹_ $_{<}\!\!\rightarrow$ '(_,_')› [*61*] *60*) **where**
$\alpha\ {}_{<}\!\!\rightarrow (P,leq) \equiv mono\_map(\alpha,Memrel(\alpha),P,leq)$

relativize functional *mono_seqspace mono_seqspace_rel*
relationalize *mono_seqspace_rel is_mono_seqspace*
synthesize *is_mono_seqspace* **from_definition assuming** *nonempty*

context *M_ZF_library*
begin

rel_closed for *mono_seqspace*
unfolding *mono_seqspace_rel_def mono_map_rel_def*

using *separation_closed separation_ball separation_imp separation_in*
lam_replacement_fst lam_replacement_snd lam_replacement_hcomp lam_replacement_constant
lam_replacement_product
lam_replacement_apply2[*THEN*[*5*] *lam_replacement_hcomp2*]
by *simp_all*

end — *M_ZF_library*

abbreviation
mono_seqspace_r (‹_ $_{<}\!\!\to$ _'(_,_')› [*61*] *60*) **where**
$\alpha\ {}_{<}\!\!\to^{M}\ (P,leq) \equiv mono\_seqspace\_rel(M,\alpha,P,leq)$

abbreviation
mono_seqspace_r_set (‹_ $_{<}\!\!\to$ _'(_,_')› [*61*] *60*) **where**
$\alpha\ {}_{<}\!\!\to^{M}\ (P,leq) \equiv mono\_seqspace\_rel(\#\#M,\alpha,P,leq)$

lemma *mono_seqspaceI*[*intro*!]:
includes *mono_map_rules*
assumes *f*: $A\to P$ $\bigwedge x\ y.\ x\in A \Longrightarrow y\in A \Longrightarrow x<y \Longrightarrow \langle f`x, f`y\rangle \in leq$ *Ord*(*A*)
shows *f*: $A\ {}_{<}\!\!\to\ (P,leq)$
using *ltI*[*OF _ Ord_in_Ord*[*of A*], *THEN* [*3*] *assms*(*2*)] *assms*(*1,3*)
unfolding *mono_seqspace_def* **by** *auto*

lemma (**in** *M_ZF_library*) *mono_seqspace_rel_char*:
assumes *M*(*A*) *M*(*P*) *M*(*leq*)
shows $A\ {}_{<}\!\!\to^{M}\ (P,leq) = \{f\in A\ {}_{<}\!\!\to\ (P,leq).\ M(f)\}$
using *assms mono_map_rel_char*
unfolding *mono_seqspace_def mono_seqspace_rel_def* **by** *simp*

lemma (**in** *M_ZF_library*) *mono_seqspace_relI*[*intro*!]:
assumes *f*: $A\to^{M} P$ $\bigwedge x\ y.\ x\in A \Longrightarrow y\in A \Longrightarrow x<y \Longrightarrow \langle f`x, f`y\rangle \in leq$
Ord(*A*) *M*(*A*) *M*(*P*) *M*(*leq*)
shows *f*: $A\ {}_{<}\!\!\to^{M}\ (P,leq)$
using *mono_seqspace_rel_char function_space_rel_char assms* **by** *auto*

lemma *mono_seqspace_is_fun*[*dest*]:
includes *mono_map_rules*
shows *j*: $A\ {}_{<}\!\!\to\ (P,leq) \Longrightarrow j\colon A\to P$
unfolding *mono_seqspace_def* **by** *auto*

lemma *mono_map_lt_le_is_mono*[*dest*]:
includes *mono_map_rules*
assumes *j*: $A\ {}_{<}\!\!\to\ (P,leq)$ $a\in A$ $c\in A$ $a\le c$ *Ord*(*A*) *refl*(*P*,*leq*)
shows $\langle j`a,j`c\rangle \in leq$
using *assms mono_map_increasing* **unfolding** *mono_seqspace_def refl_def*
by (*cases* $a=c$) (*auto dest*:*ltD*)

lemma (**in** *M_ZF_library*) *mem_mono_seqspace_abs*[*absolut*]:
assumes *M*(*f*) *M*(*A*) *M*(*P*) *M*(*leq*)

shows $f{:}A\ {}_{<}\!\!\to^{M}\ (P,leq) \longleftrightarrow\ f{:}\ A\ {}_{<}\!\!\to\ (P,leq)$
using *assms mono_map_rel_char* **unfolding** *mono_seqspace_def mono_seqspace_rel_def*
by (*simp*)

definition
$mono\_map\_lt\_le :: [i,i] \Rightarrow i$ (**infixr** ‹${}_{<}\!\!\to_{\leq}$› *60*) **where**
$\alpha\ {}_{<}\!\!\to_{\leq}\ \beta \equiv \alpha\ {}_{<}\!\!\to\ (\beta,lerel(\beta))$

lemma *mono_map_lt_leI*[*intro*!]:
includes *mono_map_rules*
assumes $f{:}\ A\to B\ \bigwedge x\ y.\ x\in A \Longrightarrow y\in A \Longrightarrow x<y \Longrightarrow f`x \leq f`y\ Ord(A)\ Ord(B)$
shows $f{:}\ A\ {}_{<}\!\!\to_{\leq}\ B$
using *assms*
unfolding *mono_map_lt_le_def* **by** *auto*

— Kunen IV.7.13, with "κ" in place of "λ"
definition
$kappa\_closed :: [i,i,i] \Rightarrow o$ (‹*_-closed'(_,_')*›) **where**
$\kappa\text{-}closed(P,leq)\ \equiv\ \forall\delta.\ \delta<\kappa\ \longrightarrow\ (\forall f{\in}\delta\ {}_{<}\!\!\to\ (P,converse(leq)).\ \exists q{\in}P.\ \forall\alpha{\in}\delta.\ \langle q,f`\alpha\rangle{\in}leq)$

relativize functional *kappa_closed kappa_closed_rel*
relationalize *kappa_closed_rel is_kappa_closed*
synthesize *is_kappa_closed* **from_definition assuming** *nonempty*

abbreviation
$kappa\_closed\_r$ (‹*_-closed–'(_,_')*› [*61*] *60*) **where**
$\kappa\text{-}closed^{M}(P,leq) \equiv kappa\_closed\_rel(M,\kappa,P,leq)$

abbreviation
$kappa\_closed\_r\_set$ (‹*_-closed–'(_,_')*› [*61*] *60*) **where**
$\kappa\text{-}closed^{M}(P,leq) \equiv kappa\_closed\_rel(\#\#M,\kappa,P,leq)$

lemma (**in** *forcing_data3*) *forcing_a_value*:
assumes $p \Vdash\ {\cdot}0{:}1{\to}2{\cdot}\ [f\_dot,\ A^v,\ B^v]\ a \in A$
$q \preceq p\ q \in \mathbb{P}\ p{\in}\mathbb{P}\ f\_dot \in M\ A{\in}M\ B{\in}M$
shows $\exists\, d{\in}\mathbb{P}.\ \exists\, b{\in}B.\ d \preceq q \wedge d \Vdash\ {\cdot}0`1\ is\ 2{\cdot}\ [f\_dot,\ a^v,\ b^v]$

proof -
from *assms*
have $q \Vdash\ {\cdot}0{:}1{\to}2{\cdot}\ [f\_dot,\ A^v,\ B^v]$
using *strengthening_lemma*[*of p* ${\cdot}0{:}1{\to}2{\cdot}$ *q* [*f_dot*, A^v, B^v]]
typed_function_type arity_typed_function_fm
by (*auto simp*: *union_abs2 union_abs1*)
from ‹$a{\in}A$› ‹$A{\in}M$›
have $a{\in}M$ **by** (*auto dest*:*transitivity*)
from ‹$q{\in}\mathbb{P}$›

Here we're using countability (via the existence of generic filters) of M as a

shortcut, to avoid a further density argument.

obtain G **where** $M\_generic(G)$ $q{\in}G$
using *generic_filter_existence* **by** *blast*
then
interpret $G\_generic3\_AC$ _ _ _ _ _ G **by** *unfold_locales*
include *G_generic1_lemmas*
note ‹$q{\in}G$›
moreover
note ‹$q \Vdash$ ·0:1→2· $[f\_dot, A^v, B^v]$› ‹$M\_generic(G)$›
moreover
note ‹$q{\in}\mathbb{P}$› ‹$f\_dot{\in}M$› ‹$B{\in}M$› ‹$A{\in}M$›
moreover from *this*
have $map(val(\ G), [f\_dot, A^v, B^v]) \in list(M[G])$ **by** *simp*
moreover from *calculation*
have $val(G,f\_dot) : A \to^{M[G]} B$
using *truth_lemma*[*of* ·0:1→2· $[f\_dot, A^v, B^v]$, *THEN iffD1*]
typed_function_type arity_typed_function_fm val_check[*OF one_in_G one_in_P*]
by (*auto simp*: *union_abs2 union_abs1 ext.mem_function_space_rel_abs*)
moreover
note ‹$a \in M$›
moreover from *calculation* **and** ‹$a{\in}A$›
have $val(G,f\_dot)$ ‘ $a \in B$ (**is** $?b \in B$)
by (*simp add*: *ext.mem_function_space_rel_abs*)
moreover from *calculation*
have $?b \in M$ **by** (*auto dest*:*transitivity*)
moreover from *calculation*
have $M[G]$, $map(val(G), [f\_dot, a^v, ?b^v]) \models$ ·0‘1 is 2·
by *simp*
ultimately
obtain r **where** $r \Vdash$ ·0‘1 is 2· $[f\_dot, a^v, ?b^v]$ $r{\in}G$ $r{\in}\mathbb{P}$
using *truth_lemma*[*of* ·0‘1 is 2· $[f\_dot, a^v, ?b^v]$, *THEN iffD2*]
fun_apply_type arity_fun_apply_fm val_check[*OF one_in_G one_in_P*]
G_subset_P
by (*auto simp*: *union_abs2 union_abs1 ext.mem_function_space_rel_abs*)
moreover from *this* **and** ‹$q{\in}G$›
obtain d **where** $d{\preceq}q$ $d{\preceq}r$ $d{\in}\mathbb{P}$ **by** *force*
moreover
note ‹$f\_dot{\in}M$› ‹$a{\in}M$› ‹$?b{\in}B$› ‹$B{\in}M$›
moreover from *calculation*
have $d \preceq q \wedge d \Vdash$ ·0‘1 is 2· $[f\_dot, a^v, ?b^v]$
using *fun_apply_type arity_fun_apply_fm*
strengthening_lemma[*of* r ·0‘1 is 2· d $[f\_dot, a^v, ?b^v]$]
by (*auto dest*:*transitivity simp add*: *union_abs2 union_abs1*)
ultimately
show *?thesis* **by** *auto*
qed

locale $M\_master\_CH = M\_master + M\_library\_DC$

sublocale *M_ZFC2_ground_CH_trans* ⊆ *M_master_CH* ##*M*
  using *replacement_dcwit_repl_body*
  by *unfold_locales* (*simp_all add*:*sep_instances del*:*setclass_iff*
    add: *transrec_replacement_def wfrec_replacement_def dcwit_repl_body_def*)

context *G_generic3_AC_CH* **begin**

context
  includes *G_generic1_lemmas*
begin

lemma *separation_check_snd_aux*:
  assumes *f_dot*∈*M* *τ*∈*M* *χ*∈*formula arity*(*χ*) ≤ 7
  shows *separation*(##*M*, *λr. M*, [*fst*(*r*), ℙ, *leq*, **1**, *f_dot*, *τ*, *snd*(*r*)^v] ⊨ *χ*)
proof -
  let *?f_fm*=*fst_fm*(1,0)
  let *?g_fm*=*hcomp_fm*(*check_fm*(6),*snd_fm*,2,0)
  note *assms*
  moreover
  have *?f_fm* ∈ *formula arity*(*?f_fm*) ≤ 7 *?g_fm* ∈ *formula arity*(*?g_fm*) ≤ 8
    using *ord_simp_union*
    unfolding *hcomp_fm_def*
    by (*simp_all add*:*arity*)
  ultimately
  show *?thesis*
    using *separation_sat_after_function*
    using *fst_abs snd_abs sats_snd_fm sats_check_fm check_abs*
    unfolding *hcomp_fm_def*
    by *simp*
qed

lemma *separation_check_fst_snd_aux* :
  assumes *f_dot*∈*M* *r*∈*M* *χ*∈*formula arity*(*χ*) ≤ 7
  shows *separation*(##*M*, *λp. M*, [*r*, ℙ, *leq*, **1**, *f_dot*, *fst*(*p*)^v, *snd*(*p*)^v] ⊨ *χ*)
proof -
  let *?ϱ*=*λz.* [*r*, ℙ, *leq*, **1**, *f_dot*, *fst*(*z*)^v, *snd*(*z*)^v]
  let *?ϱ′*=*λz.* [*fst*(*z*)^v, ℙ, *leq*, **1**, *f_dot*, *r*, *snd*(*z*)^v]
  let *?φ*= (·∃(·∃(·∃(·∃(·∃(·∃··0 = 11· ∧ ··1 = 7· ∧ ··2 = 8· ∧ ··3 = 9· ∧ ··4 = 10· ∧ ··5 = 6· ∧
    (*λp. incr_bv*(*p*)‘6)^6 (*χ*) ·····)·)·)·)·)·)
  let *?f_fm*=*hcomp_fm*(*check_fm*(5),*fst_fm*,1,0)
  let *?g_fm*=*hcomp_fm*(*check_fm*(6),*snd_fm*,2,0)
  note *assms*
  moreover
  have *?f_fm* ∈ *formula arity*(*?f_fm*) ≤ 7 *?g_fm* ∈ *formula arity*(*?g_fm*) ≤ 8
    using *ord_simp_union*
    unfolding *hcomp_fm_def*
    by (*simp_all add*:*arity*)

```
  moreover from assms
  have fm:?φ∈formula by simp
  moreover from ‹χ ∈ formula› ‹arity(χ) ≤ 7›
  have arity(χ) = 0 ∨ arity(χ) = 1 ∨ arity(χ) = 2 ∨ arity(χ) = 3
    ∨ arity(χ) = 4 ∨ arity(χ) = 5 ∨ arity(χ) = 6 ∨ arity(χ) = 7
    unfolding lt_def by auto
  with calculation and ‹χ ∈ formula›
  have ar:arity(?φ) ≤ 7
    using arity_incr_bv_lemma by safe (simp_all add: arity ord_simp_union)
  moreover from calculation
  have sep:separation(##M,λz. M,?ϱ'(z)⊨?φ)
    using separation_sat_after_function sats_check_fm check_abs
      fst_abs snd_abs
    unfolding hcomp_fm_def
    by simp
  moreover from assms
  have ?ϱ(z) ∈ list(M) if (##M)(z) for z
    using that by simp
  moreover from calculation and ‹r ∈ M› ‹χ ∈ formula›
  have (M,?ϱ(z) ⊨ χ) ⟷ (M,?ϱ'(z)⊨?φ) if (##M)(z) for z
    using that sats_incr_bv_iff[of _ _ M _ [_,_,_,_,_,_]]
    by simp
  ultimately
  show ?thesis
    using separation_cong[THEN iffD1,OF _ sep]
    by simp
qed

lemma separation_leq_and_forces_apply_aux:
  assumes f_dot∈M B∈M
  shows ∀ n∈M. separation(##M, λx. snd(x) ≼ fst(x) ∧
        (∃ b∈B. M, [snd(x), ℙ, leq, 1, f_dot, (⋃(n))^v, b^v] ⊨ forces(⋅0‘1 is 2⋅ )))
proof -
  have pred_nat_closed: pred(n)∈M if n∈M for n
    using nat_case_closed that
    unfolding pred_def
    by auto
  have separation(##M, λz. M, [snd(fst(z)), ℙ, leq, 1, f_dot, τ, snd(z)^v] ⊨ χ)
    if χ∈formula arity(χ) ≤ 7 τ∈M for χ τ
  proof -
    let ?f_fm=hcomp_fm(snd_fm,fst_fm,1,0)
    let ?g_fm=hcomp_fm(check_fm(6),snd_fm,2,0)
    note assms
    moreover
    have ?f_fm ∈ formula arity(?f_fm) ≤ 7 ?g_fm ∈ formula arity(?g_fm) ≤ 8
      using ord_simp_union
      unfolding hcomp_fm_def
      by (simp_all add:arity)
    ultimately
```

```
    show ?thesis
        using separation_sat_after_function sats_check_fm check_abs fst_abs
snd_abs that
      unfolding hcomp_fm_def
      by simp
  qed
  with assms
  show ?thesis
   using separation_in lam_replacement_constant lam_replacement_snd lam_replacement_fst
      lam_replacement_product pred_nat_closed
      arity_forces[of ·0‘1 is 2·] arity_fun_apply_fm[of 0 1 2] ord_simp_union
   by(clarify,rule_tac separation_conj,simp_all,rule_tac separation_bex,simp_all)
qed

lemma separation_leq_and_forces_apply_aux':
  assumes f_dot∈M p∈M B∈M
  shows separation
     (##M, λp .  snd(snd(p)) ≼ fst(snd(p)) ∧
    (∃ b∈B. M, [snd(snd(p)), ℙ, leq, 1, f_dot, (⋃fst(p))^v, b^v] ⊨ forces(·0‘1 is 2· )))
proof -
  have separation(##M, λz. M, [snd(snd(fst(z))), ℙ, leq, 1, f_dot, (⋃fst(fst(z)))^v,
snd(z)^v] ⊨ χ)
    if χ∈formula arity(χ) ≤ 7 for χ
  proof -
    let ?f_fm=hcomp_fm(snd_fm,hcomp_fm(snd_fm,fst_fm),1,0)
    let ?g=λz . (⋃(fst(fst(z))))^v
   let ?g_fm=hcomp_fm(check_fm(6),hcomp_fm(big_union_fm,hcomp_fm(fst_fm,fst_fm)),2,0)
    let ?h_fm=hcomp_fm(check_fm(7),snd_fm,3,0)
    note assms
    moreover
    have f_fm_facts:?f_fm ∈ formula arity(?f_fm) ≤ 6
      using ord_simp_union
      unfolding hcomp_fm_def
      by (simp_all add:arity)
    moreover from assms
    have ?g_fm ∈ formula arity(?g_fm) ≤ 7 ?h_fm ∈ formula arity(?h_fm) ≤ 8
      using ord_simp_union
      unfolding hcomp_fm_def
      by (simp_all add:arity)
    ultimately
    show ?thesis
      using separation_sat_after_function3[OF _ _ _ f_fm_facts] check_abs
        sats_check_fm that fst_abs snd_abs sats_fst_fm sats_snd_fm
      unfolding hcomp_fm_def
      by simp
  qed
  with assms
  show ?thesis
    using
```

```
    separation_conj separation_bex
    lam_replacement_constant lam_replacement_hcomp
    lam_replacement_fst lam_replacement_snd
    arity_forces[of ·0‘1 is 2·] arity_fun_apply_fm[of 0 1 2] ord_simp_union
    separation_in[OF _ lam_replacement_product]
  by simp
qed

lemma separation_closed_leq_and_forces_eq_check_aux :
  assumes A∈M r∈G τ ∈ M
  shows (##M)({q∈ℙ. ∃ h∈A. q ≼ r ∧ q ⊩ ·0 = 1· [τ, h^v]})
proof -
  have separation(##M, λz. M, [fst(z), ℙ, leq, 1, τ, snd(z)^v] ⊨ χ) if
    χ∈formula arity(χ) ≤ 6 for χ
  proof -
    let ?f_fm=fst_fm(1,0)
    let ?g_fm=hcomp_fm(check_fm(6),snd_fm,2,0)
    note assms
    moreover
    have ?f_fm ∈ formula arity(?f_fm) ≤ 6 ?g_fm ∈ formula arity(?g_fm) ≤ 7
      using ord_simp_union
      unfolding hcomp_fm_def
      by (simp_all add:arity)
    ultimately
    show ?thesis
      using separation_sat_after_function_1 sats_fst_fm that
        fst_abs snd_abs sats_snd_fm sats_check_fm check_abs
      unfolding hcomp_fm_def
      by simp
  qed
  with assms
  show ?thesis
    using separation_conj separation_in G_subset_M[THEN subsetD]
      lam_replacement_constant lam_replacement_fst lam_replacement_product
      arity_forces[of ·0 = 1·,simplified] ord_simp_union
    by(rule_tac separation_closed[OF separation_bex],simp_all)
qed

lemma separation_closed_forces_apply_aux:
  assumes B∈M f_dot∈M r∈M
  shows (##M)({⟨n,b⟩ ∈ ω × B. r ⊩ ·0‘1 is 2· [f_dot, n^v, b^v]})
 using nat_in_M assms transitivity[OF _ ‹B∈M›] nat_into_M separation_check_fst_snd_aux
    arity_forces[of ·0‘1 is 2·] arity_fun_apply_fm[of 0 1 2] ord_simp_union
  unfolding split_def
  by simp_all

— Kunen IV.6.9 (3)⇒(2), with general domain.
lemma kunen_IV_6_9_function_space_rel_eq:
  assumes ⋀p τ. p ⊩ ·0:1→2· [τ, A^v, B^v] ⟹ p∈ℙ ⟹ τ ∈ M ⟹
```

```
    ∃ q∈ℙ. ∃ h∈A →^M B. q ≼ p ∧  q ⊩ ·0 = 1· [τ, h^v] A∈M B∈M
  shows
    A →^M B = A →^M[G] B
proof (intro equalityI; clarsimp simp add:
    assms function_space_rel_char ext.function_space_rel_char)
  fix f
  assume f ∈ A → B f ∈ M[G]
  moreover from this
  obtain τ where val(G,τ) = f τ ∈ M
    using GenExtD by force
  moreover from calculation and ‹A∈M› ‹B∈M›
  obtain r where r ⊩ ·0:1→2· [τ, A^v, B^v] r∈G
    using truth_lemma[of ·0:1→2· [τ, A^v, B^v]]
        typed_function_type arity_typed_function_fm val_check[OF one_in_G
one_in_P]
    by (auto simp: union_abs2 union_abs1)
  moreover from ‹A∈M› ‹B∈M› ‹r∈G› ‹τ ∈ M›
  have {q∈ℙ. ∃ h∈A →^M B. q ≼ r ∧ q ⊩ ·0 = 1· [τ, h^v]} ∈ M (is ?D ∈ M)
    using separation_closed_leq_and_forces_eq_check_aux by auto
  moreover from calculation and assms(2-)
  have dense_below(?D, r)
    using strengthening_lemma[of r ·0:1→2· _ [τ, A^v, B^v], THEN assms(1)[of _
τ]]
      leq_transD generic_dests(1)[of r]
   by (auto simp: union_abs2 union_abs1 typed_function_type arity_typed_function_fm)
blast
  moreover from calculation
  obtain q h where h∈A →^M B q ⊩ ·0 = 1· [τ, h^v] q ≼ r q∈ℙ q∈G
    using generic_inter_dense_below[of ?D r] by blast
  note ‹q ⊩ ·0 = 1· [τ, h^v]› ‹τ∈M› ‹h∈A →^M B› ‹A∈M› ‹B∈M› ‹q∈G›
  moreover from this
  have map(val(G), [τ, h^v]) ∈ list(M[G]) h∈M
    by (auto dest:transitivity)
  ultimately
  have h = f
    using truth_lemma[of ·0=1· [τ, h^v]] val_check[OF one_in_G one_in_P]
    by (auto simp: ord_simp_union)
  with ‹h∈M›
  show f ∈ M by simp
qed
```

30.1 $(\omega + 1)$-Closed notions preserve countable sequences

```
lemma succ_omega_closed_imp_no_new_nat_sequences:
  assumes succ(ω)-closed^M(ℙ,leq) f : ω → B f∈M[G] B∈M
  shows f∈M
proof -
```

The next long block proves that the assumptions of Lemma *kunen_IV_6_9_function_space_rel_eq* are satisfied.

{
fix p f_dot
assume $p \Vdash \cdot 0{:}1{\to}2\cdot\ [f\_dot,\ \omega^v,\ B^v]$ $p{\in}\mathbb{P}$ $f\_dot{\in}M$
let $?subp{=}\{q{\in}\mathbb{P}.\ q \preceq p\}$
from ‹$p{\in}\mathbb{P}$›
have $?subp \in M$
using $first\_section\_closed[of\ \mathbb{P}\ p\ converse(leq)]$
by $(auto\ dest{:}transitivity)$
define S **where** $S \equiv \lambda n{\in}nat.$
$\{\langle q,r\rangle \in ?subp{\times}?subp.\ r \preceq q \wedge (\exists\, b{\in}B.\ r \Vdash \cdot 0{`}1\ is\ 2\cdot\ [f\_dot,\ (\bigcup(n))^v,\ b^v])\}$
(**is** $S \equiv \lambda n{\in}nat.\ ?Y(n)$)
define S' **where** $S' \equiv \lambda n{\in}nat.$
$\{\langle q,r\rangle \in ?subp{\times}?subp.\ r \preceq q \wedge (\exists\, b{\in}B.\ r \Vdash \cdot 0{`}1\ is\ 2\cdot\ [f\_dot,\ (pred(n))^v,\ b^v])\}$
— Towards proving $S \in M$.
moreover
have $S = S'$
unfolding $S\_def\ S'\_def$ **using** $pred\_nat\_eq\ lam\_cong$ **by** $auto$
moreover from ‹$B{\in}M$› ‹$?subp{\in}M$› ‹$f\_dot{\in}M$›
have $\{r \in ?subp.\ \exists\, b{\in}B.\ r \Vdash \cdot 0{`}1\ is\ 2\cdot\ [f\_dot,\ (\bigcup(n))^v,\ b^v]\} \in M$ (**is** $?X(n) \in M$)
if $n{\in}\omega$ **for** n
using $that\ separation\_check\_snd\_aux\ nat\_into\_M\ ord\_simp\_union$
$arity\_forces[of\ \cdot 0{`}1\ is\ 2\cdot]\ arity\_fun\_apply\_fm$
by$(rule\_tac\ separation\_closed[OF\ separation\_bex{,}simplified],\ simp\_all)$
moreover
have $?Y(n) = (?subp \times ?X(n)) \cap converse(leq)$ **for** n
by $(intro\ equalityI)\ auto$
moreover
note ‹$?subp \in M$› ‹$B{\in}M$› ‹$p{\in}\mathbb{P}$› ‹$f\_dot{\in}M$›
moreover from $calculation$
have $n \in \omega \Longrightarrow ?Y(n) \in M$ **for** n
using nat_into_M **by** $simp$
moreover from $calculation$
have $S \in M$
using $separation\_leq\_and\_forces\_apply\_aux\ separation\_leq\_and\_forces\_apply\_aux'$
$transitivity[OF\ ‹p{\in}\mathbb{P}›]$
unfolding $S\_def\ split\_def$
by$(rule\_tac\ lam\_replacement\_Collect'[THEN\ lam\_replacement\_imp\_lam\_closed{,}simplified],\ simp\_all)$
ultimately
have $S' \in M$
by $simp$
from ‹$p{\in}\mathbb{P}$› ‹$f\_dot{\in}M$› ‹$p \Vdash \cdot 0{:}1{\to}2\cdot\ [f\_dot,\ \omega^v,\ B^v]$› ‹$B{\in}M$›
have $exr{:}\exists\, r{\in}\mathbb{P}.\ r \preceq q \wedge (\exists\, b{\in}B.\ r \Vdash \cdot 0{`}1\ is\ 2\cdot\ [f\_dot,\ pred(n)^v,\ b^v])$
if $q \preceq p$ $q{\in}\mathbb{P}$ $n{\in}\omega$ **for** q n
using $that\ forcing\_a\_value$ **by** $(auto\ dest{:}transitivity)$
have $\forall\, q{\in}?subp.\ \forall\, n{\in}\omega.\ \exists\, r{\in}?subp.\ \langle q,r\rangle \in S'{`}n$
proof -
{

fix q n
assume $q \in$ *?subp* $n{\in}\omega$
moreover from *this*
have $q \preceq p$ $q \in \mathbb{P}$ $pred(n) = \bigcup n$
using *pred_nat_eq* **by** *simp_all*
moreover from *calculation* **and** *exr*
obtain r **where** *MM*:$r \preceq q$ $\exists\, b{\in}B.\ r \Vdash \cdot 0`1$ *is* $2{\cdot}$ $[f\_dot,\ pred(n)^v,\ b^v]$ $r{\in}\mathbb{P}$
by *blast*
moreover from *calculation* ‹$q \preceq p$› ‹$p \in \mathbb{P}$›
have $r \preceq p$
using *leq_transD*[*of* r q p] **by** *auto*
ultimately
have $\exists\, r{\in}$*?subp*. $r \preceq q \wedge (\exists\, b{\in}B.\ r \Vdash \cdot 0`1$ *is* $2{\cdot}$ $[f\_dot,\ (pred(n))^v,\ b^v])$
by *auto*
}
then
show *?thesis*
unfolding *S′_def* **by** *simp*
qed
with ‹$p{\in}\mathbb{P}$› ‹*?subp* $\in M$› ‹$S' \in M$›
obtain g **where** $g \in \omega \to^M$ *?subp* $g`0 = p$ $\forall\, n \in nat.\ \langle g`n, g`succ(n)\rangle {\in} S'`succ(n)$
using *sequence_DC*[*simplified*] *refl_leq*[*of* p] **by** *blast*
moreover from *this* **and** ‹*?subp* $\in M$›
have $g : \omega \to \mathbb{P}$ $g \in M$
using *fun_weaken_type*[*of* g ω *?subp* $\mathbb{P}$] *function_space_rel_char* **by** *auto*
ultimately
have $g : \omega\ {}_{<}{\to}^M\ (\mathbb{P}, converse(leq))$
using *decr_succ_decr*[*of* g] *leq_preord*
unfolding *S′_def* **by** (*auto simp*:*absolut intro*:*leI*)
moreover from ‹$succ(\omega)$-$closed^M(\mathbb{P}, leq)$› **and** *this*
have $\exists\, q{\in}M.\ q \in \mathbb{P} \wedge (\forall\, \alpha{\in}M.\ \alpha \in \omega \longrightarrow q \preceq g\ `\ \alpha)$
using *transitivity*[*simplified*, *of* g] *mono_seqspace_rel_closed*[*of* ω _ *converse*(*leq*)]
unfolding *kappa_closed_rel_def*
by *auto*
ultimately
obtain r **where** $r{\in}\mathbb{P}$ $r{\in}M$ $\forall\, n{\in}\omega.\ r \preceq g`n$
using *nat_into_M* **by** *auto*
with ‹$g`0 = p$›
have $r \preceq p$
by *blast*
let *?h*=$\{\langle n,b\rangle \in \omega \times B.\ r \Vdash \cdot 0`1$ *is* $2{\cdot}$ $[f\_dot,\ n^v,\ b^v]\}$
have *function*(*?h*)
proof (*rule_tac functionI*, *rule_tac ccontr*, *auto simp del*: *app_Cons*)
fix n b b'
assume $n \in \omega$ $b \neq b'$ $b \in B$ $b' \in B$
moreover
assume $r \Vdash \cdot 0`1$ *is* $2{\cdot}$ $[f\_dot,\ n^v,\ b^v]$ $r \Vdash \cdot 0`1$ *is* $2{\cdot}$ $[f\_dot,\ n^v,\ b'^v]$
moreover

```
  note ‹r ∈ ℙ›
  moreover from this
  have ¬ r ⊥ r
    by (auto intro!:refl_leq)
  moreover
  note ‹f_dot∈M› ‹B∈M›
  ultimately
  show False
    using forces_neq_apply_imp_incompatible[of r f_dot n^v b r b']
      transitivity[of _ B] by (auto dest:transitivity)
qed
moreover
have range(?h) ⊆ B
  by auto
moreover
have domain(?h) = ω
proof -
  {
    fix n
    assume n ∈ ω
    moreover from this
    have 1:(⋃(n)) = pred(n)
      using pred_nat_eq by simp
    moreover from calculation and ‹∀ n ∈ nat. ⟨g‘n,g‘succ(n)⟩∈S'‘succ(n)›
    obtain b where g‘(succ(n)) ⊩ ·0‘1 is 2· [f_dot, n^v, b^v] b∈B
      unfolding S'_def by auto
    moreover from ‹B∈M› and calculation
    have b ∈ M n ∈ M
      by (auto dest:transitivity)
    moreover
    note ‹g : ω → ℙ› ‹∀ n∈ω. r ≼ g‘n› ‹r∈ℙ› ‹f_dot∈M›
    moreover from calculation
    have r ⊩ ·0‘1 is 2· [f_dot, n^v, b^v]
      using fun_apply_type arity_fun_apply_fm
        strengthening_lemma[of g‘succ(n) ·0‘1 is 2· r [f_dot, n^v, b^v]]
      by (simp add: union_abs2 union_abs1)
    ultimately
    have ∃ b∈B. r ⊩ ·0‘1 is 2· [f_dot, n^v, b^v]
      by auto
  }
  then
  show ?thesis
    by force
qed
moreover
have relation(?h)
  unfolding relation_def by simp
moreover from ‹f_dot∈M› ‹r∈M› ‹B∈M›
have ?h ∈ M
```

```
    using separation_closed_forces_apply_aux by simp
  moreover
  note ‹B ∈ M›
  ultimately
  have ?h: ω →^M B
    using function_imp_Pi[THEN fun_weaken_type[of ?h _ range(?h) B]]
      function_space_rel_char by simp
  moreover
  note ‹p ⊩ ·0:1→2· [f_dot, ω^v, B^v]› ‹r ≼ p› ‹r∈ℙ› ‹p∈ℙ› ‹f_dot∈M› ‹B∈M›
  moreover from this
  have r ⊩ ·0:1→2· [f_dot, ω^v, B^v]
    using strengthening_lemma[of p ·0:1→2· r [f_dot, ω^v, B^v]]
      typed_function_type arity_typed_function_fm
    by (auto simp: union_abs2 union_abs1)
  moreover
  note ‹?h∈M›
  moreover from calculation
  have r ⊩ ·0 = 1· [f_dot, ?h^v]
  proof (intro definition_of_forcing[THEN iffD2] allI impI,
      simp_all add:union_abs2 union_abs1 del:app_Cons)
    fix H
    let ?f=val(H,f_dot)
    assume M_generic(H) ∧ r ∈ H
    moreover from this
    interpret g:G_generic1 _ _ _ _ _ H
      by unfold_locales simp
    note ‹r∈ℙ› ‹f_dot∈M› ‹B∈M›
    moreover from calculation
    have map(val(H), [f_dot, ω^v, B^v]) ∈ list(M[H]) r∈H
      by simp_all
    moreover from calculation and ‹r∈H› and ‹r ⊩ ·0:1→2· [f_dot, ω^v, B^v]›
    have ?f : ω → B
      using g.truth_lemma[of ·0:1→2· [f_dot, ω^v, B^v],THEN iffD1] g.one_in_G
one_in_P
        typed_function_type arity_typed_function_fm val_check
      by (auto simp: union_abs2 union_abs1)
    moreover
    have ?h‘n = ?f‘n if n ∈ ω for n
    proof -
      note ‹n ∈ ω› ‹domain(?h) = ω›
      moreover from this
      have n∈domain(?h)
        by simp
      moreover from this
      obtain b where r ⊩ ·0‘1 is 2· [f_dot, n^v, b^v] b∈B
        by force
      moreover
      note ‹function(?h)›
      moreover from calculation
```

```
      have b = ?h‘n
        using function_apply_equality by simp
      moreover
      note ‹B ∈ M›
      moreover from calculation
      have ?h‘n ∈ M
        by (auto dest:transitivity)
      moreover
      note ‹f_dot ∈ M› ‹r ∈ ℙ› ‹M_generic(H) ∧ r ∈ H› ‹map(val(H), [f_dot,
ω^v, B^v]) ∈ list(M[H])›
      moreover from calculation
      have [?f, n, ?h‘n] ∈ list(M[H])
     using M_subset_MG nat_into_M[of n] g.one_in_G by (auto dest:transitivity)
      ultimately
      show ?thesis
        using definition_of_forcing[of r ·0‘1 is 2· [f_dot, n^v, b^v],
            THEN iffD1, rule_format, of H]— without this line is slower
          val_check g.one_in_G one_in_P nat_into_M
        by (auto dest:transitivity simp add:fun_apply_type
            arity_fun_apply_fm union_abs2 union_abs1)
    qed
    with calculation and ‹B∈M› ‹?h: ω →^M B›
    have ?h = ?f
      using function_space_rel_char
      by (rule_tac fun_extension[of ?h ω λ_.B ?f]) auto
    ultimately
    show ?f = val(H, ?h^v)
      using val_check g.one_in_G one_in_P generic by simp
  qed
  ultimately
  have ∃ r∈ℙ. ∃ h∈ω →^M B. r ≼ p ∧ r ⊩ ·0 = 1· [f_dot, h^v]
    by blast
 }
 moreover
 note ‹B ∈ M› assms
 moreover from calculation
 have f : ω →^M B
   using kunen_IV_6_9_function_space_rel_eq function_space_rel_char
     ext.mem_function_space_rel_abs by auto
 ultimately
 show ?thesis
   by (auto dest:transitivity)
qed

declare mono_seqspace_rel_closed[rule del]
 — Mysteriously breaks the end of the next proof

lemma succ_omega_closed_imp_no_new_reals:
 assumes succ(ω)-closed^M(ℙ,leq)
```

```
  shows ω →^M 2 = ω →^M[G] 2
proof -
  from assms
  have ω →^M[G] 2 ⊆ ω →^M 2
   using succ_omega_closed_imp_no_new_nat_sequences function_space_rel_char
     ext.function_space_rel_char Aleph_rel_succ Aleph_rel_zero
    by auto
  then
  show ?thesis
    using function_space_rel_transfer by (intro equalityI) auto
qed

lemma succ_omega_closed_imp_Aleph_1_preserved:
  assumes succ(ω)-closed^M(ℙ,leq)
  shows ℵ_1^M = ℵ_1^M[G]
proof -
  have ℵ_1^M[G] ≤ ℵ_1^M
  proof (rule ccontr)
    assume ¬ ℵ_1^M[G] ≤ ℵ_1^M
    then
    have ℵ_1^M < ℵ_1^M[G]
      — Ridiculously complicated proof
      using Card_rel_is_Ord ext.Card_rel_is_Ord
        not_le_iff_lt[THEN iffD1] by auto
    then
    have |ℵ_1^M|^M[G] ≤ ω
      using ext.Card_rel_lt_csucc_rel_iff ext.Aleph_rel_zero
        ext.Aleph_rel_succ ext.Card_rel_nat
      by (auto intro!:ext.lt_csucc_rel_iff[THEN iffD1]
          intro:Card_rel_Aleph_rel[THEN Card_rel_is_Ord, of 1])
    then
    obtain f where f ∈ inj(ℵ_1^M,ω) f ∈ M[G]
      using ext.countable_rel_iff_cardinal_rel_le_nat[of ℵ_1^M, THEN iffD2]
      unfolding countable_rel_def lepoll_rel_def
      by auto
    then
    obtain g where g ∈ surj^M[G](ω, ℵ_1^M)
     using ext.inj_rel_imp_surj_rel[of f _ ω, OF _ zero_lt_Aleph_rel1[THEN
ltD]]
      by auto
    moreover from this
    have g : ω → ℵ_1^M g ∈ M[G]
      using ext.surj_rel_char surj_is_fun by simp_all
    moreover
    note ‹succ(ω)-closed^M(ℙ,leq)›
    ultimately
    have g ∈ surj^M(ω, ℵ_1^M) g ∈ M
      using succ_omega_closed_imp_no_new_nat_sequences
        mem_surj_abs ext.mem_surj_abs by simp_all
```

```
    then
    show False
      using surj_rel_implies_cardinal_rel_le[of g ω ℵ_1^M]
        Card_rel_nat[THEN Card_rel_cardinal_rel_eq] Card_rel_is_Ord
        not_le_iff_lt[THEN iffD2, OF _ _ nat_lt_Aleph_rel1]
      by simp
  qed
  then
  show ?thesis
    using Aleph_rel_le_Aleph_rel
    by (rule_tac le_anti_sym) simp
qed

end — bundle G_generic1_lemmas

end — G_generic3_AC

end
```

31 Forcing extension satisfying the Continuum Hypothesis

```
theory CH
  imports
    Kappa_Closed_Notions
    Cohen_Posets_Relative
begin

context M_ctm2_AC
begin

declare Fn_rel_closed[simp del, rule del, simplified setclass_iff, simp, intro]
declare Fnle_rel_closed[simp del, rule del, simplified setclass_iff, simp, intro]

abbreviation
  Coll :: i where
  Coll ≡ Fn^M(ℵ_1^M, ℵ_1^M, ω →^M 2)

abbreviation
  Colleq :: i where
  Colleq ≡ Fnle^M(ℵ_1^M, ℵ_1^M, ω →^M 2)

lemma Coll_in_M[intro,simp]: Coll ∈ M by simp

lemma Colleq_refl : refl(Coll,Colleq)
  unfolding Fnle_rel_def Fnlerel_def refl_def
  using RrelI by simp
```

31.1 Collapse forcing is sufficiently closed

lemma *succ_omega_closed_Coll*: *succ*(ω)-*closed*^{M}(*Coll*,*Colleq*)
proof -
— Case for finite sequences
have $n{\in}\omega \Longrightarrow \forall f \in n \ {}_{<}{\to}^M \ (Coll,converse(Colleq))$.
$\exists\, q{\in}M.\ q \in Coll \wedge (\forall\, \alpha{\in}M.\ \alpha \in n \longrightarrow \langle q, f\ `\ \alpha\rangle \in Colleq)$ **for** n
proof (*induct rule*:*nat_induct*)
case *0*
then
show *?case*
using *zero_lt_Aleph_rel1 zero_in_Fn_rel*
by (*auto simp del*:*setclass_iff*) (*rule bexI*[*OF _ zero_in_M*], *auto*)
next
case (*succ x*)
then
have $\forall f{\in}succ(x) \ {}_{<}{\to}^M \ (Coll,converse(Colleq)).\ \forall\, \alpha \in succ(x).\ \langle f`x, f\ `\ \alpha\rangle \in Colleq$
proof(*intro ballI*)
fix *f* α
assume $f{\in}succ(x) \ {}_{<}{\to}^M \ (Coll,converse(Colleq))\ \alpha{\in}succ(x)$
moreover from ‹$x{\in}\omega$› *this*
have $f{\in}succ(x) \ {}_{<}{\to}\ (Coll,converse(Colleq))$
using *mono_seqspace_rel_char nat_into_M*
by *simp*
moreover from *calculation succ*
consider $\alpha{\in}x \mid \alpha{=}x$
by *auto*
then
show $\langle f`x, f\ `\ \alpha\rangle \in Colleq$
proof(*cases*)
case *1*
then
have $\langle\alpha,\ x\rangle \in Memrel(succ(x))\ x{\in}succ(x)\ \alpha{\in}succ(x)$
by *auto*
with ‹$f{\in}succ(x) \ {}_{<}{\to}\ (Coll,converse(Colleq))$›
show *?thesis*
using *mono_mapD*(*2*)[*OF _* ‹$\alpha{\in}succ(x)$› *_* ‹$\langle\alpha,\ x\rangle \in Memrel(succ(x))$›]
unfolding *mono_seqspace_def*
by *auto*
next
case *2*
with ‹$f{\in}succ(x) \ {}_{<}{\to}\ (Coll,converse(Colleq))$›
show *?thesis*
using *Colleq_refl mono_seqspace_is_fun*[*THEN apply_type*]
unfolding *refl_def*
by *simp*
qed
qed
moreover

note ‹$x{\in}\omega$›
moreover from *this*
have $f`x \in Coll$ **if** f: $succ(x)\ {}_{<}{\to}^{M}\ (Coll,converse(Colleq))$ **for** f
using *that mono_seqspace_rel_char*[*simplified*, *of succ*(*x*) *Coll converse*(*Colleq*)]
nat_into_M[*simplified*] *mono_seqspace_is_fun*[*of converse*(*Colleq*)]
by (*intro apply_type*[*of _ succ*(*x*)]) (*auto simp del*:*setclass_iff*)
ultimately
show *?case*
using *transM*[*of _ Coll*]
by (*auto dest*:*transM simp del*:*setclass_iff*, *rule_tac x*=$f`x$ **in** *bexI*)
(*auto simp del*:*setclass_iff*, *simp*)
qed
moreover
— Interesting case: Countably infinite sequences.
have $\forall f{\in}M.\ f \in \omega\ {}_{<}{\to}^{M}\ (Coll,converse(Colleq)) \longrightarrow$
$(\exists q{\in}M.\ q \in Coll \wedge (\forall \alpha{\in}M.\ \alpha \in \omega \longrightarrow \langle q, f\ `\ \alpha\rangle \in Colleq))$
proof(*intro ballI impI*)
fix f
let *?rnf*=$f``\omega$
assume $f{\in}M\ f \in \omega\ {}_{<}{\to}^{M}\ (Coll,converse(Colleq))$
moreover from *this*
have $f{\in}\omega\ {}_{<}{\to}\ (Coll,converse(Colleq))\ f{\in}\omega \to Coll$
using *mono_seqspace_rel_char mono_mapD*(*2*) *nat_in_M*
by *auto*
moreover from *this*
have $countable^{M}(f`x)$ **if** $x{\in}\omega$ **for** x
using *that Fn_rel_is_function countable_iff_lesspoll_rel_Aleph_rel_one*
by *auto*
moreover from *calculation*
have *?rnf* $\in M\ f{\subseteq}\omega{\times}Coll$
using *nat_in_M image_closed Pi_iff*
by *simp_all*
moreover from *calculation*
have *1*:$\exists\, d{\in}$*?rnf*. $d \supseteq h \wedge d \supseteq g$ **if** $h \in$ *?rnf* $g \in$ *?rnf* **for** $h\ g$
proof -
from *calculation*
have *?rnf*=$\{f`x\ .\ x{\in}\omega\}$
using *image_function*[*of f* ω] *Pi_iff domain_of_fun*
by *auto*
from ‹*?rnf*=_› *that*
obtain $m\ n$ **where** *eq*:$h{=}f`m\ g{=}f`n\ n{\in}\omega\ m{\in}\omega$
by *auto*
then
have $m{\cup}n{\in}\omega\ m{\le}m{\cup}n\ n{\le}m{\cup}n$
using *Un_upper1_le Un_upper2_le nat_into_Ord* **by** *simp_all*
with *calculation eq* ‹*?rnf*=_›
have $f`(m{\cup}n){\in}$*?rnf* $f`(m{\cup}n) \supseteq h\ f`(m{\cup}n) \supseteq g$
using *Fnle_relD mono_map_lt_le_is_mono converse_refl*[*OF Colleq_refl*]
by *auto*

```
    then
    show ?thesis
      by auto
  qed
  moreover from calculation
  have ?rnf ⊆ (ℵ_1^M →^##M (nat →^M 2))
  using subset_trans[OF image_subset[OF ‹f⊆_›,of ω] Fn_rel_subset_PFun_rel]
    by simp
  moreover
  have ⋃ ?rnf ∈ (ℵ_1^M →^##M (nat →^M 2))
    using pfun_Un_filter_closed'[OF ‹?rnf⊆_› 1] ‹?rnf∈M›
    by simp
  moreover from calculation
  have ⋃ ?rnf ≺^M ℵ_1^M
    using countable_fun_imp_countable_image[of f]
     mem_function_space_rel_abs[simplified,OF nat_in_M Coll_in_M ‹f∈M›]
      countableI[OF lepoll_rel_refl]
      countable_iff_lesspoll_rel_Aleph_rel_one[of ⋃ ?rnf]
    by auto
  moreover from calculation
  have ⋃ ?rnf∈Coll
    unfolding Fn_rel_def
    by simp
  moreover from calculation
  have ⋃ ?rnf ⊇ f ‘ α  if α∈ω for α
    using that image_function[OF fun_is_function] domain_of_fun
    by auto
  ultimately
  show ∃ q∈M. q ∈ Coll ∧ (∀ α∈M. α ∈ ω ⟶ ⟨q, f ‘ α⟩ ∈ Colleq)
    using Fn_rel_is_function Fnle_relI
    by auto
 qed
 ultimately
 show ?thesis
   unfolding kappa_closed_rel_def by (auto elim!:leE dest:ltD)
qed

end — M_ctm2_AC

locale collapse_CH = G_generic3_AC_CH Fn^M(ℵ_1^##M, ℵ_1^M, ω →^M 2) Fnle^M(ℵ_1^##M,
ℵ_1^M, ω →^M 2) 0

sublocale collapse_CH ⊆ forcing_notion Coll Colleq 0
  using zero_lt_Aleph_rel1 by unfold_locales

context collapse_CH
begin

notation Leq (infixl ‹⪅› 50)
```

notation *Incompatible* (**infixl** ‹⊥› *50*)

abbreviation
  f_G :: i (‹f_G›) **where**
  $f_G \equiv \bigcup G$

lemma *f_G_in_MG*[*simp*]:
  shows $f_G \in M[G]$
  using *G_in_MG* **by** *simp*

abbreviation
  dom_dense :: $i \Rightarrow i$ **where**
  $dom\_dense(x) \equiv \{ p \in Coll . x \in domain(p) \}$

lemma *dom_dense_closed*[*intro*,*simp*]: $x \in M \Longrightarrow dom\_dense(x) \in M$
  using *separation_in_domain*[*of x*]
  by *simp*

lemma *domain_f_G*: **assumes** $x \in \aleph_1{}^M$
  shows $x \in domain(f_G)$
proof -
  have $(\lambda n \in \omega.\ 0) \in \omega \to^M 2$
    using *function_space_rel_nonempty*[*of 0 2 ω*]
    by *auto*
  with *assms*
  have $dense(dom\_dense(x))\ x \in M$
    using *dense_dom_dense InfCard_rel_Aleph_rel*[*of 1*] *transitivity*[*OF _*
      Aleph_rel_closed[*of 1*,*THEN setclass_iff*[*THEN iffD1*]]]
    unfolding *dense_def*
     by *auto*
  with *assms*
  obtain p **where** $p \in dom\_dense(x)\ p \in G$
    using *M_generic_denseD*[*of dom_dense*(*x*)]
    by *auto*
  then
  show $x \in domain(f_G)$ **by** *blast*
qed

lemma *Un_filter_is_function*:
  assumes *filter*(G)
  shows *function*($\bigcup G$)
proof -
  have $Coll \subseteq \aleph_1{}^M \rightharpoonup \#\#^M\ (\omega \to^M 2)$
    using *Fn_rel_subset_PFun_rel*
    by *simp*
  moreover
  have $\exists\ d \in Coll.\ d \supseteq f \wedge d \supseteq g$ **if** $f \in G\ g \in G$ **for** $f\ g$
    using *filter_imp_compat*[*OF assms* ‹$f \in G$› ‹$g \in G$›] *filterD*[*OF assms*]
    unfolding *compat_def compat_in_def*

```
    by auto
  ultimately
  have ∃ d ∈ ℵ₁^M ⇀##M (ω →^M 2). d ⊇ f ∧ d ⊇ g if f∈G g∈G for f g
    using rex_mono[of Coll] that by simp
  moreover
  have G⊆Coll
    using assms
    unfolding filter_def
    by simp
  moreover from this
  have G ⊆ ℵ₁^M ⇀##M (ω →^M 2)
    using assms unfolding Fn_rel_def
    by auto
  ultimately
  show ?thesis
    using pfun_Un_filter_closed[of G]
    by simp
qed

lemma f_G_funtype:
  shows f_G : ℵ₁^M → ω →^M[G] 2
proof -
  have x ∈ B ⟹ B ∈ G ⟹ x ∈ ℵ₁^M × (ω →^M[G] 2) for B x
  proof -
    assume x∈B B∈G
    moreover from this
    have x ∈ M[G]
      by (auto dest!: ext.transM simp add:G_in_MG)
    moreover from calculation
    have x ∈ ℵ₁^M × (ω → 2)
      using Fn_rel_subset_Pow[of ℵ₁^M ℵ₁^M ω →^M 2,
          OF _ _ function_space_rel_closed] function_space_rel_char
      by (auto dest!: M_genericD)
    moreover from this
    obtain z w where x=⟨z,w⟩ z∈ℵ₁^M w:ω → 2 by auto
    moreover from calculation
    have w ∈ M[G] by (auto dest:ext.transM)
    ultimately
    show ?thesis using ext.function_space_rel_char by auto
  qed
  moreover
  have function(f_G)
    using Un_filter_is_function generic
    by fast
  ultimately
  show ?thesis
    using generic domain_f_G Pi_iff by auto
qed
```

abbreviation
surj_dense :: $i \Rightarrow i$ **where**
$surj\_dense(x) \equiv \{\ p \in Coll\ .\ x \in range(p)\ \}$

lemma *surj_dense_closed*[*intro*,*simp*]:
$x \in \omega \rightarrow^{M} 2 \Longrightarrow surj\_dense(x) \in M$
using *separation_in_range transM*[*of x*] **by** *simp*

lemma *reals_sub_image_f_G*:
assumes $x \in \omega \rightarrow^{M} 2$
shows $\exists \alpha \in \aleph_1{}^{M}.\ f_G\ `\ \alpha = x$
proof -
from *assms*
have *dense*(*surj_dense*(*x*))
using *dense_surj_dense lepoll_rel_refl InfCard_rel_Aleph_rel*
unfolding *dense_def*
by *auto*
with ‹$x \in \omega \rightarrow^{M} 2$›
obtain *p* **where** $p \in surj\_dense(x)$ $p \in G$
using *M_generic_denseD*[*of surj_dense*(*x*)]
by *blast*
then
show *?thesis*
using *succ_omega_closed_Coll f_G_funtype function_apply_equality*[*of _ x f_G*]
succ_omega_closed_imp_no_new_reals[*symmetric*]
by (*auto*, *rule_tac bexI*) (*auto simp*:*Pi_def*)
qed

lemma *f_G_surj_Aleph_rel1_reals*: $f_G \in surj^{M[G]}(\aleph_1{}^{M}, \omega \rightarrow^{M[G]} 2)$
using *Aleph_rel_sub_closed*
proof (*intro ext.mem_surj_abs*[*THEN iffD2*],*simp_all*)
show $f_G \in surj(\aleph_1{}^{M}, \omega \rightarrow^{M[G]} 2)$
using *f_G_funtype G_in_MG ext.nat_into_M f_G_in_MG*
reals_sub_image_f_G succ_omega_closed_Coll
succ_omega_closed_imp_no_new_reals
unfolding *surj_def*
by *auto*
qed

lemma *continuum_rel_le_Aleph1_extension*:
includes *G_generic1_lemmas*
shows $2^{\uparrow \aleph_0{}^{M[G]}, M[G]} \leq \aleph_1{}^{M[G]}$
proof -
have $\aleph_1{}^{M} \in M[G]$ $Ord(\aleph_1{}^{M})$
using *Card_rel_is_Ord* **by** *auto*
moreover from *this*
have $\omega \rightarrow^{M[G]} 2 \lesssim^{M[G]} \aleph_1{}^{M}$
using *ext.surj_rel_implies_inj_rel*[*OF f_G_surj_Aleph_rel1_reals*]

f_G_in_MG **unfolding** *lepoll_rel_def* **by** *auto*
with ‹$Ord(\aleph_1{}^M)$›
have $|\omega \to^{M[G]} 2|^{M[G]} \leq |\aleph_1{}^M|^{M[G]}$
using *M_subset_MG*[*OF one_in_G*] *Aleph_rel_closed*[*of 1*]
by (*rule_tac ext.lepoll_rel_imp_cardinal_rel_le*) *simp_all*
ultimately
have $2^{\uparrow \aleph_0{}^{M[G]}, M[G]} \leq |\aleph_1{}^{M[G]}|^{M[G]}$
using *ext.lepoll_rel_imp_cardinal_rel_le*[*of* $\aleph_1{}^M$ $\omega \to^{M[G]} 2$]
ext.Aleph_rel_zero succ_omega_closed_Coll
succ_omega_closed_imp_Aleph_1_preserved
unfolding *cexp_rel_def* **by** *simp*
then
show $2^{\uparrow \aleph_0{}^{M[G]}, M[G]} \leq \aleph_1{}^{M[G]}$ **by** *simp*
qed

theorem *CH*: $\aleph_1{}^{M[G]} = 2^{\uparrow \aleph_0{}^{M[G]}, M[G]}$
using *continuum_rel_le_Aleph1_extension ext.Aleph_rel_succ*[*of 0*]
ext.Aleph_rel_zero ext.csucc_rel_le[*of* $2^{\uparrow \aleph_0{}^{M[G]}, M[G]}$ ω]
ext.Card_rel_cexp_rel ext.cantor_cexp_rel[*of* ω] *ext.Card_rel_nat*
le_anti_sym
by *auto*

end — *collapse_CH*

31.2 Models of fragments of $ZFC + CH$

theorem *ctm_of_CH*:
assumes
$M \approx \omega$ *Transset*(M)
$M \models ZC \cup \{\cdot Replacement(p) \cdot . p \in overhead\_CH\}$
$\Phi \subseteq formula$ $M \models \{ \cdot Replacement(ground\_repl\_fm(\varphi)) \cdot . \varphi \in \Phi\}$
shows
$\exists N.$
$M \subseteq N \wedge N \approx \omega \wedge Transset(N) \wedge N \models ZC \cup \{\cdot CH \cdot\} \cup \{ \cdot Replacement(\varphi) \cdot . \varphi \in \Phi\} \wedge$
$(\forall \alpha.\ Ord(\alpha) \longrightarrow (\alpha \in M \longleftrightarrow \alpha \in N))$
proof -
from ‹$M \models ZC \cup \{\cdot Replacement(p) \cdot . p \in overhead\_CH\}$›
interpret *M_ZFC3 M*
using *M_satT_overhead_imp_M_ZF3* **unfolding** *overhead_CH_def overhead_notCH_def* **by** *auto*
from ‹$M \models ZC \cup \{\cdot Replacement(p) \cdot . p \in overhead\_CH\}$› ‹$Transset(M)$›
interpret *M_ZF_ground_CH_trans M*
using *M_satT_imp_M_ZF_ground_CH_trans*
unfolding *ZC_def* **by** *auto*
from ‹$M \approx \omega$›
obtain *enum* **where** $enum \in bij(\omega, M)$
using *eqpoll_sym* **unfolding** *eqpoll_def* **by** *blast*
then

interpret *M_ctm2_AC_CH M enum* **by** *unfold_locales*
interpret *forcing_data1 Coll Colleq 0 M enum*
using *zero_in_Fn_rel*[*of* $\aleph_1{}^M$ $\aleph_1{}^M$ ω $\to^M$ *2*]
zero_top_Fn_rel[*of* _ $\aleph_1{}^M$ $\aleph_1{}^M$ ω $\to^M$ *2*]
preorder_on_Fnle_rel[*of* $\aleph_1{}^M$ $\aleph_1{}^M$ ω $\to^M$ *2*]
zero_lt_Aleph_rel1
by *unfold_locales simp_all*
obtain *G* **where** *M_generic*(*G*)
using *generic_filter_existence*[*OF one_in_P*]
by *auto*
moreover from *this*
interpret *collapse_CH M enum G* **by** *unfold_locales*
have $\aleph_1{}^{M[G]} = 2^{\uparrow \aleph_0{}^{M[G]}, M[G]}$
using *CH* **.**
then
have *M*[*G*], [] ⊨ ·*CH*·
using *ext.is_ContHyp_iff*
by (*simp add*:*ContHyp_rel_def*)
then
have *M*[*G*] ⊨ *ZC* ∪ {·*CH*·}
using *ext.M_satT_ZC* **by** *auto*
moreover
have *Transset*(*M*[*G*]) **using** *Transset_MG* **.**
moreover
have *M* ⊆ *M*[*G*] **using** *M_subset_MG*[*OF one_in_G*] *generic* **by** *simp*
moreover
note ‹*M* ⊨ { ·*Replacement*(*ground_repl_fm*(*φ*))· . *φ* ∈ Φ}› ‹Φ ⊆ *formula*›
ultimately
show *?thesis*
using *Ord_MG_iff MG_eqpoll_nat satT_ground_repl_fm_imp_satT_ZF_replacement_fm*[*of* Φ]
by (*rule_tac x*=*M*[*G*] **in** *exI*,*blast*)
qed

corollary *ctm_ZFC_imp_ctm_CH*:
assumes
M ≈ *ω* *Transset*(*M*) *M* ⊨ *ZFC*
shows
∃ *N*.
M ⊆ *N* ∧ *N* ≈ *ω* ∧ *Transset*(*N*) ∧ *N* ⊨ *ZFC* ∪ {·*CH*·} ∧
(∀ *α*. *Ord*(*α*) ⟶ (*α* ∈ *M* ⟷ *α* ∈ *N*))
proof -
from *assms*
have ∃ *N*.
M ⊆ *N* ∧
N ≈ *ω* ∧
Transset(*N*) ∧
N ⊨ *ZC* ∧ *N* ⊨ {·*CH*·} ∧ *N* ⊨ {·*Replacement*(*x*)· . *x* ∈ *formula*} ∧ (∀ *α*. *Ord*(*α*) ⟶ *α* ∈ *M* ⟷ *α* ∈ *N*)

using *ctm_of_CH*[*of M formula*] *satT_ZFC_imp_satT_ZC*[*of M*]
satT_mono[*OF _ ground_repl_fm_sub_ZFC*, *of M*]
satT_mono[*OF _ ZF_replacement_overhead_CH_sub_ZFC*, *of M*]
satT_mono[*OF _ ZF_replacement_fms_sub_ZFC*, *of M*]
by (*simp add*: *satT_Un_iff*)
then
obtain *N* **where** $N \models ZC$ $N \models \{\cdot CH \cdot\}$ $N \models \{\cdot Replacement(x) \cdot \ldots x \in formula\}$
$M \subseteq N$ $N \approx \omega$ $Transset(N)$ $(\forall\, \alpha.\ Ord(\alpha) \longrightarrow \alpha \in M \longleftrightarrow \alpha \in N)$
by *auto*
moreover from *this*
have $N \models ZFC$
using *satT_ZC_ZF_replacement_imp_satT_ZFC*
by *auto*
moreover from *this* **and** $\langle N \models \{\cdot CH \cdot\}\rangle$
have $N \models ZFC \cup \{\cdot CH \cdot\}$
using *satT_ZC_ZF_replacement_imp_satT_ZFC*
by *auto*
ultimately
show *?thesis*
by *auto*
qed

end

32 From M to $\mathcal{V}$

theory *Absolute_Versions*
imports
CH
ZF.Cardinal_AC
begin

hide_const (**open**) *Order.pred*

32.1 Locales of a class M hold in $\mathcal{V}$

interpretation *V*: *M_trivial* $\mathcal{V}$
using *Union_ax_absolute upair_ax_absolute*
by *unfold_locales auto*

lemmas *bad_simps* = *V.nonempty V.Forall_in_M_iff V.Inl_in_M_iff V.Inr_in_M_iff*
V.succ_in_M_iff V.singleton_in_M_iff V.Equal_in_M_iff V.Member_in_M_iff
V.Nand_in_M_iff
V.Cons_in_M_iff V.pair_in_M_iff V.upair_in_M_iff

lemmas *bad_M_trivial_simps*[*simp del*] = *V.Forall_in_M_iff V.Equal_in_M_iff*
V.nonempty

lemmas *bad_M_trivial_rules*[*rule del*] = *V.pair_in_MI V.singleton_in_MI*

```
V.pair_in_MD V.nat_into_M
  V.depth_closed V.length_closed V.nat_case_closed V.separation_closed
  V.Un_closed V.strong_replacement_closed V.nonempty

interpretation V:M_basic 𝒱
  using power_ax_absolute separation_absolute replacement_absolute
  by unfold_locales auto

interpretation V:M_eclose 𝒱
  by unfold_locales (auto intro:separation_absolute replacement_absolute
      simp:iterates_replacement_def wfrec_replacement_def)

lemmas bad_M_basic_rules[simp del, rule del] =
  V.cartprod_closed V.finite_funspace_closed V.converse_closed
  V.list_case'_closed V.pred_closed

interpretation V:M_cardinal_arith 𝒱
  by unfold_locales (auto intro:separation_absolute replacement_absolute
    simp add:iterates_replacement_def wfrec_replacement_def lam_replacement_def)

lemmas bad_M_cardinals_rules[simp del, rule del] =
  V.iterates_closed V.M_nat V.trancl_closed V.rvimage_closed

interpretation V:M_cardinal_arith_jump 𝒱
  by unfold_locales (auto intro:separation_absolute replacement_absolute
      simp:wfrec_replacement_def)

lemma choice_ax_Universe: choice_ax(𝒱)
proof -
  {
    fix x
    obtain f where f ∈ surj(|x|,x)
      using cardinal_eqpoll unfolding eqpoll_def bij_def by fast
    moreover
    have Ord(|x|) by simp
    ultimately
    have ∃ a. Ord(a) ∧ (∃ f. f ∈ surj(a,x))
      by fast
  }
  then
  show ?thesis  unfolding choice_ax_def rall_def rex_def
    by simp
qed

interpretation V:M_master 𝒱
  using choice_ax_Universe
  by unfold_locales (auto intro:separation_absolute replacement_absolute
      simp:lam_replacement_def transrec_replacement_def wfrec_replacement_def
      is_wfrec_def M_is_recfun_def)
```

named_theorems *V_simps*

— To work systematically, ASCII versions of "_absolute" theorems as those below are preferable.
lemma *eqpoll_rel_absolute*[*V_simps*]: $x \approx^{\mathcal{V}} y \longleftrightarrow x \approx y$
unfolding *eqpoll_def* **using** *V.def_eqpoll_rel* **by** *auto*

lemma *cardinal_rel_absolute*[*V_simps*]: $|x|^{\mathcal{V}} = |x|$
unfolding *cardinal_def cardinal_rel_def* **by** (*simp add*: *V_simps*)

lemma *Card_rel_absolute*[*V_simps*]:$Card^{\mathcal{V}}(a) \longleftrightarrow Card(a)$
unfolding *Card_rel_def Card_def* **by** (*simp only*: *V_simps*)

lemma *csucc_rel_absolute*[*V_simps*]:$(a^+)^{\mathcal{V}} = a^+$
unfolding *csucc_rel_def csucc_def* **by** (*simp add*: *V_simps*)

lemma *function_space_rel_absolute*[*V_simps*]:$x \rightarrow^{\mathcal{V}} y = x \rightarrow y$
using *V.function_space_rel_char* **by** (*simp add*: *V_simps*)

lemma *cexp_rel_absolute*[*V_simps*]:$x^{\uparrow y,\mathcal{V}} = x^{\uparrow y}$
unfolding *cexp_rel_def cexp_def* **by** (*simp only*: *V_simps*)

lemma *HAleph_rel_absolute*[*V_simps*]:*HAleph_rel*($\mathcal{V}$,*a*,*b*) = *HAleph*(*a*,*b*)
unfolding *HAleph_rel_def HAleph_def* **by** (*auto simp add*: *V_simps*)

lemma *Aleph_rel_absolute*[*V_simps*]: $Ord(x) \Longrightarrow \aleph_x{}^{\mathcal{V}} = \aleph_x$
proof -
assume *Ord*(*x*)
have $\aleph_x{}^{\mathcal{V}}$ = *transrec*(*x*, $\lambda a\ b$. *HAleph_rel*($\mathcal{V}$,*a*,*b*))
unfolding *Aleph_rel_def* **by** *simp*
also
have ... = *transrec*(*x*, *HAleph*)
by (*simp only*: *V_simps*)
also from ‹*Ord*(*x*)›
have ... = $\aleph_x$
using *Aleph'_eq_Aleph* **unfolding** *Aleph'_def* **by** *simp*
finally
show *?thesis* **.**
qed

Example of absolute lemmas obtained from the relative versions. Note the *only* declarations

lemma *Ord_cardinal_idem'*: $Ord(A) \Longrightarrow ||A|| = |A|$
using *V.Ord_cardinal_rel_idem* **by** (*simp only*: *V_simps*)

lemma *Aleph_succ'*: $Ord(\alpha) \Longrightarrow \aleph_{succ(\alpha)} = \aleph_\alpha{}^+$
using *V.Aleph_rel_succ* **by** (*simp only*: *V_simps*)

These two results are new, first obtained in relative form (not ported).

lemma *csucc_cardinal*:
  assumes $Ord(\kappa)$ **shows** $|\kappa|^+ = \kappa^+$
  using *assms V.csucc_rel_cardinal_rel* **by** (*simp only*:*V_simps*)

lemma *csucc_le_mono*:
  assumes $\kappa \leq \nu$ **shows** $\kappa^+ \leq \nu^+$
  using *assms V.csucc_rel_le_mono* **by** (*simp only*:*V_simps*)

Example of transferring results from a transitive model to $\mathcal{V}$

lemma (**in** *M_Perm*) *eqpoll_rel_transfer_absolute*:
  assumes $M(A)$ $M(B)$ $A \approx^M B$
  shows $A \approx B$
proof -
  interpret *M_N_Perm M* $\mathcal{V}$
    by (*unfold_locales*, *simp only*:*V_simps*)
  from *assms*
  show *?thesis* **using** *eqpoll_rel_transfer*
    by (*simp only*:*V_simps*)
qed

The "relationalized" *CH* with respect to $\mathcal{V}$ corresponds to the real *CH*.

lemma *is_ContHyp_iff_CH*: *is_ContHyp*($\mathcal{V}$) $\longleftrightarrow$ *ContHyp*
  using *V.is_ContHyp_iff*
  by (*auto simp add*:*ContHyp_rel_def ContHyp_def V_simps*)

end

33 Main definitions of the development

theory *Definitions_Main*
  imports
    Absolute_Versions
begin

This theory gathers the main definitions of the `Transitive_Models` session and the present one.

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

The main theorem of this session and all of its relevant definitions appear in Section 33.4. The reader trusting all the libraries on which our development is based, might jump directly to Section 33.3, which treats relative cardinal arithmetic as implemented in `Transitive_Models`. But in case one wants to dive deeper, the following sections treat some basic concepts of the ZF

logic (Section 33.1) and in the ZF-Constructible library (Section 33.2) on which our definitions are built.

declare [[*show_question_marks*=*false*]]

33.1 ZF

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

thm *bij_def*[*unfolded inj_def surj_def*]

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

thm *eqpoll_def*

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

thm *Transset_def*

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

thm *Ord_def*

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

thm *lt_def le_iff*

$$i < j \equiv i \in j \wedge Ord(j) \\ i \leq j \longleftrightarrow i < j \vee i = j \wedge Ord(j)$$

With the concepts of empty set and successor in place,

lemma *empty_def′*: $\forall x.\ x \notin 0$ **by** *simp*
lemma *succ_def′*: $succ(i) = i \cup \{i\}$ **by** *blast*

we can define the set of natural numbers ω. In the sources, it is defined as a fixpoint, but here we just write its characterization as the first limit ordinal.

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

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

Then, addition and predecessor on ω are inductively characterized as follows:

thm *add_0_right add_succ_right pred_0 pred_succ_eq*

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

Lists on a set A can be characterized by being recursively generated from the empty list [] and the operation *Cons* that adds a new element to the left end; the induction theorem for them shows that the characterization is "complete".

thm *Nil Cons list.induct*

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

Length, concatenation, and nth element of lists are recursively characterized as follows.

thm *length.simps app.simps nth_0 nth_Cons*

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

We have the usual Haskell-like notation for iterated applications of *Cons*:

lemma *Cons_app*: $[a,b,c] = Cons(a,Cons(b,Cons(c,[])))$ **..**

Relative quantifiers restrict the range of the bound variable to a class M of type $i \Rightarrow o$; that is, a truth-valued function with set arguments.

lemma $\forall x[M].\ P(x) \equiv \forall x.\ M(x) \longrightarrow P(x)$
$\exists x[M].\ P(x) \equiv \exists x.\ M(x) \wedge P(x)$
unfolding *rall_def rex_def* **.**

Finally, a set can be viewed ("cast") as a class using the following function of type $i \Rightarrow i \Rightarrow o$.

thm *setclass_iff*

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

33.2 Relative concepts

A list of relative concepts (mostly from the ZF-Constructible library) follows next.

thm *big_union_def*

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

thm *upair_def*

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

thm *pair_def*

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

thm *successor_def*[*unfolded is_cons_def union_def*]

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

thm *empty_def*

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

thm *transitive_set_def*[*unfolded subset_def*]

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

thm *ordinal_def*

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

thm *image_def*

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

thm *fun_apply_def*

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

thm *is_function_def*

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

thm *is_relation_def*

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

thm *is_domain_def*

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

thm *typed_function_def*

$$typed\_function(M,\ A,\ B,\ r) \equiv$$
$$is\_function(M,\ r) \wedge$$
$$is\_relation(M,\ r) \wedge$$
$$is\_domain(M,\ r,\ A) \wedge$$
$$(\forall\, u[M].\ u \in r \longrightarrow (\forall\, x[M].\ \forall\, y[M].\ pair(M,\ x,\ y,\ u) \longrightarrow y \in B))$$

thm *is_function_space_def*[*unfolded is_funspace_def*]
function_space_rel_def surjection_def

$$is\_function\_space(M,\ A,\ B,\ fs) \equiv$$
$$M(fs) \wedge (\forall\, f[M].\ f \in fs \longleftrightarrow typed\_function(M,\ A,\ B,\ f))$$
$$A \to^M B \equiv THE\ d.\ is\_function\_space(M,\ A,\ B,\ d)$$
$$surjection(M,\ A,\ B,\ f) \equiv$$
$$typed\_function(M,\ A,\ B,\ f) \wedge$$
$$(\forall\, y[M].\ y \in B \longrightarrow (\exists\, x[M].\ x \in A \wedge is\_apply(M,\ f,\ x,\ y)))$$

Relative version of the *ZFC* axioms

thm *extensionality_def*

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

thm *foundation_ax_def*

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

thm *upair_ax_def*

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

thm *Union_ax_def*

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

thm *power_ax_def*[*unfolded powerset_def subset_def*]

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

thm *infinity_ax_def*

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

thm *choice_ax_def*

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

thm *separation_def*

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

thm *univalent_def*

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

thm *strong_replacement_def*

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

Internalized formulas

"Codes" for formulas (as sets) are constructed from natural numbers using *Member*, *Equal*, *Nand*, and *Forall*.

thm *Member Equal Nand Forall formula.induct*

$[\![x \in \omega;\ y \in \omega]\!] \Longrightarrow \cdot x \in y\cdot \in formula$
$[\![x \in \omega;\ y \in \omega]\!] \Longrightarrow \cdot x = y\cdot \in formula$
$[\![p \in formula;\ q \in formula]\!] \Longrightarrow \cdot\neg(p \wedge q)\cdot \in formula$
$p \in formula \Longrightarrow (\cdot\forall\, p\cdot) \in formula$
$[\![x \in formula;\ \bigwedge x\ y.\ [\![x \in \omega;\ y \in \omega]\!] \Longrightarrow P(\cdot x \in y\cdot);$
$\ \bigwedge x\ y.\ [\![x \in \omega;\ y \in \omega]\!] \Longrightarrow P(\cdot x = y\cdot);$
$\ \bigwedge p\ q.\ [\![p \in formula;\ P(p);\ q \in formula;\ P(q)]\!] \Longrightarrow P(\cdot\neg(p \wedge q)\cdot);$
$\ \bigwedge p.\ [\![p \in formula;\ P(p)]\!] \Longrightarrow P((\cdot\forall\, p\cdot))]\!]$
$\Longrightarrow P(x)$

Definitions for the other connectives and the internal existential quantifier are also provided. For instance, negation:

thm *Neg_def*

$\cdot\neg p\cdot \equiv \cdot\neg(p \wedge p)\cdot$

thm *arity.simps*

$arity(\cdot x \in y\cdot) = succ(x) \cup succ(y)$
$arity(\cdot x = y\cdot) = succ(x) \cup succ(y)$
$arity(\cdot\neg(p \wedge q)\cdot) = arity(p) \cup arity(q)$
$arity((\cdot\forall\, p\cdot)) = pred(arity(p))$

We have the satisfaction relation between $\in$-models and first order formulas (given a "environment" list representing the assignment of free variables),

thm *mem_iff_sats equal_iff_sats sats_Nand_iff sats_Forall_iff*

$[\![nth(i,\ env) = x;\ nth(j,\ env) = y;\ env \in list(A)]\!]$
$\Longrightarrow x \in y \longleftrightarrow A,\ env \models \cdot i \in j\cdot$
$[\![nth(i,\ env) = x;\ nth(j,\ env) = y;\ env \in list(A)]\!]$
$\Longrightarrow x = y \longleftrightarrow A,\ env \models \cdot i = j\cdot$
$env \in list(A) \Longrightarrow (A,\ env \models \cdot\neg(p \wedge q)\cdot) \longleftrightarrow \neg\ ((A,\ env \models p) \wedge (A,\ env \models q))$
$env \in list(A) \Longrightarrow (A,\ env \models (\cdot\forall\, p\cdot)) \longleftrightarrow (\forall\, x{\in}A.\ A,\ Cons(x,\ env) \models p)$

as well as the satisfaction of an arbitrary set of sentences.

thm *satT_def*

$A \models \Phi \equiv \forall\, \varphi{\in}\Phi.\ A,\ [] \models \varphi$

The internalized (viz. as elements of the set *formula*) version of the axioms follow next.

thm *ZF_union_iff_sats ZF_power_iff_sats ZF_pairing_iff_sats*
ZF_foundation_iff_sats ZF_extensionality_iff_sats
ZF_infinity_iff_sats sats_ZF_separation_fm_iff
sats_ZF_replacement_fm_iff ZF_choice_iff_sats

$Union\_ax(\#\#A) \longleftrightarrow A,\ [] \models \cdot Union\ Ax\cdot$
$power\_ax(\#\#A) \longleftrightarrow A,\ [] \models \cdot Powerset\ Ax\cdot$
$upair\_ax(\#\#A) \longleftrightarrow A,\ [] \models \cdot Pairing\cdot$
$foundation\_ax(\#\#A) \longleftrightarrow A,\ [] \models \cdot Foundation\cdot$
$extensionality(\#\#A) \longleftrightarrow A,\ [] \models \cdot Extensionality\cdot$
$infinity\_ax(\#\#A) \longleftrightarrow A,\ [] \models \cdot Infinity\cdot$
$\varphi \in formula \Longrightarrow$
$(M,\ [] \models \cdot Separation(\varphi)\cdot) \longleftrightarrow$
$(\forall\, env{\in}list(M).$
$\quad arity(\varphi) \le 1 +_\omega length(env) \longrightarrow separation(\#\#M,\ \lambda x.\ M,\ [x]\ @\ env \models \varphi))$
$\varphi \in formula \Longrightarrow$
$(M,\ [] \models \cdot Replacement(\varphi)\cdot) \longleftrightarrow (\forall\, env.\ replacement\_assm(M,\ env,\ \varphi))$
$choice\_ax(\#\#A) \longleftrightarrow A,\ [] \models \cdot AC\cdot$

Above, we use the following:

thm *replacement_assm_def*

$replacement\_assm(M,\ env,\ \varphi) \equiv$
$\varphi \in formula \longrightarrow$
$env \in list(M) \longrightarrow$
$arity(\varphi) \le 2 +_\omega length(env) \longrightarrow$
$strong\_replacement(\#\#M,\ \lambda x\ y.\ M,\ [x,\ y]\ @\ env \models \varphi)$

Finally, the axiom sets are defined as follows.

thm *ZF_fin_def ZF_schemes_def Zermelo_fms_def ZC_def ZF_def ZFC_def*

ZF_fin ≡
{·*Extensionality*·, ·*Foundation*·, ·*Pairing*·, ·*Union Ax*·, ·*Infinity*·,
·*Powerset Ax*·}
ZF_schemes ≡
{·*Separation*(*p*)· . *p* ∈ *formula*} ∪ {·*Replacement*(*p*)· . *p* ∈ *formula*}
·*Z*· ≡ *ZF_fin* ∪ {·*Separation*(*p*)· . *p* ∈ *formula*}
ZC ≡ ·*Z*· ∪ {·*AC*·}
ZF ≡ *ZF_schemes* ∪ *ZF_fin*
ZFC ≡ *ZF* ∪ {·*AC*·}

33.3 Relativization of infinitary arithmetic

In order to state the defining property of the relative equipotence relation, we work under the assumptions of the locale *M_cardinals*. They comprise a finite set of instances of Separation and Replacement to prove closure properties of the transitive class *M*.

lemma (**in** *M_cardinals*) *eqpoll_def'*:
  assumes *M*(*A*) *M*(*B*) **shows** *A* ≈^{M} *B* ⟷ (∃*f*[*M*]. *f* ∈ *bij*(*A*,*B*))
  using *assms* **unfolding** *eqpoll_rel_def* **by** *auto*

Below, μ denotes the minimum operator on the ordinals.

lemma *cardinalities_defs*:
  fixes *M*::*i*⇒*o*
  shows
    |*A*|^M ≡ μ *i*. *M*(*i*) ∧ *i* ≈^{M} *A*
    Card^M(α) ≡ α = |α|^M
    $\kappa^{\uparrow\nu,M}$ ≡ |ν →^{M} κ|^M
    $(\kappa^+)^M$ ≡ μ *x*. *M*(*x*) ∧ *Card*^{M}(*x*) ∧ κ < *x*
  unfolding *cardinal_rel_def cexp_rel_def*
    csucc_rel_def Card_rel_def **.**

context *M_aleph*
begin

Analogous to the previous Lemma *eqpoll_def'*, we are now under the assumptions of the locale *M_aleph*. The axiom instances included are sufficient to state and prove the defining properties of the relativized *Aleph* function (in particular, the required ability to perform transfinite recursions).

thm *Aleph_rel_zero Aleph_rel_succ Aleph_rel_limit*

$\aleph_0{}^M = \omega$
⟦*Ord*(α); *M*(α)⟧ ⟹ $\aleph_{succ(\alpha)}{}^M = (\aleph_\alpha{}^{M+})^M$
⟦*Limit*(α); *M*(α)⟧ ⟹ $\aleph_\alpha{}^M = (\bigcup j{\in}\alpha.\ \aleph_j{}^M)$

end — *M_aleph*

lemma *ContHyp_rel_def'*:
fixes $N{::}i{\Rightarrow}o$
shows
$CH^N \equiv \aleph_1{}^N = 2^{\uparrow\aleph_0{}^N,N}$
unfolding *ContHyp_rel_def* **.**

Under appropriate hypotheses (this time, from the locale *M_ZF_library*), CH^M is equivalent to its fully relational version *is_ContHyp*. As a sanity check, we see that if the transitive class is indeed $\mathcal{V}$, we recover the original CH.

thm *M_ZF_library.is_ContHyp_iff is_ContHyp_iff_CH*[*unfolded ContHyp_def*]

$M\_ZF\_library(M) \Longrightarrow is\_ContHyp(M) \longleftrightarrow CH^M$
$is\_ContHyp(\mathcal{V}) \longleftrightarrow \aleph_1 = 2^{\uparrow\aleph_0}$

In turn, the fully relational version evaluated on a nonempty transitive A is equivalent to the satisfaction of the first-order formula $\cdot CH\cdot$.

thm *is_ContHyp_iff_sats*

$[\![env \in list(A);\ 0 \in A]\!] \Longrightarrow is\_ContHyp(\#\#A) \longleftrightarrow A,\ env \models \cdot CH\cdot$

33.4 Forcing

Our first milestone was to obtain a proper extension using forcing. Its original proof didn't required the previous developments involving the relativization of material on cardinal arithmetic. Now it is derived from a stronger result, namely *extensions_of_ctms* below.

thm *extensions_of_ctms_ZF*

$[\![M \approx \omega;\ Transset(M);\ M \models ZF]\!]$
$\Longrightarrow \exists N.\ M \subseteq N \wedge$
$N \approx \omega \wedge$
$Transset(N) \wedge$
$N \models ZF \wedge$
$M \neq N \wedge (\forall \alpha.\ Ord(\alpha) \longrightarrow \alpha \in M \longleftrightarrow \alpha \in N) \wedge ((M, [] \models \cdot AC\cdot) \longrightarrow N \models ZFC)$

We can finally state our main results, namely, the existence of models for $ZFC + CH$ and $ZFC + \neg CH$ under the assumption of a ctm of ZFC.

thm *ctm_ZFC_imp_ctm_not_CH*

$[\![M \approx \omega;\ \mathit{Transset}(M);\ M \models \mathit{ZFC}]\!]$
$\Longrightarrow \exists N.\ M \subseteq N \wedge$
$\quad N \approx \omega \wedge$
$\quad \mathit{Transset}(N) \wedge N \models \mathit{ZFC} \cup \{\cdot\neg\cdot \mathit{CH}\cdot\cdot\} \wedge (\forall \alpha.\ \mathit{Ord}(\alpha) \longrightarrow \alpha \in M \longleftrightarrow \alpha \in N)$

thm *ctm_ZFC_imp_ctm_CH*

$[\![M \approx \omega;\ \mathit{Transset}(M);\ M \models \mathit{ZFC}]\!]$
$\Longrightarrow \exists N.\ M \subseteq N \wedge$
$\quad N \approx \omega \wedge$
$\quad \mathit{Transset}(N) \wedge N \models \mathit{ZFC} \cup \{\cdot \mathit{CH}\cdot\} \wedge (\forall \alpha.\ \mathit{Ord}(\alpha) \longrightarrow \alpha \in M \longleftrightarrow \alpha \in N)$

These results can be strengthened by enumerating six finite sets of replacement instances which are sufficient to develop forcing and for the construction of the aforementioned models: *instances1_fms* through *instances3_fms*, *instances_ground_fms*, and *instances_ground_notCH_fms*, which are then collected into the 31-element set *overhead_notCH*. For example, we have:

thm *instances1_fms_def*

instances1_fms ≡
{*eclose_closed_fm*, *eclose_abs_fm*, *wfrec_rank_fm*, *transrec_VFrom_fm*}

thm *overhead_def overhead_notCH_def*

overhead ≡ *instances1_fms* ∪ *instances_ground_fms*
overhead_notCH ≡
overhead ∪ *instances2_fms* ∪ *instances3_fms* ∪ *instances_ground_notCH_fms*
overhead_CH ≡ *overhead_notCH* ∪ {*dc_abs_fm*}

One further instance is needed to force CH, with a total count of 32 instances:

thm *overhead_CH_def*

overhead_CH ≡ *overhead_notCH* ∪ {*dc_abs_fm*}

thm *extensions_of_ctms*

$[\![M \approx \omega;\ \mathit{Transset}(M);\ M \models \cdot Z\cdot \cup \{\cdot \mathit{Replacement}(p)\cdot \ .\ p \in \mathit{overhead}\};$
$\Phi \subseteq \mathit{formula};\ M \models \{\cdot \mathit{Replacement}(\mathit{ground\_repl\_fm}(\varphi))\cdot \ .\ \varphi \in \Phi\}]\!]$
$\Longrightarrow \exists N.\ M \subseteq N \wedge$
$N \approx \omega \wedge$
$\mathit{Transset}(N) \wedge$
$M \neq N \wedge$
$(\forall \alpha.\ \mathit{Ord}(\alpha) \longrightarrow \alpha \in M \longleftrightarrow \alpha \in N) \wedge$
$((M,\ [] \models \cdot AC\cdot) \longrightarrow N,\ [] \models \cdot AC\cdot) \wedge$
$N \models \cdot Z\cdot \cup \{\cdot \mathit{Replacement}(\varphi)\cdot \ .\ \varphi \in \Phi\}$

thm *ctm_of_not_CH*

$[\![M \approx \omega;\ \mathit{Transset}(M);\ M \models ZC \cup \{\cdot \mathit{Replacement}(p)\cdot \ .\ p \in \mathit{overhead\_notCH}\};$
$\Phi \subseteq \mathit{formula};\ M \models \{\cdot \mathit{Replacement}(\mathit{ground\_repl\_fm}(\varphi))\cdot \ .\ \varphi \in \Phi\}]\!]$
$\Longrightarrow \exists N.\ M \subseteq N \wedge$
$N \approx \omega \wedge$
$\mathit{Transset}(N) \wedge$
$N \models ZC \cup \{\cdot \neg \cdot CH\cdot\cdot\} \cup \{\cdot \mathit{Replacement}(\varphi)\cdot \ .\ \varphi \in \Phi\} \wedge$
$(\forall \alpha.\ \mathit{Ord}(\alpha) \longrightarrow \alpha \in M \longleftrightarrow \alpha \in N)$

thm *ctm_of_CH*

$[\![M \approx \omega;\ \mathit{Transset}(M);\ M \models ZC \cup \{\cdot \mathit{Replacement}(p)\cdot \ .\ p \in \mathit{overhead\_CH}\};$
$\Phi \subseteq \mathit{formula};\ M \models \{\cdot \mathit{Replacement}(\mathit{ground\_repl\_fm}(\varphi))\cdot \ .\ \varphi \in \Phi\}]\!]$
$\Longrightarrow \exists N.\ M \subseteq N \wedge$
$N \approx \omega \wedge$
$\mathit{Transset}(N) \wedge$
$N \models ZC \cup \{\cdot CH\cdot\} \cup \{\cdot \mathit{Replacement}(\varphi)\cdot \ .\ \varphi \in \Phi\} \wedge$
$(\forall \alpha.\ \mathit{Ord}(\alpha) \longrightarrow \alpha \in M \longleftrightarrow \alpha \in N)$

In the above three statements, the function *ground_repl_fm* takes an element φ of *formula* and returns the replacement instance in the ground model that produces the φ-replacement instance in the generic extension. The next result is stated in the context *G_generic1*, which assumes the existence of a generic filter.

context *G_generic1*
begin

thm *sats_ground_repl_fm_imp_sats_ZF_replacement_fm*

$[\![\varphi \in \mathit{formula};\ M,\ [] \models \cdot \mathit{Replacement}(\mathit{ground\_repl\_fm}(\varphi))\cdot]\!]$
$\Longrightarrow M[G],\ [] \models \cdot \mathit{Replacement}(\varphi)\cdot$

end — *G_generic1*

end

34 Some demonstrations

theory *Demonstrations*
imports
Definitions_Main
begin

The following theory is only intended to explore some details of the formalization and to show the appearance of relevant internalized formulas. It is **not** intended as the entry point of the session. For that purpose, consult *Independence_CH.Definitions_Main*

The snippet (by M. Pagano) commented out below outputs a directed graph picturing the locale structure.

locale *Demo* = *M_trivial* + *M_AC* +
fixes t_1 t_2
assumes
ts_in_nat[*simp*]: $t_1 \in \omega$ $t_2 \in \omega$
and
power_infty: *power_ax*(*M*) *M*(ω)
begin

The next fake lemma is intended to explore the instances of the axiom schemes that are needed to build our forcing models. They are categorized as plain replacements (using *strong_replacement*), "lambda-replacements" using a higher order function, replacements to perform transfinite and general well-founded recursion (using *transrec_replacement* and *wfrec_replacement* respectively) and for the construction of fixpoints (using *iterates_replacement*). Lastly, separations instances.

lemma
assumes
sorried_replacements:
$\bigwedge P$. *strong_replacement*(*M*,*P*)
$\bigwedge F$. *lam_replacement*(*M*,*F*)
$\bigwedge Q\ S$. *iterates_replacement*(*M*,*Q*,*S*)
$\bigwedge Q\ S$. *wfrec_replacement*(*M*,*Q*,*S*)
$\bigwedge Q\ S$. *transrec_replacement*(*M*,*Q*,*S*)
and
sorried_separations: $\bigwedge Q$. *separation*(*M*,*Q*)
shows
M_master(*M*)
apply *unfold_locales*
apply
(*simp_all add*:
sorried_replacements(*1-2*)
sorried_separations
power_infty)
oops

— NOTE: Only for pretty-printing purposes, overrides previous fundamental notations
no_notation *mem* (**infixl** ‹∈› *50*)
no_notation *conj* (**infixr** ‹∧› *35*)
no_notation *disj* (**infixr** ‹∨› *30*)
no_notation *iff* (**infixr** ‹⟷› *25*)
no_notation *imp* (**infixr** ‹⟶› *25*)
no_notation *not* (‹¬ _› [*40*] *40*)
no_notation *All* (‹'(∀_')›)
no_notation *Ex* (‹'(∃_')›)

no_notation *Member* (‹·_ ∈/ _·›)
no_notation *Equal* (‹·_ =/ _·›)
no_notation *Nand* (‹·¬'(_ ∧/ _')·›)
no_notation *And* (‹·_ ∧/ _·›)
no_notation *Or* (‹·_ ∨/ _·›)
no_notation *Iff* (‹·_ ↔/ _·›)
no_notation *Implies* (‹·_ →/ _·›)
no_notation *Neg* (‹·¬_·›)
no_notation *Forall* (‹'(·∀(/_)·')›)
no_notation *Exists* (‹'(·∃(/_)·')›)

notation *Member* (**infixl** ‹∈› *50*)
notation *Equal* (**infixl** ‹≡› *50*)
notation *Nand* (‹¬'(_ ∧/ _')›)
notation *And* (**infixr** ‹∧› *35*)
notation *Or* (**infixr** ‹∨› *30*)
notation *Iff* (**infixr** ‹⟷› *25*)
notation *Implies* (**infixr** ‹⟶› *25*)
notation *Neg* (‹¬ _› [*40*] *40*)
notation *Forall* (‹'(∀_')›)
notation *Exists* (‹'(∃_')›)

lemma *forces*(t_1∈t_2) = (*0* ∈ *1* ∧ *forces_mem_fm*(*1*, *2*, *0*, t_1+$_\omega$*4*, t_2+$_\omega$*4*))
  unfolding *forces_def* **by** *simp*

definition *forces_0_mem_1* **where** *forces_0_mem_1*≡*forces_mem_fm*(*1*,*2*,*0*,t_1+$_\omega$*4*,t_2+$_\omega$*4*)
thm *forces_0_mem_1_def*[
    unfolded frc_at_fm_def ftype_fm_def
    name1_fm_def name2_fm_def snd_snd_fm_def hcomp_fm_def
    ecloseN_fm_def eclose_n1_fm_def eclose_n2_fm_def
    is_eclose_fm_def mem_eclose_fm_def eclose_n_fm_def
    is_If_fm_def least_fm_def Replace_fm_def Collect_fm_def
    fm_definitions,*simplified*]

named_theorems *incr_bv_new_simps*

schematic_goal *incr_bv_Neg*:
mem(*n*,*ω*) ⟹ *mem*(*φ*,*formula*) ⟹ *incr_bv*(*Neg*(*φ*))‘*n* = *?x*
unfolding *Neg_def* **by** *simp*

schematic_goal *incr_bv_Exists* [*incr_bv_new_simps*]:
mem(*n*,*ω*) ⟹ *mem*(*φ*,*formula*) ⟹ *incr_bv*(*Exists*(*φ*))‘*n* = *?x*
unfolding *Exists_def* **by** (*simp add*: *incr_bv_Neg*)

— The two renamings involved in the definition of *forces* depend on the recursive function *incr_bv*. Here we have an apparently exponential bottleneck, since all the propositional connectives (even *Neg*) duplicate the appearances of *incr_bv*.
Not even the double negation of an atomic formula can be managed by the system (in version 2021-1).

end — *Demo*

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, in: N. Peltier, V. Sofronie-Stokkermans (Eds.), Automated Reasoning. 10th International Joint Conference, IJCAR 2020, Paris, France, July 1–4, 2020, Proceedings, Part II, Lecture Notes in Artificial Intelligence **12167**, Springer International Publishing: 221–235 (2020).

[4] L.C. Paulson, K. Grabczewski, Mechanizing set theory, *J. Autom. Reasoning* **17**: 291–323 (1996).