

Transitive Models of Fragments of ZF

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

February 27, 2022

Abstract

We extend the ZF-Constructibility library by relativizing theories of the Isabelle/ZF and Delta System Lemma sessions to a transitive class. We also relativize Paulson’s work on Aleph and our former treatment of the Axiom of Dependent Choices. This work is a prerequisite to our formalization of the independence of the Continuum Hypothesis.

Contents

1	Introduction	4
2	Auxiliary results on arithmetic	5
2.1	Some results in ordinal arithmetic	8
3	Various results missing from ZF.	11
4	Renaming of variables in internalized formulas	15
4.1	Renaming of free variables	15
4.2	Renaming of formulas	23
5	Automatic synthesis of formulas	28
6	Aids to internalize formulas	37
7	Some enhanced theorems on recursion	40
8	The binder <i>Least</i>	48
8.1	Uniqueness, absoluteness and closure under <i>Least</i>	49

*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.

9	Fully relational versions of higher order construct	51
10	Automatic relativization of terms and formulas	54
10.1	Discipline of relativization of basic concepts	74
10.2	Discipline for <i>Pow</i>	78
10.3	Discipline for <i>PiP</i>	80
10.4	Discipline for <i>Pi</i>	83
10.5	Auxiliary ported results on <i>Pi_rel</i> , now unused	86
11	Arities of internalized formulas	87
11.1	Discipline for $\lambda A B. A \rightarrow B$	97
11.2	Discipline for <i>Collect</i> terms.	100
11.3	Discipline for <i>inj</i>	100
11.4	Discipline for <i>surj</i>	104
11.5	Discipline for <i>bij</i>	108
11.6	Discipline for (\approx)	109
11.7	Discipline for (\lesssim)	111
11.8	Discipline for (\prec)	112
12	Relativization of the cumulative hierarchy	116
12.1	Formula synthesis	122
13	Replacements using Lambdas	125
13.1	Replacement instances obtained through Powerset	131
13.2	Particular instances	163
14	Relative, Choice-less Cardinal Numbers	169
14.1	The Schroeder-Bernstein Theorem	171
14.2	lesspoll_rel: contributions by Krzysztof Grabczewski	176
15	Porting from <i>ZF.Cardinal</i>	178
16	Relative, Choice-less Cardinal Arithmetic	196
16.1	Cardinal addition	201
16.1.1	Cardinal addition is commutative	201
16.1.2	Cardinal addition is associative	201
16.1.3	0 is the identity for addition	202
16.1.4	Addition by another cardinal	202
16.1.5	Monotonicity of addition	203
16.1.6	Addition of finite cardinals is "ordinary" addition	203
16.2	Cardinal multiplication	204
16.2.1	Cardinal multiplication is commutative	204
16.2.2	Cardinal multiplication is associative	205
16.2.3	Cardinal multiplication distributes over addition	205
16.2.4	Multiplication by 0 yields 0	206

16.2.5	1 is the identity for multiplication	206
16.3	Some inequalities for multiplication	206
16.3.1	Multiplication by a non-zero cardinal	207
16.3.2	Monotonicity of multiplication	207
16.4	Multiplication of finite cardinals is "ordinary" multiplication .	207
16.5	Infinite Cardinals are Limit Ordinals	208
16.5.1	Toward's Kunen's Corollary 10.13 (1)	224
16.6	For Every Cardinal Number There Exists A Greater One . .	225
16.7	Basic Properties of Successor Cardinals	229
16.7.1	Theorems by Krzysztof Grabczewski, proofs by lcp . .	230
17	Relative, Cardinal Arithmetic Using AC	239
17.1	Strengthened Forms of Existing Theorems on Cardinals . . .	242
17.2	The relationship between cardinality and le-pollence	243
17.3	Other Applications of AC	245
18	Relativization of Finite Functions	248
18.1	The set of finite binary sequences	249
18.2	Representation of finite functions	250
19	Library of basic ZF results	257
20	Lambda-replacements required for cardinal inequalities	279
21	Cardinal Arithmetic under Choice	290
21.1	Miscellaneous	290
21.2	Countable and uncountable sets	295
21.3	Results on Aleph_rels	299
21.4	Applications of transfinite recursive constructions	304
21.5	Results on relative cardinal exponentiation	314
22	The Delta System Lemma, Relativized	316
23	Relative DC	325
24	Cohen forcing notions	335
24.0.1	image, preimage, domain, range	351
24.0.2	Domain, range and field	352
24.0.3	Relations, functions and application	352
24.0.4	Composition of relations	353
24.0.5	Some Facts About Separation Axioms	354
24.0.6	Functions and function space	355

1 Introduction

As it was explained in [1, Sect. 3] and elsewhere, relativization of concepts is a key tool to obtain results in forcing.

In this session, we cast some theories in relative form, in a way that they now refer to a fixed class M as the universe of discourse. Whenever it was possible, we tried to minimize the changes to the structure and proof scripts. For this reason, some comments of the original text as well as outdated **apply** commands appear profusely in the following theories.

A repeated pattern that appears is that the relativized result can be proved *mutatis mutandis*, with remaining proof obligations that the objects constructed actually belong to the model M . Another aspect was that the management of higher order constructs always posed some extra problems, already noted by Paulson [2, Sect. 7.3].

We proceed to enumerate the theories that were “ported” to relative form, briefly commenting on each of them. Below, we refer to the original theories as the *source* and, correspondingly, call *target* the relativized version. We omit the `.thy` suffixes.

1. From *ZF*:

- (a) **Univ**. Here we decided to relativize only the term **Vfrom** that constructs the cumulative hierarchy up to some ordinal length and starting from an arbitrary set.
- (b) **Cardinal**. There are two targets for this source, **Least** and **Cardinal_Relative**. Both require some fair amount of preparation, trying to take advantage of absolute concepts. It is not straightforward to compare source and targets in a line-by-line fashion at this point.
- (c) **CardinalArith**. The hardest part was to formalize the cardinal successor function. We also disregarded the part treating finite cardinals since it is an absolute concept. Apart from that, the relative version closely parallels the source.
- (d) **Cardinal_AC**. After some boilerplate, porting was rather straightforward, excepting cardinal arithmetic involving the higher-order union operator.

2. From *ZF-Constructible*:

- (a) **Normal**. The target here is **Aleph_Relative** since that is the only concept that we ported. Instead of porting all the machinery of normal functions (since it involved higher-order variables), we particularized the results for the Aleph function. We also used

an alternative definition of the latter that worked better with our relativization discipline.

3. From *Delta_System_Lemma*:

- (a) **ZF_Library**. The target includes a big section of auxiliary lemmas and commands that aid the relativization. We needed to make explicit the witnesses (mainly functions) in some of the existential results proved in the source, since only in that way we would be able to show that they belonged to the model.
- (b) **Cardinal_Library**. Porting was relatively straightforward; most of the extra work laid in adjusting locale assumptions to obtain an appropriate context to state and prove the theorems.
- (c) **Delta_System**. Same comments as in the case of **Cardinal_Library** apply here.

4. From *Forcing*:

- (a) **Pointed_DC**. This case was similar to **Cardinal_AC** above, although a bit of care was needed to handle the recursive construction. Also, a fraction of the theory **AC** from *ZF* was ported here as it was a prerequisite. A complete relativization of **AC** would be desirable but still missing.

2 Auxiliary results on arithmetic

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

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

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

lemma *nat_succD* : $m \in \text{nat} \implies \text{succ}(n) \in \text{succ}(m) \implies n \in m$
by (*drule_tac j=succ(m) in ltI, auto elim:ltD*)

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

lemma *in_n_in_nat* : $m \in \text{nat} \implies n \in m \implies n \in \text{nat}$
by(*drule ltI[of n], auto simp add: lt_nat_in_nat*)

lemma *in_succ_in_nat* : $m \in \text{nat} \implies n \in \text{succ}(m) \implies n \in \text{nat}$
by(*auto simp add:in_n_in_nat*)

lemma *ltI_neg* : $x \in \text{nat} \implies j \leq x \implies j \neq x \implies j < x$
by (*simp add: le_iff*)

lemma *succ_pred_eq* : $m \in \text{nat} \implies m \neq 0 \implies \text{succ}(\text{pred}(m)) = m$
by (*auto elim: natE*)

lemma *succ_ltI* : $\text{succ}(j) < n \implies j < n$
by (*simp add: succ_leE*[*OF leI*])

lemma *succ_In* : $n \in \text{nat} \implies \text{succ}(j) \in n \implies j \in n$
by (*rule succ_ltI*[*THEN ltD*], *auto intro: ltI*)

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

lemma *succpred_leI* : $n \in \text{nat} \implies n \leq \text{succ}(\text{pred}(n))$
by (*auto elim: natE*)

lemma *succpred_n0* : $\text{succ}(n) \in p \implies p \neq 0$
by (*auto*)

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

lemma *succ_in* : $\text{succ}(x) \leq y \implies x \in y$
by (*auto dest:ltD*)

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

lemma *pred_type* : $m \in \text{nat} \implies n \leq m \implies n \in \text{nat}$
by (*rule leE, auto simp: in_n_in_nat ltD*)

lemma *pred_le* : $m \in \text{nat} \implies n \leq \text{succ}(m) \implies \text{pred}(n) \leq m$
by (*rule_tac n=n in natE, auto simp add: pred_type*[*of succ(m)*])

lemma *pred_le2* : $n \in \text{nat} \implies m \in \text{nat} \implies \text{pred}(n) \leq m \implies n \leq \text{succ}(m)$
by (*subgoal_tac n ∈ nat, rule_tac n=n in natE, auto*)

lemma *Un_leD1* : $\text{Ord}(i) \implies \text{Ord}(j) \implies \text{Ord}(k) \implies i \cup j \leq k \implies i \leq k$
by (*rule Un_least_lt_iff*[*THEN iffD1*[*THEN conjunct1*]], *simp_all*)

lemma *Un_leD2* : $\text{Ord}(i) \implies \text{Ord}(j) \implies \text{Ord}(k) \implies i \cup j \leq k \implies j \leq k$
by (*rule Un_least_lt_iff*[*THEN iffD1*[*THEN conjunct2*]], *simp_all*)

lemma *gt1* : $n \in \text{nat} \implies i \in n \implies i \neq 0 \implies i \neq 1 \implies 1 < i$
by (*rule_tac n=i in natE, erule in_n_in_nat, auto intro: Ord_0_lt*)

lemma *pred_mono* : $m \in \text{nat} \implies n \leq m \implies \text{pred}(n) \leq \text{pred}(m)$
by (*rule_tac n=n in natE, auto simp add: le_in_nat, erule_tac n=m in natE, auto*)

lemma *succ_mono* : $m \in \text{nat} \implies n \leq m \implies \text{succ}(n) \leq \text{succ}(m)$
by *auto*

lemma *union_abs1* :
 $\llbracket i \leq j \rrbracket \implies i \cup j = j$

```

by (rule Un_absorb1,erule le_imp_subset)

lemma union_abs2 :
   $\llbracket i \leq j \rrbracket \implies j \cup i = j$ 
  by (rule Un_absorb2,erule le_imp_subset)

lemma ord_un_max :  $\text{Ord}(i) \implies \text{Ord}(j) \implies i \cup j = \max(i,j)$ 
  using max_def union_abs1 not_lt_iff_le leI union_abs2
  by auto

lemma ord_max_ty :  $\text{Ord}(i) \implies \text{Ord}(j) \implies \text{Ord}(\max(i,j))$ 
  unfolding max_def by simp

lemmas ord_simp_union = ord_un_max ord_max_ty max_def

lemma le_succ :  $x \in \text{nat} \implies x \leq \text{succ}(x)$  by simp

lemma le_pred :  $x \in \text{nat} \implies \text{pred}(x) \leq x$ 
  using pred_le[OF le_succ] pred_succ_eq
  by simp

lemma not_le_anti_sym :  $x \in \text{nat} \implies y \in \text{nat} \implies \neg x \leq y \implies \neg y \leq x \implies y = x$ 
  using Ord_linear not_le_iff_lt ltD lt_trans
  by auto

lemma Un_le_compat :  $o \leq p \implies q \leq r \implies \text{Ord}(o) \implies \text{Ord}(p) \implies \text{Ord}(q) \implies$ 
 $\text{Ord}(r) \implies o \cup q \leq p \cup r$ 
  using le_trans[of q r p $\cup$ r,OF Un_upper2_le] le_trans[of o p p $\cup$ r,OF Un_upper1_le]
  ord_simp_union
  by auto

lemma Un_le :  $p \leq r \implies q \leq r \implies$ 
 $\text{Ord}(p) \implies \text{Ord}(q) \implies \text{Ord}(r) \implies$ 
 $p \cup q \leq r$ 
  using ord_simp_union by auto

lemma Un_leI3 :  $o \leq r \implies p \leq r \implies q \leq r \implies$ 
 $\text{Ord}(o) \implies \text{Ord}(p) \implies \text{Ord}(q) \implies \text{Ord}(r) \implies$ 
 $o \cup p \cup q \leq r$ 
  using ord_simp_union by auto

lemma diff_mono :
  assumes  $m \in \text{nat } n \in \text{nat } p \in \text{nat } m < n \ p \leq m$ 
  shows  $m \# -p < n \# -p$ 
proof -
  from assms
  have  $m \# -p \in \text{nat } m \# -p \# +p = m$ 
  using add_diff_inverse2 by simp_all

```

```

with assms
show ?thesis
  using less_diff_conv[of n p m #- p, THEN iffD2] by simp
qed

lemma pred_Un:
   $x \in \text{nat} \implies y \in \text{nat} \implies \text{Arith.pred}(\text{succ}(x) \cup y) = x \cup \text{Arith.pred}(y)$ 
   $x \in \text{nat} \implies y \in \text{nat} \implies \text{Arith.pred}(x \cup \text{succ}(y)) = \text{Arith.pred}(x) \cup y$ 
  using pred_Un_distrib pred_succ_eq by simp_all

lemma le_natI :  $j \leq n \implies n \in \text{nat} \implies j \in \text{nat}$ 
  by(drule ltD, rule in_n_in_nat, rule nat_succ_iff[THEN iffD2, of n], simp_all)

lemma le_natE :  $n \in \text{nat} \implies j < n \implies j \in n$ 
  by(rule ltE[of j n], simp+)

lemma leD : assumes  $n \in \text{nat} \ j \leq n$ 
  shows  $j < n \mid j = n$ 
  using leE[OF <j≤n>, of j<n | j = n] by auto

lemma pred_nat_eq :
  assumes  $n \in \text{nat}$ 
  shows  $\text{Arith.pred}(n) = \bigcup n$ 
  using assms
proof(induct)
  case 0
  then show ?case by simp
next
  case (succ x)
  then show ?case using Arith.pred_succ_eq Ord_Union_succ_eq
    by simp
qed

```

2.1 Some results in ordinal arithmetic

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

```

lemma max_cong :
  assumes  $x \leq y \ \text{Ord}(y) \ \text{Ord}(z)$ 
  shows  $\max(x, y) \leq \max(y, z)$ 
proof (cases y ≤ z)
  case True
  then show ?thesis
    unfolding max_def using assms by simp
next
  case False
  then have  $z \leq y$  using assms not_le_iff_lt leI by simp
  then show ?thesis
    unfolding max_def using assms by simp

```


qed

lemma *max_commutes* :
 assumes $Ord(x)$ $Ord(y)$
 shows $max(x,y) = max(y,x)$
 using *assms Un_commute ord_simp_union(1) ord_simp_union(1)[symmetric]*
 by *auto*

lemma *max_cong2* :
 assumes $x \leq y$ $Ord(y)$ $Ord(z)$ $Ord(x)$
 shows $max(x,z) \leq max(y,z)$
proof -
 from *assms*
 have $x \cup z \leq y \cup z$
 using *lt_Ord Ord_Un Un_mono[OF le_imp_subset[OF $\langle x \leq y \rangle$]] subset_imp_le*
 by *auto*
 then show *?thesis*
 using *ord_simp_union $\langle Ord(x) \rangle \langle Ord(z) \rangle \langle Ord(y) \rangle$ by simp*
 qed

lemma *max_D1* :
 assumes $x = y$ $w < z$ $Ord(x)$ $Ord(w)$ $Ord(z)$ $max(x,w) = max(y,z)$
 shows $z \leq y$
proof -
 from *assms*
 have $w < x \cup w$ using *Un_upper2_lt[OF $\langle w < z \rangle$] assms ord_simp_union* by *simp*
 then
 have $w < x$ using *assms lt_Un_iff[of x w w] lt_not_refl* by *auto*
 then
 have $y = y \cup z$ using *assms max_commutes ord_simp_union assms leI* by *simp*
 then
 show *?thesis* using *Un_leD2 assms* by *simp*
 qed

lemma *max_D2* :
 assumes $w = y \vee w = z$ $x < y$ $Ord(x)$ $Ord(w)$ $Ord(y)$ $Ord(z)$ $max(x,w) = max(y,z)$
 shows $x < w$
proof -
 from *assms*
 have $x < z \cup y$ using *Un_upper2_lt[OF $\langle x < y \rangle$] by simp*
 then
 consider (a) $x < y$ | (b) $x < w$
 using *assms ord_simp_union* by *simp*
 then show *?thesis* **proof** (cases)
 case a
 consider (c) $w = y$ | (d) $w = z$

```

    using assms by auto
  then show ?thesis proof (cases)
    case c
    with a show ?thesis by simp
  next
    case d
    with a
    show ?thesis
    proof (cases  $y < w$ )
      case True
      then show ?thesis using lt_trans[OF  $\langle x < y \rangle$ ] by simp
    next
      case False
      then
        have  $w \leq y$ 
          using not_lt_iff_le[OF assms(5) assms(4)] by simp
        with  $\langle w = z \rangle$ 
        have  $\max(z, y) = y$  unfolding max_def using assms by simp
        with assms
        have  $\dots = x \cup w$  using ord_simp_union_max_commutes by simp
        then show ?thesis using le_Un_iff assms by blast
      qed
    qed
  next
    case b
    then show ?thesis .
  qed
qed

lemma oadd_lt_mono2 :
  assumes Ord(n) Ord( $\alpha$ ) Ord( $\beta$ )  $\alpha < \beta$   $x < n$   $y < n$   $0 < n$ 
  shows  $n ** \alpha ++ x < n ** \beta ++ y$ 
proof -
  consider (0)  $\beta = 0$  | (s)  $\gamma$  where Ord( $\gamma$ )  $\beta = \text{succ}(\gamma)$  | (l) Limit( $\beta$ )
  using Ord_cases[OF  $\langle \text{Ord}(\beta) \rangle$ , of ?thesis] by force
  then show ?thesis
  proof cases
    case 0
    then show ?thesis using  $\langle \alpha < \beta \rangle$  by auto
  next
    case s
    then
      have  $\alpha \leq \gamma$  using  $\langle \alpha < \beta \rangle$  using leI by auto
      then
        have  $n ** \alpha \leq n ** \gamma$  using omult_le_mono[OF  $\_ \langle \alpha \leq \gamma \rangle$ ]  $\langle \text{Ord}(n) \rangle$  by simp
        then
          have  $n ** \alpha ++ x < n ** \gamma ++ n$  using oadd_lt_mono[OF  $\_ \langle x < n \rangle$ ] by simp
          also
          have  $\dots = n ** \beta$  using  $\langle \beta = \text{succ}(\_) \rangle$  omult_succ  $\langle \text{Ord}(\beta) \rangle$   $\langle \text{Ord}(n) \rangle$  by simp

```

```

    finally
    have  $n ** \alpha ++ x < n ** \beta$  by auto
    then
    show ?thesis using oadd_le_self  $\langle \text{Ord}(\beta) \rangle$  lt_trans2  $\langle \text{Ord}(n) \rangle$  by auto
next
case l
have  $\text{Ord}(x)$  using  $\langle x < n \rangle$  lt_Ord by simp
with l
have  $\text{succ}(\alpha) < \beta$  using Limit_has_succ  $\langle \alpha < \beta \rangle$  by simp
have  $n ** \alpha ++ x < n ** \alpha ++ n$ 
    using oadd_lt_mono [OF le_refl [OF Ord_omult [OF  $\langle \text{Ord}(\alpha) \rangle$ ]]]  $\langle x < n \rangle$ 
 $\langle \text{Ord}(n) \rangle$  by simp
also
have  $\dots = n ** \text{succ}(\alpha)$  using omult_succ  $\langle \text{Ord}(\alpha) \rangle$   $\langle \text{Ord}(n) \rangle$  by simp
finally
have  $n ** \alpha ++ x < n ** \text{succ}(\alpha)$  by simp
with  $\langle \text{succ}(\alpha) < \beta \rangle$ 
have  $n ** \alpha ++ x < n ** \beta$  using lt_trans omult_lt_mono  $\langle \text{Ord}(n) \rangle$   $\langle 0 < n \rangle$ 
by auto
    then show ?thesis using oadd_le_self  $\langle \text{Ord}(\beta) \rangle$  lt_trans2  $\langle \text{Ord}(n) \rangle$  by auto
qed
qed
end

```

3 Various results missing from ZF.

theory *ZF_Miscellanea*

imports

ZF

Nat_Miscellanea

begin

lemma *funcI* : $f \in A \rightarrow B \implies a \in A \implies b = f \text{ ` } a \implies \langle a, b \rangle \in f$
by (*simp_all* *add: apply_Pair*)

lemma *vimage_fun_sing*:

assumes $f \in A \rightarrow B$ $b \in B$

shows $\{a \in A . f \text{ ` } a = b\} = f \text{ `` } \{b\}$

using *assms* *vimage_singleton_iff* *function_apply_equality* *Pi_iff* *funcI* **by** *auto*

lemma *image_fun_subset*: $S \in A \rightarrow B \implies C \subseteq A \implies \{S \text{ ` } x . x \in C\} = S \text{ `` } C$

using *image_function* [*symmetric*, *of* *S* *C*] *domain_of_fun* *Pi_iff* **by** *auto*

lemma *subset_Diff_Un*: $X \subseteq A \implies A = (A - X) \cup X$ **by** *auto*

lemma *Diff_bij*:

assumes $\forall A \in F. X \subseteq A$ **shows** $(\lambda A \in F. A - X) \in \text{bij}(F, \{A - X. A \in F\})$

using *assms* **unfolding** *bij_def* *inj_def* *surj_def*

by (*auto* *intro: lam_type*, *subst* *subset_Diff_Un* [*of* *X*]) *auto*

lemma *function_space_nonempty*:

assumes $b \in B$
shows $(\lambda x \in A. b) : A \rightarrow B$
using *assms lam_type* **by** *force*

lemma *vimage_lam*: $(\lambda x \in A. f(x)) \text{ -'' } B = \{ x \in A . f(x) \in B \}$

using *lam_funtime*[*of* A f , *THEN* [2] *domain_type*]
lam_funtime[*of* A f , *THEN* [2] *apply_equality*] *lamI*[*of* $_ A$ f]
by *auto blast*

lemma *range_fun_subset_codomain*:

assumes $h : B \rightarrow C$
shows $\text{range}(h) \subseteq C$
unfolding *range_def domain_def converse_def* **using** *range_type*[*OF* $_ \text{assms}$]
by *auto*

lemma *Pi_rangeD*:

assumes $f \in \text{Pi}(A, B)$ $b \in \text{range}(f)$
shows $\exists a \in A. f'a = b$
using *assms apply_equality*[*OF* $_ \text{assms}(1)$, *of* $_ b$]
domain_type[*OF* $_ \text{assms}(1)$] **by** *auto*

lemma *Pi_range_eq*: $f \in \text{Pi}(A, B) \implies \text{range}(f) = \{f'x . x \in A\}$

using *Pi_rangeD*[*of* $f A B$] *apply_rangeI*[*of* $f A B$]
by *blast*

lemma *Pi_vimage_subset* : $f \in \text{Pi}(A, B) \implies f \text{ -'' } C \subseteq A$

unfolding *Pi_def* **by** *auto*

definition

minimum :: $i \Rightarrow i \Rightarrow i$ **where**
minimum(r, B) $\equiv \text{THE } b. \text{first}(b, B, r)$

lemma *minimum_in*: $\llbracket \text{well_ord}(A, r); B \subseteq A; B \neq 0 \rrbracket \implies \text{minimum}(r, B) \in B$

using *the_first_in* **unfolding** *minimum_def* **by** *simp*

lemma *well_ord_surj_imp_inj_inverse*:

assumes *well_ord*(A, r) $h \in \text{surj}(A, B)$
shows $(\lambda b \in B. \text{minimum}(r, \{a \in A. h'a = b\})) \in \text{inj}(B, A)$

proof -

let $?f = \lambda b \in B. \text{minimum}(r, \{a \in A. h'a = b\})$
have $\text{minimum}(r, \{a \in A . h' a = b\}) \in \{a \in A. h'a = b\}$ **if** $b \in B$ **for** b

proof -

from $\langle h \in \text{surj}(A, B) \rangle$ *that*

have $\{a \in A. h'a = b\} \neq 0$

unfolding *surj_def* **by** *blast*

with $\langle \text{well_ord}(A, r) \rangle$

```

    show  $\text{minimum}(r, \{a \in A. h'a=b\}) \in \{a \in A. h'a=b\}$ 
    using minimum_in by blast
qed
moreover from this
have  $?f : B \rightarrow A$ 
    using lam_type[of  $B \_ \lambda_. A$ ] by simp
moreover
have  $?f' w = ?f' x \implies w = x$  if  $w \in B$   $x \in B$  for  $w x$ 
proof -
  from calculation that
  have  $w = h' \text{minimum}(r, \{a \in A. h'a=w\})$ 
     $x = h' \text{minimum}(r, \{a \in A. h'a=x\})$ 
    by simp_all
  moreover
  assume  $?f' w = ?f' x$ 
  moreover from this and that
  have  $\text{minimum}(r, \{a \in A. h'a = w\}) = \text{minimum}(r, \{a \in A. h'a = x\})$ 
    unfolding minimum_def by simp_all
  moreover from calculation(1,2,4)
  show  $w=x$  by simp
qed
ultimately
show ?thesis
  unfolding inj_def by blast
qed

```

```

lemma well_ord_surj_imp_lepoll:
  assumes  $\text{well\_ord}(A, r)$   $h \in \text{surj}(A, B)$ 
  shows  $B \lesssim A$ 
    unfolding lepoll_def using well_ord_surj_imp_inj_inverse[OF assms]
    by blast

```

— New result

```

lemma surj_imp_well_ord:
  assumes  $\text{well\_ord}(A, r)$   $h \in \text{surj}(A, B)$ 
  shows  $\exists s. \text{well\_ord}(B, s)$ 
    using assms lepoll_well_ord[OF well_ord_surj_imp_lepoll]
    by force

```

```

lemma Pow_sing :  $\text{Pow}(\{a\}) = \{0, \{a\}\}$ 
proof(intro equalityI, simp_all)
  have  $z \in \{0, \{a\}\}$  if  $z \subseteq \{a\}$  for  $z$ 
    using that by auto
  then
  show  $\text{Pow}(\{a\}) \subseteq \{0, \{a\}\}$  by auto
qed

```

```

lemma Pow_cons:
  shows  $\text{Pow}(\text{cons}(a, A)) = \text{Pow}(A) \cup \{\{a\} \cup X \mid X: \text{Pow}(A)\}$ 

```

```

using Un_Pow_subset Pow_sing
proof(intro equalityI,auto simp add:Un_Pow_subset)
{
  fix C D
  assume  $\bigwedge B . B \in \text{Pow}(A) \implies C \neq \{a\} \cup B \ C \subseteq \{a\} \cup A \ D \in C$ 
  moreover from this
  have  $\forall x \in C . x = a \vee x \in A$  by auto
  moreover from calculation
  consider  $(a) \ D = a \mid (b) \ D \in A$  by auto
  from this
  have  $D \in A$ 
  proof(cases)
    case a
    with calculation show ?thesis by auto
  next
    case b
    then show ?thesis by simp
  qed
}
then show  $\bigwedge x \ x a . (\forall xa \in \text{Pow}(A) . x \neq \{a\} \cup xa) \implies x \subseteq \text{cons}(a, A) \implies xa \in$ 
 $x \implies xa \in A$ 
by auto
qed

lemma app_nm :
  assumes  $n \in \text{nat} \ m \in \text{nat} \ f \in n \rightarrow m \ x \in \text{nat}$ 
  shows  $f'x \in \text{nat}$ 
proof(cases x \in n)
  case True
    then show ?thesis using assms in_n_in_nat apply_type by simp
  next
    case False
    then show ?thesis using assms apply_0 domain_of_fun by simp
  qed

lemma Upair_eq_cons:  $\text{Upair}(a,b) = \{a,b\}$ 
  unfolding cons_def by auto

lemma converse_apply_eq :  $\text{converse}(f) \text{ `` } x = \bigcup (f \text{ `` } \{x\})$ 
  unfolding apply_def vimage_def by simp

lemmas app_fun = apply_iff[THEN iffD1]

lemma Finite_imp_lespoll_nat:
  assumes Finite(A)
  shows  $A \prec \text{nat}$ 
  using assms subset_imp_lepoll[OF naturals_subset_nat] eq_lepoll_trans
  n_lespoll_nat eq_lespoll_trans
  unfolding Finite_def lespoll_def by auto

```

end

4 Renaming of variables in internalized formulas

```
theory Renaming
  imports
    ZF_Miscellanea
    ZF-Constructible.Formula
begin
```

4.1 Renaming of free variables

definition

```
union_fun :: [i,i,i,i] ⇒ i where
union_fun(f,g,m,p) ≡ λj ∈ m ∪ p . if j∈m then f'j else g'j
```

lemma *union_fun_type*:

```
assumes f ∈ m → n
         g ∈ p → q
shows union_fun(f,g,m,p) ∈ m ∪ p → n ∪ q
```

proof -

```
let ?h=union_fun(f,g,m,p)
```

have

```
D: ?h'x ∈ n ∪ q if x ∈ m ∪ p for x
```

proof (*cases* x ∈ m)

```
case True
```

then have

```
x ∈ m ∪ p by simp
```

```
with ⟨x∈m⟩
```

```
have ?h'x = f'x
```

```
unfolding union_fun_def beta by simp
```

```
with ⟨f ∈ m → n⟩ ⟨x∈m⟩
```

```
have ?h'x ∈ n by simp
```

```
then show ?thesis ..
```

next

```
case False
```

```
with ⟨x ∈ m ∪ p⟩
```

```
have x ∈ p
```

```
by auto
```

```
with ⟨x∉m⟩
```

```
have ?h'x = g'x
```

```
unfolding union_fun_def using beta by simp
```

```
with ⟨g ∈ p → q⟩ ⟨x∈p⟩
```

```
have ?h'x ∈ q by simp
```

```
then show ?thesis ..
```

qed

```
have A:function(?h) unfolding union_fun_def using function_lam by simp
```

```
have x ∈ (m ∪ p) × (n ∪ q) if x ∈ ?h for x
```

```

    using that lamE[of  $x \in m \cup p \implies x \in (m \cup p) \times (n \cup q)$ ] D unfolding
union_fun_def
  by auto
  then have  $B: ?h \subseteq (m \cup p) \times (n \cup q) ..$ 
  have  $m \cup p \subseteq \text{domain}(?h)$ 
    unfolding union_fun_def using domain_lam by simp
  with A B
  show ?thesis using Pi_iff [THEN iffD2] by simp
qed

```

```

lemma union_fun_action :
  assumes
     $env \in \text{list}(M)$ 
     $env' \in \text{list}(M)$ 
     $\text{length}(env) = m \cup p$ 
     $\forall i. i \in m \longrightarrow \text{nth}(f'i, env') = \text{nth}(i, env)$ 
     $\forall j. j \in p \longrightarrow \text{nth}(g'j, env') = \text{nth}(j, env)$ 
  shows  $\forall i. i \in m \cup p \longrightarrow \text{nth}(i, env) = \text{nth}(\text{union\_fun}(f, g, m, p)'i, env')$ 
proof -
  let ?h = union_fun(f, g, m, p)
  have  $\text{nth}(x, env) = \text{nth}(?h'x, env')$  if  $x \in m \cup p$  for x
    using that
  proof (cases  $x \in m$ )
    case True
    with  $\langle x \in m \rangle$ 
    have  $?h'x = f'x$ 
      unfolding union_fun_def beta by simp
    with  $\text{assms } \langle x \in m \rangle$ 
    have  $\text{nth}(x, env) = \text{nth}(?h'x, env')$  by simp
    then show ?thesis .
  next
    case False
    with  $\langle x \in m \cup p \rangle$ 
    have
       $x \in p \wedge x \notin m$  by auto
    then
      have  $?h'x = g'x$ 
        unfolding union_fun_def beta by simp
      with  $\text{assms } \langle x \in p \rangle$ 
      have  $\text{nth}(x, env) = \text{nth}(?h'x, env')$  by simp
      then show ?thesis .
    qed
  then show ?thesis by simp
qed

```

```

lemma id_fn_type :
  assumes  $n \in \text{nat}$ 

```



```

shows  $id(n) \in n \rightarrow n$ 
unfolding  $id\_def$  using  $\langle n \in nat \rangle$  by simp

lemma  $id\_fn\_action$ :
  assumes  $n \in nat \ env \in list(M)$ 
  shows  $\bigwedge j . j < n \implies nth(j, env) = nth(id(n) 'j, env)$ 
proof -
  show  $nth(j, env) = nth(id(n) 'j, env)$  if  $j < n$  for  $j$  using that  $\langle n \in nat \rangle$  ltD by
simp
qed

definition
   $rsum :: [i, i, i, i, i] \Rightarrow i$  where
   $rsum(f, g, m, n, p) \equiv \lambda j \in m \# + p . \text{if } j < m \text{ then } f 'j \text{ else } (g '(j \# - m)) \# + n$ 

lemma  $sum\_inl$ :
  assumes  $m \in nat \ n \in nat$ 
   $f \in m \rightarrow n \ x \in m$ 
  shows  $rsum(f, g, m, n, p) 'x = f 'x$ 
proof -
  from  $\langle m \in nat \rangle$ 
  have  $m \leq m \# + p$ 
  using add_le_self[of  $m$ ] by simp
  with assms
  have  $x \in m \# + p$ 
  using ltI[of  $x \ m$ ] lt_trans2[of  $x \ m \ m \# + p$ ] ltD by simp
  from assms
  have  $x < m$ 
  using ltI by simp
  with  $\langle x \in m \# + p \rangle$ 
  show ?thesis unfolding  $rsum\_def$  by simp
qed

lemma  $sum\_inr$ :
  assumes  $m \in nat \ n \in nat \ p \in nat$ 
   $g \in p \rightarrow q \ m \leq x \ x < m \# + p$ 
  shows  $rsum(f, g, m, n, p) 'x = g '(x \# - m) \# + n$ 
proof -
  from assms
  have  $x \in nat$ 
  using in_n_in_nat[of  $m \# + p$ ] ltD
  by simp
  with assms
  have  $\neg x < m$ 
  using not_lt_iff_le[THEN iffD2] by simp
  from assms
  have  $x \in m \# + p$ 
  using ltD by simp

```

```

with  $\langle \neg x < m \rangle$ 
show ?thesis unfolding rsum_def by simp
qed

```

lemma *sum_action* :

```

assumes  $m \in \text{nat } n \in \text{nat } p \in \text{nat } q \in \text{nat}$ 
 $f \in m \rightarrow n \ g \in p \rightarrow q$ 
 $\text{env} \in \text{list}(M)$ 
 $\text{env}' \in \text{list}(M)$ 
 $\text{env1} \in \text{list}(M)$ 
 $\text{env2} \in \text{list}(M)$ 
 $\text{length}(\text{env}) = m$ 
 $\text{length}(\text{env1}) = p$ 
 $\text{length}(\text{env}') = n$ 
 $\bigwedge i. i < m \implies \text{nth}(i, \text{env}) = \text{nth}(f' i, \text{env}')$ 
 $\bigwedge j. j < p \implies \text{nth}(j, \text{env1}) = \text{nth}(g' j, \text{env2})$ 
shows  $\forall i. i < m \# + p \longrightarrow$ 
 $\text{nth}(i, \text{env} @ \text{env1}) = \text{nth}(\text{rsum}(f, g, m, n, p) 'i, \text{env}' @ \text{env2})$ 

```

proof -

```

let ?h = rsum(f, g, m, n, p)
from  $\langle m \in \text{nat} \rangle \langle n \in \text{nat} \rangle \langle q \in \text{nat} \rangle$ 
have  $m \leq m \# + p \ n \leq n \# + q \ q \leq n \# + q$ 
using add_le_self[of m] add_le_self2[of n q] by simp_all
from  $\langle p \in \text{nat} \rangle$ 
have  $p = (m \# + p) \# - m$  using diff_add_inverse2 by simp
have  $\text{nth}(x, \text{env} @ \text{env1}) = \text{nth}(\text{?h}' x, \text{env}' @ \text{env2})$  if  $x < m \# + p$  for x
proof (cases x < m)
case True
then
have 2:  $\text{?h}' x = f' x \ x \in m \ f' x \in n \ x \in \text{nat}$ 
using assms sum_inl ltD apply_type[of f m _ x] in_n_in_nat by simp_all
with  $\langle x < m \rangle$  assms
have  $f' x < n \ f' x < \text{length}(\text{env}') \ f' x \in \text{nat}$ 
using ltI in_n_in_nat by simp_all
with 2  $\langle x < m \rangle$  assms
have  $\text{nth}(x, \text{env} @ \text{env1}) = \text{nth}(x, \text{env})$ 
using nth_append[OF  $\langle \text{env} \in \text{list}(M) \rangle$ ]  $\langle x \in \text{nat} \rangle$  by simp
also
have
 $\dots = \text{nth}(f' x, \text{env}')$ 
using 2  $\langle x < m \rangle$  assms by simp
also
have  $\dots = \text{nth}(f' x, \text{env}' @ \text{env2})$ 
using nth_append[OF  $\langle \text{env}' \in \text{list}(M) \rangle$ ]  $\langle f' x < \text{length}(\text{env}') \rangle \langle f' x \in \text{nat} \rangle$  by simp
also
have  $\dots = \text{nth}(\text{?h}' x, \text{env}' @ \text{env2})$ 
using 2 by simp
finally

```

```

have nth(x, env @ env1) = nth(?h'x,env'@env2) .
then show ?thesis .
next
case False
have x∈nat
  using that in_n_in_nat[of m#+p x] ltD ⟨p∈nat⟩ ⟨m∈nat⟩ by simp
with ⟨length(env) = m⟩
have m≤x length(env) ≤ x
  using not_lt_iff_le ⟨m∈nat⟩ ⟨¬x<m⟩ by simp_all
with ⟨¬x<m⟩ ⟨length(env) = m⟩
have 2 : ?h'x = g'(x#-m)#+n ¬ x < length(env)
  unfolding rsum_def
  using sum_inr that beta ltD by simp_all
from assms ⟨x∈nat⟩ ⟨p=m#+p#-m⟩
have x#-m < p
  using diff_mono[OF _ _ _ ⟨x<m#+p⟩ ⟨m≤x⟩] by simp
then have x#-m∈p using ltD by simp
with ⟨g∈p→q⟩
have g'(x#-m) ∈ q by simp
with ⟨q∈nat⟩ ⟨length(env') = n⟩
have g'(x#-m) < q g'(x#-m)∈nat using ltI in_n_in_nat by simp_all
with ⟨q∈nat⟩ ⟨n∈nat⟩
have (g'(x#-m))#+n < n#+q n ≤ g'(x#-m)#+n ¬ g'(x#-m)#+n < length(env')
  using add_lt_mono1[of g'(x#-m) _ n, OF _ ⟨q∈nat⟩]
  add_le_self2[of n] ⟨length(env') = n⟩
  by simp_all
from assms ⟨¬ x < length(env)⟩ ⟨length(env) = m⟩
have nth(x, env @ env1) = nth(x#-m, env1)
  using nth_append[OF ⟨env∈list(M)⟩ ⟨x∈nat⟩] by simp
also
have ... = nth(g'(x#-m), env2)
  using assms ⟨x#-m < p⟩ by simp
also
have ... = nth((g'(x#-m)#+n)#-length(env'), env2)
  using ⟨length(env') = n⟩
  diff_add_inverse2 ⟨g'(x#-m)∈nat⟩
  by simp
also
have ... = nth((g'(x#-m)#+n), env'@env2)
  using nth_append[OF ⟨env'∈list(M)⟩] ⟨n∈nat⟩ ⟨¬ g'(x#-m)#+n < length(env')⟩
  by simp
also
have ... = nth(?h'x, env'@env2)
  using 2 by simp
finally
have nth(x, env @ env1) = nth(?h'x, env'@env2) .
then show ?thesis .
qed
then show ?thesis by simp

```

qed

lemma *sum_type* :

assumes $m \in \text{nat}$ $n \in \text{nat}$ $p \in \text{nat}$ $q \in \text{nat}$

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

shows $rsum(f, g, m, n, p) \in (m\# + p) \rightarrow (n\# + q)$

proof -

let $?h = rsum(f, g, m, n, p)$

from $\langle m \in \text{nat} \rangle \langle n \in \text{nat} \rangle \langle q \in \text{nat} \rangle$

have $m \leq m\# + p$ $n \leq n\# + q$ $q \leq n\# + q$

using *add_le_self*[*of m*] *add_le_self2*[*of n q*] **by** *simp_all*

from $\langle p \in \text{nat} \rangle$

have $p = (m\# + p)\# - m$ **using** *diff_add_inverse2* **by** *simp*

{fix x

assume $1: x \in m\# + p$ $x < m$

with 1 **have** $?h'x = f'x$ $x \in m$

using *assms sum_inl ltD* **by** *simp_all*

with $\langle f \in m \rightarrow n \rangle$

have $?h'x \in n$ **by** *simp*

with $\langle n \in \text{nat} \rangle$ **have** $?h'x < n$ **using** *ltI* **by** *simp*

with $\langle n \leq n\# + q \rangle$

have $?h'x < n\# + q$ **using** *lt_trans2* **by** *simp*

then

have $?h'x \in n\# + q$ **using** *ltD* **by** *simp*

}

then have $1: ?h'x \in n\# + q$ **if** $x \in m\# + p$ $x < m$ **for** x **using** *that* .

{fix x

assume $1: x \in m\# + p$ $m \leq x$

then have $x < m\# + p$ $x \in \text{nat}$ **using** *ltI in_n_in_nat*[*of m\# + p*] *ltD* **by** *simp_all*

with 1

have $2: ?h'x = g'(x\# - m)\# + n$

using *assms sum_inr ltD* **by** *simp_all*

from *assms* $\langle x \in \text{nat} \rangle \langle p = m\# + p\# - m \rangle$

have $x\# - m < p$ **using** *diff_mono*[*OF* $____ \langle x < m\# + p \rangle \langle m \leq x \rangle$] **by** *simp*

then have $x\# - m \in p$ **using** *ltD* **by** *simp*

with $\langle g \in p \rightarrow q \rangle$

have $g'(x\# - m) \in q$ **by** *simp*

with $\langle q \in \text{nat} \rangle$ **have** $g'(x\# - m) < q$ **using** *ltI* **by** *simp*

with $\langle q \in \text{nat} \rangle$

have $(g'(x\# - m))\# + n < n\# + q$ **using** *add_lt_mono1*[*of g'(x\# - m)* $____ n, OF ____$

$\langle q \in \text{nat} \rangle$] **by** *simp*

with 2

have $?h'x \in n\# + q$ **using** *ltD* **by** *simp*

}

then have $2: ?h'x \in n\# + q$ **if** $x \in m\# + p$ $m \leq x$ **for** x **using** *that* .

have

$D: ?h'x \in n\# + q$ **if** $x \in m\# + p$ **for** x

using *that*

proof (*cases* $x < m$)

```

    case True
    then show ?thesis using 1 that by simp
next
    case False
    with ⟨m∈nat⟩ have m≤x using not_lt_iff_le that in_n_in_nat[of m#+p]
by simp
    then show ?thesis using 2 that by simp
qed
have A:function(?h) unfolding rsum_def using function_lam by simp
have x∈(m#+p)×(n#+q) if x∈?h for x
    using that lamE[of x m#+p _ x∈(m#+p)×(n#+q)] D unfolding
rsum_def
    by auto
then have B:?h⊆(m#+p)×(n#+q) ..
have m#+p⊆domain(?h)
    unfolding rsum_def using domain_lam by simp
with A B
show ?thesis using Pi_iff [THEN iffD2] by simp
qed

```

lemma *sum_type_id* :

```

assumes
  f ∈ length(env)→length(env')
  env ∈ list(M)
  env' ∈ list(M)
  env1 ∈ list(M)
shows
  rsum(f,id(length(env1)),length(env),length(env'),length(env1)) ∈
    (length(env)#+length(env1)) → (length(env')#+length(env1))
using assms length_type_id_fn_type sum_type
by simp

```

lemma *sum_type_id_aux2* :

```

assumes
  f ∈ m→n
  m ∈ nat n ∈ nat
  env1 ∈ list(M)
shows
  rsum(f,id(length(env1)),m,n,length(env1)) ∈
    (m#+length(env1)) → (n#+length(env1))
using assms id_fn_type sum_type
by auto

```

lemma *sum_action_id* :

```

assumes
  env ∈ list(M)
  env' ∈ list(M)
  f ∈ length(env)→length(env')
  env1 ∈ list(M)

```

$\bigwedge i . i < \text{length}(\text{env}) \implies \text{nth}(i, \text{env}) = \text{nth}(f'i, \text{env}')$
shows $\bigwedge i . i < \text{length}(\text{env})\# + \text{length}(\text{env1}) \implies$
 $\text{nth}(i, \text{env}@\text{env1}) = \text{nth}(\text{rsum}(f, \text{id}(\text{length}(\text{env1})), \text{length}(\text{env}), \text{length}(\text{env}'), \text{length}(\text{env1})) 'i, \text{env}'@\text{env1})$
proof -
from *assms*
have $\text{length}(\text{env}) \in \text{nat}$ **(is** $?m \in _$ **)** **by** *simp*
from *assms* **have** $\text{length}(\text{env}') \in \text{nat}$ **(is** $?n \in _$ **)** **by** *simp*
from *assms* **have** $\text{length}(\text{env1}) \in \text{nat}$ **(is** $?p \in _$ **)** **by** *simp*
note $\text{lenv} = \text{id_fn_action}[OF \langle ?p \in \text{nat} \rangle \langle \text{env1} \in \text{list}(M) \rangle]$
note $\text{lenv_ty} = \text{id_fn_type}[OF \langle ?p \in \text{nat} \rangle]$
{
fix i
assume $i < \text{length}(\text{env})\# + \text{length}(\text{env1})$
have $\text{nth}(i, \text{env}@\text{env1}) = \text{nth}(\text{rsum}(f, \text{id}(\text{length}(\text{env1})), ?m, ?n, ?p) 'i, \text{env}'@\text{env1})$
using $\text{sum_action}[OF \langle ?m \in \text{nat} \rangle \langle ?n \in \text{nat} \rangle \langle ?p \in \text{nat} \rangle \langle ?p \in \text{nat} \rangle \langle f \in ?m \rightarrow ?n \rangle$
 $\text{lenv_ty} \langle \text{env} \in \text{list}(M) \rangle \langle \text{env}' \in \text{list}(M) \rangle$
 $\langle \text{env1} \in \text{list}(M) \rangle \langle \text{env1} \in \text{list}(M) \rangle _$
 $_ _ \text{assms}(5) \text{lenv}$
 $] \langle i < ?m\# + \text{length}(\text{env1}) \rangle$ **by** *simp*
}
then show $\bigwedge i . i < ?m\# + \text{length}(\text{env1}) \implies$
 $\text{nth}(i, \text{env}@\text{env1}) = \text{nth}(\text{rsum}(f, \text{id}(?p), ?m, ?n, ?p) 'i, \text{env}'@\text{env1})$ **by** *simp*
qed

lemma *sum_action_id_aux* :

assumes

$f \in m \rightarrow n$

$\text{env} \in \text{list}(M)$

$\text{env}' \in \text{list}(M)$

$\text{env1} \in \text{list}(M)$

$\text{length}(\text{env}) = m$

$\text{length}(\text{env}') = n$

$\text{length}(\text{env1}) = p$

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

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

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

using *assms* *length_type* *id_fn_type* *sum_action_id*

by *auto*

definition

$\text{sum_id} :: [i, i] \Rightarrow i$ **where**

$\text{sum_id}(m, f) \equiv \text{rsum}(\lambda x \in 1. x, f, 1, 1, m)$

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

by (*unfold* *sum_id_def*, *subst* *sum_inl*, *auto*)

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

by(*subgoal_tac* $x \in \text{nat}$, *unfold* *sum_id_def*, *subst* *sum_inr*,
simp_all add: ltI, simp_all add: app_nm in_n_in_nat)

lemma *sum_id_tc_aux* :
 $p \in \text{nat} \implies q \in \text{nat} \implies f \in p \rightarrow q \implies \text{sum_id}(p, f) \in 1\# + p \rightarrow 1\# + q$
by (*unfold* *sum_id_def*, *rule* *sum_type*, *simp_all*)

lemma *sum_id_tc* :
 $n \in \text{nat} \implies m \in \text{nat} \implies f \in n \rightarrow m \implies \text{sum_id}(n, f) \in \text{succ}(n) \rightarrow \text{succ}(m)$
by(*rule* *ssubst*[*of succ*($n \rightarrow \text{succ}(m)$) $1\# + n \rightarrow 1\# + m$],
simp, *rule* *sum_id_tc_aux*, *simp_all*)

4.2 Renaming of formulas

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

primrec

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

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

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

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

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

by(*simp*, *rule* *subsetD*, *rule* *le_imp_subset*, *assumption*, *simp*)

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

by(*simp*, *rule* *subsetD*, *rule* *le_imp_subset*, *assumption*, *simp*)

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

by(*simp*, *rule* *subsetD*, *rule* *le_imp_subset*, *assumption*, *simp*)

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

by(*simp*, *rule* *subsetD*, *rule* *le_imp_subset*, *assumption*, *simp*)

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

by (*simp*, *rule* *Un_upper1_le*, *simp+*)

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

by (*simp*, *rule* *Un_upper2_le*, *simp+*)

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

by(*auto* *simp add: le_trans*[*OF Un_upper1_le*[*of* *arity*(p) *arity*(q)]])

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

$\text{arity}(q) \leq n$
by (*auto simp add: le_trans[OF Un_upper2_le[of arity(p) arity(q)]]*)

lemma *ren_tc* : $p \in \text{formula} \implies$
 $(\bigwedge n\ m\ f . n \in \text{nat} \implies m \in \text{nat} \implies f \in n \rightarrow m \implies \text{ren}(p)\ 'n\ 'm\ 'f \in \text{formula})$
by (*induct set:formula, auto simp add: app_nm sum_id_tc*)

lemma *arity_ren* :
fixes p
assumes $p \in \text{formula}$
shows $\bigwedge n\ m\ f . n \in \text{nat} \implies m \in \text{nat} \implies f \in n \rightarrow m \implies \text{arity}(p) \leq n \implies \text{arity}(\text{ren}(p)\ 'n\ 'm\ 'f) \leq m$
using *assms*
proof (*induct set:formula*)
case (*Member x y*)
then have $f\ 'x \in m\ f\ 'y \in m$
using *Member assms* **by** (*simp add: arity_meml apply_funtype, simp add: arity_memr apply_funtype*)
then show $?case$ **using** *Member* **by** (*simp add: Un_least_lt ltI*)
next
case (*Equal x y*)
then have $f\ 'x \in m\ f\ 'y \in m$
using *Equal assms* **by** (*simp add: arity_eql apply_funtype, simp add: arity_eqr apply_funtype*)
then show $?case$ **using** *Equal* **by** (*simp add: Un_least_lt ltI*)
next
case (*Nand p q*)
then have $\text{arity}(p) \leq \text{arity}(\text{Nand}(p, q))$
 $\text{arity}(q) \leq \text{arity}(\text{Nand}(p, q))$
by (*subst nand_ar1, simp, simp, simp, subst nand_ar2, simp+*)
then have $\text{arity}(p) \leq n$
and $\text{arity}(q) \leq n$ **using** *Nand*
by (*rule_tac j=arity(Nand(p,q)) in le_trans, simp, simp*)
then have $\text{arity}(\text{ren}(p)\ 'n\ 'm\ 'f) \leq m$ **and** $\text{arity}(\text{ren}(q)\ 'n\ 'm\ 'f) \leq m$
using *Nand* **by** *auto*
then show $?case$ **using** *Nand* **by** (*simp add: Un_least_lt*)
next
case (*Forall p*)
from *Forall* **have** $\text{succ}(n) \in \text{nat}\ \text{succ}(m) \in \text{nat}$ **by** *auto*
from *Forall* **have** $2: \text{sum_id}(n, f) \in \text{succ}(n) \rightarrow \text{succ}(m)$ **by** (*simp add: sum_id_tc*)
from *Forall* **have** $3: \text{arity}(p) \leq \text{succ}(n)$ **by** (*rule_tac n=arity(p) in natE, simp+*)
then have $\text{arity}(\text{ren}(p)\ ' \text{succ}(n)\ ' \text{succ}(m)\ ' \text{sum_id}(n, f)) \leq \text{succ}(m)$ **using**
 $\text{Forall}\ \langle \text{succ}(n) \in \text{nat} \rangle\ \langle \text{succ}(m) \in \text{nat} \rangle\ 2$ **by** *force*
then show $?case$ **using** *Forall* $2\ 3$ *ren_tc* *arity_type* *pred_le* **by** *auto*
qed

lemma *arity_forallE* : $p \in \text{formula} \implies m \in \text{nat} \implies \text{arity}(\text{Forall}(p)) \leq m \implies$


```

arity(p) ≤ succ(m)
by(rule_tac n=arity(p) in natE,erule arity_type,simp+)

lemma env_coincidence_sum_id :
  assumes m ∈ nat n ∈ nat
    ρ ∈ list(A) ρ' ∈ list(A)
    f ∈ n → m
    ∧ i . i < n ⇒ nth(i,ρ) = nth(f'i,ρ')
    a ∈ A j ∈ succ(n)
  shows nth(j,Cons(a,ρ)) = nth(sum_id(n,f) 'j,Cons(a,ρ'))
proof -
  let ?g=sum_id(n,f)
  have succ(n) ∈ nat using ⟨n∈nat⟩ by simp
  then have j ∈ nat using ⟨j∈succ(n)⟩ in_n in_nat by blast
  then have nth(j,Cons(a,ρ)) = nth(?g'j,Cons(a,ρ'))
  proof (cases rule:natE[OF ⟨j∈nat⟩])
    case 1
    then show ?thesis using assms sum_id0 by simp
  next
    case (2 i)
    with ⟨j∈succ(n)⟩ have succ(i)∈succ(n) by simp
    with ⟨n∈nat⟩ have i ∈ n using nat_succD assms by simp
    have f'i ∈ m using ⟨f∈n→m⟩ apply_type ⟨i∈n⟩ by simp
    then have f'i ∈ nat using in_n in_nat ⟨m∈nat⟩ by simp
    have nth(succ(i),Cons(a,ρ)) = nth(i,ρ) using ⟨i∈nat⟩ by simp
    also have ... = nth(f'i,ρ') using assms ⟨i∈n⟩ ltI by simp
    also have ... = nth(succ(f'i),Cons(a,ρ')) using ⟨f'i∈nat⟩ by simp
    also have ... = nth(?g'succ(i),Cons(a,ρ'))
    using assms sum_idS[OF ⟨n∈nat⟩ ⟨m∈nat⟩ ⟨f∈n→m⟩ ⟨i ∈ n⟩] cases by
  simp
  finally have nth(succ(i),Cons(a,ρ)) = nth(?g'succ(i),Cons(a,ρ')) .
  then show ?thesis using ⟨j=succ(i)⟩ by simp
qed
then show ?thesis .
qed

lemma sats_iff_sats_ren :
  assumes φ ∈ formula
  shows [ n ∈ nat ; m ∈ nat ; ρ ∈ list(M) ; ρ' ∈ list(M) ; f ∈ n → m ;
    arity(φ) ≤ n ;
    ∧ i . i < n ⇒ nth(i,ρ) = nth(f'i,ρ') ] ⇒
    sats(M,φ,ρ) ⇔ sats(M,ren(φ) 'n'm'f,ρ')
  using ⟨φ ∈ formula⟩
proof(induct φ arbitrary:n m ρ ρ' f)
  case (Member x y)
  have ren(Member(x,y)) 'n'm'f = Member(f'x,f'y) using Member assms arity_type
  by force
  moreover
  have x ∈ n using Member arity_mem1 by simp

```

```

moreover
  have  $y \in n$  using Member arity_memr by simp
  ultimately
  show ?case using Member ltI by simp
next
  case (Equal x y)
    have  $\text{ren}(\text{Equal}(x,y))'n'm'f = \text{Equal}(f'x,f'y)$  using Equal assms arity_type by
    force
    moreover
    have  $x \in n$  using Equal arity_eql by simp
    moreover
    have  $y \in n$  using Equal arity_eqr by simp
    ultimately show ?case using Equal ltI by simp
next
  case (Nand p q)
    have  $\text{ren}(\text{Nand}(p,q))'n'm'f = \text{Nand}(\text{ren}(p)'n'm'f, \text{ren}(q)'n'm'f)$  using Nand by
    simp
    moreover
    have  $\text{arity}(p) \leq n$  using Nand nand_ar1D by simp
    moreover from this
    have  $i \in \text{arity}(p) \implies i \in n$  for  $i$  using subsetD[OF le_imp_subset[OF  $\langle \text{arity}(p) \leq n \rangle$ ]] by simp
    moreover from this
    have  $i \in \text{arity}(p) \implies \text{nth}(i, \varrho) = \text{nth}(f'i, \varrho')$  for  $i$  using Nand ltI by simp
    moreover from this
    have  $\text{sats}(M, p, \varrho) \longleftrightarrow \text{sats}(M, \text{ren}(p)'n'm'f, \varrho')$  using  $\langle \text{arity}(p) \leq n \rangle$  Nand by simp
    have  $\text{arity}(q) \leq n$  using Nand nand_ar2D by simp
    moreover from this
    have  $i \in \text{arity}(q) \implies i \in n$  for  $i$  using subsetD[OF le_imp_subset[OF  $\langle \text{arity}(q) \leq n \rangle$ ]] by simp
    moreover from this
    have  $i \in \text{arity}(q) \implies \text{nth}(i, \varrho) = \text{nth}(f'i, \varrho')$  for  $i$  using Nand ltI by simp
    moreover from this
    have  $\text{sats}(M, q, \varrho) \longleftrightarrow \text{sats}(M, \text{ren}(q)'n'm'f, \varrho')$  using assms  $\langle \text{arity}(q) \leq n \rangle$  Nand
    by simp
    ultimately
    show ?case using Nand by simp
next
  case (Forall p)
    have  $0: \text{ren}(\text{Forall}(p))'n'm'f = \text{Forall}(\text{ren}(p)'succ(n)'succ(m)'sum\_id(n,f))$ 
    using Forall by simp
    have  $1: sum\_id(n,f) \in succ(n) \rightarrow succ(m)$  (is  $?g \in \_$ ) using sum_id_tc Forall
    by simp
    then have  $2: \text{arity}(p) \leq succ(n)$ 
    using Forall le_trans[of  $\_ succ(pred(\text{arity}(p)))$ ] succpred_leI by simp
    have  $succ(n) \in nat$   $succ(m) \in nat$  using Forall by auto
    then have  $A: \bigwedge j. j < succ(n) \implies \text{nth}(j, \text{Cons}(a, \varrho)) = \text{nth}(?g'j, \text{Cons}(a, \varrho'))$ 
    if  $a \in M$  for  $a$ 
    using that env_coincidence_sum_id Forall ltD by force

```

```

    have
      sats(M,p,Cons(a,ρ)) ⟷ sats(M,ren(p)‘succ(n)‘succ(m)‘?g,Cons(a,ρ)) if a∈M
  for a
  proof -
    have C:Cons(a,ρ) ∈ list(M) Cons(a,ρ)∈list(M) using Forall that by auto
    have sats(M,p,Cons(a,ρ)) ⟷ sats(M,ren(p)‘succ(n)‘succ(m)‘?g,Cons(a,ρ))
      using Forall(2)[OF ⟨succ(n)∈nat⟩ ⟨succ(m)∈nat⟩ C(1) C(2) 1 2 A[OF
⟨a∈M⟩]] by simp
    then show ?thesis .
  qed
  then show ?case using Forall 0 1 2 by simp
qed

end
theory Utils
  imports ZF-Constructible.Formula
begin

This theory encapsulates some ML utilities

ML_file⟨Utils.ml⟩

end
theory Renaming_Auto
  imports
    Renaming
    Utils
keywords
  rename :: thy_decl % ML
and
  simple_rename :: thy_decl % ML
and
  src
and
  tgt
abbrevs
  simple_rename =

begin

lemmas nat_succI = nat_succ_iff[THEN iffD2]
ML_file⟨Renaming_ML.ml⟩
ML⟨
  open Renaming_ML

  fun renaming_def mk_ren name from to ctxt =
    let val to = to |> Syntax.read_term ctxt
        val from = from |> Syntax.read_term ctxt
        val (tc_lemma,action_lemma,fvs,r) = mk_ren from to ctxt
        val (tc_lemma,action_lemma) = (fix_vars tc_lemma fvs ctxt , fix_vars

```

```

action_lemma fvs ctxt)
  val ren_fun_name = Binding.name (name ^ _fn)
  val ren_fun_def = Binding.name (name ^ _fn_def)
  val ren_thm = Binding.name (name ^ _thm)
in
  Local_Theory.note ((ren_thm, []), [tc_lemma, action_lemma]) ctxt |> snd
|>
  Local_Theory.define ((ren_fun_name, NoSyn), ((ren_fun_def, []), r)) |> snd
end;
>

ML<
local

  val ren_parser = Parse.position (Parse.string --
    (Parse.$$$ src |-- Parse.string |-- Parse.$$$ tgt -- Parse.string));

  val _ =
    Outer_Syntax.local_theory command_keyword <rename> ML setup for synthetic
    definitions
      (ren_parser >> (fn ((name, (from, to)), _) => renaming_def sum_rename
        name from to ))

  val _ =
    Outer_Syntax.local_theory command_keyword <simple_rename> ML setup
    for synthetic definitions
      (ren_parser >> (fn ((name, (from, to)), _) => renaming_def ren_thm name
        from to ))

in
end
>
end

```

5 Automatic synthesis of formulas

```

theory Synthetic_Definition
imports
  Utils
keywords
  synthesize :: thy_decl % ML
and
  synthesize_notc :: thy_decl % ML
and
  generate_schematic :: thy_decl % ML
and
  arity_theorem :: thy_decl % ML
and
  manual_schematic :: thy_goal_stmt % ML

```

```

and
  manual_arity :: thy_goal_stmt % ML
and
  from_schematic
and
  for
and
  from_definition
and
  assuming
and
  intermediate

begin

named_theorems fm_definitions Definitions of synthesized formulas.

named_theorems iff_sats Theorems for synthesising formulas.

named_theorems arity Theorems for arity of formulas.

named_theorems arity_aux Auxiliary theorems for calculating arities.

ML<
val $' = curry ((op $) o swap)
infix $'

infix 6 &&&
val op &&& = Utils.&&&

infix 6 ***
val op *** = Utils.***

fun arity_goal intermediate def_name lthy =
  let
    val thm = Proof_Context.get_thm lthy (def_name ^ _def)
    val (_, tm, _) = Utils.thm_concl_tm lthy (def_name ^ _def)
    val (def, tm) = tm |> Utils.dest_eq_tms'
    fun first_lambdas (Abs (body as (_, ty, _))) =
      if ty = @{typ i}
      then (op ::) (Utils.dest_abs body |>> Utils.var_i ||> first_lambdas)
      else Utils.dest_abs body |> first_lambdas o #2
      | first_lambdas _ = []
    val (def, vars) = Term.strip_comb def
    val vs = vars @ first_lambdas tm
    val def = fold (op $') vs def
    val hyps = map (fn v => Utils.mem_ v Utils.nat_ |> Utils.tp) vs
    val concl = @{const IFOL.eq(i)} $ (@{const arity} $ def) $ Var ((ar, 0), @{typ
i})

```

```

    val g_iff = Logic.list_implies (hypos, Utils.tp concl)
    val attrs = if intermediate then [] else @{attributes [arity]}
  in
    (g_iff, arity_ ^ def_name ^ (if intermediate then ' else ), attrs, thm, vs)
  end

fun manual_arity intermediate def_name pos lthy =
  let
    val (goal, thm_name, attrs, _, _) = arity_goal intermediate def_name lthy
  in
    Proof.theorem NONE (fn thms => Utils.display theorem pos
      o Local_Theory.note ((Binding.name thm_name,
        attrs), hd thms))
    [[(goal, [])] lthy
  end

fun prove_arity thms goal ctxt =
  let
    val rules = (Named_Theorems.get ctxt named_theorems <arity>) @
      (Named_Theorems.get ctxt named_theorems <arity_aux>)
  in
    Goal.prove ctxt [] [] goal
    (K (rewrite_goal_tac ctxt thms 1 THEN Method.insert_tac ctxt rules 1 THEN
      asm_simp_tac ctxt 1))
  end

fun auto_arity intermediate def_name pos lthy =
  let
    val (goal, thm_name, attrs, def_thm, vs) = arity_goal intermediate def_name
    lthy
    val intermediate_text = if intermediate then intermediate else
    val help = You can manually prove the arity_theorem by typing:\n
      ^ manual_arity ^ intermediate_text ^ for \ ^ def_name ^ \n
    val thm = prove_arity [def_thm] goal lthy
      handle ERROR s => help ^ \n\n ^ s |> Exn.reraise o ERROR
    val thm = Utils.fix_vars thm (map Utils.freeName vs) lthy
  in
    Local_Theory.note ((Binding.name thm_name, attrs), [thm]) lthy |> Utils.display
    theorem pos
  end

fun prove_tc_form goal thms ctxt =
  Goal.prove ctxt [] [] goal (K (rewrite_goal_tac ctxt thms 1 THEN TypeCheck.typecheck_tac
    ctxt))

fun prove_sats_tm thm_auto thms goal ctxt =
  let
    val ctxt' = ctxt |> Simplifier.add_simp (hd thm_auto)
  in

```

```

    Goal.prove ctxt [] [] goal
    (K (rewrite_goal_tac ctxt thms 1 THEN PARALLEL_ALLGOALS (asm_simp_tac
    ctxt'))))
  end

```

```

fun prove_sats_iff goal ctxt = Goal.prove ctxt [] [] goal (K (asm_simp_tac ctxt
1))

```

```

fun is_mem (@{const mem} $ _ $ _) = true
  | is_mem _ = false

```

```

fun pre_synth_thm_sats term set env vars vs lthy =
  let
    val (_, tm, ctxt1) = Utils.thm_concl_tm lthy term
    val (thm_refs, ctxt2) = Variable.import true [Proof_Context.get_thm lthy term]
    ctxt1 |>> #2
    val vs' = map (Thm.term_of o #2) vs
    val vars' = map (Thm.term_of o #2) vars
    val r_tm = tm |> Utils.dest_lhs_def |> fold (op $') vs'
    val sats = @{const apply} $ (@{const satisfies} $ set $ r_tm) $ env
    val sats' = @{const IFOL.eq(i)} $ sats $ (@{const succ} $ @{const zero})
  in
    { vars = vars'
    , vs = vs'
    , sats = sats'
    , thm_refs = thm_refs
    , lthy = ctxt2
    , env = env
    , set = set
    }
  end

```

```

fun synth_thm_sats_gen name lhs hypos pos attribs aux_funs environment lthy =
  let
    val ctxt = (#prepare_ctxt aux_funs) lthy
    val ctxt = Utils.add_to_context (Utils.freeName (#set environment)) ctxt
    val (new_vs, ctxt') = (#create_variables aux_funs) (#vs environment, ctxt)
    val new_hyps = (#create_hyps aux_funs) (#vs environment, new_vs)
    val concl = (#make_concl aux_funs) (lhs, #sats environment, new_vs)
    val g_iff = Logic.list_implies (new_hyps @ hypos, Utils.tp concl)
    val thm = (#prover aux_funs) g_iff ctxt'
    val thm = Utils.fix_vars thm (map Utils.freeName ((#vars environment) @
    new_vs)) lthy
  in
    Local_Theory.note ((name, attribs), [thm]) lthy |> Utils.display theorem pos
  end

```

```

fun synth_thm_sats_iff def_name lhs hypos pos environment =
  let

```

```

    val subst = Utils.zip_with (I *** I) (#vs environment)
    fun subst_nth (@{const nth} $ v $ _) new_vs = AList.lookup (op =) (subst
new_vs) v |> the
    | subst_nth (t1 $ t2) new_vs = (subst_nth t1 new_vs) $ (subst_nth t2 new_vs)
    | subst_nth (Abs (v, ty, t)) new_vs = Abs (v, ty, subst_nth t new_vs)
    | subst_nth t _ = t
    val name = Binding.name (def_name ^ __iff_sats)
    val iff_sats_attrib = @{attributes [iff_sats]}
    val aux_funs = { prepare_ctxt = fold Utils.add_to_context (map Utils.freeName
(#vs environment))
                    , create_variables = fn (vs, ctxt) => Variable.variant_fixes (map
Utils.freeName vs) ctxt |>> map Utils.var_i
                    , create_hyps = fn (vs, new_vs) => Utils.zip_with (fn (v, nv) =>
Utils.eq_ (Utils.nth v (#env environment)) nv) vs new_vs |> map Utils.tp
                    , make_concl = fn (lhs, rhs, new_vs) => @{const IFOL.iff} $
(subst_nth lhs new_vs) $ rhs
                    , prover = prove_sats_iff
                    }
    in
    synth_thm_sats_gen name lhs hyps pos iff_sats_attrib aux_funs environment
end

fun synth_thm_sats_fm def_name lhs hyps pos thm_auto environment =
let
    val name = Binding.name (sats_ ^ def_name ^ _fm)
    val simp_attrib = @{attributes [simp]}
    val aux_funs = { prepare_ctxt = I
                    , create_variables = K [] *** I
                    , create_hyps = K []
                    , make_concl = fn (rhs, lhs, _) => @{const IFOL.iff} $ lhs $ rhs
                    , prover = prove_sats_tm thm_auto (#thm_refs environment)
                    }
    in
    synth_thm_sats_gen name lhs hyps pos simp_attrib aux_funs environment
end

fun synth_thm_tc def_name term hyps vars pos lthy =
let
    val (_,tm,ctxt1) = Utils.thm_concl_tm lthy term
    val (thm_refs,ctxt2) = Variable.import true [Proof_Context.get_thm lthy term]
ctxt1 |>> #2
    val vars' = map (Thm.term_of o #2) vars
    val tc_attrib = @{attributes [TC]}
    val r_tm = tm |> Utils.dest_lhs_def |> fold (op $) vars'
    val concl = @{const mem} $ r_tm $ @{const formula}
    val g = Logic.list_implies(hyps, Utils.tp concl)
    val thm = prove_tc_form g thm_refs ctxt2
    val name = Binding.name (def_name ^ _fm_type)
    val thm = Utils.fix_vars thm (map Utils.freeName vars') ctxt2

```



```

in
  Local_Theory.note ((name, tc_attrib), [thm]) lthy |> Utils.display theorem pos
end

fun synthetic_def def_name thm_ref pos tc auto thy =
  let
    val thm = Proof_Context.get_thm thy thm_ref
    val thm_vars = rev (Term.add_vars (Thm.full_prop_of thm) [])
    val ((_, inst), thm_tms, _) = Variable.import true [thm] thy
    val vars = map (fn v => (v, the (Vars.lookup inst v))) thm_vars
    val (tm, hyps) = thm_tms |> hd |> Thm.concl_of &&& Thm.premis_of
    val (lhs, rhs) = tm |> Utils.dest_iff_tms o Utils.dest_trueprop
    val ((set, t), env) = rhs |> Utils.dest_sats_fm
    fun relevant ts (@{const mem} $ t $ _) =
      (not (t = @{term 0})) andalso
      (not (Term.is_Free t) orelse member (op =) ts (t |> Term.dest_Free |>
#1))
      | relevant _ _ = false
    val t_vars = sort_strings (Term.add_free_names t [])
    val vs = filter (Ord_List.member String.compare t_vars o #1 o #1 o #1) vars
    val at = fold_rev (lambda o Thm.term_of o #2) vs t
    val hyps' = filter (relevant t_vars o Utils.dest_trueprop) hyps
    val def_attrs = @{attributes [fm_definitions]}
  in
    Local_Theory.define ((Binding.name (def_name ^ _fm), NoSyn),
      ((Binding.name (def_name ^ _fm_def), def_attrs), at)) thy
    |>> (#2 #> I *** single) |> Utils.display theorem pos |>
    (if tc then synth_thm_tc def_name (def_name ^ _fm_def) hyps' vs pos else
I) |>
    (if auto then
      pre_synth_thm_sats (def_name ^ _fm_def) set env vars vs
      #> I &&& #lthy
      #> #1 &&& uncurry (synth_thm_sats_fm def_name lhs hyps pos thm_tms)
      #> uncurry (synth_thm_sats_iff def_name lhs hyps pos)
    else I)
  end

fun prove_schematic thms goal ctxt =
  let
    val rules = Named_Theorems.get ctxt named_theorems <iff_sats>
  in
    Goal.prove ctxt [] [] goal
    (K (rewrite_goal_tac ctxt thms 1 THEN REPEAT1 (CHANGED (resolve_tac
ctxt rules 1 ORELSE asm_simp_tac ctxt 1))))
  end

val valid_assumptions = [ (nonempty, Utils.mem_ @{term 0})
]

```

```

fun pre_schematic_def target assuming lthy =
let
  val thm = Proof_Context.get_thm lthy (target ^ _def)
  val (vars, tm, ctxt1) = Utils.thm_concl_tm lthy (target ^ _def)
  val (const, tm) = tm |> Utils.dest_eq_tms' o Utils.dest_trueprop |>> #1 o
strip_comb
  val t_vars = sort_strings (Term.add_free_names tm [])
  val vs = List.filter (#1 #> #1 #> #1 #> Ord_List.member String.compare
t_vars) vars
    |> List.filter ((curry op = @{typ i}) o #2 o #1)
    |> List.map (Utils.var_i o #1 o #1 o #1)
  fun first_lambdas (Abs (body as (_, ty, _))) =
    if ty = @{typ i}
    then (op ::) (Utils.dest_abs body |>> Utils.var_i ||> first_lambdas)
    else Utils.dest_abs body |> first_lambdas o #2
    | first_lambdas _ = []
  val vs = vs @ (first_lambdas tm)
  val ctxt1' = fold Utils.add_to_context (map Utils.freeName vs) ctxt1
  val (set, ctxt2) = Variable.variant_fixes [A] ctxt1' |>> Utils.var_i o hd
  val class = @{const setclass} $ set
  val (env, ctxt3) = Variable.variant_fixes [env] ctxt2 |>> Utils.var_i o hd
  val assumptions = filter (member (op =) assuming o #1) valid_assumptions
|> map #2
  val hyps = (List.map (fn v => Utils.tp (Utils.mem_ v Utils.nat_)) vs)
    @ [Utils.tp (Utils.mem_ env (Utils.list_set))]
    @ Utils.zip_with (fn (f,x) => Utils.tp (f x)) assumptions (replicate
(length assumptions) set)
  val args = class :: map (fn v => Utils.nth_ v env) vs
  val (fm_name, ctxt4) = Variable.variant_fixes [fm] ctxt3 |>> hd
  val fm_type = fold (K (fn acc => Type (fun, [@{typ i}, acc]))) vs @{typ i}
  val fm = Var ((fm_name, 0), fm_type)
  val lhs = fold (op $') args const
  val fm_app = fold (op $') vs fm
  val sats = @{const apply} $ (@{const satisfies} $ set $ fm_app) $ env
  val rhs = @{const IFOL.eq(i)} $ sats $ (@{const succ} $ @{const zero})
  val concl = @{const IFOL.iff} $ lhs $ rhs
  val schematic = Logic.list_implies (hyps, Utils.tp concl)
in
  (schematic, ctxt4, thm, set, env, vs)
end

fun str_join _ [] =
| str_join _ [s] = s
| str_join c (s :: ss) = s ^ c ^ (str_join c ss)

fun schematic_def def_name target assuming pos lthy =
let
  val (schematic, ctxt, thm, set, env, vs) = pre_schematic_def target assuming
lthy

```

```

    val assuming_text = if null assuming then else assuming ^ (map (fn s => \
^ s ^\) assuming |> str_join )
    val help = You can manually prove the schematic_goal by typing:\n
    ^ manual_schematic [sch_name] for \ ^ target ^\ ^ assuming_text ^\n
    ^ And then complete the synthesis with:\n
    ^ synthesize \ ^ target ^\ from_schematic [sch_name]\n
    ^ In both commands, «sch_name» must be the same and, if omitted,
will be defaulted to sats_ ^ target ^ _fm_auto.\n
    ^ You can also try adding new assumptions and/or synthesizing definitions
of sub-terms.
    val thm = prove_schematic [thm] schematic ctxt
    handle ERROR s => help ^\n\n ^ s |> Exn.reraise o ERROR
    val thm = Utils.fix_vars thm (map Utils.freeName (set :: env :: vs)) lthy
in
    Local_Theory.note ((Binding.name def_name, []), [thm]) lthy |> Utils.display
theorem pos
end

fun schematic_synthetic_def def_name target assuming pos tc auto =
  (synthetic_def def_name (sats_ ^ def_name ^ _fm_auto) pos tc auto)
  o (schematic_def (sats_ ^ def_name ^ _fm_auto) target assuming pos)

fun manual_schematic def_name target assuming pos lthy =
  let
    val (schematic, _, _, _, _, _) = pre_schematic_def target assuming lthy
  in
    Proof.theorem NONE (fn thmss => Utils.display theorem pos
    o Local_Theory.note ((Binding.name def_name, []),
hd thmss))
    [[(schematic, [])]] lthy
  end
end

```

ML

local

```

    val simple_from_schematic_synth_constdecl =
      Parse.string --| (Parse.$$$ from_schematic)
      >> (fn bndg => synthetic_def bndg (sats_ ^ bndg ^ _fm_auto))

    val full_from_schematic_synth_constdecl =
      Parse.string -- ((Parse.$$$ from_schematic |-- Parse.thm))
      >> (fn (bndg, thm) => synthetic_def bndg (#1 (thm |>> Facts.ref_name)))

    val full_from_definition_synth_constdecl =
      Parse.string -- ((Parse.$$$ from_definition |-- Parse.string)) -- (Scan.optional
      (Parse.$$$ assuming |-- Scan.repeat Parse.string) [])
      >> (fn ((bndg, target), assuming) => schematic_synthetic_def bndg target
      assuming)

```

```

val simple_from_definition_synth_constdecl =
  Parse.string -- (Parse.$$$ from_definition |-- (Scan.optional (Parse.$$$ assuming
|-- Scan.repeat Parse.string)) [])
  >> (fn (bndg, assuming) => schematic_synthetic_def bndg bndg assuming)

val synth_constdecl =
  Parse.position (full_from_schematic_synth_constdecl || simple_from_schematic_synth_constdecl
  || full_from_definition_synth_constdecl
  || simple_from_definition_synth_constdecl)

val full_schematic_decl =
  Parse.string -- ((Parse.$$$ for |-- Parse.string)) -- (Scan.optional (Parse.$$$
assuming |-- Scan.repeat Parse.string) [])

val simple_schematic_decl =
  (Parse.$$$ for |-- Parse.string >> (fn name => sats_ ^ name ^ _fm_auto)
&&& I) -- (Scan.optional (Parse.$$$ assuming |-- Scan.repeat Parse.string) [])

val schematic_decl = Parse.position (full_schematic_decl || simple_schematic_decl)

val _ =
  Outer_Syntax.local_theory command_keyword⟨synthesize⟩ ML setup for
synthetic definitions
  (synth_constdecl >> (fn (f,p) => f p true true))

val _ =
  Outer_Syntax.local_theory command_keyword⟨synthesize_notc⟩ ML setup
for synthetic definitions
  (synth_constdecl >> (fn (f,p) => f p false false))

val _ = Outer_Syntax.local_theory command_keyword⟨generate_schematic⟩
ML setup for schematic goals
  (schematic_decl >> (fn (((bndg,target), assuming),p) => schematic_def
bndg target assuming p))

val _ = Outer_Syntax.local_theory_to_proof command_keyword⟨manual_schematic⟩
ML setup for schematic goals
  (schematic_decl >> (fn (((bndg,target), assuming),p) => manual_schematic
bndg target assuming p))

val arity_parser = Parse.position ((Scan.option (Parse.$$$ intermediate) >>
is_some) -- (Parse.$$$ for |-- Parse.string))

val _ = Outer_Syntax.local_theory_to_proof command_keyword⟨manual_arity⟩
ML setup for arities
  (arity_parser >> (fn ((intermediate, target), pos) => manual_arity
intermediate target pos))

val _ = Outer_Syntax.local_theory command_keyword⟨arity_theorem⟩ ML

```

```

setup for arities
  (arity_parser >> (fn ((intermediate, target), pos) => auto_arity intermediate
    target pos))

in

end
>

```

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

```
end
```

6 Aids to internalize formulas

```
theory Internalizations
```

```
  imports
```

```
    ZF-Constructible.DPow_absolute
```

```
    Synthetic_Definition
```

```
begin
```

```
definition
```

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

```
definition
```

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

```
lemma empty_intf :
```

```
  infinity_ax(M) ⟹
    (∃ z[M]. empty(M,z))
by (auto simp add: empty_def infinity_ax_def)
```

```
lemma Transset_intf :
```

```
  Transset(M) ⟹ y ∈ x ⟹ x ∈ M ⟹ y ∈ M
by (simp add: Transset_def, auto)
```

```
definition
```

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

```
lemma (in M_basic) choice_ax_abs :
```

```
  choice_ax(M) ⟷ (∀ x[M]. ∃ a[M]. ∃ f[M]. Ord(a) ∧ f ∈ surj(a,x))
unfolding choice_ax_def
```

by *simp*

notation *Member* ($\langle \cdot \in / _ \rangle$)
notation *Equal* ($\langle \cdot = / _ \rangle$)
notation *Nand* ($\langle \cdot \neg' (_ \wedge / _) \rangle$)
notation *And* ($\langle \cdot \wedge / _ \rangle$)
notation *Or* ($\langle \cdot \vee / _ \rangle$)
notation *Iff* ($\langle \cdot \leftrightarrow / _ \rangle$)
notation *Implies* ($\langle \cdot \rightarrow / _ \rangle$)
notation *Neg* ($\langle \cdot \neg _ \rangle$)
notation *Forall* ($\langle \cdot (\forall (_)) \rangle$)
notation *Exists* ($\langle \cdot (\exists (_)) \rangle$)

notation *subset_fm* ($\langle \cdot \subseteq / _ \rangle$)
notation *succ_fm* ($\langle \cdot \text{succ}'(_) \text{ is } _ \rangle$)
notation *empty_fm* ($\langle \cdot \text{is empty} \rangle$)
notation *fun_apply_fm* ($\langle \cdot _ \text{ is } _ \rangle$)
notation *big_union_fm* ($\langle \cdot \bigcup _ \text{ is } _ \rangle$)
notation *upair_fm* ($\langle \cdot \{ _, _ \} \text{ is } _ \rangle$)
notation *ordinal_fm* ($\langle \cdot \text{is ordinal} \rangle$)

abbreviation

fm_surjection :: $[i, i, i] \Rightarrow i (\langle _ \text{ surjects } _ \text{ to } _ \rangle)$ **where**
fm_surjection(*f*, *A*, *B*) \equiv *surjection_fm*(*A*, *B*, *f*)

abbreviation

fm_typedfun :: $[i, i, i] \Rightarrow i (\langle _ : _ \rightarrow _ \rangle)$ **where**
fm_typedfun(*f*, *A*, *B*) \equiv *typed_function_fm*(*A*, *B*, *f*)

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

lemma *nth_closed* :

assumes *env* ∈ *list*(*A*) *0* ∈ *A*
shows *nth*(*n*, *env*) ∈ *A*
using *assms* **unfolding** *nth_def* **by** (*induct env*; *simp*)

lemma *mem_model_iff_sats* [*iff_sats*]:

$[| 0 \in A; \text{nth}(i, \text{env}) = x; \text{env} \in \text{list}(A)|]$
 $\implies (x \in A) \longleftrightarrow \text{sats}(A, \text{Exists}(\text{Equal}(0, 0)), \text{env})$
using *nth_closed*[*of env A i*]
by *auto*

lemma *subset_iff_sats*[*iff_sats*]:

$\text{nth}(i, \text{env}) = x \implies \text{nth}(j, \text{env}) = y \implies i \in \text{nat} \implies j \in \text{nat} \implies$
 $\text{env} \in \text{list}(A) \implies \text{subset}(\#\#A, x, y) \longleftrightarrow \text{sats}(A, \text{subset_fm}(i, j), \text{env})$
using *sats_subset_fm'* **by** *simp*

lemma *not_mem_model_iff_sats* [*iff_sats*]:

$[| 0 \in A; \text{nth}(i, \text{env}) = x; \text{env} \in \text{list}(A)|]$

```

==> (∀ x . x ∉ A) ↔ sats(A, Neg(Exists(Equal(0,0))), env)
by auto

lemma top_iff_sats [iff_sats]:
  env ∈ list(A) ⇒ 0 ∈ A ⇒ sats(A, Exists(Equal(0,0)), env)
by auto

lemma prefix1_iff_sats [iff_sats]:
  assumes
    x ∈ nat env ∈ list(A) 0 ∈ A a ∈ A
  shows
    a = nth(x,env) ↔ sats(A, Equal(0,x#+1), Cons(a,env))
    nth(x,env) = a ↔ sats(A, Equal(x#+1,0), Cons(a,env))
    a ∈ nth(x,env) ↔ sats(A, Member(0,x#+1), Cons(a,env))
    nth(x,env) ∈ a ↔ sats(A, Member(x#+1,0), Cons(a,env))
  using assms nth_closed
  by simp_all

lemma prefix2_iff_sats [iff_sats]:
  assumes
    x ∈ nat env ∈ list(A) 0 ∈ A a ∈ A b ∈ A
  shows
    b = nth(x,env) ↔ sats(A, Equal(1,x#+2), Cons(a,Cons(b,env)))
    nth(x,env) = b ↔ sats(A, Equal(x#+2,1), Cons(a,Cons(b,env)))
    b ∈ nth(x,env) ↔ sats(A, Member(1,x#+2), Cons(a,Cons(b,env)))
    nth(x,env) ∈ b ↔ sats(A, Member(x#+2,1), Cons(a,Cons(b,env)))
  using assms nth_closed
  by simp_all

lemma prefix3_iff_sats [iff_sats]:
  assumes
    x ∈ nat env ∈ list(A) 0 ∈ A a ∈ A b ∈ A c ∈ A
  shows
    c = nth(x,env) ↔ sats(A, Equal(2,x#+3), Cons(a,Cons(b,Cons(c,env))))
    nth(x,env) = c ↔ sats(A, Equal(x#+3,2), Cons(a,Cons(b,Cons(c,env))))
    c ∈ nth(x,env) ↔ sats(A, Member(2,x#+3), Cons(a,Cons(b,Cons(c,env))))
    nth(x,env) ∈ c ↔ sats(A, Member(x#+3,2), Cons(a,Cons(b,Cons(c,env))))
  using assms nth_closed
  by simp_all

lemmas FOL_sats_iff = sats_Nand_iff sats_Forall_iff sats_Neg_iff sats_And_iff
  sats_Or_iff sats_Implies_iff sats_Iff_iff sats_Exists_iff

lemma nth_ConsI: [nth(n,l) = x; n ∈ nat] ⇒ nth(succ(n), Cons(a,l)) = x
by simp

lemmas nth_rules = nth_0 nth_ConsI nat_0I nat_succI
lemmas sep_rules = nth_0 nth_ConsI FOL_iff_sats function_iff_sats
  fun_plus_iff_sats successor_iff_sats

```

omega_iff_sats FOL_sats_iff Replace_iff_sats

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

lemmas *fm_defs* =

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

lemmas *formulas_def [fm_definitions] = fm_defs*

is_iterates_fm_def iterates_MH_fm_def is_wfrec_fm_def is_recfun_fm_def
is_transrec_fm_def
is_nat_case_fm_def quasinat_fm_def number1_fm_def ordinal_fm_def finite_ordinal_fm_def
cartprod_fm_def sum_fm_def Inr_fm_def Inl_fm_def
formula_functor_fm_def
Memrel_fm_def transset_fm_def subset_fm_def pre_image_fm_def restriction_fm_def
list_functor_fm_def tl_fm_def quaselist_fm_def Cons_fm_def Nil_fm_def

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

declare *rtran_closure_iff_sats [iff_sats] tran_closure_iff_sats [iff_sats]*
is_eclose_iff_sats [iff_sats]

arity_theorem *for rtran_closure_fm*

arity_theorem *for tran_closure_fm*

arity_theorem *for rtran_closure_mem_fm*

end

7 Some enhanced theorems on recursion

theory *Recursion_Thms*

imports *ZF-Constructible.Datatype_absolute*

begin

— Removing arities from inherited simpset

declare *arity_And [simp del] arity_Or [simp del] arity_Implies [simp del]*
arity_Exists [simp del] arity_Iff [simp del]
arity_subset_fm [simp del] arity_ordinal_fm [simp del] arity_transset_fm [simp del]

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

lemma *fld_restrict_eq : a ∈ A ⟹ (r ∩ A × A) - ``{a} = (r - ``{a} ∩ A)*
by (*force*)

lemma *fld_restrict_mono* : $\text{relation}(r) \implies A \subseteq B \implies r \cap A \times A \subseteq r \cap B \times B$
by (*auto*)

lemma *fld_restrict_dom* :
assumes $\text{relation}(r)$ $\text{domain}(r) \subseteq A$ $\text{range}(r) \subseteq A$
shows $r \cap A \times A = r$
proof (*rule equalityI,blast,rule subsetI*)
{ **fix** x
assume xr : $x \in r$
from xr *assms* **have** $\exists a b . x = \langle a, b \rangle$ **by** (*simp add: relation_def*)
then obtain $a b$ **where** $\langle a, b \rangle \in r$ $\langle a, b \rangle \in r \cap A \times A$ $x \in r \cap A \times A$
using *assms xr*
by force
then have $x \in r \cap A \times A$ **by simp**
}
then show $x \in r \implies x \in r \cap A \times A$ **for** x .
qed

definition *tr_down* :: $[i, i] \Rightarrow i$
where $\text{tr_down}(r, a) = (r^+)^{-1} \{a\}$

lemma *tr_downD* : $x \in \text{tr_down}(r, a) \implies \langle x, a \rangle \in r^+$
by (*simp add: tr_down_def vimage_singleton_iff*)

lemma *pred_down* : $\text{relation}(r) \implies r^{-1} \{a\} \subseteq \text{tr_down}(r, a)$
by (*simp add: tr_down_def vimage_mono r_subset_trancl*)

lemma *tr_down_mono* : $\text{relation}(r) \implies x \in r^{-1} \{a\} \implies \text{tr_down}(r, x) \subseteq \text{tr_down}(r, a)$
by (*rule subsetI, simp add: tr_down_def, auto dest: underD, force simp add: underI*
r_into_trancl trancl_trans)

lemma *rest_eq* :
assumes $\text{relation}(r)$ **and** $r^{-1} \{a\} \subseteq B$ **and** $a \in B$
shows $r^{-1} \{a\} = (r \cap B \times B)^{-1} \{a\}$
proof (*intro equalityI subsetI*)
fix x
assume $x \in r^{-1} \{a\}$
then
have $x \in B$ **using** *assms* **by** (*simp add: subsetD*)
from $\langle x \in r^{-1} \{a\} \rangle$
have $\langle x, a \rangle \in r$ **using** *underD* **by simp**
then
show $x \in (r \cap B \times B)^{-1} \{a\}$ **using** $\langle x \in B \rangle \langle a \in B \rangle$ *underI* **by simp**
next
from *assms*
show $x \in (r \cap B \times B)^{-1} \{a\}$ **if** $x \in (r \cap B \times B)^{-1} \{a\}$ **for** x
using *vimage_mono that* **by auto**
qed

lemma *wfrec_restr_eq* : $r' = r \cap A \times A \implies \text{wfrec}[A](r, a, H) = \text{wfrec}(r', a, H)$
by (*simp add: wfrec_on_def*)

lemma *wfrec_restr* :
assumes *rr*: *relation*(*r*) **and** *wfr*: *wf*(*r*)
shows $a \in A \implies \text{tr_down}(r, a) \subseteq A \implies \text{wfrec}(r, a, H) = \text{wfrec}[A](r, a, H)$
proof (*induct a arbitrary: A rule: wf_induct_raw[OF wfr]*)
case (1 *a*)
have *wfRa* : *wf*[*A*](*r*)
using *wf_subset wfr wf_on_def Int_lower1* **by** *simp*
from *pred_down rr*
have $r - \{a\} \subseteq \text{tr_down}(r, a)$.
with 1
have $r - \{a\} \subseteq A$ **by** (*force simp add: subset_trans*)
{
fix *x*
assume $x_a : x \in r - \{a\}$
with $\langle r - \{a\} \subseteq A \rangle$
have $x \in A$..
from *pred_down rr*
have $b : r - \{x\} \subseteq \text{tr_down}(r, x)$.
then
have $\text{tr_down}(r, x) \subseteq \text{tr_down}(r, a)$
using *tr_down_mono x_a rr* **by** *simp*
with 1
have $\text{tr_down}(r, x) \subseteq A$ **using** *subset_trans* **by** *force*
have $\langle x, a \rangle \in r$ **using** *x_a underD* **by** *simp*
with 1 $\langle \text{tr_down}(r, x) \subseteq A \rangle \langle x \in A \rangle$
have $\text{wfrec}(r, x, H) = \text{wfrec}[A](r, x, H)$ **by** *simp*
}
then
have $x \in r - \{a\} \implies \text{wfrec}(r, x, H) = \text{wfrec}[A](r, x, H)$ **for** x .
then
have *Eq1* : $(\lambda x \in r - \{a\} . \text{wfrec}(r, x, H)) = (\lambda x \in r - \{a\} . \text{wfrec}[A](r, x, H))$
using *lam_cong* **by** *simp*

from *assms*
have $\text{wfrec}(r, a, H) = H(a, \lambda x \in r - \{a\} . \text{wfrec}(r, x, H))$ **by** (*simp add: wfrec*)
also
have $\dots = H(a, \lambda x \in r - \{a\} . \text{wfrec}[A](r, x, H))$
using *assms Eq1* **by** *simp*
also from 1 $\langle r - \{a\} \subseteq A \rangle$
have $\dots = H(a, \lambda x \in (r \cap A \times A) - \{a\} . \text{wfrec}[A](r, x, H))$
using *assms restr_eq* **by** *simp*
also from $\langle a \in A \rangle$
have $\dots = H(a, \lambda x \in (r - \{a\}) \cap A . \text{wfrec}[A](r, x, H))$
using *fld_restrict_eq* **by** *simp*
also from $\langle a \in A \rangle \langle \text{wf}[A](r) \rangle$

```

    have ... = wfrec[A](r,a,H) using wfrec_on by simp
    finally show ?case .
qed

lemmas wfrec_tr_down = wfrec_restr[OF _ _ subset_refl]

lemma wfrec_trans_restr : relation(r)  $\implies$  wf(r)  $\implies$  trans(r)  $\implies$  r-“{a}  $\subseteq$  A  $\implies$ 
a  $\in$  A  $\implies$ 
  wfrec(r, a, H) = wfrec[A](r, a, H)
  by(subgoal_tac tr_down(r,a)  $\subseteq$  A,auto simp add : wfrec_restr tr_down_def
  trancl_eq_r)

lemma field_trancl : field(r $^+$ ) = field(r)
  by (blast intro: r_into_trancl dest!: trancl_type [THEN subsetD])

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

lemma RrelI : x  $\in$  A  $\implies$  y  $\in$  A  $\implies$  R(x,y)  $\implies$   $\langle$ x,y $\rangle \in$  Rrel(R,A)
  unfolding Rrel_def by simp

lemma Rrel_mem: Rrel(mem,x) = Memrel(x)
  unfolding Rrel_def Memrel_def ..

lemma relation_Rrel: relation(Rrel(R,d))
  unfolding Rrel_def relation_def by simp

lemma field_Rrel: field(Rrel(R,d))  $\subseteq$  d
  unfolding Rrel_def by auto

lemma Rrel_mono : A  $\subseteq$  B  $\implies$  Rrel(R,A)  $\subseteq$  Rrel(R,B)
  unfolding Rrel_def by blast

lemma Rrel_restr_eq : Rrel(R,A)  $\cap$  B $\times$ B = Rrel(R,A $\cap$ B)
  unfolding Rrel_def by blast

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

  using Rrel_mem field_Rrel by blast

lemma restrict_trancl_Rrel:
  assumes R(w,y)
  shows restrict(f,Rrel(R,d)-“{y}”w
    = restrict(f,(Rrel(R,d) $^+$ )-“{y}”w
proof (cases y $\in$ d)
  let ?r=Rrel(R,d) and ?s=(Rrel(R,d)) $^+$ 

```

```

case True
show ?thesis
proof (cases w ∈ d)
  case True
  with ⟨y ∈ d⟩ assms
  have ⟨w, y⟩ ∈ ?r
    unfolding Rrel_def by blast
  then
  have ⟨w, y⟩ ∈ ?s
    using r_subset_trancl[of ?r] relation_Rrel[of R d] by blast
  with ⟨⟨w, y⟩ ∈ ?r⟩
  have w ∈ ?r-“{y} w ∈ ?s-“{y}
    using vimage_singleton_iff by simp_all
  then
  show ?thesis by simp
next
case False
then
have w ∉ domain(restrict(f, ?r-“{y}))
  using subsetD[OF field_Rrel[of R d]] by auto
moreover from ⟨w ∉ d⟩
have w ∉ domain(restrict(f, ?s-“{y}))
  using subsetD[OF field_Rrel[of R d], of w] field_trancl[of ?r]
  fieldI1[of w y ?s] by auto
ultimately
have restrict(f, ?r-“{y})'w = 0 restrict(f, ?s-“{y})'w = 0
  unfolding apply_def by auto
then show ?thesis by simp
qed
next
let ?r = Rrel(R, d)
let ?s = ?r+
case False
then
have ?r-“{y} = 0
  unfolding Rrel_def by blast
then
have w ∉ ?r-“{y} by simp
with ⟨y ∉ d⟩ assms
have y ∉ field(?s)
  using field_trancl_subsetD[OF field_Rrel[of R d]] by force
then
have w ∉ ?s-“{y}
  using vimage_singleton_iff by blast
with ⟨w ∉ ?r-“{y}⟩
show ?thesis by simp
qed

```

lemma restrict_trans_eq:

```

assumes  $w \in y$ 
shows  $\text{restrict}(f, \text{Memrel}(\text{eclose}(\{x\})) - \{\{y\}\})'w$ 
       $= \text{restrict}(f, (\text{Memrel}(\text{eclose}(\{x\}))^\wedge +) - \{\{y\}\})'w$ 
using assms restrict_trancl_Rrel[of mem] Rrel_mem by (simp)

lemma wf_eq_trancl:
  assumes  $\bigwedge f y . H(y, \text{restrict}(f, R - \{\{y\}\})) = H(y, \text{restrict}(f, R^\wedge + - \{\{y\}\}))$ 
  shows  $\text{wfrec}(R, x, H) = \text{wfrec}(R^\wedge +, x, H)$  (is  $\text{wfrec}(?r, \_, \_) = \text{wfrec}(?r', \_, \_)$ )
proof -
  have  $\text{wfrec}(R, x, H) = \text{wftrec}(?r^\wedge +, x, \lambda y f . H(y, \text{restrict}(f, ?r - \{\{y\}\})))$ 
    unfolding wfrec_def ..
  also
  have  $\dots = \text{wftrec}(?r^\wedge +, x, \lambda y f . H(y, \text{restrict}(f, (?r^\wedge +) - \{\{y\}\})))$ 
    using assms by simp
  also
  have  $\dots = \text{wfrec}(?r^\wedge +, x, H)$ 
    unfolding wfrec_def using trancl_eq_r[OF relation_trancl trans_trancl] by
simp
  finally
  show ?thesis .
qed

lemma transrec_equal_on_Ord:
assumes
   $\bigwedge x f . \text{Ord}(x) \implies \text{foo}(x, f) = \text{bar}(x, f)$ 
   $\text{Ord}(\alpha)$ 
shows
   $\text{transrec}(\alpha, \text{foo}) = \text{transrec}(\alpha, \text{bar})$ 
proof -
  have  $\text{transrec}(\beta, \text{foo}) = \text{transrec}(\beta, \text{bar})$  if  $\text{Ord}(\beta)$  for  $\beta$ 
    using that
  proof (induct rule: trans_induct)
    case (step  $\beta$ )
    have  $\text{transrec}(\beta, \text{foo}) = \text{foo}(\beta, \lambda x \in \beta . \text{transrec}(x, \text{foo}))$ 
      using def_transrec[of  $\lambda x . \text{transrec}(x, \text{foo})$ ] foo] by blast
    also from assms and step
    have  $\dots = \text{bar}(\beta, \lambda x \in \beta . \text{transrec}(x, \text{foo}))$ 
      by simp
    also from step
    have  $\dots = \text{bar}(\beta, \lambda x \in \beta . \text{transrec}(x, \text{bar}))$ 
      by (auto)
    also
    have  $\dots = \text{transrec}(\beta, \text{bar})$ 
      using def_transrec[of  $\lambda x . \text{transrec}(x, \text{bar})$ ] bar, symmetric]
      by blast
    finally
    show  $\text{transrec}(\beta, \text{foo}) = \text{transrec}(\beta, \text{bar})$  .
  qed
with assms

```

show *?thesis* **by** *simp*
qed

— Next theorem is very similar to $\llbracket \bigwedge x f. \text{Ord}(x) \implies ?foo(x, f) = ?bar(x, f); \text{Ord}(?\alpha) \rrbracket \implies \text{transrec}(?\alpha, ?foo) = \text{transrec}(?\alpha, ?bar)$

lemma (in *M_eclose*) *transrec_equal_on_M*:

assumes

$\bigwedge x f. M(x) \implies M(f) \implies foo(x, f) = bar(x, f)$
 $\bigwedge \beta. M(\beta) \implies \text{transrec_replacement}(M, is_foo, \beta) \text{ relation2}(M, is_foo, foo)$
 $\text{strong_replacement}(M, \lambda x y. y = \langle x, \text{transrec}(x, foo) \rangle)$
 $\forall x[M]. \forall g[M]. \text{function}(g) \longrightarrow M(foo(x, g))$
 $M(\alpha) \text{ Ord}(\alpha)$

shows

$\text{transrec}(\alpha, foo) = \text{transrec}(\alpha, bar)$

proof -

have $M(\text{transrec}(x, foo))$ **if** $\text{Ord}(x)$ **and** $M(x)$ **for** x

using *that assms transrec_closed[of is_foo]*

by *simp*

have $\text{transrec}(\beta, foo) = \text{transrec}(\beta, bar)$ $M(\text{transrec}(\beta, foo))$ **if** $\text{Ord}(\beta)$ $M(\beta)$ **for** β

using *that*

proof (*induct rule:trans_induct*)

case (*step* β)

moreover

assume $M(\beta)$

moreover

note $\langle \text{Ord}(\beta) \implies M(\beta) \implies M(\text{transrec}(\beta, foo)) \rangle$

ultimately

show $M(\text{transrec}(\beta, foo))$ **by** *blast*

with *step* $\langle M(\beta) \rangle$ $\langle \bigwedge x. \text{Ord}(x) \implies M(x) \implies M(\text{transrec}(x, foo)) \rangle$

$\langle \text{strong_replacement}(M, \lambda x y. y = \langle x, \text{transrec}(x, foo) \rangle) \rangle$

have $M(\lambda x \in \beta. \text{transrec}(x, foo))$

using *Ord_in_Ord transM[of _ β]*

by (*rule_tac lam_closed*) *auto*

have $\text{transrec}(\beta, foo) = foo(\beta, \lambda x \in \beta. \text{transrec}(x, foo))$

using *def_transrec[of $\lambda x. \text{transrec}(x, foo)$ foo]* **by** *blast*

also from *assms* **and** $\langle M(\lambda x \in \beta. \text{transrec}(x, foo)) \rangle$ $\langle M(\beta) \rangle$

have $\dots = bar(\beta, \lambda x \in \beta. \text{transrec}(x, foo))$

by *simp*

also from *step* **and** $\langle M(\beta) \rangle$

have $\dots = bar(\beta, \lambda x \in \beta. \text{transrec}(x, bar))$

using *transM[of _ β]* **by** (*auto*)

also

have $\dots = \text{transrec}(\beta, bar)$

using *def_transrec[of $\lambda x. \text{transrec}(x, bar)$ bar , *symmetric*]*

by *blast*

finally

show $\text{transrec}(\beta, foo) = \text{transrec}(\beta, bar)$.

qed

with *assms*

show ?thesis by simp
qed

```

lemma ordermap_restr_eq:
  assumes well_ord(X,r)
  shows ordermap(X, r) = ordermap(X, r ∩ X × X)
proof -
  let ?A = λx. Order.pred(X, x, r)
  let ?B = λx. Order.pred(X, x, r ∩ X × X)
  let ?F = λx f. f “ ?A(x)
  let ?G = λx f. f “ ?B(x)
  let ?P = λ z. z ∈ X ⟶ wfrec(r ∩ X × X, z, λx f. f “ ?A(x)) = wfrec(r ∩ X ×
X, z, λx f. f “ ?B(x))
  have pred_eq:
    Order.pred(X, x, r ∩ X × X) = Order.pred(X, x, r) if x ∈ X for x
  unfolding Order.pred_def using that by auto
  from assms
  have wf_onX: wf(r ∩ X × X) unfolding well_ord_def wf_on_def by simp
  {
    have ?P(z) for z
    proof(induct rule: wf_induct[where P=?P, OF wf_onX])
      case (1 x)
      {
        assume x ∈ X
        from 1
        have lam_eq:
          (λw ∈ (r ∩ X × X). -“ {x}. wfrec(r ∩ X × X, w, ?F)) =
          (λw ∈ (r ∩ X × X). -“ {x}. wfrec(r ∩ X × X, w, ?G)) (is ?L=?R)
        proof -
          have wfrec(r ∩ X × X, w, ?F) = wfrec(r ∩ X × X, w, ?G) if
w ∈ (r ∩ X × X) -“ {x} for w
          using 1 that by auto
          then show ?thesis using lam_cong[OF refl] by simp
        qed
        then
        have wfrec(r ∩ X × X, x, ?F) = ?L “ ?A(x)
          using wfrec[OF wf_onX, of x ?F] by simp
        also have ... = ?R “ ?B(x)
          using lam_eq pred_eq[OF ‹x ∈ ‹_›] by simp
        also
        have ... = wfrec(r ∩ X × X, x, ?G)
          using wfrec[OF wf_onX, of x ?G] by simp
        finally
        have wfrec(r ∩ X × X, x, ?F) = wfrec(r ∩ X × X, x, ?G) by simp
      }
    }
  then
  show ?case by simp
qed

```

```

    }
  then
    show ?thesis
      unfolding ordermap_def wfrec_on_def using Int_ac by simp
qed

end

```

8 The binder *Least*

```

theory Least
  imports
    Internalizations

```

```
begin
```

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

```

lemma Least_Ord:  $(\mu \alpha. R(\alpha)) = (\mu \alpha. Ord(\alpha) \wedge R(\alpha))$ 
  unfolding Least_def by (simp add:lt_Ord)

```

```

lemma Ord_Least_cong:
  assumes  $\bigwedge y. Ord(y) \implies R(y) \longleftrightarrow Q(y)$ 
  shows  $(\mu \alpha. R(\alpha)) = (\mu \alpha. Q(\alpha))$ 

```

```

proof -
  from assms
  have  $(\mu \alpha. Ord(\alpha) \wedge R(\alpha)) = (\mu \alpha. Ord(\alpha) \wedge Q(\alpha))$ 
    by simp
  then
  show ?thesis using Least_Ord by simp
qed

```

definition

```

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

```

definition

```

least_fm ::  $[i,i] \Rightarrow i$  where
least_fm(q,i)  $\equiv And(ordinal\_fm(i),$ 
   $Or(And(empty\_fm(i),Forall(Implies(ordinal\_fm(0),Neg(q)))),$ 
     $And(Exists(And(q,Equal(0,succ(i)))),$ 
       $Forall(Implies(And(ordinal\_fm(0),Member(0,succ(i))),Neg(q))))))$ 

```

```

lemma least_fm_type[TC] :  $i \in nat \implies q \in formula \implies least\_fm(q,i) \in formula$ 
  unfolding least_fm_def
  by simp

```


lemmas *basic_fm_simps* = *sats_subset_fm'* *sats_transset_fm'* *sats_ordinal_fm'*

lemma *sats_least_fm* :

assumes *p_iff_sats*:

$\bigwedge a. a \in A \implies P(a) \longleftrightarrow \text{sats}(A, p, \text{Cons}(a, \text{env}))$

shows

$\llbracket y \in \text{nat}; \text{env} \in \text{list}(A) ; 0 \in A \rrbracket$
 $\implies \text{sats}(A, \text{least_fm}(p, y), \text{env}) \longleftrightarrow$
 $\text{least}(\#\#A, P, \text{nth}(y, \text{env}))$

using *nth_closed_p_iff_sats* **unfolding** *least_def* *least_fm_def*

by (*simp add:basic_fm_simps*)

lemma *least_iff_sats* [*iff_sats*]:

assumes *is_Q_iff_sats*:

$\bigwedge a. a \in A \implies \text{is_Q}(a) \longleftrightarrow \text{sats}(A, q, \text{Cons}(a, \text{env}))$

shows

$\llbracket \text{nth}(j, \text{env}) = y; j \in \text{nat}; \text{env} \in \text{list}(A); 0 \in A \rrbracket$
 $\implies \text{least}(\#\#A, \text{is_Q}, y) \longleftrightarrow \text{sats}(A, \text{least_fm}(q, j), \text{env})$

using *sats_least_fm* [*OF is_Q_iff_sats, of j, symmetric*]

by *simp*

lemma *least_conj*: $a \in M \implies \text{least}(\#\#M, \lambda x. x \in M \wedge Q(x), a) \longleftrightarrow \text{least}(\#\#M, Q, a)$

unfolding *least_def* **by** *simp*

context *M_trivial*

begin

8.1 Uniqueness, absoluteness and closure under *Least*

lemma *unique_least*:

assumes *M(a)* *M(b)* *least(M, Q, a)* *least(M, Q, b)*

shows *a=b*

proof -

from *assms*

have *Ord(a)* *Ord(b)*

unfolding *least_def*

by *simp_all*

then

consider (*le*) *a ∈ b* | *a=b* | (*ge*) *b ∈ a*

using *Ord_linear*[*OF* $\langle \text{Ord}(a) \rangle$ $\langle \text{Ord}(b) \rangle$] **by** *auto*

then

show *?thesis*

proof(*cases*)

case *le*

then show *?thesis* **using** *assms* **unfolding** *least_def* **by** *auto*

next

case *ge*

then show *?thesis* **using** *assms* **unfolding** *least_def* **by** *auto*

qed
qed

lemma *least_abs*:

assumes $\bigwedge x. Q(x) \implies \text{Ord}(x) \implies \exists y[M]. Q(y) \wedge \text{Ord}(y) \ M(a)$

shows $\text{least}(M, Q, a) \longleftrightarrow a = (\mu x. Q(x))$

unfolding *least_def*

proof (*cases* $\forall b[M]. \text{Ord}(b) \longrightarrow \neg Q(b)$; *intro iffI*; *simp add:assms*)

case *True*

with *assms*

have $\neg (\exists i. \text{Ord}(i) \wedge Q(i))$ **by** *blast*

then

show $0 = (\mu x. Q(x))$ **using** *Least_0* **by** *simp*

then

show $\text{ordinal}(M, \mu x. Q(x)) \wedge (\text{empty}(M, \text{Least}(Q)) \vee Q(\text{Least}(Q)))$

by *simp*

next

assume $\exists b[M]. \text{Ord}(b) \wedge Q(b)$

then

obtain *i* **where** $M(i) \ \text{Ord}(i) \ Q(i)$ **by** *blast*

assume $a = (\mu x. Q(x))$

moreover

note $\langle M(a) \rangle$

moreover from $\langle Q(i) \rangle \ \langle \text{Ord}(i) \rangle$

have $Q(\mu x. Q(x))$ **(is ?G)**

by (*blast intro:LeastI*)

moreover

have $(\forall b[M]. \text{Ord}(b) \wedge b \in (\mu x. Q(x)) \longrightarrow \neg Q(b))$ **(is ?H)**

using *less_LeastE[of Q _ False]*

by (*auto, drule_tac ltI, simp, blast*)

ultimately

show $\text{ordinal}(M, \mu x. Q(x)) \wedge (\text{empty}(M, \mu x. Q(x)) \wedge (\forall b[M]. \text{Ord}(b) \longrightarrow \neg Q(b)) \vee ?G \wedge ?H)$

by *simp*

next

assume $1: \exists b[M]. \text{Ord}(b) \wedge Q(b)$

then

obtain *i* **where** $M(i) \ \text{Ord}(i) \ Q(i)$ **by** *blast*

assume $\text{Ord}(a) \wedge (a = 0 \wedge (\forall b[M]. \text{Ord}(b) \longrightarrow \neg Q(b)) \vee Q(a) \wedge (\forall b[M].$

$\text{Ord}(b) \wedge b \in a \longrightarrow \neg Q(b)))$

with *1*

have $\text{Ord}(a) \ Q(a) \ \forall b[M]. \text{Ord}(b) \wedge b \in a \longrightarrow \neg Q(b)$

by *blast+*

moreover from *this* **and** *assms*

have $\text{Ord}(b) \implies b \in a \implies \neg Q(b)$ **for** *b*

by (*auto dest:transM*)

moreover from *this* **and** $\langle \text{Ord}(a) \rangle$

have $b < a \implies \neg Q(b)$ **for** *b*

unfolding *lt_def* **using** *Ord_in_Ord* **by** *blast*

```

ultimately
show  $a = (\mu x. Q(x))$ 
  using Least_equality by simp
qed

```

```

lemma Least_closed:
  assumes  $\bigwedge x. Q(x) \implies \text{Ord}(x) \implies \exists y[M]. Q(y) \wedge \text{Ord}(y)$ 
  shows  $M(\mu x. Q(x))$ 
  using assms Least_le[of Q] Least_0[of Q]
  by (cases  $(\exists i[M]. \text{Ord}(i) \wedge Q(i))$ ) (force dest:transM ltD)+

```

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

```

lemma least_abs':
  assumes  $\bigwedge x. Q(x) \implies M(x) M(a)$ 
  shows  $\text{least}(M, Q, a) \longleftrightarrow a = (\mu x. Q(x))$ 
  using assms least_abs[of Q] by auto

```

```

lemma Least_closed':
  assumes  $\bigwedge x. Q(x) \implies M(x)$ 
  shows  $M(\mu x. Q(x))$ 
  using assms Least_closed[of Q] by auto

```

```

end — M_trivial

```

```

end

```

9 Fully relational versions of higher order construct

```

theory Higher_Order_Constructs
  imports
    Recursion_Thms
    Least
begin

syntax
  __sats :: [i, i, i]  $\Rightarrow o$  ((_, _  $\models$  _) [36,36,36] 25)

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

```

```

definition
  is_If :: [i  $\Rightarrow o, o, i, i, i$ ]  $\Rightarrow o$  where
  is_If(M, b, t, f, r)  $\equiv (b \longrightarrow r=t) \wedge (\neg b \longrightarrow r=f)$ 

```

```

lemma (in M_trans) If_abs:
  is_If(M, b, t, f, r)  $\longleftrightarrow r = \text{If}(b, t, f)$ 
by (simp add: is_If_def)

```

```

definition

```

```

is_If_fm :: [i,i,i,i] => i where
is_If_fm( $\varphi, t, f, r$ )  $\equiv$  Or(And( $\varphi, \text{Equal}(t, r)$ ), And(Neg( $\varphi$ ), Equal( $f, r$ )))

lemma is_If_fm_type [TC]:  $\varphi \in \text{formula} \implies t \in \text{nat} \implies f \in \text{nat} \implies r \in \text{nat}$ 
 $\implies$ 
is_If_fm( $\varphi, t, f, r$ )  $\in \text{formula}$ 
unfolding is_If_fm_def by auto

lemma sats_is_If_fm:
assumes Qsats:  $Q \longleftrightarrow A, \text{env} \models \varphi \text{ env} \in \text{list}(A)$ 
shows is_If_fm( $\#\#A, Q, \text{nth}(t, \text{env}), \text{nth}(f, \text{env}), \text{nth}(r, \text{env})$ )  $\longleftrightarrow A, \text{env} \models$ 
is_If_fm( $\varphi, t, f, r$ )
using assms unfolding is_If_def is_If_fm_def by auto

lemma is_If_fm_iff_sats [iff_sats]:
assumes Qsats:  $Q \longleftrightarrow A, \text{env} \models \varphi$  and
nth( $t, \text{env}$ ) = ta nth( $f, \text{env}$ ) = fa nth( $r, \text{env}$ ) = ra
 $t \in \text{nat } f \in \text{nat } r \in \text{nat } \text{env} \in \text{list}(A)$ 
shows is_If_fm( $\#\#A, Q, ta, fa, ra$ )  $\longleftrightarrow A, \text{env} \models \text{is\_If\_fm}(\varphi, t, f, r)$ 
using assms sats_is_If_fm[of  $Q A \varphi \text{env } t f r$ ] by simp

lemma arity_is_If_fm [arity]:
 $\varphi \in \text{formula} \implies t \in \text{nat} \implies f \in \text{nat} \implies r \in \text{nat} \implies$ 
arity(is_If_fm( $\varphi, t, f, r$ )) = arity( $\varphi$ )  $\cup \text{succ}(t) \cup \text{succ}(r) \cup \text{succ}(f)$ 
unfolding is_If_fm_def
by auto

definition
is_The :: [i=>o,i=>o,i] => o where
is_The( $M, Q, i$ )  $\equiv (Q(i) \wedge (\exists x[M]. Q(x) \wedge (\forall y[M]. Q(y) \longrightarrow y = x))) \vee$ 
 $(\neg(\exists x[M]. Q(x) \wedge (\forall y[M]. Q(y) \longrightarrow y = x))) \wedge \text{empty}(M, i)$ 

lemma (in  $M\_trans$ ) The_abs:
assumes  $\bigwedge x. Q(x) \implies M(x) M(a)$ 
shows is_The( $M, Q, a$ )  $\longleftrightarrow a = (\text{THE } x. Q(x))$ 
proof (cases  $\exists x[M]. Q(x) \wedge (\forall y[M]. Q(y) \longrightarrow y = x)$ )
case True
with assms
show ?thesis
unfolding is_The_def
by (intro iffI the_equality[symmetric])
(auto, blast intro:theI)
next
case False
with  $\langle \bigwedge x. Q(x) \implies M(x) \rangle$ 
have  $\neg(\exists x. Q(x) \wedge (\forall y. Q(y) \longrightarrow y = x))$ 
by auto

```

```

then
have The(Q) = 0
  by (intro the_0) auto
with assms and False
show ?thesis
  unfolding is_The_def
  by auto
qed

```

definition

$is_recursor :: [i \Rightarrow o, i, [i, i, i] \Rightarrow o, i, i] \Rightarrow o$ **where**
 $is_recursor(M, a, is_b, k, r) \equiv is_transrec(M, \lambda n f ntc. is_nat_case(M, a,$
 $\lambda m bmfm.$
 $\exists fm[M]. fun_apply(M, f, m, fm) \wedge is_b(m, fm, bmfm), n, ntc), k, r)$

lemma (in M_eclose) recursor_abs:

assumes Ord(k) **and**
 $types: M(a) M(k) M(r)$ **and**
 $b_iff: \bigwedge m f bmfm. M(m) \Rightarrow M(f) \Rightarrow M(bmfm) \Rightarrow is_b(m, f, bmfm) \longleftrightarrow bmfm$
 $= b(m, f)$ **and**
 $b_closed: \bigwedge m f bmfm. M(m) \Rightarrow M(f) \Rightarrow M(b(m, f))$ **and**
 $repl: transrec_replacement(M, \lambda n f ntc. is_nat_case(M, a,$
 $\lambda m bmfm. \exists fm[M]. fun_apply(M, f, m, fm) \wedge is_b(m, fm, bmfm), n, ntc),$
 $k)$
shows
 $is_recursor(M, a, is_b, k, r) \longleftrightarrow r = recursor(a, b, k)$
unfolding $is_recursor_def$ $recursor_def$
using $assms$
apply (rule_tac transrec_abs)
apply (auto simp: relation2_def)
apply (rule nat_case_abs[THEN iffD1, **where** $is_b1 = \lambda m bmfm.$
 $\exists fm[M]. fun_apply(M, _, m, fm) \wedge is_b(m, fm, bmfm)$])
apply (auto simp: relation1_def)
apply (rule nat_case_abs[THEN iffD2, **where** $is_b1 = \lambda m bmfm.$
 $\exists fm[M]. fun_apply(M, _, m, fm) \wedge is_b(m, fm, bmfm)$])
apply (auto simp: relation1_def)
done

definition

$is_wfrec_on :: [i \Rightarrow o, [i, i, i] \Rightarrow o, i, i, i, i] \Rightarrow o$ **where**
 $is_wfrec_on(M, MH, A, r, a, z) == is_wfrec(M, MH, r, a, z)$

lemma (in M_trancl) trans_wfrec_on_abs:

$[| wf(r); trans(r); relation(r); M(r); M(a); M(z);$
 $wfrec_replacement(M, MH, r); relation2(M, MH, H);$
 $\forall x[M]. \forall g[M]. function(g) \longrightarrow M(H(x, g));$
 $r - \{a\} \subseteq A; a \in A |]$

```

==> is_wfrec_on(M,MH,A,r,a,z) <=> z==wfrec[A](r,a,H)
using trans_wfrec_abs wfrec_trans_restr
unfolding is_wfrec_on_def by simp

end

```

10 Automatic relativization of terms and formulas

Relativization of terms and formulas. Relativization of formulas shares relativized terms as far as possible; assuming that the witnesses for the relativized terms are always unique.

```

theory Relativization
imports
  ZF-Constructible.Datatype_absolute
  Higher_Order_Constructs
keywords
  relativize :: thy_decl % ML
and
  relativize_tm :: thy_decl % ML
and
  reldb_add :: thy_decl % ML
and
  reldb_rem :: thy_decl % ML
and
  relationalize :: thy_decl % ML
and
  rel_closed :: thy_goal_stmt % ML
and
  is_iff_rel :: thy_goal_stmt % ML
and
  univalent :: thy_goal_stmt % ML
and
  absolute
and
  functional
and
  relational
and
  external
and
  for

begin

ML_file⟨Relativization_Database.ml⟩

ML⟨
  structure Absoluteness = Named_Thms

```

```

    (val name = @{binding absolut}
      val description = Theorems of absolute terms and predicates.)
  >
setup⟨Absoluteness.setup⟩

lemmas relative_abs =
  M_trans.empty_abs
  M_trans.pair_abs
  M_trivial.cartprod_abs
  M_trans.union_abs
  M_trans.inter_abs
  M_trans.setdiff_abs
  M_trans.Union_abs
  M_trivial.cons_abs

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

  M_trivial.nat_case_abs

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

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

  M_trivial.list_case_abs
  M_trivial.hd_abs
  M_trivial.tl_abs
  M_trivial.least_abs'
  M_eclose.transrec_abs

```

```

M_trans.If_abs
M_trans.The_abs
M_eclose.recursor_abs
M_trancl.trans_wfrec_abs
M_trancl.trans_wfrec_on_abs

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

declare relative_abs[absolut]
declare datatype_abs[absolut]

ML⟨
signature Relativization =
  sig
    structure Data: GENERIC_DATA
    val Rel_add: attribute
    val Rel_del: attribute
    val add_rel_const : Database.mode -> term -> term -> Data.T -> Data.T
    val add_constant : Database.mode -> string -> string -> Proof.context ->
Proof.context
    val rem_constant : (term -> Data.T -> Data.T) -> string -> Proof.context ->
Proof.context
    val db: Data.T
    val init_db : Data.T -> theory -> theory
    val get_db : Proof.context -> Data.T
    val relativ_fm: bool -> bool -> term -> Data.T -> (term * (term * term)) list
* Proof.context * term list * bool -> term -> term * ((term * (term * term)) list
* term list * term list * Proof.context)
    val relativ_tm: bool -> bool -> term option -> term -> Data.T -> (term *
(term * term)) list * Proof.context -> term -> term * (term * (term * term)) list
* Proof.context
    val read_new_const : Proof.context -> string -> term
    val relativ_tm_frm': bool -> bool -> term -> Data.T -> Proof.context -> term
-> term option * term
    val relativize_def: bool -> bool -> bool -> bstring -> string -> Position.T ->

```



```

Proof.context -> Proof.context
  val relativize_tm: bool -> bstring -> string -> Position.T -> Proof.context ->
Proof.context
  val rel_closed_goal : string -> Position.T -> Proof.context -> Proof.state
  val iff_goal : string -> Position.T -> Proof.context -> Proof.state
  val univalent_goal : string -> Position.T -> Proof.context -> Proof.state
end

structure Relativization : Relativization = struct

infix 6 &&&
val op &&& = Utils.&&&

infix 6 ***
val op *** = Utils.***

infix 6 @@
val op @@ = Utils.@@

infix 6 ---
val op --- = Utils.---

fun insert_abs2rel ((t, u), db) = ((t, u), Database.insert Database.abs2rel (t, t) db)

fun insert_rel2is ((t, u), db) = Database.insert Database.rel2is (t, u) db

(* relativization db of relation constructors *)
val db = [ (@{const relation}, @{const Relative.is_relation})
, (@{const function}, @{const Relative.is_function})
, (@{const mem}, @{const mem})
, (@{const True}, @{const True})
, (@{const False}, @{const False})
, (@{const Memrel}, @{const membership})
, (@{const trancl}, @{const tran_closure})
, (@{const IFOL.eq(i)}, @{const IFOL.eq(i)})
, (@{const Subset}, @{const Relative.subset})
, (@{const quasinat}, @{const Relative.is_quasinat})
, (@{const apply}, @{const Relative.fun_apply})
, (@{const Upair}, @{const Relative.upair})
]
|> List.foldr (insert_rel2is o insert_abs2rel) Database.empty
|> Database.insert Database.abs2is (@{const Pi}, @{const is_funspace})

fun var_i v = Free (v, @{typ i})
fun var_io v = Free (v, @{typ i => o})
val const_name = #1 o dest_Const

val lookup_tm = AList.lookup (op aconv)
val update_tm = AList.update (op aconv)

```

```

val join_tm = AList.join (op aconv) (K #1)

val conj_ = Utils.binop @{const IFOL.conj}

(* generic data *)
structure Data = Generic_Data
(
  type T = Database.db
  val empty = Database.empty (* Should we initialize this outside this file? *)
  val merge = Database.merge
);

fun init_db db = Context.theory_map (Data.put db)

fun get_db thy = Data.get (Context.Proof thy)

val read_const = Proof_Context.read_const {proper = true, strict = true}
val read_new_const = Proof_Context.read_term_pattern

fun add_rel_const mode c t = Database.insert mode (c, t)

fun get_consts thm =
  let val (c_rel, rhs) = Thm.concl_of thm |> Utils.dest_trueprop |>
    Utils.dest_iff_tms |>> head_of
  in case try Utils.dest_eq_tms rhs of
    SOME tm => (c_rel, tm |> #2 |> head_of)
  | NONE => (c_rel, rhs |> Utils.dest_mem_tms |> #2 |> head_of)
  end

fun add_rule thm rs =
  let val (c_rel, c_abs) = get_consts thm
  (* in (add_rel_const Database.rel2is c_abs c_rel o add_rel_const Database.abs2rel
  c_abs c_abs) rs *)
  in (add_rel_const Database.abs2rel c_abs c_abs o add_rel_const Database.abs2is
  c_abs c_rel) rs
  end

fun get_mode is_functional relationalising = if relationalising then Database.rel2is
else if is_functional then Database.abs2rel else Database.abs2is

fun add_constant mode abs rel thy =
  let
    val c_abs = read_new_const thy abs
    val c_rel = read_new_const thy rel
    val db_map = Data.map (Database.insert mode (c_abs, c_rel))
    fun add_to_context ctxt' = Context.proof_map db_map ctxt'
    fun add_to_theory ctxt' = Local_Theory.raw_theory (Context.theory_map
    db_map) ctxt'
  in

```

```

    Local_Theory.target (add_to_theory o add_to_context) thy
  end

fun rem_constant rem_op c thy =
  let
    val c = read_new_const thy c
    val db_map = Data.map (rem_op c)
    fun add_to_context ctxt' = Context.proof_map db_map ctxt'
    fun add_to_theory ctxt' = Local_Theory.raw_theory (Context.theory_map
db_map) ctxt'
  in
    Local_Theory.target (add_to_theory o add_to_context) thy
  end

val del_rel_const = Database.remove_abs

fun del_rule thm = del_rel_const (thm |> get_consts |> #2)

val Rel_add =
  Thm.declaration_attribute (fn thm => fn context =>
    Data.map (add_rule (Thm.trim_context thm)) context);

val Rel_del =
  Thm.declaration_attribute (fn thm => fn context =>
    Data.map (del_rule (Thm.trim_context thm)) context);

(* Conjunction of a list of terms *)
fun conjs [] = @{term IFOL.True}
  | conjs (fs as _ :: _) = foldr1 (uncurry conj_) fs

(* Produces a relativized existential quantification of the term t *)
fun rex p t (Free v) = @{const rex} $ p $ lambda (Free v) t
  | rex _ t (Bound _) = t
  | rex _ t tm = raise TERM (rex shouldn't handle this.,[tm,t])

(* Constants that do not take the class predicate *)
val absolute_rels = [ @{const ZF_Base.mem}
  , @{const IFOL.eq(i)}
  , @{const Memrel}
  , @{const True}
  , @{const False}
  ]

(* Creates the relational term corresponding to a term of type i. If the last
argument is (SOME v) then that variable is not bound by an existential
quantifier.
*)
fun close_rel_tm pred tm tm_var rs =
  let val news = filter (not o (fn x => is_Free x orelse is_Bound x) o #1) rs

```

```

    val (vars, tms) = split_list (map #2 news) ||> (curry op @) (the_list tm)
    val vars = case tm_var of
      SOME w => filter (fn v => not (v = w)) vars
    | NONE => vars
  in fold (fn v => fn t => rex pred (incr_boundvars 1 t) v) vars (conjs tms)
end

fun relativ_tms ____ rs ctxt [] = ([], rs, ctxt)
  | relativ_tms is_functional relationalising pred rel_db rs ctxt (u :: us) =
    let val (w_u, rs_u, ctxt_u) = relativ_tm is_functional relationalising NONE
    pred rel_db (rs, ctxt) u
    val (w_us, rs_us, ctxt_us) = relativ_tms is_functional relationalising pred
    rel_db rs_u ctxt_u us
    in (w_u :: w_us, join_tm (rs_u , rs_us), ctxt_us)
    end
and
  (* The result of the relativization of a term is a triple consisting of
  a. the relativized term (it can be a free or a bound variable but also a Collect)
  b. a list of (term * (term, term)), taken as a map, which is used
     to reuse relativization of different occurrences of the same term. The
     first element is the original term, the second its relativized version,
     and the last one is the predicate corresponding to it.
  c. the resulting context of created variables.
  *)
relativ_tm is_functional relationalising mv pred rel_db (rs, ctxt) tm =
  let
    (* relativization of a fully applied constant *)
    fun mk_rel_const mv c (args, after) abs_args ctxt =
      case Database.lookup (get_mode is_functional relationalising) c rel_db of
        SOME p =>
          let
            val args' = List.filter (not o member (op =) (Utils.frees p)) args
            val (v, ctxt1) =
              the_default
                (Variable.variant_fixes [] ctxt ||>> var_i o hd)
                (Utils.map_option (I &&& K ctxt) mv)
            val args' =
              (* FIXME: This special case for functional relativization of sigma
              should not be needed *)
              if c = @{const Sigma} andalso is_functional
              then
                let
                  val t = hd args'
                  val t' = Abs (uu_, @{typ i}, (hd o tl) args' ||> incr_boundvars 1)
                in
                  [t, t']
                end
              else
                args'
          end

```

```

    val arg_list = if after then abs_args @ args' else args' @ abs_args
    val r_tm =
      if is_functional
      then list_comb (p, if p = c then arg_list else pred :: arg_list)
      else list_comb (p, if (not o null) args' andalso hd args' = pred then
arg_list @ [v] else pred :: arg_list @ [v])
    in
      if is_functional
      then (r_tm, r_tm, ctxt)
      else (v, r_tm, ctxt1)
    end
  | NONE => raise TERM (Constant ^ const_name c ^ is not present in
the db. , nil)
  (* relativization of a partially applied constant *)
  fun relativ_app mv mctxt tm abs_args (Const c) (args, after) rs =
    let
      val (w_ts, rs_ts, ctxt_ts) = relativ_tms is_functional relationalising
pred rel_db rs (the_default ctxt mctxt) args
      val (w_tm, r_tm, ctxt_tm) = mk_rel_const mv (Const c) (w_ts, after)
abs_args ctxt_ts
      val rs_ts' = if is_functional then rs_ts else update_tm (tm, (w_tm,
r_tm)) rs_ts
    in
      (w_tm, rs_ts', ctxt_tm)
    end
  | relativ_app _ _ _ _ t _ _ =
    raise TERM (Tried to relativize an application with a non-constant in
head position,[t])

  (* relativization of non dependent product and sum *)
  fun relativ_app_no_dep mv tm c t t' rs =
    if loose_bvar1 (t', 0)
    then
      raise TERM (A dependency was found when trying to relativize, [tm])
    else
      relativ_app mv NONE tm [] c ([t, incr_boundvars ~ 1 t'], false) rs

  fun relativ_replace mv t body after ctxt' =
    let
      val (v, b) = Utils.dest_abs body |>> var_i ||> after
      val (b', (rs', ctxt'')) =
        relativ_fm is_functional relationalising pred rel_db (rs, ctxt', single v,
false) b |>> incr_boundvars 1 ||> #1 &&& #4
    in
      relativ_app mv (SOME ctxt'') tm [lambda v b'] @ {const Replace} ([t], false)
rs'
    end

  fun get_abs_body (Abs body) = body

```

```

| get_abs_body t = raise TERM (Term is not Abs, [t])

fun go _ (Var _) = raise TERM (Var: Is this possible?,[])
| go mv (@{const Replace} $ t $ Abs body) = relativ_replace mv t body I ctxt
(* It is easier to rewrite RepFun as Replace before relativizing,
   since { f(x) . x ∈ t } = { y . x ∈ t, y = f(x) } *)
| go mv (@{const RepFun} $ t $ Abs body) =
  let
    val (y, ctxt') = Variable.variant_fixes [] ctxt |>> var_i o hd
  in
    relativ_replace mv t body (lambda y o Utils.eq_ y o incr_boundvars 1)
  ctxt'
  end
| go mv (@{const Collect} $ t $ pc) =
  let
    val (pc', (rs', ctxt')) = relativ_fm is_functional relationalising pred
rel_db (rs, ctxt, [], false) pc ||> #1 &&& #4
  in
    relativ_app mv (SOME ctxt') tm [pc'] @ {const Collect} ([t], false) rs'
  end
| go mv (@{const Least} $ pc) =
  let
    val (pc', (rs', ctxt')) = relativ_fm is_functional relationalising pred
rel_db (rs, ctxt, [], false) pc ||> #1 &&& #4
  in
    relativ_app mv (SOME ctxt') tm [pc'] @ {const Least} ([], false) rs'
  end
| go mv (@{const transrec} $ t $ Abs body) =
  let
    val (res, ctxt') = Variable.variant_fixes [if is_functional then __aux else
] ctxt |>> var_i o hd
    val (x, b') = Utils.dest_abs body |>> var_i
    val (y, b) = get_abs_body b' |> Utils.dest_abs |>> var_i
    val p = Utils.eq_ res b |> lambda res
    val (p', (rs', ctxt'')) = relativ_fm is_functional relationalising pred
rel_db (rs, ctxt', [x, y], true) p |>> incr_boundvars 3 ||> #1 &&& #4
    val p' = if is_functional then p' |> #2 o Utils.dest_eq_tms o #2 o
Utils.dest_abs o get_abs_body else p'
  in
    relativ_app mv (SOME ctxt'') tm [p' |> lambda x o lambda y] @ {const
transrec} ([t], not is_functional) rs'
  end
| go mv (tm as @ {const Sigma} $ t $ Abs (_, _, t')) =
  relativ_app_no_dep mv tm @ {const Sigma} t t' rs
| go mv (tm as @ {const Pi} $ t $ Abs (_, _, t')) =
  relativ_app_no_dep mv tm @ {const Pi} t t' rs
| go mv (tm as @ {const bool_of_o} $ t) =
  let
    val (t', (rs', ctxt')) = relativ_fm is_functional relationalising pred rel_db

```

```

(rs, ctxt, [], false) t ||> #1 &&& #4
  in
    relativ_app mv (SOME ctxt') tm [t'] @ {const bool_of_o} ([], false) rs'
  end
| go mv (tm as @ {const If} $ b $ t $ t') =
  let
    val (br, (rs', ctxt')) = relativ_fm is_functional relationalising pred
rel_db (rs, ctxt, [], false) b ||> #1 &&& #4
  in
    relativ_app mv (SOME ctxt') tm [br] @ {const If} ([t, t'], true) rs'
  end
| go mv (@ {const The} $ pc) =
  let
    val (pc', (rs', ctxt')) = relativ_fm is_functional relationalising pred
rel_db (rs, ctxt, [], false) pc ||> #1 &&& #4
  in
    relativ_app mv (SOME ctxt') tm [pc'] @ {const The} ([], false) rs'
  end
| go mv (@ {const recursor} $ t $ Abs body $ t') =
  let
    val (res, ctxt') = Variable.variant_fixes [if is_functional then __aux else
] ctxt |>> var_i o hd
    val (x, b') = Utils.dest_abs body |>> var_i
    val (y, b) = get_abs_body b' |> Utils.dest_abs |>> var_i
    val p = Utils.eq_res b |> lambda res
    val (p', (rs', ctxt'')) = relativ_fm is_functional relationalising pred
rel_db (rs, ctxt', [x, y], true) p |>> incr_boundvars 3 ||> #1 &&& #4
    val p' = if is_functional then p' |> #2 o Utils.dest_eq_tms o #2 o
Utils.dest_abs o get_abs_body else p'
    val (tr, rs'', ctxt''') = relativ_tm is_functional relationalising NONE
pred rel_db (rs', ctxt'') t
  in
    relativ_app mv (SOME ctxt''') tm [tr, p' |> lambda x o lambda y]
@ {const recursor} ([t'], true) rs''
  end
| go mv (@ {const wfrec} $ t1 $ t2 $ Abs body) =
  let
    val (res, ctxt') = Variable.variant_fixes [if is_functional then __aux else
] ctxt |>> var_i o hd
    val (x, b') = Utils.dest_abs body |>> var_i
    val (y, b) = get_abs_body b' |> Utils.dest_abs |>> var_i
    val p = Utils.eq_res b |> lambda res
    val (p', (rs', ctxt'')) = relativ_fm is_functional relationalising pred
rel_db (rs, ctxt', [x, y], true) p |>> incr_boundvars 3 ||> #1 &&& #4
    val p' = if is_functional then p' |> #2 o Utils.dest_eq_tms o #2 o
Utils.dest_abs o get_abs_body else p'
  in
    relativ_app mv (SOME ctxt'') tm [p' |> lambda x o lambda y] @ {const
wfrec} ([t1, t2], not is_functional) rs'
  end

```

```

    end
    | go mv (@{const wfrec_on} $ t1 $ t2 $ t3 $ Abs body) =
      let
        val (res, ctxt') = Variable.variant_fixes [if is_functional then _aux else
] ctxt |>> var_i o hd
        val (x, b') = Utils.dest_abs body |>> var_i
        val (y, b) = get_abs_body b' |> Utils.dest_abs |>> var_i
        val p = Utils.eq_res b |> lambda res
          val (p', (rs', ctxt'')) = relativ_fm is_functional relationalising pred
rel_db (rs, ctxt', [x, y], true) p |>> incr_boundvars 3 ||> #1 &&& #4
          val p' = if is_functional then p' |> #2 o Utils.dest_eq_tms o #2 o
Utils.dest_abs o get_abs_body else p'
        in
          relativ_app mv (SOME ctxt'') tm [p' |> lambda x o lambda y] @ {const
wfrec_on} ([t1,t2,t3], not is_functional) rs'
        end
      | go mv (@{const Lambda} $ t $ Abs body) =
        let
          val (res, ctxt') = Variable.variant_fixes [if is_functional then _aux else
] ctxt |>> var_i o hd
          val (x, b) = Utils.dest_abs body |>> var_i
          val p = Utils.eq_res b |> lambda res
            val (p', (rs', ctxt'')) = relativ_fm is_functional relationalising pred
rel_db (rs, ctxt', [x], true) p |>> incr_boundvars 2 ||> #1 &&& #4
            val p' = if is_functional then p' |> #2 o Utils.dest_eq_tms o #2 o
Utils.dest_abs o get_abs_body else p'
            val (tr, rs'', ctxt''') = relativ_tm is_functional relationalising NONE
pred rel_db (rs', ctxt'') t
          in
            relativ_app mv (SOME ctxt''') tm [tr, p' |> lambda x] @ {const Lambda}
([], true) rs''
          end
        (* The following are the generic cases *)
        | go mv (tm as Const _) = relativ_app mv NONE tm [] tm ([], false) rs
        | go mv (tm as _ $ _) = (strip_comb tm ||> I &&& K false |> uncurry
(relativ_app mv NONE tm [])) rs
        | go _ tm = if is_functional then (tm, rs, ctxt) else (tm, update_tm
(tm,(tm,tm)) rs, ctxt)

        (* we first check if the term has been already relativized as a variable *)
        in case lookup_tm rs tm of
          NONE => go mv tm
          | SOME (w, _) => (w, rs, ctxt)
        end
      and
      relativ_fm is_functional relationalising pred rel_db (rs, ctxt, vs, is_term) fm =
      let
        (* relativization of a fully applied constant *)

```



```

fun relativ_app (ctxt, rs) c args = case Database.lookup (get_mode is_functional
relationalising) c rel_db of
  SOME p =>
    let (* flag indicates whether the relativized constant is absolute or not. *)
      val flag = not (exists (curry op aconv c) absolute_rels orelse c = p)
      val (args, rs_ts, ctxt') = relativ_tms is_functional relationalising pred rel_db
    rs ctxt args
    (* TODO: Verify if next line takes care of locales' definitions *)
    val args' = List.filter (not o member (op =) (Utils.frees p)) args
    val args'' = if not (null args') andalso hd args' = pred then args' else pred ::
args'
    val tm = list_comb (p, if flag then args'' else args')
    (* TODO: Verify if next line is necessary *)
    val news = filter (not o (fn x => is_Free x orelse is_Bound x) o #1) rs_ts
    val (vars, tms) = split_list (map #2 news)
    (* val vars = filter (fn v => not (v = tm)) vars *) (* Verify if this line is
necessary *)
    in (tm, (rs_ts, vars, tms, ctxt'))
    end
  | NONE => raise TERM (Constant ^ const_name c ^ is not present in the
db. , nil)

fun close_fm quantifier (f, (rs, vars, tms, ctxt)) =
  let
    fun contains_b0 t = loose_bvar1 (t, 0)

    fun contains_extra_var t = fold (fn v => fn acc => acc orelse fold_atrms
(fn t => fn acc => t = v orelse acc) t false) vs false

    fun contains_b0_extra t = contains_b0 t orelse contains_extra_var t

    (* t1 $ v ↦ t2 iff v ∈ FV(t2) *)
    fun chained_frees (_, $ v) t2 = member (op =) (Utils.frees t2) v
    | chained_frees t _ = raise TERM (Malformed term, [t])

    val tms_to_close = filter contains_b0_extra tms |> Utils.reachable chained_frees
tms
    val tms_to_keep = map (incr_boundvars ~1) (tms --- tms_to_close)
    val vars_to_close = inter (op =) (map (List.last o #2 o strip_comb)
tms_to_close) vars
    val vars_to_keep = vars --- vars_to_close
    val new_rs =
      rs
    |> filter (fn (k, (v, rel)) => not (contains_b0_extra k orelse contains_b0_extra
v orelse contains_b0_extra rel))
    |> map (fn (k, (v, rel)) => (incr_boundvars ~1 k, (incr_boundvars ~1 v,
incr_boundvars ~1 rel)))

    val f' =

```

```

    if not is_term andalso not quantifier andalso is_functional
    then pred $ Bound 0 :: (map (curry (op $) pred) vs) @ [f]
    else [f]
  in
    (fold (fn v => fn t => rex pred (incr_boundvars 1 t) v) vars_to_close (conjs
(f' @ tms_to_close)),
    (new_rs, vars_to_keep, tms_to_keep, ctxt))
  end

(* Handling of bounded quantifiers. *)
fun bquant (ctxt, rs) quant conn dom pred =
  let val (v, pred') = Utils.dest_abs pred |>> var_i
  in
    go (ctxt, rs, false) (quant $ (lambda v o incr_boundvars 1) (conn $ (@{const
mem} $ v $ dom) $ pred'))
  end
and
bind_go (ctxt, rs) const f f' =
  let
    val (r , (rs1, vars1, tms1, ctxt1)) = go (ctxt, rs, false) f
    val (r', (rs2, vars2, tms2, ctxt2)) = go (ctxt1, rs1, false) f'
  in
    (const $ r $ r', (rs2, vars1 @@ vars2, tms1 @@ tms2, ctxt2))
  end
and
relativ_eq_var (ctxt, rs) v t =
  let
    val (_, rs', ctxt') = relativ_tm is_functional relationalising (SOME v)
  in
    pred rel_db (rs, ctxt) t
    val f = lookup_tm rs' t |> #2 o the
    val rs'' = filter (not o (curry (op =) t) o #1) rs'
    val news = filter (not o (fn x => is_Free x orelse is_Bound x) o #1) rs''
    val (vars, tms) = split_list (map #2 news)
  in
    (f, (rs'', vars, tms, ctxt'))
  end
and
relativ_eq (ctxt, rs) t1 t2 =
  if is_functional orelse ((is_Free t1 orelse is_Bound t1) andalso (is_Free t2
orelse is_Bound t2)) then
    relativ_app (ctxt, rs) @{const IFOL.eq(i)} [t1, t2]
  else if is_Free t1 orelse is_Bound t1 then
    relativ_eq_var (ctxt, rs) t1 t2
  else if is_Free t2 orelse is_Bound t2 then
    relativ_eq_var (ctxt, rs) t2 t1
  else
    relativ_app (ctxt, rs) @{const IFOL.eq(i)} [t1, t2]
and
go (ctxt, rs, _ ) (@{const IFOL.conj} $ f $ f') = bind_go (ctxt, rs)

```

```

@{const IFOL.conj} f f'
  | go (ctxt, rs, _) (@{const IFOL.disj} $ f $ f') = bind_go (ctxt, rs)
@{const IFOL.disj} f f'
  | go (ctxt, rs, _) (@{const IFOL.Not} $ f) = go (ctxt, rs, false) f |>>
((curry op $) (@{const IFOL.Not}))
  | go (ctxt, rs, _) (@{const IFOL.iff} $ f $ f') = bind_go (ctxt, rs)
@{const IFOL.iff} f f'
  | go (ctxt, rs, _) (@{const IFOL.imp} $ f $ f') = bind_go (ctxt, rs)
@{const IFOL.imp} f f'
  | go (ctxt, rs, _) (@{const IFOL.All(i)} $ f) = go (ctxt, rs, true) f |>>
((curry op $) (@{const OrdQuant.rall} $ pred))
  | go (ctxt, rs, _) (@{const IFOL.Ex(i)} $ f) = go (ctxt, rs, true) f |>>
((curry op $) (@{const OrdQuant.rex} $ pred))
  | go (ctxt, rs, _) (@{const Bex} $ f $ Abs p) = bquant (ctxt, rs) @{const
Ex(i)} @{const IFOL.conj} f p
  | go (ctxt, rs, _) (@{const Ball} $ f $ Abs p) = bquant (ctxt, rs) @{const
All(i)} @{const IFOL.imp} f p
  | go (ctxt, rs, _) (@{const rall} $ _ $ p) = go (ctxt, rs, true) p |>>
(curry op $) (@{const rall} $ pred)
  | go (ctxt, rs, _) (@{const rex} $ _ $ p) = go (ctxt, rs, true) p |>>
(curry op $) (@{const rex} $ pred)
  | go (ctxt, rs, _) (@{const IFOL.eq(i)} $ t1 $ t2) = relativ_eq (ctxt, rs)
t1 t2
  | go (ctxt, rs, _) (Const c) = relativ_app (ctxt, rs) (Const c) []
  | go (ctxt, rs, _) (tm as _ $ _) = strip_comb tm |> uncurry (relativ_app
(ctxt, rs))
  | go (ctxt, rs, quantifier) (Abs (v, _, t)) =
    let
      val new_rs = map (fn (k, (v, rel)) => (incr_boundvars 1 k, (incr_boundvars
1 v, incr_boundvars 1 rel))) rs
    in
      go (ctxt, new_rs, false) t |> close_fm quantifier |>> lambda (var_i v)
    end
  | go _ t = raise TERM (Relativization of formulas cannot handle this case.,[t])
in
  go (ctxt, rs, false) fm
end

```

```

fun relativ_tm_frm' is_functional relationalising cls_pred db ctxt tm =
  let
    fun get_bounds (l as Abs _) = op @@ (strip_abs l |>> map (op #1) ||>
get_bounds)
      | get_bounds (t as _ $ _) = strip_comb t |> op :: |> map get_bounds |> flat
      | get_bounds _ = []

    val ty = fastype_of tm
    val initial_ctxt = fold Utils.add_to_context (get_bounds tm) ctxt
  in

```

```

case ty of
  @{typ i} =>
    let
      val (w, rs, _) = relativ_tm is_functional relationalising NONE cls_pred
db ([], initial_ctxt) tm
    in
      if is_functional
      then (NONE, w)
      else (SOME w, close_rel_tm cls_pred NONE (SOME w) rs)
    end
  | @{typ o} =>
    let
      fun close_fm (f, (_, vars, tms, _)) =
        fold (fn v => fn t => rex cls_pred (incr_boundvars 1 t) v) vars (conjs
(f :: tms))
    in
      (NONE, relativ_fm is_functional relationalising cls_pred db ([], initial_ctxt,
[], false) tm |> close_fm)
    end
  | ty' => raise TYPE (We can relativize only terms of types i and o, [ty'], [tm])
end

fun lname ctxt = Local_Theory.full_name ctxt o Binding.name

fun destroy_first_lambdas (Abs (body as (_, ty, _))) =
  Utils.dest_abs body ||> destroy_first_lambdas |> (#1 o #2) &&& ((fn v =>
Free (v, ty)) *** #2) ||> op ::
  | destroy_first_lambdas t = (t, [])

fun freeType (Free (_, ty)) = ty
  | freeType t = raise TERM (freeType, [t])

fun relativize_def is_external is_functional relationalising def_name thm_ref pos
lthy =
  let
    val ctxt = lthy
    val (vars, tm, ctxt1) = Utils.thm_concl_tm ctxt (thm_ref ^ _def)
    val db' = Data.get (Context.Proof lthy)
    val (tm, lambdavors) = tm |> destroy_first_lambdas o #2 o Utils.dest_eq_tms'
o Utils.dest_trueprop
    val ctxt1 = fold Utils.add_to_context (map Utils.freeName lambdavors) ctxt1
    val (cls_pred, ctxt1, vars, lambdavors) =
      if (not o null) vars andalso (#2 o #1 o hd) vars = @{typ i => o} then
        ((Thm.term_of o #2 o hd) vars, ctxt1, tl vars, lambdavors)
      else if null vars andalso (not o null) lambdavors andalso (freeType o hd)
lambdavors = @{typ i => o} then
        (hd lambdavors, ctxt1, vars, tl lambdavors)
      else Variable.variant_fixes [N] ctxt1 |>> var_io o hd |> (fn (cls, ctxt) =>
(cls, ctxt, vars, lambdavors))

```

```

    val db' = db' |> Database.insert Database.abs2rel (cls_pred, cls_pred)
      o Database.insert Database.rel2is (cls_pred, cls_pred)
    val (v,t) = relativ_tm_frm' is_functional relationalising cls_pred db' ctxt1 tm
    val t_vars = sort_strings (Term.add_free_names tm [])
    val vs' = List.filter (#1 #> #1 #> #1 #> Ord_List.member String.compare
t_vars) vars
    val vs = cls_pred :: map (Thm.term_of o #2) vs' @ lambdavar @ the_list v
    val at = List.foldr (uncurry lambda) t vs
    val abs_const = read_const lthy (if is_external then thm_ref else lname lthy
thm_ref)
    fun new_const ctxt' = read_new_const ctxt' def_name
    fun db_map ctxt' =
      Data.map (add_rel_const (get_mode is_functional relationalising) abs_const
(new_const ctxt'))
    fun add_to_context ctxt' = Context.proof_map (db_map ctxt') ctxt'
    fun add_to_theory ctxt' = Local_Theory.raw_theory (Context.theory_map
(db_map ctxt')) ctxt'
  in
    lthy
    |> Local_Theory.define ((Binding.name def_name, NoSyn), ((Binding.name
(def_name ^_def), []), at))
    |>> (#2 #> (fn (s,t) => (s,[t])))
    |> Utils.display theorem pos
    |> Local_Theory.target (add_to_theory o add_to_context)
  end

fun relativize_tm is_functional def_name term pos lthy =
  let
    val ctxt = lthy
    val (cls_pred, ctxt1) = Variable.variant_fixes [N] ctxt |>> var_io o hd
    val tm = Syntax.read_term ctxt1 term
    val db' = Data.get (Context.Proof lthy)
    val db' = db' |> Database.insert Database.abs2rel (cls_pred, cls_pred)
      o Database.insert Database.rel2is (cls_pred, cls_pred)
    val vs' = Variable.add_frees ctxt1 tm []
    val ctxt2 = fold Utils.add_to_context (map #1 vs') ctxt1
    val (v,t) = relativ_tm_frm' is_functional false cls_pred db' ctxt2 tm
    val vs = cls_pred :: map Free vs' @ the_list v
    val at = List.foldr (uncurry lambda) t vs
  in
    lthy
    |> Local_Theory.define ((Binding.name def_name, NoSyn), ((Binding.name
(def_name ^_def), []), at))
    |>> (#2 #> (fn (s,t) => (s,[t])))
    |> Utils.display theorem pos
  end

val op $' = curry ((op $) o swap)
infix $'

```

```

fun is_free_i (Free (_, @{typ i})) = true
  | is_free_i _ = false

fun rel_closed_goal target pos lthy =
  let
    val (_, tm, _) = Utils.thm_concl_tm lthy (target ^ _rel_def)
    val (def, tm) = tm |> Utils.dest_eq_tms'
    fun first_lambdas (Abs (body as (_, ty, _))) =
      if ty = @{typ i}
      then (op ::) (Utils.dest_abs body |>> Utils.var_i ||> first_lambdas)
      else Utils.dest_abs body |> first_lambdas o #2
      | first_lambdas _ = []
    val (def, vars) = Term.strip_comb def ||> filter is_free_i
    val vs = vars @ first_lambdas tm
    val class = Free (M, @{typ i ⇒ o})
    val def = fold (op $') (class :: vs) def
    val hyps = map (fn v => class $ v |> Utils.tp) vs
    val concl = class $ def
    val goal = Logic.list_implies (hyps, Utils.tp concl)
    val attribs = @{attributes [intro, simp]}
  in
    Proof.theorem NONE (fn thmss => Utils.display theorem pos
      o Local_Theory.note ((Binding.name (target ^
        _rel_closed), attribs), hd thmss))
    [(goal, [])] lthy
  end

fun iff_goal target pos lthy =
  let
    val (_, tm, ctxt') = Utils.thm_concl_tm lthy (target ^ _rel_def)
    val (_, is_def, ctxt) = Utils.thm_concl_tm ctxt' (is_ ^ target ^ _def)
    val is_def = is_def |> Utils.dest_eq_tms' |> #1 |> Term.strip_comb |> #1
    val (def, tm) = tm |> Utils.dest_eq_tms'
    fun first_lambdas (Abs (body as (_, ty, _))) =
      if ty = @{typ i}
      then (op ::) (Utils.dest_abs body |>> Utils.var_i ||> first_lambdas)
      else Utils.dest_abs body |> first_lambdas o #2
      | first_lambdas _ = []
    val (def, vars) = Term.strip_comb def ||> filter is_free_i
    val vs = vars @ first_lambdas tm
    val class = Free (M, @{typ i ⇒ o})
    val def = fold (op $') (class :: vs) def
    val ty = fastype_of def
    val res = if ty = @{typ i}
      then Variable.variant_fixes [res] ctxt |> SOME o Utils.var_i o hd o
    #1
    else NONE
    val is_def = fold (op $') (class :: vs @ the_list res) is_def
  end

```

```

    val hyps = map (fn v => class $ v |> Utils.tp) (vs @ the_list res)
    val concl = @{const IFOL.iff} $ is_def
                $ (if ty = @{typ i} then (@{const IFOL.eq(i)} $ the res $ def) else def)
    val goal = Logic.list_implies (hyps, Utils.tp concl)
  in
    Proof.theorem NONE (fn thmss => Utils.display theorem pos
                        o Local_Theory.note ((Binding.name (is_ ^ target ^
_iff), []), hd thmss))
    [[(goal, [])] lthy
  end

fun univalent_goal target pos lthy =
  let
    val (_, tm, ctxt) = Utils.thm_concl_tm lthy (is_ ^ target ^ _def)
    val (def, tm) = tm |> Utils.dest_eq_tms'
    fun first_lambdas (Abs (body as (_, ty, _))) =
      if ty = @{typ i}
      then (op ::) (Utils.dest_abs body |>> Utils.var_i ||> first_lambdas)
      else Utils.dest_abs body |> first_lambdas o #2
    | first_lambdas _ = []
    val (def, vars) = Term.strip_comb def ||> filter is_free_i
    val vs = vars @ first_lambdas tm
    val n = length vs
    val vs = List.take (vs, n - 2)
    val class = Free (M, @{typ i => o})
    val def = fold (op $') (class :: vs) def
    val v = Variable.variant_fixes [A] ctxt |> Utils.var_i o hd o #1
    val hyps = map (fn v => class $ v |> Utils.tp) (v :: vs)
    val concl = @{const Relative.univalent} $ class $ v $ def
    val goal = Logic.list_implies (hyps, Utils.tp concl)
  in
    Proof.theorem NONE (fn thmss => Utils.display theorem pos
                        o Local_Theory.note ((Binding.name (univalent_is_
^target), []), hd thmss))
    [[(goal, [])] lthy
  end

end
>

ML
local
  val full_mode_parser =
    Scan.option (((Parse.$$$ functional |-- Parse.$$$ relational) >> K Database.rel2is)
    || (((Scan.option (Parse.$$$ absolute)) |-- Parse.$$$ functional)
    >> K Database.abs2rel)
    || (((Scan.option (Parse.$$$ absolute)) |-- Parse.$$$ relational) >>
    K Database.abs2is))
    >> (fn mode => the_default Database.abs2is mode)

```

```

val reldb_parser =
  Parse.position (full_mode_parser -- (Parse.string -- Parse.string));

val singlemode_parser = (Parse.$$$ absolute >> K Database.remove_abs)
  || (Parse.$$$ functional >> K Database.remove_rel)
  || (Parse.$$$ relational >> K Database.remove_is)

val reldb_rem_parser = Parse.position (singlemode_parser -- Parse.string)

val mode_parser =
  Scan.option ((Parse.$$$ relational >> K false) || (Parse.$$$ functional >>
K true))
  >> (fn mode => if is_none mode then false else the mode)

val relativize_parser =
  Parse.position (mode_parser -- (Parse.string -- Parse.string) -- (Scan.optional
(Parse.$$$ external >> K true) false));

val _ =
  Outer_Syntax.local_theory command_keyword⟨reldb_add⟩ ML setup for
adding relativized/absolute pairs
  (reldb_parser >> (fn ((mode, (abs_term, rel_term)), _) =>
    Relativization.add_constant mode abs_term rel_term))

val _ =
  Outer_Syntax.local_theory command_keyword⟨reldb_rem⟩ ML setup for
adding relativized/absolute pairs
  (reldb_rem_parser >> (uncurry Relativization.rem_constant o #1))

val _ =
  Outer_Syntax.local_theory command_keyword⟨relativize⟩ ML setup for
relativizing definitions
  (relativize_parser >> (fn (((is_functional, (bndg, thm)), is_external), pos)
=>
    Relativization.relativize_def is_external is_functional false thm bndg pos))

val _ =
  Outer_Syntax.local_theory command_keyword⟨relativize_tm⟩ ML setup for
relativizing definitions
  (relativize_parser >> (fn (((is_functional, (bndg, term)), _), pos) =>
    Relativization.relativize_tm is_functional term bndg pos))

val _ =
  Outer_Syntax.local_theory command_keyword⟨relationalize⟩ ML setup for
relativizing definitions
  (relativize_parser >> (fn (((is_functional, (bndg, thm)), is_external), pos)
=>
    Relativization.relativize_def is_external is_functional true thm bndg pos))

```



```

    val _ =
      Outer_Syntax.local_theory_to_proof command_keyword⟨rel_closed⟩ ML
    setup for rel_closed theorem
      (Parse.position (Parse.$$$ for |-- Parse.string) >> (fn (target,pos) =>
        Relativization.rel_closed_goal target pos))

    val _ =
      Outer_Syntax.local_theory_to_proof command_keyword⟨is_iff_rel⟩ ML
    setup for rel_closed theorem
      (Parse.position (Parse.$$$ for |-- Parse.string) >> (fn (target,pos) =>
        Relativization.iff_goal target pos))

    val _ =
      Outer_Syntax.local_theory_to_proof command_keyword⟨univalent⟩ ML
    setup for rel_closed theorem
      (Parse.position (Parse.$$$ for |-- Parse.string) >> (fn (target,pos) =>
        Relativization.univalent_goal target pos))

    val _ =
      Theory.setup
        (Attrib.setup binding⟨Rel⟩ (Attrib.add_del Relativization.Rel_add Relativization.Rel_del)
          declaration of relativization rule) ;
    in
    end
  ›
setup⟨Relativization.init_db Relativization.db ›

declare relative_abs[Rel]

declare datatype_abs[Rel]

ML⟨
  val db = Relativization.get_db @ {context}
  ›

end
theory Discipline_Base
  imports
    ZF-Constructible.Rank
    ZF_Miscellanea
    Relativization

begin

declare [[syntax_ambiguity_warning = false]]

```

10.1 Discipline of relativization of basic concepts

definition

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

lemma (in $M_trivial$) $singleton_abs[simp]$:
 $\llbracket M(x) ; M(s) \rrbracket \Longrightarrow is_singleton(M, x, s) \longleftrightarrow s = \{x\}$
unfolding $is_singleton_def$ **using** $nonempty$ **by** $simp$

synthesize $singleton$ **from_definition** $is_singleton$

lemma (in $M_trivial$) $singleton_closed$ $[simp]$:
 $M(x) \Longrightarrow M(\{x\})$
by $simp$

lemma (in $M_trivial$) $Upair_closed[simp]$: $M(a) \Longrightarrow M(b) \Longrightarrow M(Upair(a, b))$
using $Upair_eq_cons$ **by** $simp$

lemma (in $M_trivial$) $upair_closed[simp]$: $M(x) \Longrightarrow M(y) \Longrightarrow M(\{x, y\})$
by $simp$

The following named theorems gather instances of transitivity that arise from closure theorems

named_theorems $trans_closed$

definition

$is_hcomp :: [i \Rightarrow o, i \Rightarrow i \Rightarrow o, i \Rightarrow i \Rightarrow o, i, i] \Rightarrow o$ **where**
 $is_hcomp(M, is_f, is_g, a, w) \equiv \exists z[M]. is_g(a, z) \wedge is_f(z, w)$

lemma (in $M_trivial$) is_hcomp_abs :
assumes
 $is_f_abs: \bigwedge a z. M(a) \Longrightarrow M(z) \Longrightarrow is_f(a, z) \longleftrightarrow z = f(a)$ **and**
 $is_g_abs: \bigwedge a z. M(a) \Longrightarrow M(z) \Longrightarrow is_g(a, z) \longleftrightarrow z = g(a)$ **and**
 $g_closed: \bigwedge a. M(a) \Longrightarrow M(g(a))$
 $M(a) \ M(w)$
shows
 $is_hcomp(M, is_f, is_g, a, w) \longleftrightarrow w = f(g(a))$
unfolding is_hcomp_def **using** $assms$ **by** $simp$

definition

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

lemma $sats_hcomp_fm$:

assumes
 $f_iff_sats: \bigwedge a b z. a \in nat \Longrightarrow b \in nat \Longrightarrow z \in M \Longrightarrow$
 $is_f(nth(a, Cons(z, env)), nth(b, Cons(z, env))) \longleftrightarrow sats(M, pf(a, b), Cons(z, env))$

and

$$g_iff_sats: \bigwedge a \ b \ z. a \in nat \implies b \in nat \implies z \in M \implies$$

$$is_g(nth(a, Cons(z, env)), nth(b, Cons(z, env))) \longleftrightarrow sats(M, pg(a, b), Cons(z, env))$$
and

$$a \in nat \ w \in nat \ env \in list(M)$$
shows

$$sats(M, hcomp_fm(pf, pg, a, w), env) \longleftrightarrow is_hcomp(\#\#M, is_f, is_g, nth(a, env), nth(w, env))$$
proof -
have $sats(M, pf(0, succ(w)), Cons(x, env)) \longleftrightarrow is_f(x, nth(w, env))$ **if** $x \in M$
 $w \in nat$ **for** $x \ w$
using $f_iff_sats[of \ 0 \ succ(w) \ x]$ **that by simp**
moreover
have $sats(M, pg(succ(a), 0), Cons(x, env)) \longleftrightarrow is_g(nth(a, env), x)$ **if** $x \in M \ a \in nat$
for $x \ a$
using $g_iff_sats[of \ succ(a) \ 0 \ x]$ **that by simp**
ultimately
show $?thesis$ **unfolding** $hcomp_fm_def$ is_hcomp_def **using** $assms$ **by simp**
qed

definition

$hcomp_r :: [i \Rightarrow o, [i \Rightarrow o, i, i] \Rightarrow o, [i \Rightarrow o, i, i] \Rightarrow o, i, i] \Rightarrow o$ **where**
 $hcomp_r(M, is_f, is_g, a, w) \equiv \exists z[M]. is_g(M, a, z) \wedge is_f(M, z, w)$

definition

$is_hcomp2_2 :: [i \Rightarrow o, [i \Rightarrow o, i, i, i] \Rightarrow o, [i \Rightarrow o, i, i, i] \Rightarrow o, [i \Rightarrow o, i, i, i] \Rightarrow o, i, i, i] \Rightarrow o$ **where**
 $is_hcomp2_2(M, is_f, is_g1, is_g2, a, b, w) \equiv \exists g1ab[M]. \exists g2ab[M].$
 $is_g1(M, a, b, g1ab) \wedge is_g2(M, a, b, g2ab) \wedge is_f(M, g1ab, g2ab, w)$

lemma ($in \ M_trivial$) $hcomp_abs$:

assumes

$is_f_abs: \bigwedge a \ z. M(a) \implies M(z) \implies is_f(M, a, z) \longleftrightarrow z = f(a)$ **and**
 $is_g_abs: \bigwedge a \ z. M(a) \implies M(z) \implies is_g(M, a, z) \longleftrightarrow z = g(a)$ **and**
 $g_closed: \bigwedge a. M(a) \implies M(g(a))$
 $M(a) \ M(w)$

shows

$hcomp_r(M, is_f, is_g, a, w) \longleftrightarrow w = f(g(a))$
unfolding $hcomp_r_def$ **using** $assms$ **by simp**

lemma $hcomp_uniqueness$:

assumes

$uniq_is_f:$
 $\bigwedge r \ d \ d'. M(r) \implies M(d) \implies M(d') \implies is_f(M, r, d) \implies is_f(M, r, d') \implies$
 $d = d'$

and

$uniq_is_g:$
 $\bigwedge r \ d \ d'. M(r) \implies M(d) \implies M(d') \implies is_g(M, r, d) \implies is_g(M, r, d') \implies$
 $d = d'$

and

$M(a) \ M(w) \ M(w')$
 $hcomp_r(M, is_f, is_g, a, w)$
 $hcomp_r(M, is_f, is_g, a, w')$
shows
 $w = w'$
proof -
from *assms*
obtain $z \ z'$ **where** $is_g(M, a, z) \ is_g(M, a, z')$
 $is_f(M, z, w) \ is_f(M, z', w')$
 $M(z) \ M(z')$
unfolding *hcomp_r_def* **by** *blast*
moreover from *this* **and** *uniq_is_g* **and** $\langle M(a) \rangle$
have $z = z'$ **by** *blast*
moreover note *uniq_is_f* **and** $\langle M(w) \rangle \ \langle M(w') \rangle$
ultimately
show *?thesis* **by** *blast*
qed

lemma *hcomp_witness*:
assumes
 $wit_is_f: \bigwedge r. M(r) \implies \exists d[M]. is_f(M, r, d)$ **and**
 $wit_is_g: \bigwedge r. M(r) \implies \exists d[M]. is_g(M, r, d)$ **and**
 $M(a)$
shows
 $\exists w[M]. hcomp_r(M, is_f, is_g, a, w)$
proof -
from $\langle M(a) \rangle$ **and** *wit_is_g*
obtain z **where** $is_g(M, a, z) \ M(z)$ **by** *blast*
moreover from *this* **and** *wit_is_f*
obtain w **where** $is_f(M, z, w) \ M(w)$ **by** *blast*
ultimately
show *?thesis*
using *assms* **unfolding** *hcomp_r_def* **by** *auto*
qed

lemma (*in* *M_trivial*) *hcomp2_2_abs*:
assumes
 $is_f_abs: \bigwedge r1 \ r2 \ z. M(r1) \implies M(r2) \implies M(z) \implies is_f(M, r1, r2, z) \longleftrightarrow z = f(r1, r2)$ **and**
 $is_g1_abs: \bigwedge r1 \ r2 \ z. M(r1) \implies M(r2) \implies M(z) \implies is_g1(M, r1, r2, z) \longleftrightarrow z = g1(r1, r2)$ **and**
 $is_g2_abs: \bigwedge r1 \ r2 \ z. M(r1) \implies M(r2) \implies M(z) \implies is_g2(M, r1, r2, z) \longleftrightarrow z = g2(r1, r2)$ **and**
 $types: M(a) \ M(b) \ M(w) \ M(g1(a, b)) \ M(g2(a, b))$
shows
 $is_hcomp2_2(M, is_f, is_g1, is_g2, a, b, w) \longleftrightarrow w = f(g1(a, b), g2(a, b))$
unfolding *is_hcomp2_2_def* **using** *assms*
— We only need some particular cases of the abs assumptions

by *simp*

lemma *hcomp2_2_uniqueness*:

assumes

uniq_is_f:

$\bigwedge r1\ r2\ d\ d'.\ M(r1) \implies M(r2) \implies M(d) \implies M(d') \implies$
 $\text{is_f}(M, r1, r2, d) \implies \text{is_f}(M, r1, r2, d') \implies d = d'$

and

uniq_is_g1:

$\bigwedge r1\ r2\ d\ d'.\ M(r1) \implies M(r2) \implies M(d) \implies M(d') \implies \text{is_g1}(M, r1, r2, d)$
 $\implies \text{is_g1}(M, r1, r2, d') \implies$
 $d = d'$

and

uniq_is_g2:

$\bigwedge r1\ r2\ d\ d'.\ M(r1) \implies M(r2) \implies M(d) \implies M(d') \implies \text{is_g2}(M, r1, r2, d)$
 $\implies \text{is_g2}(M, r1, r2, d') \implies$
 $d = d'$

and

$M(a)\ M(b)\ M(w)\ M(w')$
 $\text{is_hcomp2_2}(M, \text{is_f}, \text{is_g1}, \text{is_g2}, a, b, w)$
 $\text{is_hcomp2_2}(M, \text{is_f}, \text{is_g1}, \text{is_g2}, a, b, w')$

shows

$w = w'$

proof -

from *assms*

obtain $z\ z'\ y\ y'$ **where** $\text{is_g1}(M, a, b, z) \text{ is_g1}(M, a, b, z')$

$\text{is_g2}(M, a, b, y) \text{ is_g2}(M, a, b, y')$

$\text{is_f}(M, z, y, w) \text{ is_f}(M, z', y', w')$

$M(z)\ M(z')\ M(y)\ M(y')$

unfolding *is_hcomp2_2_def* **by** *force*

moreover from this and $\text{uniq_is_g1}\ \text{uniq_is_g2}$ **and** $\langle M(a) \rangle\ \langle M(b) \rangle$

have $z = z'\ y = y'$ **by** *blast+*

moreover note uniq_is_f **and** $\langle M(w) \rangle\ \langle M(w') \rangle$

ultimately

show *?thesis* **by** *blast*

qed

lemma *hcomp2_2_witness*:

assumes

wit_is_f: $\bigwedge r1\ r2.\ M(r1) \implies M(r2) \implies \exists d[M].\ \text{is_f}(M, r1, r2, d)$ **and**

wit_is_g1: $\bigwedge r1\ r2.\ M(r1) \implies M(r2) \implies \exists d[M].\ \text{is_g1}(M, r1, r2, d)$ **and**

wit_is_g2: $\bigwedge r1\ r2.\ M(r1) \implies M(r2) \implies \exists d[M].\ \text{is_g2}(M, r1, r2, d)$ **and**
 $M(a)\ M(b)$

shows

$\exists w[M].\ \text{is_hcomp2_2}(M, \text{is_f}, \text{is_g1}, \text{is_g2}, a, b, w)$

proof -

from $\langle M(a) \rangle\ \langle M(b) \rangle$ **and** *wit_is_g1*

obtain *g1a* **where** $\text{is_g1}(M, a, b, g1a)\ M(g1a)$ **by** *blast*

moreover from $\langle M(a) \rangle\ \langle M(b) \rangle$ **and** *wit_is_g2*

obtain $g2a$ **where** $is_g2(M, a, b, g2a) \ M(g2a)$ **by** *blast*
moreover from *calculation* **and** wit_is_f
obtain w **where** $is_f(M, g1a, g2a, w) \ M(w)$ **by** *blast*
ultimately
show *?thesis*
using *assms* **unfolding** $is_hcomp2_2_def$ **by** *auto*
qed

lemma (in $M_trivial$) *extensionality_trans*:

assumes
 $M(d) \wedge (\forall x[M]. x \in d \longleftrightarrow P(x))$
 $M(d') \wedge (\forall x[M]. x \in d' \longleftrightarrow P(x))$
shows
 $d = d'$
proof -
from *assms*
have $\forall x. x \in d \longleftrightarrow P(x) \wedge M(x)$
using $transM[of_d]$ **by** *auto*
moreover from *assms*
have $\forall x. x \in d' \longleftrightarrow P(x) \wedge M(x)$
using $transM[of_d']$ **by** *auto*
ultimately
show *?thesis* **by** *auto*
qed

definition

$lt_rel :: [i \Rightarrow o, i, i] \Rightarrow o$ **where**
 $lt_rel(M, a, b) \equiv a \in b \wedge ordinal(M, b)$

lemma (in M_trans) $lt_abs[absolut]: M(a) \Longrightarrow M(b) \Longrightarrow lt_rel(M, a, b) \longleftrightarrow a < b$
unfolding lt_rel_def lt_def **by** *auto*

definition

$le_rel :: [i \Rightarrow o, i, i] \Rightarrow o$ **where**
 $le_rel(M, a, b) \equiv \exists sb[M]. successor(M, b, sb) \wedge lt_rel(M, a, sb)$

lemma (in $M_trivial$) $le_abs[absolut]: M(a) \Longrightarrow M(b) \Longrightarrow le_rel(M, a, b) \longleftrightarrow a \leq b$
unfolding le_rel_def **by** (*simp add: absolut*)

10.2 Discipline for Pow

definition

$is_Pow :: [i \Rightarrow o, i, i] \Rightarrow o$ **where**
 $is_Pow(M, A, z) \equiv M(z) \wedge (\forall x[M]. x \in z \longleftrightarrow subset(M, x, A))$

definition

$Pow_rel :: [i \Rightarrow o, i] \Rightarrow i \ (\langle Pow_rel'(_) \rangle)$ **where**
 $Pow_rel(M, r) \equiv THE\ d.\ is_Pow(M, r, d)$

abbreviation

$Pow_r_set :: [i,i] \Rightarrow i \ (\langle Pow_r'(_) \rangle)$ **where**
 $Pow_r_set(M) \equiv Pow_rel(\#\#M)$

context M_basic **begin****lemma** $is_Pow_uniqueness$:**assumes** $M(r)$ $is_Pow(M,r,d) \ is_Pow(M,r,d')$ **shows** $d=d'$ **using** $assms \ extensionality_trans$ **unfolding** is_Pow_def **by** $simp$ **lemma** $is_Pow_witness$: $M(r) \Longrightarrow \exists d[M]. \ is_Pow(M,r,d)$ **using** $power_ax \ unfolding \ power_ax_def \ powerset_def \ is_Pow_def$ **by** $simp$ — We have to do this by hand, using axioms**lemma** is_Pow_closed : $\llbracket M(r); is_Pow(M,r,d) \rrbracket \Longrightarrow M(d)$ **unfolding** $is_Pow_def \ by \ simp$ **lemma** $Pow_rel_closed[intro,simp]$: $M(r) \Longrightarrow M(Pow_rel(M,r))$ **unfolding** Pow_rel_def **using** $is_Pow_closed \ theI[OF \ ex1I[of \ \lambda d. \ is_Pow(M,r,d)], \ OF \ is_Pow_uniqueness[of \ r]]$ $is_Pow_witness$ **by** $fastforce$ **lemmas** $trans_Pow_rel_closed[trans_closed] = transM[OF \ trans_Pow_rel_closed]$

The proof of f_rel_iff lemma is schematic and it can reused by copy-paste replacing appropriately.

lemma Pow_rel_iff :**assumes** $M(r) \ M(d)$ **shows** $is_Pow(M,r,d) \longleftrightarrow d = Pow_rel(M,r)$ **proof** ($intro \ iffI$)**assume** $d = Pow_rel(M,r)$ **with** $assms$ **show** $is_Pow(M, r, d)$ **using** $is_Pow_uniqueness[of \ r] \ is_Pow_witness$ $theI[OF \ ex1I[of \ \lambda d. \ is_Pow(M,r,d)], \ OF \ is_Pow_uniqueness[of \ r]]$ **unfolding** Pow_rel_def **by** $auto$

```

next
  assume is_Pow(M, r, d)
  with assms
  show d = Pow_rel(M, r)
    using is_Pow_uniqueness unfolding Pow_rel_def
    by (auto del:the_equality intro:the_equality[symmetric])
qed

```

The next "def_" result really corresponds to $?A \in \text{Pow}(?B) \longleftrightarrow ?A \subseteq ?B$

```

lemma def_Pow_rel: M(A)  $\implies$  M(r)  $\implies$  A  $\in$  Pow_rel(M, r)  $\longleftrightarrow$  A  $\subseteq$  r
  using Pow_rel_iff[OF Pow_rel_closed, of r r]
  unfolding is_Pow_def by simp

```

```

lemma Pow_rel_char: M(r)  $\implies$  Pow_rel(M, r) = {A  $\in$  Pow(r). M(A)}
proof -
  assume M(r)
  moreover from this
  have x  $\in$  Pow_rel(M, r)  $\implies$  x  $\subseteq$  r M(x)  $\implies$  x  $\subseteq$  r  $\implies$  x  $\in$  Pow_rel(M, r) for x
    using def_Pow_rel by (auto intro!:trans_closed)
  ultimately
  show ?thesis
    using trans_closed by blast
qed

```

```

lemma mem_Pow_rel_abs: M(a)  $\implies$  M(r)  $\implies$  a  $\in$  Pow_rel(M, r)  $\longleftrightarrow$  a  $\in$  Pow(r)
  using Pow_rel_char by simp

```

end — *M_basic*

10.3 Discipline for *PiP*

```

definition
  PiP_rel:: [i $\Rightarrow$ o, i, i] $\Rightarrow$ o where
  PiP_rel(M, A, f)  $\equiv$   $\exists$  df[M]. is_domain(M, f, df)  $\wedge$  subset(M, A, df)  $\wedge$ 
    is_function(M, f)

```

```

context M_basic
begin

```

```

lemma def_PiP_rel:
  assumes
    M(A) M(f)
  shows
    PiP_rel(M, A, f)  $\longleftrightarrow$  A  $\subseteq$  domain(f)  $\wedge$  function(f)
  using assms unfolding PiP_rel_def by simp

```

end — *M_basic*

definition — FIX THIS: not completely relational. Can it be?

$Sigfun :: [i, i \Rightarrow i] \Rightarrow i$ **where**
 $Sigfun(x, B) \equiv \bigcup_{y \in B(x)}. \{ \langle x, y \rangle \}$

lemma $Sigma_Sigfun$: $Sigma(A, B) = \bigcup \{ Sigfun(x, B) \mid x \in A \}$
unfolding $Sigma_def$ $Sigfun_def$..

definition — FIX THIS: not completely relational. Can it be?

$is_Sigfun :: [i \Rightarrow o, i, i \Rightarrow i, i] \Rightarrow o$ **where**
 $is_Sigfun(M, x, B, Sd) \equiv M(Sd) \wedge (\exists RB[M]. is_Replace(M, B(x), \lambda y z. z = \{ \langle x, y \rangle \}, RB) \wedge big_union(M, RB, Sd))$

context $M_trivial$
begin

lemma is_Sigfun_abs :

assumes
 $strong_replacement(M, \lambda y z. z = \{ \langle x, y \rangle \})$
 $M(x) \ M(B(x)) \ M(Sd)$

shows
 $is_Sigfun(M, x, B, Sd) \longleftrightarrow Sd = Sigfun(x, B)$

proof -

have $\bigcup \{ z \mid y \in B(x), z = \{ \langle x, y \rangle \} \} = (\bigcup_{y \in B(x)}. \{ \langle x, y \rangle \})$ **by** *auto*

then

show *?thesis*

using *assms transM[OF_⟨M(B(x))⟩ Replace_abs*

unfolding is_Sigfun_def $Sigfun_def$ **by** *auto*

qed

lemma $Sigfun_closed$:

assumes
 $strong_replacement(M, \lambda y z. y \in B(x) \wedge z = \{ \langle x, y \rangle \})$
 $M(x) \ M(B(x))$

shows

$M(Sigfun(x, B))$

using *assms transM[OF_⟨M(B(x))⟩ RepFun_closed2*

unfolding $Sigfun_def$ **by** *simp*

lemmas $trans_Sigfun_closed[trans_closed] = transM[OF_Sigfun_closed]$

end — $M_trivial$

definition

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

```

is_Sigma(M,A,B,S)  $\equiv$  M(S)  $\wedge$  ( $\exists$  RSf[M].
  is_Replace(M,A, $\lambda x z. z = \text{Sigfun}(x,B),\text{RSf}$ )  $\wedge$  big_union(M,RSf,S))

locale M_Pi = M_basic +
assumes
  Pi_separation: M(A)  $\implies$  separation(M, PiP_rel(M,A))
and
  Pi_replacement:
    M(x)  $\implies$  M(y)  $\implies$ 
      strong_replacement(M,  $\lambda ya z. ya \in y \wedge z = \{\langle x, ya \rangle\}$ )
    M(y)  $\implies$ 
      strong_replacement(M,  $\lambda x z. z = (\bigcup xa \in y. \{\langle x, xa \rangle\})$ )

locale M_Pi_assumptions = M_Pi +
fixes A B
assumes
  Pi_assumptions:
    M(A)
     $\bigwedge x. x \in A \implies M(B(x))$ 
     $\forall x \in A. \text{strong\_replacement}(M, \lambda y z. y \in B(x) \wedge z = \{\langle x, y \rangle\})$ 
    strong_replacement(M,  $\lambda x z. z = \text{Sigfun}(x,B)$ )
begin

lemma Sigma_abs[simp]:
assumes
  M(S)
shows
  is_Sigma(M,A,B,S)  $\longleftrightarrow$  S = Sigma(A,B)
proof -
have  $\bigcup \{z . x \in A, z = \text{Sigfun}(x, B)\} = (\bigcup x \in A. \text{Sigfun}(x, B))$ 
by auto
with assms
show ?thesis
using Replace_abs[of A  $\_ \lambda x z. z = \text{Sigfun}(x,B)$ ]
  Sigfun_closed Sigma_Sigfun[of A B] transM[of  $\_ A$ ]
  Pi_assumptions is_Sigfun_abs
unfolding is_Sigma_def by simp
qed

lemma Sigma_closed[intro,simp]: M(Sigma(A,B))
proof -
have  $(\bigcup x \in A. \text{Sigfun}(x, B)) = \bigcup \{z . x \in A, z = \text{Sigfun}(x, B)\}$ 
by auto
then
show ?thesis
using Sigma_Sigfun[of A B] transM[of  $\_ A$ ]
  Sigfun_closed Pi_assumptions
by simp
qed

```

lemmas *trans_Sigma_closed*[*trans_closed*] = *transM*[*OF __ Sigma_closed*]

end — *M_Pi_assumptions*

10.4 Discipline for Pi

definition

is_Pi :: [$i \Rightarrow o, i, i \Rightarrow i, i] \Rightarrow o$ **where**
is_Pi(M, A, B, I) $\equiv M(I) \wedge (\exists S[M]. \exists PS[M]. is_Sigma(M, A, B, S) \wedge$
is_Pow(M, S, PS) \wedge
is_Collect($M, PS, PiP_rel(M, A, I)$)

definition

Pi_rel :: [$i \Rightarrow o, i, i \Rightarrow i$] $\Rightarrow i$ ($\langle Pi_ '(_, _) \rangle$) **where**
Pi_rel(M, A, B) $\equiv THE\ d.\ is_Pi(M, A, B, d)$

abbreviation

Pi_r_set :: [$i, i, i \Rightarrow i$] $\Rightarrow i$ ($\langle Pi_ '(_, _) \rangle$) **where**
Pi_r_set(M, A, B) $\equiv Pi_rel(\#\#M, A, B)$

context *M_Pi_assumptions*

begin

lemma *is_Pi_uniqueness*:

assumes

is_Pi(M, A, B, d) *is_Pi*(M, A, B, d')

shows

$d = d'$

using *assms Pi_assumptions extensionality_trans*

Pow_rel_iff

unfolding *is_Pi_def* **by** *simp*

lemma *is_Pi_witness*: $\exists d[M]. is_Pi(M, A, B, d)$

using *Pow_rel_iff Pi_separation Pi_assumptions*

unfolding *is_Pi_def* **by** *simp*

lemma *is_Pi_closed* : *is_Pi*(M, A, B, d) $\implies M(d)$

unfolding *is_Pi_def* **by** *simp*

lemma *Pi_rel_closed*[*intro, simp*]: $M(Pi_rel(M, A, B))$

proof -

have *is_Pi*($M, A, B, THE\ xa.\ is_Pi(M, A, B, xa)$)

using *Pi_assumptions*

theI[*OF ex1I*[*of is_Pi*(M, A, B)], *OF __ is_Pi_uniqueness*]

is_Pi_witness is_Pi_closed

by *auto*

```

then show ?thesis
  using is_Pi_closed
  unfolding Pi_rel_def
  by simp
qed

```

— From this point on, the higher order variable y must be explicitly instantiated, and proof methods are slower

```

lemmas trans_Pi_rel_closed[trans_closed] = transM[OF _ Pi_rel_closed]

```

```

lemma Pi_rel_iff:
  assumes M(d)
  shows is_Pi(M,A,B,d)  $\longleftrightarrow$  d = Pi_rel(M,A,B)
proof (intro iffI)
  assume d = Pi_rel(M,A,B)
  moreover
  note assms
  moreover from this
  obtain e where M(e) is_Pi(M,A,B,e)
    using is_Pi_witness by blast
  ultimately
  show is_Pi(M, A, B, d)
    using is_Pi_uniqueness is_Pi_witness is_Pi_closed
    theI[OF exII[of is_Pi(M,A,B)], OF _ is_Pi_uniqueness, of e]
    unfolding Pi_rel_def
    by simp
next
  assume is_Pi(M, A, B, d)
  with assms
  show d = Pi_rel(M,A,B)
    using is_Pi_uniqueness is_Pi_closed unfolding Pi_rel_def
    by (blast del:the_equality intro:the_equality[symmetric])
qed

```

```

lemma def_Pi_rel:
  Pi_rel(M,A,B) = {f $\in$ Pow_rel(M,Sigma(A,B)). A $\subseteq$ domain(f)  $\wedge$  function(f)}
proof -
  have Pi_rel(M,A, B)  $\subseteq$  Pow_rel(M,Sigma(A,B))
    using Pi_assumptions Pi_rel_iff[of Pi_rel(M,A,B)] Pow_rel_iff
    unfolding is_Pi_def by auto
  moreover
  have f  $\in$  Pi_rel(M,A, B)  $\implies$  A $\subseteq$ domain(f)  $\wedge$  function(f) for f
    using Pi_assumptions Pi_rel_iff[of Pi_rel(M,A,B)]
    def_PiP_rel[of A f] trans_closed Pow_rel_iff
    unfolding is_Pi_def by simp
  moreover
  have f  $\in$  Pow_rel(M,Sigma(A,B))  $\implies$  A $\subseteq$ domain(f)  $\wedge$  function(f)  $\implies$  f  $\in$ 
    Pi_rel(M,A, B) for f

```

```

    using  $Pi\_rel\_iff$ [of  $Pi\_rel(M,A,B)$ ]  $Pi\_assumptions$ 
    def  $PiP\_rel$ [of  $A\ f$ ]  $trans\_closed\ Pow\_rel\_iff$ 
    unfolding  $is\_Pi\_def$  by  $simp$ 
    ultimately
    show  $?thesis$  by  $force$ 
qed

```

```

lemma  $Pi\_rel\_char$ :  $Pi\_rel(M,A,B) = \{f \in Pi(A,B). M(f)\}$ 
  using  $Pi\_assumptions\ def\_Pi\_rel\ Pow\_rel\_char[OF\ Sigma\_closed]$  unfolding
 $Pi\_def$ 
  by  $fastforce$ 

```

```

lemma  $mem\_Pi\_rel\_abs$ :
  assumes  $M(f)$ 
  shows  $f \in Pi\_rel(M,A,B) \longleftrightarrow f \in Pi(A,B)$ 
  using  $assms\ Pi\_rel\_char$  by  $simp$ 

```

```

end —  $M\_Pi\_assumptions$ 

```

The next locale (and similar ones below) are used to show the relationship between versions of simple (i.e. $\Sigma_1^{ZF}, \Pi_1^{ZF}$) concepts in two different transitive models.

```

locale  $M\_N\_Pi\_assumptions = M:M\_Pi\_assumptions + N:M\_Pi\_assumptions$ 
N for N +
  assumes
     $M\_imp\_N:M(x) \implies N(x)$ 
begin

```

```

lemma  $Pi\_rel\_transfer$ :  $Pi^M(A,B) \subseteq Pi^N(A,B)$ 
  using  $M.Pi\_rel\_char\ N.Pi\_rel\_char\ M\_imp\_N$  by  $auto$ 

```

```

end —  $M\_N\_Pi\_assumptions$ 

```

```

locale  $M\_Pi\_assumptions\_0 = M\_Pi\_assumptions\_0$ 
begin

```

This is used in the proof of AC_Pi_rel

```

lemma  $Pi\_rel\_empty1[simp]$ :  $Pi^M(0,B) = \{0\}$ 
  using  $Pi\_assumptions\ Pow\_rel\_char$ 
  by  $(unfold\ def\_Pi\_rel\ function\_def)\ (auto)$ 

```

```

end —  $M\_Pi\_assumptions\_0$ 

```

```

context  $M\_Pi\_assumptions$ 
begin

```

10.5 Auxiliary ported results on Pi_rel , now unused

```

lemma  $Pi\_rel\_iff'$ :
  assumes  $types:M(f)$ 
  shows
     $f \in Pi\_rel(M,A,B) \longleftrightarrow function(f) \wedge f \subseteq Sigma(A,B) \wedge A \subseteq domain(f)$ 
  using  $assms\ Pow\_rel\_char$ 
  by ( $simp\ add: def\_Pi\_rel, blast$ )

lemma  $lam\_type\_M$ :
  assumes  $M(A) \wedge x. x \in A \implies M(B(x))$ 
     $\wedge x. x \in A \implies b(x) \in B(x)\ strong\_replacement(M, \lambda x\ y. y = \langle x, b(x) \rangle)$ 
  shows  $(\lambda x \in A. b(x)) \in Pi\_rel(M,A,B)$ 
proof ( $auto\ simp\ add: lam\_def\ def\_Pi\_rel\ function\_def$ )
  from  $assms$ 
  have  $M(\{\langle x, b(x) \rangle . x \in A\})$ 
    using  $Pi\_assumptions\ transM[OF\_ \langle M(A) \rangle]$ 
    by ( $rule\_tac\ RepFun\_closed, auto\ intro!: transM[OF\_ \langle \wedge x. x \in A \implies M(B(x)) \rangle]$ )
  with  $assms$ 
  show  $\{\langle x, b(x) \rangle . x \in A\} \in Pow^M(Sigma(A, B))$ 
    using  $Pow\_rel\_char$  by  $auto$ 
qed

end —  $M\_Pi\_assumptions$ 

locale  $M\_Pi\_assumptions2 = M\_Pi\_assumptions +$ 
   $PiC: M\_Pi\_assumptions \_ \_ C$  for  $C$ 
begin

lemma  $Pi\_rel\_type$ :
  assumes  $f \in Pi^M(A,C) \wedge x. x \in A \implies f'x \in B(x)$ 
    and  $types: M(f)$ 
  shows  $f \in Pi^M(A,B)$ 
  using  $assms\ Pi\_assumptions$ 
  by ( $simp\ only: Pi\_rel\_iff'\ PiC.Pi\_rel\_iff'$ )
    ( $blast\ dest: function\_apply\_equality$ )

lemma  $Pi\_rel\_weaken\_type$ :
  assumes  $f \in Pi^M(A,B) \wedge x. x \in A \implies B(x) \subseteq C(x)$ 
    and  $types: M(f)$ 
  shows  $f \in Pi^M(A,C)$ 
  using  $assms\ Pi\_assumptions$ 
  by ( $simp\ only: Pi\_rel\_iff'\ PiC.Pi\_rel\_iff'$ )
    ( $blast\ intro: Pi\_rel\_type\ dest: apply\_type$ )

end —  $M\_Pi\_assumptions2$ 

end

```

11 Arities of internalized formulas

```

theory Arities
  imports
    Internalizations
    Discipline_Base
begin

lemmas FOL_arities [simp del, arity] = arity_And arity_Or arity_Implies arity_Iff
arity_Exists

declare pred_Un_distrib[arity_aux]

context
  notes FOL_arities[simp]
begin

lemma arity_upair_fm [arity] :  $\llbracket t1 \in \text{nat} ; t2 \in \text{nat} ; up \in \text{nat} \rrbracket \implies$ 
  arity(upair_fm(t1,t2,up)) =  $\bigcup \{succ(t1), succ(t2), succ(up)\}$ 
  unfolding upair_fm_def
  using union_abs1 union_abs2 pred_Un
  by auto

lemma arity_pair_fm [arity] :  $\llbracket t1 \in \text{nat} ; t2 \in \text{nat} ; p \in \text{nat} \rrbracket \implies$ 
  arity(pair_fm(t1,t2,p)) =  $\bigcup \{succ(t1), succ(t2), succ(p)\}$ 
  unfolding pair_fm_def
  using arity_upair_fm union_abs1 union_abs2 pred_Un
  by auto

lemma arity_composition_fm [arity] :
   $\llbracket r \in \text{nat} ; s \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{composition\_fm}(r,s,t)) = \bigcup \{succ(r), succ(s),$ 
  succ(t)\}
  unfolding composition_fm_def
  using arity_pair_fm union_abs1 union_abs2 pred_Un_distrib
  by auto

lemma arity_domain_fm [arity] :
   $\llbracket r \in \text{nat} ; z \in \text{nat} \rrbracket \implies \text{arity}(\text{domain\_fm}(r,z)) = succ(r) \cup succ(z)$ 
  unfolding domain_fm_def
  using arity_pair_fm union_abs1 union_abs2 pred_Un_distrib
  by auto

lemma arity_range_fm [arity] :
   $\llbracket r \in \text{nat} ; z \in \text{nat} \rrbracket \implies \text{arity}(\text{range\_fm}(r,z)) = succ(r) \cup succ(z)$ 
  unfolding range_fm_def
  using arity_pair_fm union_abs1 union_abs2 pred_Un_distrib
  by auto

```

```

lemma arity_union_fm [arity] :
   $\llbracket x \in \text{nat} ; y \in \text{nat} ; z \in \text{nat} \rrbracket \implies \text{arity}(\text{union\_fm}(x,y,z)) = \bigcup \{ \text{succ}(x), \text{succ}(y), \text{succ}(z) \}$ 
  unfolding union_fm_def
  using union_abs1 union_abs2 pred_Un_distrib
  by auto

lemma arity_image_fm [arity] :
   $\llbracket x \in \text{nat} ; y \in \text{nat} ; z \in \text{nat} \rrbracket \implies \text{arity}(\text{image\_fm}(x,y,z)) = \bigcup \{ \text{succ}(x), \text{succ}(y), \text{succ}(z) \}$ 
  unfolding image_fm_def
  using arity_pair_fm union_abs1 union_abs2 pred_Un_distrib
  by auto

lemma arity_pre_image_fm [arity] :
   $\llbracket x \in \text{nat} ; y \in \text{nat} ; z \in \text{nat} \rrbracket \implies \text{arity}(\text{pre\_image\_fm}(x,y,z)) = \bigcup \{ \text{succ}(x), \text{succ}(y), \text{succ}(z) \}$ 
  unfolding pre_image_fm_def
  using arity_pair_fm union_abs1 union_abs2 pred_Un_distrib
  by auto

lemma arity_big_union_fm [arity] :
   $\llbracket x \in \text{nat} ; y \in \text{nat} \rrbracket \implies \text{arity}(\text{big\_union\_fm}(x,y)) = \text{succ}(x) \cup \text{succ}(y)$ 
  unfolding big_union_fm_def
  using union_abs1 union_abs2 pred_Un_distrib
  by auto

lemma arity_fun_apply_fm [arity] :
   $\llbracket x \in \text{nat} ; y \in \text{nat} ; f \in \text{nat} \rrbracket \implies$ 
   $\text{arity}(\text{fun\_apply\_fm}(f,x,y)) = \text{succ}(f) \cup \text{succ}(x) \cup \text{succ}(y)$ 
  unfolding fun_apply_fm_def
  using arity_upair_fm arity_image_fm arity_big_union_fm union_abs2 pred_Un_distrib
  by auto

lemma arity_field_fm [arity] :
   $\llbracket r \in \text{nat} ; z \in \text{nat} \rrbracket \implies \text{arity}(\text{field\_fm}(r,z)) = \text{succ}(r) \cup \text{succ}(z)$ 
  unfolding field_fm_def
  using arity_pair_fm arity_domain_fm arity_range_fm arity_union_fm
   $\text{union\_abs1 union\_abs2 pred\_Un\_distrib}$ 
  by auto

lemma arity_empty_fm [arity]:
   $\llbracket r \in \text{nat} \rrbracket \implies \text{arity}(\text{empty\_fm}(r)) = \text{succ}(r)$ 
  unfolding empty_fm_def
  using union_abs1 union_abs2 pred_Un_distrib
  by simp

lemma arity_cons_fm [arity] :

```



```

lemma arity_cons_fm [arity] :
   $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \implies \text{arity}(\text{cons\_fm}(x, y, z)) = \text{succ}(x) \cup \text{succ}(y) \cup \text{succ}(z)$ 
unfolding cons_fm_def
using arity_upair_fm arity_union_fm union_abs2 pred_Un_distrib
by auto

lemma arity_succ_fm [arity] :
   $\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \implies \text{arity}(\text{succ\_fm}(x, y)) = \text{succ}(x) \cup \text{succ}(y)$ 
unfolding succ_fm_def
using arity_cons_fm
by auto

lemma arity_number1_fm [arity] :
   $\llbracket r \in \text{nat} \rrbracket \implies \text{arity}(\text{number1\_fm}(r)) = \text{succ}(r)$ 
unfolding number1_fm_def
using arity_empty_fm arity_succ_fm union_abs1 union_abs2 pred_Un_distrib
by simp

lemma arity_function_fm [arity] :
   $\llbracket r \in \text{nat} \rrbracket \implies \text{arity}(\text{function\_fm}(r)) = \text{succ}(r)$ 
unfolding function_fm_def
using arity_pair_fm union_abs1 union_abs2 pred_Un_distrib
by simp

lemma arity_relation_fm [arity] :
   $\llbracket r \in \text{nat} \rrbracket \implies \text{arity}(\text{relation\_fm}(r)) = \text{succ}(r)$ 
unfolding relation_fm_def
using arity_pair_fm union_abs1 union_abs2 pred_Un_distrib
by simp

lemma arity_restriction_fm [arity] :
   $\llbracket r \in \text{nat} ; z \in \text{nat} ; A \in \text{nat} \rrbracket \implies \text{arity}(\text{restriction\_fm}(A, z, r)) = \text{succ}(A) \cup \text{succ}(r)$ 
 $\cup \text{succ}(z)$ 
unfolding restriction_fm_def
using arity_pair_fm union_abs2 pred_Un_distrib
by auto

lemma arity_typed_function_fm [arity] :
   $\llbracket x \in \text{nat} ; y \in \text{nat} ; f \in \text{nat} \rrbracket \implies$ 
 $\text{arity}(\text{typed\_function\_fm}(f, x, y)) = \bigcup \{ \text{succ}(f), \text{succ}(x), \text{succ}(y) \}$ 
unfolding typed_function_fm_def
using arity_pair_fm arity_relation_fm arity_function_fm arity_domain_fm
  union_abs2 pred_Un_distrib
by auto

lemma arity_subset_fm [arity] :
   $\llbracket x \in \text{nat} ; y \in \text{nat} \rrbracket \implies \text{arity}(\text{subset\_fm}(x, y)) = \text{succ}(x) \cup \text{succ}(y)$ 
unfolding subset_fm_def
using union_abs2 pred_Un_distrib

```

by *auto*

lemma *arity_transset_fm* [arity] :
 $\llbracket x \in \text{nat} \rrbracket \implies \text{arity}(\text{transset_fm}(x)) = \text{succ}(x)$
unfolding *transset_fm_def*
using *arity_subset_fm union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_ordinal_fm* [arity] :
 $\llbracket x \in \text{nat} \rrbracket \implies \text{arity}(\text{ordinal_fm}(x)) = \text{succ}(x)$
unfolding *ordinal_fm_def*
using *arity_transset_fm union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_limit_ordinal_fm* [arity] :
 $\llbracket x \in \text{nat} \rrbracket \implies \text{arity}(\text{limit_ordinal_fm}(x)) = \text{succ}(x)$
unfolding *limit_ordinal_fm_def*
using *arity_ordinal_fm arity_succ_fm arity_empty_fm union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_finite_ordinal_fm* [arity] :
 $\llbracket x \in \text{nat} \rrbracket \implies \text{arity}(\text{finite_ordinal_fm}(x)) = \text{succ}(x)$
unfolding *finite_ordinal_fm_def*
using *arity_ordinal_fm arity_limit_ordinal_fm arity_succ_fm arity_empty_fm union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_omega_fm* [arity] :
 $\llbracket x \in \text{nat} \rrbracket \implies \text{arity}(\text{omega_fm}(x)) = \text{succ}(x)$
unfolding *omega_fm_def*
using *arity_limit_ordinal_fm union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_cartprod_fm* [arity] :
 $\llbracket A \in \text{nat} ; B \in \text{nat} ; z \in \text{nat} \rrbracket \implies \text{arity}(\text{cartprod_fm}(A, B, z)) = \text{succ}(A) \cup \text{succ}(B) \cup \text{succ}(z)$
unfolding *cartprod_fm_def*
using *arity_pair_fm union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_singleton_fm* [arity] :
 $\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{singleton_fm}(x, t)) = \text{succ}(x) \cup \text{succ}(t)$
unfolding *singleton_fm_def cons_fm_def*
using *arity_union_fm arity_upair_fm arity_empty_fm union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_Memrel_fm* [arity] :
 $\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{Memrel_fm}(x, t)) = \text{succ}(x) \cup \text{succ}(t)$
unfolding *Memrel_fm_def*

```

using arity_pair_fm union_abs2 pred_Un_distrib
by auto

lemma arity_quasinat_fm [arity] :
   $\llbracket x \in \text{nat} \rrbracket \implies \text{arity}(\text{quasinat\_fm}(x)) = \text{succ}(x)$ 
  unfolding quasinat_fm_def cons_fm_def
  using arity_succ_fm arity_empty_fm
    union_abs2 pred_Un_distrib
  by auto

lemma arity_is_recfun_fm [arity] :
   $\llbracket p \in \text{formula} ; v \in \text{nat} ; n \in \text{nat} ; Z \in \text{nat} ; i \in \text{nat} \rrbracket \implies \text{arity}(p) = i \implies$ 
   $\text{arity}(\text{is\_recfun\_fm}(p, v, n, Z)) = \text{succ}(v) \cup \text{succ}(n) \cup \text{succ}(Z) \cup \text{pred}(\text{pred}(\text{pred}(\text{pred}(i))))$ 
  unfolding is_recfun_fm_def
  using arity_upair_fm arity_pair_fm arity_pre_image_fm arity_restriction_fm
    union_abs2 pred_Un_distrib
  by auto

lemma arity_is_wfrec_fm [arity] :
   $\llbracket p \in \text{formula} ; v \in \text{nat} ; n \in \text{nat} ; Z \in \text{nat} ; i \in \text{nat} \rrbracket \implies \text{arity}(p) = i \implies$ 
   $\text{arity}(\text{is\_wfrec\_fm}(p, v, n, Z)) = \text{succ}(v) \cup \text{succ}(n) \cup \text{succ}(Z) \cup \text{pred}(\text{pred}(\text{pred}(\text{pred}(\text{pred}(i)))))$ 
  unfolding is_wfrec_fm_def
  using arity_succ_fm arity_is_recfun_fm
    union_abs2 pred_Un_distrib
  by auto

lemma arity_is_nat_case_fm [arity] :
   $\llbracket p \in \text{formula} ; v \in \text{nat} ; n \in \text{nat} ; Z \in \text{nat} ; i \in \text{nat} \rrbracket \implies \text{arity}(p) = i \implies$ 
   $\text{arity}(\text{is\_nat\_case\_fm}(v, p, n, Z)) = \text{succ}(v) \cup \text{succ}(n) \cup \text{succ}(Z) \cup \text{pred}(\text{pred}(i))$ 
  unfolding is_nat_case_fm_def
  using arity_succ_fm arity_empty_fm arity_quasinat_fm
    union_abs2 pred_Un_distrib
  by auto

lemma arity_iterates_MH_fm [arity] :
  assumes isF  $\in \text{formula}$   $v \in \text{nat}$   $n \in \text{nat}$   $g \in \text{nat}$   $z \in \text{nat}$   $i \in \text{nat}$ 
     $\text{arity}(\text{isF}) = i$ 
  shows  $\text{arity}(\text{iterates\_MH\_fm}(\text{isF}, v, n, g, z)) =$ 
     $\text{succ}(v) \cup \text{succ}(n) \cup \text{succ}(g) \cup \text{succ}(z) \cup \text{pred}(\text{pred}(\text{pred}(\text{pred}(i))))$ 
proof -
  let  $? \varphi = \text{Exists}(\text{And}(\text{fun\_apply\_fm}(\text{succ}(\text{succ}(\text{succ}(g))), 2, 0), \text{Forall}(\text{Implies}(\text{Equal}(0,$ 
     $2), \text{isF}))))$ 
  let  $?ar = \text{succ}(\text{succ}(\text{succ}(g))) \cup \text{pred}(\text{pred}(i))$ 
  from assms
  have  $\text{arity}(\text{?} \varphi) = ?ar$   $? \varphi \in \text{formula}$ 
    using arity_fun_apply_fm
     $\text{union\_abs1}$   $\text{union\_abs2}$   $\text{pred\_Un\_distrib}$   $\text{succ\_Un\_distrib}$   $\text{Un\_assoc}[\text{symmetric}]$ 
    by simp_all
  then

```

```

show ?thesis
  unfolding iterates_MH_fm_def
  using arity_is_nat_case_fm[OF ‹?φ∈_› _ _ _ ‹arity(?φ) = ?ar›] assms
pred_succ_eq pred_Un_distrib
  by auto
qed

```

```

lemma arity_is_iterates_fm [arity] :
  assumes p∈formula v∈nat n∈nat Z∈nat i∈nat
    arity(p) = i
  shows arity(is_iterates_fm(p,v,n,Z)) = succ(v) ∪ succ(n) ∪ succ(Z) ∪
    pred(pred(pred(pred(pred(pred(pred(pred(pred(pred(i))))))))))
proof -
  let ?φ = iterates_MH_fm(p, 7#+v, 2, 1, 0)
  let ?ψ = is_wfrec_fm(?φ, 0, succ(succ(n)),succ(succ(Z)))
  from ‹v∈_›
  have arity(?φ) = (8#+v) ∪ pred(pred(pred(pred(i)))) ?φ∈formula
    using assms arity_iterates_MH_fm union_abs2
    by simp_all
  then
  have arity(?ψ) = succ(succ(succ(n))) ∪ succ(succ(succ(Z))) ∪ (3#+v) ∪
    pred(pred(pred(pred(pred(pred(pred(pred(pred(i))))))))
    using assms arity_is_wfrec_fm[OF ‹?φ∈_› _ _ _ _ ‹arity(?φ) = _›]
union_abs1 pred_Un_distrib
  by auto
  then
  show ?thesis
    unfolding is_iterates_fm_def
    using arity_Memrel_fm arity_succ_fm assms union_abs1 pred_Un_distrib
    by auto
qed

```

```

lemma arity_eclose_n_fm [arity] :
  assumes A∈nat x∈nat t∈nat
  shows arity(eclose_n_fm(A,x,t)) = succ(A) ∪ succ(x) ∪ succ(t)
proof -
  let ?φ = big_union_fm(1,0)
  have arity(?φ) = 2 ?φ∈formula
    using arity_big_union_fm union_abs2
    by simp_all
  with assms
  show ?thesis
    unfolding eclose_n_fm_def
    using arity_is_iterates_fm[OF ‹?φ∈_› _ _ _,of _ _ _ 2]
    by auto
qed

```

```

lemma arity_mem_eclose_fm [arity] :
  assumes x∈nat t∈nat

```

shows $\text{arity}(\text{mem_eclose_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$
proof -
let $? \varphi = \text{eclose_n_fm}(x \# + 2, 1, 0)$
from $\langle x \in \text{nat} \rangle$
have $\text{arity}(\varphi) = x \# + 3$
using $\text{arity_eclose_n_fm union_abs2}$
by *simp*
with *assms*
show $?thesis$
unfolding mem_eclose_fm_def
using $\text{arity_finite_ordinal_fm union_abs2 pred_Un_distrib}$
by *simp*
qed

lemma $\text{arity_is_eclose_fm} [\text{arity}] :$
 $\llbracket x \in \text{nat} ; t \in \text{nat} \rrbracket \implies \text{arity}(\text{is_eclose_fm}(x,t)) = \text{succ}(x) \cup \text{succ}(t)$
unfolding is_eclose_fm_def
using $\text{arity_mem_eclose_fm union_abs2 pred_Un_distrib}$
by *auto*

lemma $\text{arity_Collect_fm} [\text{arity}] :$
assumes $x \in \text{nat } y \in \text{nat } p \in \text{formula}$
shows $\text{arity}(\text{Collect_fm}(x,p,y)) = \text{succ}(x) \cup \text{succ}(y) \cup \text{pred}(\text{arity}(p))$
unfolding Collect_fm_def
using $\text{assms pred_Un_distrib}$
by *auto*

schematic_goal $\text{arity_least_fm}' :$
assumes
 $i \in \text{nat } q \in \text{formula}$
shows
 $\text{arity}(\text{least_fm}(q,i)) \equiv ?ar$
unfolding least_fm_def
using $\text{assms pred_Un_distrib arity_And arity_Or arity_Neg arity_Implies}$
 arity_ordinal_fm
 $\text{arity_empty_fm Un_assoc[symmetric] Un_commute}$
by *auto*

lemma $\text{arity_least_fm} [\text{arity}] :$
assumes
 $i \in \text{nat } q \in \text{formula}$
shows
 $\text{arity}(\text{least_fm}(q,i)) = \text{succ}(i) \cup \text{pred}(\text{arity}(q))$
using $\text{assms arity_least_fm}'$
by *auto*

lemma $\text{arity_Replace_fm} [\text{arity}] :$
 $\llbracket p \in \text{formula} ; v \in \text{nat} ; n \in \text{nat} ; i \in \text{nat} \rrbracket \implies \text{arity}(p) = i \implies$
 $\text{arity}(\text{Replace_fm}(v,p,n)) = \text{succ}(n) \cup (\text{succ}(v) \cup \text{Arith.pred}(\text{Arith.pred}(i)))$

```

unfolding Replace_fm_def
using union_abs2 pred_Un_distrib
by simp

lemma arity_lambda_fm [arity] :
   $\llbracket p \in \text{formula}; v \in \text{nat}; n \in \text{nat}; i \in \text{nat} \rrbracket \implies \text{arity}(p) = i \implies$ 
 $\text{arity}(\text{lambda\_fm}(p, v, n)) = \text{succ}(n) \cup (\text{succ}(v) \cup \text{Arith.pred}(\text{Arith.pred}(\text{Arith.pred}(i))))$ 
unfolding lambda_fm_def
using arity_pair_fm pred_Un_distrib union_abs1 union_abs2
by simp

lemma arity_transrec_fm [arity] :
   $\llbracket p \in \text{formula}; v \in \text{nat}; n \in \text{nat}; i \in \text{nat} \rrbracket \implies \text{arity}(p) = i \implies$ 
 $\text{arity}(\text{is\_transrec\_fm}(p, v, n)) = \text{succ}(v) \cup \text{succ}(n) \cup (\text{pred}^8(i))$ 
unfolding is_transrec_fm_def
using arity_Un_assoc[symmetric] pred_Un_distrib
by simp

end — FOL_arities

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

end
theory Discipline_Function
  imports
    Arities
begin

Discipline for fst arity_theorem for empty_fm
arity_theorem for upair_fm
arity_theorem for pair_fm
definition
  is_fst ::  $(i \Rightarrow o) \Rightarrow i \Rightarrow i \Rightarrow o$  where
  is_fst( $M, x, t$ )  $\equiv (\exists z[M]. \text{pair}(M, t, z, x)) \vee$ 
 $(\neg(\exists z[M]. \exists w[M]. \text{pair}(M, w, z, x)) \wedge \text{empty}(M, t))$ 
synthesize fst from_definition is_fst
notation fst_fm ( $\langle \cdot \text{fst}'(\cdot) \text{ is } \cdot \rangle$ )

arity_theorem for fst_fm

definition fst_rel ::  $[i \Rightarrow o, i] \Rightarrow i$  where
  fst_rel( $M, p$ )  $\equiv \text{THE } d. M(d) \wedge \text{is\_fst}(M, p, d)$ 

reldb_add relational fst is_fst
reldb_add functional fst fst_rel

definition
  is_snd ::  $(i \Rightarrow o) \Rightarrow i \Rightarrow i \Rightarrow o$  where

```

```

is_snd(M,x,t) ≡ (∃ z[M]. pair(M,z,t,x)) ∨
                 (¬(∃ z[M]. ∃ w[M]. pair(M,z,w,x)) ∧ empty(M,t))
synthesize snd from_definition is_snd
notation snd_fm (⟦snd'(_') is _⟧)
arity_theorem for snd_fm

definition snd_rel :: [i⇒o,i] ⇒ i where
  snd_rel(M,p) ≡ THE d. M(d) ∧ is_snd(M,p,d)

reldb_add relational snd is_snd
reldb_add functional snd snd_rel

context M_trans
begin

lemma fst_snd_closed:
  assumes M(p)
  shows M(fst(p)) ∧ M(snd(p))
  unfolding fst_def snd_def using assms
  by (cases ∃ a. ∃ b. p = ⟨a, b⟩; auto)

lemma fst_closed[intro,simp]: M(x) ⇒ M(fst(x))
  using fst_snd_closed by auto

lemma snd_closed[intro,simp]: M(x) ⇒ M(snd(x))
  using fst_snd_closed by auto

lemma fst_abs [absolut]:
  ⟦M(p); M(x)⟧ ⇒ is_fst(M,p,x) ⟷ x = fst(p)
  unfolding is_fst_def fst_def
  by (cases ∃ a. ∃ b. p = ⟨a, b⟩; auto)

lemma snd_abs [absolut]:
  ⟦M(p); M(y)⟧ ⇒ is_snd(M,p,y) ⟷ y = snd(p)
  unfolding is_snd_def snd_def
  by (cases ∃ a. ∃ b. p = ⟨a, b⟩; auto)

lemma empty_rel_abs : M(x) ⇒ M(0) ⇒ x = 0 ⟷ x = (THE d. M(d) ∧
empty(M, d))
  unfolding the_def
  using transM
  by auto

lemma fst_rel_abs:
  assumes M(p)
  shows fst(p) = fst_rel(M,p)
  using fst_abs assms
  unfolding fst_def fst_rel_def

```

```

    by (cases  $\exists a. \exists b. p = \langle a, b \rangle$ ; auto; rule_tac the_equality[symmetric], simp_all)

lemma snd_rel_abs:
  assumes  $M(p)$ 
  shows  $snd(p) = snd\_rel(M, p)$ 
  using snd_abs assms
  unfolding snd_def snd_rel_def
  by (cases  $\exists a. \exists b. p = \langle a, b \rangle$ ; auto; rule_tac the_equality[symmetric], simp_all)

end —  $M\_trans$ 

relativize functional first first_rel external
relativize functional minimum minimum_rel external
context  $M\_trans$ 
begin

lemma minimum_closed[simp, intro]:
  assumes  $M(A)$ 
  shows  $M(minimum(r, A))$ 
  using first_is_elem the_equality_if transM[OF  $\langle M(A) \rangle$ ]
  by (cases  $\exists x. first(x, A, r)$ ; auto simp: minimum_def)

lemma first_abs :
  assumes  $M(B)$ 
  shows  $first(z, B, r) \longleftrightarrow first\_rel(M, z, B, r)$ 
  unfolding first_def first_rel_def using assms by auto

lemma minimum_abs:
  assumes  $M(B)$ 
  shows  $minimum(r, B) = minimum\_rel(M, r, B)$ 
proof -
  from assms
  have  $first(b, B, r) \longleftrightarrow M(b) \wedge first\_rel(M, b, B, r)$  for  $b$ 
    using first_abs
  proof (auto)
    fix  $b$ 
    assume  $first\_rel(M, b, B, r)$ 
    with  $\langle M(B) \rangle$ 
    have  $b \in B$  using first_abs first_is_elem by simp
    with  $\langle M(B) \rangle$ 
    show  $M(b)$  using transM[OF  $\langle b \in B \rangle$ ] by simp
  qed
  with assms
  show ?thesis unfolding minimum_rel_def minimum_def
    by simp
qed

end —  $M\_trans$ 

```


11.1 Discipline for $\lambda A \ B. A \rightarrow B$

definition

$is_function_space :: [i \Rightarrow o, i, i, i] \Rightarrow o$ **where**
 $is_function_space(M, A, B, fs) \equiv M(fs) \wedge is_funspace(M, A, B, fs)$

definition

$function_space_rel :: [i \Rightarrow o, i, i] \Rightarrow i$ **where**
 $function_space_rel(M, A, B) \equiv THE\ d.\ is_function_space(M, A, B, d)$

reldb_rem absolute Pi

reldb_add relational $Pi\ is_function_space$

reldb_add functional $Pi\ function_space_rel$

abbreviation

$function_space_r :: [i, i \Rightarrow o, i] \Rightarrow i$ ($\langle _ \rightarrow _ \rangle [61, 1, 61]\ 60$) **where**
 $A \rightarrow^M B \equiv function_space_rel(M, A, B)$

abbreviation

$function_space_r_set :: [i, i, i] \Rightarrow i$ ($\langle _ \rightarrow _ \rangle [61, 1, 61]\ 60$) **where**
 $function_space_r_set(A, M) \equiv function_space_rel(\#\#M, A)$

context M_Pi

begin

lemma $is_function_space_uniqueness:$

assumes

$M(r)\ M(B)$

$is_function_space(M, r, B, d)\ is_function_space(M, r, B, d')$

shows

$d = d'$

using $assms\ extensionality_trans$

unfolding $is_function_space_def\ is_funspace_def$

by $simp$

lemma $is_function_space_witness:$

assumes $M(A)\ M(B)$

shows $\exists d[M].\ is_function_space(M, A, B, d)$

proof -

from $assms$

interpret $M_Pi_assumptions\ M\ A\ \lambda_.\ B$

using $Pi_replacement\ Pi_separation$

by $unfold_locales\ (auto\ dest:transM\ simp\ add:Sigfun_def)$

have $\forall f[M].\ f \in Pi_rel(M, A, \lambda_.\ B) \longleftrightarrow f \in A \rightarrow B$

using Pi_rel_char **by** $simp$

with $assms$

show $?thesis$ **unfolding** $is_funspace_def\ is_function_space_def$ **by** $auto$

qed

lemma $is_function_space_closed :$

is_function_space(*M*,*A*,*B*,*d*) \implies *M*(*d*)
unfolding *is_function_space_def* **by** *simp*

— adding closure to simpset and claset

lemma *function_space_rel_closed*[*intro, simp*]:

assumes *M*(*x*) *M*(*y*)

shows *M*(*function_space_rel*(*M*,*x*,*y*))

proof -

have *is_function_space*(*M*, *x*, *y*, *THE xa. is_function_space*(*M*, *x*, *y*, *xa*))

using *assms*

theI[*OF ex1I*[*of is_function_space*(*M*,*x*,*y*)], *OF is_function_space_uniqueness*[*of*
x y]]

is_function_space_witness

by *auto*

then show *?thesis*

using *assms is_function_space_closed*

unfolding *function_space_rel_def*

by *blast*

qed

lemmas *trans_function_space_rel_closed*[*trans_closed*] = *transM*[*OF function_space_rel_closed*]

lemma *function_space_rel_iff*:

assumes *M*(*x*) *M*(*y*) *M*(*d*)

shows *is_function_space*(*M*,*x*,*y*,*d*) \longleftrightarrow *d* = *function_space_rel*(*M*,*x*,*y*)

proof (*intro iffI*)

assume *d* = *function_space_rel*(*M*,*x*,*y*)

moreover

note *assms*

moreover from *this*

obtain *e* **where** *M*(*e*) *is_function_space*(*M*,*x*,*y*,*e*)

using *is_function_space_witness* **by** *blast*

ultimately

show *is_function_space*(*M*, *x*, *y*, *d*)

using *is_function_space_uniqueness*[*of x y*] *is_function_space_witness*

theI[*OF ex1I*[*of is_function_space*(*M*,*x*,*y*)], *OF is_function_space_uniqueness*[*of*
x y], *of e*]

unfolding *function_space_rel_def*

by *auto*

next

assume *is_function_space*(*M*, *x*, *y*, *d*)

with *assms*

show *d* = *function_space_rel*(*M*,*x*,*y*)

using *is_function_space_uniqueness* **unfolding** *function_space_rel_def*

by (*blast del:the_equality intro:the_equality[symmetric]*)

qed

lemma *def_function_space_rel*:

```

    assumes  $M(A) \ M(y)$ 
    shows  $function\_space\_rel(M, A, y) = Pi\_rel(M, A, \lambda\_ . y)$ 
  proof -
    from assms
    interpret  $M\_Pi\_assumptions \ M \ A \ \lambda\_ . \ y$ 
    using Pi\_replacement Pi\_separation
    by unfold_locales (auto dest:transM simp add:Sigfun_def)
    from assms
    have  $x \in function\_space\_rel(M, A, y) \longleftrightarrow x \in Pi\_rel(M, A, \lambda\_ . y)$  if  $M(x)$  for  $x$ 
    using that
    function\_space\_rel iff[of A y, OF _ function\_space\_rel\_closed, of A y]
    def_Pi_rel Pi_rel_char Pow_rel_char
    unfolding is\_function\_space\_def is\_funspace\_def by (simp add:Pi_def)
    with assms
    show ?thesis — At this point, quoting "trans_rules" doesn't work
    using transM[OF function\_space\_rel\_closed, OF _  $\langle M(A) \rangle \langle M(y) \rangle$ ]
    transM[OF Pi\_rel\_closed] by blast
  qed

```

```

lemma function\_space\_rel\_char:
  assumes  $M(A) \ M(y)$ 
  shows  $function\_space\_rel(M, A, y) = \{f \in A \rightarrow y. \ M(f)\}$ 
  proof -
    from assms
    interpret  $M\_Pi\_assumptions \ M \ A \ \lambda\_ . \ y$ 
    using Pi\_replacement Pi\_separation
    by unfold_locales (auto dest:transM simp add:Sigfun_def)
    show ?thesis
    using assms def\_function\_space\_rel Pi\_rel\_char
    by simp
  qed

```

```

lemma mem\_function\_space\_rel\_abs:
  assumes  $M(A) \ M(y) \ M(f)$ 
  shows  $f \in function\_space\_rel(M, A, y) \longleftrightarrow f \in A \rightarrow y$ 
  using assms function\_space\_rel\_char by simp

end —  $M\_Pi$ 

```

```

locale  $M\_N\_Pi = M:M\_Pi + N:M\_Pi \ N$  for  $N +$ 
  assumes
     $M\_imp\_N:M(x) \implies N(x)$ 
  begin

```

```

lemma function\_space\_rel\_transfer:  $M(A) \implies M(B) \implies$ 

```

```

      function_space_rel(M,A,B) ⊆ function_space_rel(N,A,B)
    using M.function_space_rel_char N.function_space_rel_char
    by (auto dest!:M_imp_N)

end — M_N_Pi

```

abbreviation

is_apply \equiv *fun_apply*
 — It is not necessary to perform the Discipline for *is_apply* since it is absolute in this context

11.2 Discipline for *Collect* terms.

We have to isolate the predicate involved and apply the Discipline to it.

definition

```

injP_rel:: [i⇒o,i,i]⇒o where
injP_rel(M,A,f) ≡ ∀ w[M]. ∀ x[M]. ∀ fw[M]. ∀ fx[M]. w∈A ∧ x∈A ∧
    is_apply(M,f,w,fw) ∧ is_apply(M,f,x,fx) ∧ fw=fx ⟶ w=x

```

synthesize *injP_rel* **from_definition** **assuming** *nonempty*

arity_theorem **for** *injP_rel_fm*

context *M_basic*

begin

— I'm undecided on keeping the relative quantifiers here. Same with *surjP* below.
 It might relieve from changing $?P(?x) \implies \exists x. ?P(x)$
 $(\bigwedge x. ?P(x)) \implies \forall x. ?P(x)$ to $\llbracket ?P(?x); ?M(?x) \rrbracket \implies \exists x[?M]. ?P(x)$
 $(\bigwedge x. ?M(x) \implies ?P(x)) \implies \forall x[?M]. ?P(x)$ in some proofs. I wonder if this escalates well. Assuming that all terms appearing in the "def_" theorem are in *M* and using $\llbracket ?y \in ?x; M(?x) \rrbracket \implies M(?y)$, it might do.

lemma *def_injP_rel*:

assumes

M(A) M(f)

shows

$\text{injP_rel}(M,A,f) \longleftrightarrow (\forall w[M]. \forall x[M]. w \in A \wedge x \in A \wedge f'w=f'x \longrightarrow w=x)$

using *assms* **unfolding** *injP_rel_def* **by** *simp*

end — *M_basic*

11.3 Discipline for *inj*

definition

```

is_inj :: [i⇒o,i,i,i]⇒o where
is_inj(M,A,B,I) ≡ M(I) ∧ (∃ F[M]. is_function_space(M,A,B,F) ∧
    is_Collect(M,F,injP_rel(M,A,I))

```

declare *typed_function_iff_sats* *Collect_iff_sats* [*iff_sats*]

synthesize *is_funspace* **from_definition** **assuming** *nonempty*
arity_theorem **for** *is_funspace_fm*

synthesize *is_function_space* **from_definition** **assuming** *nonempty*
notation *is_function_space_fm* ($\langle _ \rightarrow _ \text{ is } _ \rangle$)

arity_theorem **for** *is_function_space_fm*

synthesize *is_inj* **from_definition** **assuming** *nonempty*
notation *is_inj_fm* ($\langle _ \text{ inj }'(_, _) \text{ is } _ \rangle$)

arity_theorem **intermediate** **for** *is_inj_fm*

lemma *arity_is_inj_fm*[*arity*]:

$A \in \text{nat} \implies$

$B \in \text{nat} \implies I \in \text{nat} \implies \text{arity}(\text{is_inj_fm}(A, B, I)) = \text{succ}(A) \cup \text{succ}(B) \cup \text{succ}(I)$

using *arity_is_inj_fm'* **by** (*auto simp: pred_Un_distrib arity*)

definition

inj_rel :: $[i \Rightarrow o, i, i] \Rightarrow i \ (\langle \text{inj_}'(_, _) \rangle)$ **where**

$\text{inj_rel}(M, A, B) \equiv \text{THE } d. \text{is_inj}(M, A, B, d)$

abbreviation

inj_r_set :: $[i, i, i] \Rightarrow i \ (\langle \text{inj_}'(_, _) \rangle)$ **where**

$\text{inj_r_set}(M) \equiv \text{inj_rel}(\#\#M)$

locale *M_inj* = *M_Pi* +

assumes

$\text{injP_separation}: M(r) \implies \text{separation}(M, \text{injP_rel}(M, r))$

begin

lemma *is_inj_uniqueness*:

assumes

$M(r) \ M(B)$

$\text{is_inj}(M, r, B, d) \ \text{is_inj}(M, r, B, d')$

shows

$d = d'$

using *assms function_space_rel_iff extensionality_trans*

unfolding *is_inj_def* **by** *simp*

lemma *is_inj_witness*: $M(r) \implies M(B) \implies \exists d[M]. \text{is_inj}(M, r, B, d)$

using *injP_separation function_space_rel_iff*

unfolding *is_inj_def* **by** *simp*

```

lemma is_inj_closed :
  is_inj(M,x,y,d)  $\implies$  M(d)
  unfolding is_inj_def by simp

lemma inj_rel_closed[intro,simp]:
  assumes M(x) M(y)
  shows M(inj_rel(M,x,y))
proof -
  have is_inj(M, x, y, THE xa. is_inj(M, x, y, xa))
    using assms
      theI[OF ex1I[of is_inj(M,x,y)], OF _ is_inj_uniqueness[of x y]]
      is_inj_witness
    by auto
  then show ?thesis
    using assms is_inj_closed
    unfolding inj_rel_def
    by blast
qed

lemmas trans_inj_rel_closed[trans_closed] = transM[OF _ inj_rel_closed]

lemma inj_rel_iff:
  assumes M(x) M(y) M(d)
  shows is_inj(M,x,y,d)  $\longleftrightarrow$  d = inj_rel(M,x,y)
proof (intro iffI)
  assume d = inj_rel(M,x,y)
  moreover
  note assms
  moreover from this
  obtain e where M(e) is_inj(M,x,y,e)
    using is_inj_witness by blast
  ultimately
  show is_inj(M, x, y, d)
    using is_inj_uniqueness[of x y] is_inj_witness
      theI[OF ex1I[of is_inj(M,x,y)], OF _ is_inj_uniqueness[of x y], of e]
    unfolding inj_rel_def
    by auto
next
  assume is_inj(M, x, y, d)
  with assms
  show d = inj_rel(M,x,y)
    using is_inj_uniqueness unfolding inj_rel_def
    by (blast del:the_equality intro:the_equality[symmetric])
qed

lemma def_inj_rel:
  assumes M(A) M(B)
  shows inj_rel(M,A,B) =
    {f  $\in$  function_space_rel(M,A,B).  $\forall w[M]. \forall x[M]. w \in A \wedge x \in A \wedge f'w =$ 

```

```

 $f \cdot x \longrightarrow w = x$ 
  (is _ = Collect(_, ?P))
proof -
  from assms
  have  $\text{inj\_rel}(M, A, B) \subseteq \text{function\_space\_rel}(M, A, B)$ 
    using  $\text{inj\_rel\_iff}[of\ A\ B\ \text{inj\_rel}(M, A, B)]\ \text{function\_space\_rel\_iff}$ 
    unfolding  $\text{is\_inj\_def}$  by auto
  moreover from assms
  have  $f \in \text{inj\_rel}(M, A, B) \implies ?P(f)$  for  $f$ 
    using  $\text{inj\_rel\_iff}[of\ A\ B\ \text{inj\_rel}(M, A, B)]\ \text{function\_space\_rel\_iff}$ 
    def  $\text{injP\_rel}\ \text{transM}[OF\ \text{function\_space\_rel\_closed},\ OF\ \langle M(A) \rangle\ \langle M(B) \rangle]$ 
    unfolding  $\text{is\_inj\_def}$  by auto
  moreover from assms
  have  $f \in \text{function\_space\_rel}(M, A, B) \implies ?P(f) \implies f \in \text{inj\_rel}(M, A, B)$  for  $f$ 
    using  $\text{inj\_rel\_iff}[of\ A\ B\ \text{inj\_rel}(M, A, B)]\ \text{function\_space\_rel\_iff}$ 
    def  $\text{injP\_rel}\ \text{transM}[OF\ \text{function\_space\_rel\_closed},\ OF\ \langle M(A) \rangle\ \langle M(B) \rangle]$ 
    unfolding  $\text{is\_inj\_def}$  by auto
  ultimately
  show ?thesis by force
qed

lemma  $\text{inj\_rel\_char}$ :
  assumes  $M(A)\ M(B)$ 
  shows  $\text{inj\_rel}(M, A, B) = \{f \in \text{inj}(A, B).\ M(f)\}$ 
proof -
  from assms
  interpret  $M\_Pi\_assumptions\ M\ A\ \lambda\_.\ B$ 
    using  $Pi\_replacement\ Pi\_separation$ 
    by unfold_locales (auto dest:  $\text{transM}\ \text{simp}\ \text{add:}\ \text{Sigfun\_def}$ )
  from assms
  show ?thesis
    using  $\text{def\_inj\_rel}[OF\ \text{assms}]\ \text{def\_function\_space\_rel}[OF\ \text{assms}]$ 
     $\text{transM}[OF\ \langle M(A) \rangle]\ \text{Pi\_rel\_char}$ 
    unfolding  $\text{inj\_def}$ 
    by auto
qed

end —  $M\_inj$ 

locale  $M\_N\_inj = M:M\_inj + N:M\_inj\ N$  for  $N +$ 
  assumes
     $M\_imp\_N:M(x) \implies N(x)$ 
begin

lemma  $\text{inj\_rel\_transfer}$ :  $M(A) \implies M(B) \implies \text{inj\_rel}(M, A, B) \subseteq \text{inj\_rel}(N, A, B)$ 
  using  $M.\text{inj\_rel\_char}\ N.\text{inj\_rel\_char}$ 
  by (auto dest!:  $M\_imp\_N$ )

```

end — M_N_inj

definition

$surjP_rel :: [i \Rightarrow o, i, i, i] \Rightarrow o$ **where**
 $surjP_rel(M, A, B, f) \equiv$
 $\forall y[M]. \exists x[M]. \exists fx[M]. y \in B \longrightarrow x \in A \wedge is_apply(M, f, x, fx) \wedge fx = y$

synthesize $surjP_rel$ **from_definition** **assuming** $nonempty$

context M_basic

begin

lemma def_surjP_rel :

assumes

$M(A) \ M(B) \ M(f)$

shows

$surjP_rel(M, A, B, f) \longleftrightarrow (\forall y[M]. \exists x[M]. y \in B \longrightarrow x \in A \wedge f'x = y)$

using $assms$ **unfolding** $surjP_rel_def$ **by** $auto$

end — M_basic

11.4 Discipline for $surj$

definition

$is_surj :: [i \Rightarrow o, i, i, i] \Rightarrow o$ **where**
 $is_surj(M, A, B, I) \equiv M(I) \wedge (\exists F[M]. is_function_space(M, A, B, F) \wedge$
 $is_Collect(M, F, surjP_rel(M, A, B), I))$

synthesize is_surj **from_definition** **assuming** $nonempty$

notation is_surj_fm ($\langle \cdot \rangle$ $surj'$ ($_$, $_$) is $_$ \rangle)

definition

$surj_rel :: [i \Rightarrow o, i, i] \Rightarrow i$ ($\langle surj'(_$, $_$) \rangle) **where**
 $surj_rel(M, A, B) \equiv THE\ d.\ is_surj(M, A, B, d)$

abbreviation

$surj_r_set :: [i, i, i] \Rightarrow i$ ($\langle surj'(_$, $_$) \rangle) **where**
 $surj_r_set(M) \equiv surj_rel(\#\#M)$

locale $M_surj = M_Pi +$

assumes

$surjP_separation: M(A) \Longrightarrow M(B) \Longrightarrow separation(M, \lambda x. surjP_rel(M, A, B, x))$

begin


```

lemma is_surj_uniqueness:
  assumes
     $M(r) \ M(B)$ 
     $is\_surj(M, r, B, d) \ is\_surj(M, r, B, d')$ 
  shows
     $d = d'$ 
  using assms function_space_rel_iff extensionality_trans
  unfolding is_surj_def by simp

lemma is_surj_witness:  $M(r) \implies M(B) \implies \exists d[M]. \ is\_surj(M, r, B, d)$ 
  using surjP_separation function_space_rel_iff
  unfolding is_surj_def by simp

lemma is_surj_closed :
   $is\_surj(M, x, y, d) \implies M(d)$ 
  unfolding is_surj_def by simp

lemma surj_rel_closed[intro, simp]:
  assumes  $M(x) \ M(y)$ 
  shows  $M(surj\_rel(M, x, y))$ 
proof -
  have  $is\_surj(M, x, y, \text{THE } xa. \ is\_surj(M, x, y, xa))$ 
  using assms
    theI[OF ex1I[of is_surj(M, x, y)], OF _ is_surj_uniqueness[of x y]]
    is_surj_witness
  by auto
  then show ?thesis
  using assms is_surj_closed
  unfolding surj_rel_def
  by blast
qed

lemmas trans_surj_rel_closed[trans_closed] = transM[OF _ surj_rel_closed]

lemma surj_rel_iff:
  assumes  $M(x) \ M(y) \ M(d)$ 
  shows  $is\_surj(M, x, y, d) \longleftrightarrow d = surj\_rel(M, x, y)$ 
proof (intro iffI)
  assume  $d = surj\_rel(M, x, y)$ 
  moreover
  note assms
  moreover from this
  obtain e where  $M(e) \ is\_surj(M, x, y, e)$ 
  using is_surj_witness by blast
  ultimately
  show  $is\_surj(M, x, y, d)$ 
  using is_surj_uniqueness[of x y] is_surj_witness
    theI[OF ex1I[of is_surj(M, x, y)], OF _ is_surj_uniqueness[of x y], of e]
  unfolding surj_rel_def

```

```

    by auto
next
  assume is_surj(M, x, y, d)
  with assms
  show d = surj_rel(M, x, y)
    using is_surj_uniqueness unfolding surj_rel_def
    by (blast del:the_equality intro:the_equality[symmetric])
qed

lemma def_surj_rel:
  assumes M(A) M(B)
  shows surj_rel(M, A, B) =
    {f ∈ function_space_rel(M, A, B). ∀ y[M]. ∃ x[M]. y ∈ B ⟶ x ∈ A ∧ f'x=y }
    (is _ = Collect(_, ?P))
proof -
  from assms
  have surj_rel(M, A, B) ⊆ function_space_rel(M, A, B)
    using surj_rel_iff[of A B surj_rel(M, A, B)] function_space_rel_iff
    unfolding is_surj_def by auto
  moreover from assms
  have f ∈ surj_rel(M, A, B) ⟹ ?P(f) for f
    using surj_rel_iff[of A B surj_rel(M, A, B)] function_space_rel_iff
    def_surjP_rel transM[OF _ function_space_rel_closed, OF _ ⟨M(A)⟩
  ⟨M(B)⟩]
    unfolding is_surj_def by auto
  moreover from assms
  have f ∈ function_space_rel(M, A, B) ⟹ ?P(f) ⟹ f ∈ surj_rel(M, A, B) for f
    using surj_rel_iff[of A B surj_rel(M, A, B)] function_space_rel_iff
    def_surjP_rel transM[OF _ function_space_rel_closed, OF _ ⟨M(A)⟩
  ⟨M(B)⟩]
    unfolding is_surj_def by auto
  ultimately
  show ?thesis by force
qed

lemma surj_rel_char:
  assumes M(A) M(B)
  shows surj_rel(M, A, B) = {f ∈ surj(A, B). M(f)}
proof -
  from assms
  interpret M_Pi_assumptions M A λ_. B
    using Pi_replacement Pi_separation
    by unfold_locales (auto dest:transM simp add:Sigfun_def)
  from assms
  show ?thesis
    using def_surj_rel[OF assms] def_function_space_rel[OF assms]
    transM[OF _ ⟨M(A)⟩] transM[OF _ ⟨M(B)⟩] Pi_rel_char
    unfolding surj_def
    by auto

```

```

qed

end — M_surj

locale M_N_surj = M:M_surj + N:M_surj N for N +
  assumes
    M_imp_N:M(x)  $\implies$  N(x)
begin

lemma surj_rel_transfer: M(A)  $\implies$  M(B)  $\implies$  surj_rel(M,A,B)  $\subseteq$  surj_rel(N,A,B)
  using M.surj_rel_char N.surj_rel_char
  by (auto dest!:M_imp_N)

end — M_N_surj


definition
  is_Int :: [i $\implies$ o,i,i,i] $\implies$ o where
    is_Int(M,A,B,I)  $\equiv$  M(I)  $\wedge$  ( $\forall x[M]. x \in I \longleftrightarrow x \in A \wedge x \in B$ )


reldb_rem relational inter
reldb_add absolute relational ZF_Base.Int is_Int


synthesize is_Int from_definition assuming nonempty
notation is_Int_fm ( $\langle \_ \cap \_ \text{ is } \_ \rangle$ )


context M_basic
begin

lemma is_Int_closed :
  is_Int(M,A,B,I)  $\implies$  M(I)
  unfolding is_Int_def by simp

lemma is_Int_abs:
  assumes
    M(A) M(B) M(I)
  shows
    is_Int(M,A,B,I)  $\longleftrightarrow$  I = A  $\cap$  B
  using assms transM[OF _  $\langle M(B) \rangle$ ] transM[OF _  $\langle M(I) \rangle$ ]
  unfolding is_Int_def by blast

lemma is_Int_uniqueness:
  assumes
    M(r) M(B)
    is_Int(M,r,B,d) is_Int(M,r,B,d')
  shows
    d=d'
proof -

```

```

have  $M(d)$  and  $M(d')$ 
  using assms is_Int_closed by simp+
then show ?thesis
  using assms is_Int_abs by simp
qed

```

Note: $\llbracket M(?A); M(?B) \rrbracket \implies M(?A \cap ?B)$ already in *ZF-Constructible.Relative*.

end — *M_basic*

11.5 Discipline for *bij*

```

reldb_add functional inj inj_rel
reldb_add functional relational inj_rel is_inj
reldb_add functional surj surj_rel
reldb_add functional relational surj_rel is_surj
relativize functional bij bij_rel external
relationalize bij_rel is_bij

```

```

synthesize is_bij from_definition assuming nonempty
notation is_bij_fm ( $\langle \cdot \text{bij}'(\_, \_) \text{ is } \_ \rangle$ )

```

abbreviation

```

bij_r_class ::  $[i \Rightarrow o, i, i] \Rightarrow i (\langle \text{bij}'(\_, \_) \rangle)$  where
bij_r_class  $\equiv$  bij_rel

```

abbreviation

```

bij_r_set ::  $[i, i, i] \Rightarrow i (\langle \text{bij}'(\_, \_) \rangle)$  where
bij_r_set(M)  $\equiv$  bij_rel( $\#\#M$ )

```

```

locale M_Perm = M_Pi + M_inj + M_surj
begin

```

```

lemma is_bij_closed : is_bij(M, f, y, d)  $\implies M(d)$ 
  unfolding is_bij_def using is_Int_closed is_inj_witness is_surj_witness by
auto

```

```

lemma bij_rel_closed[intro, simp]:
  assumes  $M(x) \ M(y)$ 
  shows  $M(\text{bij\_rel}(M, x, y))$ 
  unfolding bij_rel_def
  using assms Int_closed surj_rel_closed inj_rel_closed
  by auto

```

lemmas *trans_bij_rel_closed*[*trans_closed*] = *transM*[*OF _ bij_rel_closed*]

lemma *bij_rel_iff*:

assumes $M(x) \ M(y) \ M(d)$
shows $is_bij(M, x, y, d) \longleftrightarrow d = bij_rel(M, x, y)$
unfolding *is_bij_def* *bij_rel_def*
using *assms surj_rel_iff inj_rel_iff is_Int_abs*
by *auto*

lemma *def_bij_rel*:

assumes $M(A) \ M(B)$
shows $bij_rel(M, A, B) = inj_rel(M, A, B) \cap surj_rel(M, A, B)$
using *assms bij_rel_iff inj_rel_iff surj_rel_iff*
is_Int_abs— For absolute terms, "*_abs*" replaces "*_iff*". Also, in this case "*_closed*" is in the simpset.
unfolding *is_bij_def* **by** *simp*

lemma *bij_rel_char*:

assumes $M(A) \ M(B)$
shows $bij_rel(M, A, B) = \{f \in bij(A, B). \ M(f)\}$
using *assms def_bij_rel inj_rel_char surj_rel_char*
unfolding *bij_def*— Unfolding this might be a pattern already
by *auto*

end — *M_Perm*

locale *M_N_Perm* = *M_N_Pi* + *M_N_inj* + *M_N_surj* + *M:M_Perm* +
N:M_Perm N

begin

lemma *bij_rel_transfer*: $M(A) \implies M(B) \implies bij_rel(M, A, B) \subseteq bij_rel(N, A, B)$
using *M.bij_rel_char N.bij_rel_char*
by (*auto dest!: M_imp_N*)

end — *M_N_Perm*

11.6 Discipline for (\approx)

relativize functional *eqpoll eqpoll_rel external*
relationalize *eqpoll_rel is_eqpoll*

synthesize *is_eqpoll from_definition* **assuming** *nonempty*
arity_theorem for *is_eqpoll_fm*
notation *is_eqpoll_fm* ($\langle \cdot \approx \cdot \rangle$)

context *M_Perm* **begin**

```

is_iff_rel for eqpoll
  using bij_rel_iff unfolding is_eqpoll_def eqpoll_rel_def by simp

end — M_Perm

abbreviation
  eqpoll_r ::  $[i, i \Rightarrow o, i] \Rightarrow o$  ( $\langle \_ \approx - \_ \rangle [51, 1, 51] 50$ ) where
     $A \approx^M B \equiv eqpoll\_rel(M, A, B)$ 

abbreviation
  eqpoll_r_set ::  $[i, i, i] \Rightarrow o$  ( $\langle \_ \approx - \_ \rangle [51, 1, 51] 50$ ) where
    eqpoll_r_set(A, M)  $\equiv eqpoll\_rel(\#\#M, A)$ 

context M_Perm
begin

lemma def_eqpoll_rel:
  assumes
    M(A) M(B)
  shows
     $eqpoll\_rel(M, A, B) \longleftrightarrow (\exists f[M]. f \in bij\_rel(M, A, B))$ 
  using assms bij_rel_iff
  unfolding eqpoll_rel_def by simp

end — M_Perm

context M_N_Perm
begin

lemma eqpoll_rel_transfer: assumes  $A \approx^M B$  M(A) M(B)
  shows  $A \approx^N B$ 
proof -
  note assms
  moreover from this
  obtain f where  $f \in bij^M(A, B)$  N(f)
    using M.def_eqpoll_rel by (auto dest!: M_imp_N)
  moreover from calculation
  have  $f \in bij^N(A, B)$ 
    using bij_rel_transfer by (auto)
  ultimately
  show ?thesis
    using N.def_eqpoll_rel by (blast dest!: M_imp_N)
qed

end — M_N_Perm

```

11.7 Discipline for (\lesssim)

relativize functional *lepoll lepoll_rel external*

relationalize *lepoll_rel is_lepoll*

synthesize *is_lepoll from_definition assuming nonempty*

notation *is_lepoll_fm* $(\langle \cdot \lesssim \cdot \rangle)$

arity_theorem for *is_lepoll_fm*

context *M_inj* **begin**

is_iff_rel for *lepoll*

using *inj_rel_iff* **unfolding** *is_lepoll_def lepoll_rel_def* **by** *simp*

end — *M_inj*

abbreviation

lepoll_r :: $[i, i \Rightarrow o, i] \Rightarrow o$ $(\langle \cdot \lesssim \cdot \rangle [51, 1, 51] \ 50)$ **where**

$A \lesssim^M B \equiv \text{lepoll_rel}(M, A, B)$

abbreviation

lepoll_r_set :: $[i, i, i] \Rightarrow o$ $(\langle \cdot \lesssim \cdot \rangle [51, 1, 51] \ 50)$ **where**

$\text{lepoll_r_set}(A, M) \equiv \text{lepoll_rel}(\#\#M, A)$

context *M_Perm*

begin

lemma *def_lepoll_rel*:

assumes

$M(A) \ M(B)$

shows

$\text{lepoll_rel}(M, A, B) \longleftrightarrow (\exists f[M]. f \in \text{inj_rel}(M, A, B))$

using *assms inj_rel_iff*

unfolding *lepoll_rel_def* **by** *simp*

end — *M_Perm*

context *M_N_Perm*

begin

lemma *lepoll_rel_transfer*: **assumes** $A \lesssim^M B \ M(A) \ M(B)$

shows $A \lesssim^N B$

proof -

note *assms*

moreover from *this*

obtain *f* **where** $f \in \text{inj}^M(A, B) \ N(f)$

using *M.def_lepoll_rel* **by** *(auto dest!: M_imp_N)*

moreover from *calculation*

have $f \in \text{inj}^N(A, B)$

```

    using inj_rel_transfer by (auto)
  ultimately
  show ?thesis
    using N.def_lepoll_rel by (blast dest!:M_imp_N)
qed

end — M_N_Perm

```

11.8 Discipline for (\prec)

```

relativize functional lesspoll lesspoll_rel external
relationalize lesspoll_rel is_lesspoll

```

```

synthesize is_lesspoll from_definition assuming nonempty
notation is_lesspoll_fm ( $\langle \cdot \prec \cdot \rangle$ )
arity_theorem for is_lesspoll_fm

```

```

context M_Perm begin

```

```

is_iff_rel for lesspoll
  using is_lepoll_iff is_eqpoll_iff
  unfolding is_lesspoll_def lesspoll_rel_def by simp

```

```

end — M_Perm

```

abbreviation

$lesspoll_r :: [i, i \Rightarrow o, i] \Rightarrow o \ (\langle _ \prec _ \rangle [51, 1, 51] \ 50)$ **where**
 $A \prec^M B \equiv lesspoll_rel(M, A, B)$

abbreviation

$lesspoll_r_set :: [i, i, i] \Rightarrow o \ (\langle _ \prec _ \rangle [51, 1, 51] \ 50)$ **where**
 $lesspoll_r_set(A, M) \equiv lesspoll_rel(\#\#M, A)$

Since $lesspoll_rel$ is defined as a propositional combination of older terms, there is no need for a separate “def” theorem for it.

Note that $lesspoll_rel$ is neither Σ_1^{ZF} nor Π_1^{ZF} , so there is no “transfer” theorem for it.

end

theory Discipline_Cardinal

imports

Discipline_Function

begin

declare [[syntax_ambiguity_warning = false]]

relativize functional cardinal cardinal_rel **external**

relationalize cardinal_rel is_cardinal

synthesize is_cardinal **from_definition** assuming nonempty

notation $is_cardinal_fm$ ($\langle cardinal'(_) is_ \rangle$)

abbreviation

$cardinal_r :: [i, i \Rightarrow o] \Rightarrow i \ (\langle | _ | _ \rangle)$ **where**
 $|x|^M \equiv cardinal_rel(M, x)$

abbreviation

$cardinal_r_set :: [i, i] \Rightarrow i \ (\langle | _ | _ \rangle)$ **where**
 $|x|^M \equiv cardinal_rel(\#\# M, x)$

context $M_trivial$ **begin**

rel_closed **for** $cardinal$

using $Least_closed'[of \ \lambda i. M(i) \wedge i \approx^M A]$

unfolding $cardinal_rel_def$

by $simp$

end

manual_arity **intermediate** **for** is_Int_fm

unfolding $is_Int_fm_def$

using $arity_pred_Un_distrib$

by $(simp)$

arity_theorem **for** is_Int_fm

arity_theorem **for** $is_funspace_fm$

arity_theorem **for** $is_function_space_fm$

arity_theorem **for** $surjP_rel_fm$

arity_theorem **intermediate** **for** is_surj_fm

lemma $arity_is_surj_fm$ $[arity]$:

$A \in nat \Longrightarrow B \in nat \Longrightarrow I \in nat \Longrightarrow arity(is_surj_fm(A, B, I)) = succ(A) \cup succ(B) \cup succ(I)$

using $arity_is_surj_fm' \ pred_Un_distrib$

by $auto$

arity_theorem **for** $injP_rel_fm$

arity_theorem **intermediate** **for** is_inj_fm

lemma $arity_is_inj_fm$ $[arity]$:

$A \in nat \Longrightarrow B \in nat \Longrightarrow I \in nat \Longrightarrow arity(is_inj_fm(A, B, I)) = succ(A) \cup succ(B) \cup succ(I)$

using $arity_is_inj_fm' \ pred_Un_distrib$

by $auto$

```

arity__theorem for is_bij_fm

arity__theorem for is_eqpoll_fm

arity__theorem for is_cardinal_fm

context M_Perm begin

is_iff_rel for cardinal
  using least_abs'[of  $\lambda i. M(i) \wedge i \approx^M A$ ]
  is_eqpoll_iff
  unfolding is_cardinal_def cardinal_rel_def
  by simp
end

reldb_add functional Ord Ord
reldb_add relational Ord ordinal
reldb_add functional lt lt
reldb_add relational lt lt_rel
synthesize lt_rel from_definition
notation lt_rel_fm ( $\langle \cdot < \cdot \rangle$ )
arity__theorem intermediate for lt_rel_fm

lemma arity_lt_rel_fm[arity]:  $a \in \text{nat} \implies b \in \text{nat} \implies \text{arity}(\text{lt\_rel\_fm}(a, b)) =$ 
 $\text{succ}(a) \cup \text{succ}(b)$ 
  using arity_lt_rel_fm'
  by auto

relativize functional Card Card_rel external
relationalize Card_rel is_Card
synthesize is_Card from_definition assuming nonempty
notation is_Card_fm ( $\langle \cdot \text{Card}'(\_) \cdot \rangle$ )
arity__theorem for is_Card_fm

notation Card_rel ( $\langle \text{Card}'(\_) \rangle$ )

lemma (in M_Perm) is_Card_iff:  $M(A) \implies \text{is\_Card}(M, A) \longleftrightarrow \text{Card}^M(A)$ 
  using is_cardinal_iff
  unfolding is_Card_def Card_rel_def by simp

abbreviation
  Card_r_set ::  $[i, i] \Rightarrow o$  ( $\langle \text{Card}'(\_) \rangle$ ) where
   $\text{Card}^M(i) \equiv \text{Card\_rel}(\#\#M, i)$ 

relativize functional InfCard InfCard_rel external
relationalize InfCard_rel is_InfCard
synthesize is_InfCard from_definition assuming nonempty
notation is_InfCard_fm ( $\langle \cdot \text{InfCard}'(\_) \cdot \rangle$ )
arity__theorem for is_InfCard_fm

```

notation $InfCard_rel$ ($\langle InfCard_('(_')) \rangle$)

abbreviation

$InfCard_r_set :: [i,i] \Rightarrow o$ ($\langle InfCard_('(_')) \rangle$) **where**
 $InfCard^M(i) \equiv InfCard_rel(\#\#M,i)$

relativize functional $cadd$ $cadd_rel$ **external**

abbreviation

$cadd_r :: [i,i \Rightarrow o,i] \Rightarrow i$ ($\langle _ \oplus _ \rangle [66,1,66] 65$) **where**
 $A \oplus^M B \equiv cadd_rel(M,A,B)$

context M_basic **begin**

rel_closed **for** $cadd$

using $cardinal_rel_closed$

unfolding $cadd_rel_def$

by $simp$

end

relationalize $cadd_rel$ is_cadd

manual_schematic **for** is_cadd **assuming** $nonempty$

unfolding is_cadd_def

by ($rule$ iff_sats sum_iff_sats | $simp$) +

synthesize is_cadd **from_schematic**

arity_theorem **for** sum_fm

arity_theorem **for** is_cadd_fm

context M_Perm **begin**

is_iff_rel **for** $cadd$

using $is_cardinal_iff$

unfolding is_cadd_def $cadd_rel_def$

by $simp$

end

relativize functional $cmult$ $cmult_rel$ **external**

abbreviation

$cmult_r :: [i,i \Rightarrow o,i] \Rightarrow i$ ($\langle _ \otimes _ \rangle [66,1,66] 65$) **where**
 $A \otimes^M B \equiv cmult_rel(M,A,B)$

relationalize $cmult_rel$ is_cmult

```

declare cartprod_iff_sats [iff_sats]

synthesize is_cmult from_definition assuming nonempty

arity_theorem for is_cmult_fm

context M_Perm begin

  rel_closed for cmult
    using cardinal_rel_closed
    unfolding cmult_rel_def
    by simp

  is_iff_rel for cmult
    using is_cardinal_iff
    unfolding is_cmult_def cmult_rel_def
    by simp

end

end

```

12 Relativization of the cumulative hierarchy

```

theory Univ_Relative
  imports
    ZF-Constructible.Rank
    ZF.Univ
    Internalizations
    Recursion_Thms
    Discipline_Cardinal

  begin

    declare arity_ordinal_fm[arity]

    context M_trivial
    begin
      declare powerset_abs[simp]

      lemma family_union_closed:  $\llbracket \text{strong\_replacement}(M, \lambda x y. y = f(x)); M(A); \forall x \in A. M(f(x)) \rrbracket$ 
         $\implies M(\bigcup_{x \in A} f(x))$ 
        using RepFun_closed ..

      lemma family_union_closed':  $\llbracket \text{strong\_replacement}(M, \lambda x y. x \in A \wedge y = f(x)); M(A); \forall x \in A. M(f(x)) \rrbracket$ 
         $\implies M(\bigcup_{x \in A} f(x))$ 
        using RepFun_closed2
    end
  end

```

```

    by simp

end — M_trivial

definition
  Powapply :: [i,i] ⇒ i where
    Powapply(f,y) ≡ Pow(f'y)

reldb_add functional Pow Pow_rel
reldb_add relational Pow is_Pow

declare Replace_iff_sats[iff_sats]
synthesize is_Pow from_definition assuming nonempty
arity_theorem for is_Pow_fm

relativize functional Powapply Powapply_rel
relationalize Powapply_rel is_Powapply
synthesize is_Powapply from_definition assuming nonempty
arity_theorem for is_Powapply_fm

notation Powapply_rel (⟨Powapply- '(_,'_')⟩)

context M_basic
begin

rel_closed for Powapply
  unfolding Powapply_rel_def
  by simp

is_iff_rel for Powapply
  using Pow_rel_iff
  unfolding is_Powapply_def Powapply_rel_def
  by simp

end — M_basic

definition
  HVfrom :: [i,i,i] ⇒ i where
    HVfrom(A,x,f) ≡ A ∪ (⋃ y∈x. Powapply(f,y))

relativize functional HVfrom HVfrom_rel
relationalize HVfrom_rel is_HVfrom
synthesize is_HVfrom from_definition assuming nonempty
arity_theorem intermediate for is_HVfrom_fm

lemma arity_is_HVfrom_fm:
  A ∈ nat ⇒
  x ∈ nat ⇒
  f ∈ nat ⇒

```

```

    d ∈ nat ⇒
    arity(is_HVfrom_fm(A, x, f, d)) = succ(A) ∪ succ(d) ∪ (succ(x) ∪ succ(f))
    using arity_is_HVfrom_fm' arity_is_Powapply_fm
    by(simp,subst arity_Replace_fm[of ·.(∃·0 = 0·) ∧ ·.(∃·0 = 0·) ∧ is_Powapply_fm(succ(succ(succ(succ(f))))),
    0, 1) .. succ(succ(x)) 1])
    (simp_all,simp add:arity_pred_Un_distrib )

notation HVfrom_rel (⟨HVfrom'(_,_,_')⟩)

locale M_HVfrom = M_eclose +
  assumes
    Powapply_replacement:
      M(f) ⇒ strong_replacement(M,λy z. z = PowapplyM(f,y))
begin

is_iff_rel for HVfrom
proof -
  assume assms:M(A) M(x) M(f) M(res)
  then
    have is_Replace(M,x,λy z. z = PowapplyM(f,y),r) ⟷ r = {z . y∈x, z =
    PowapplyM(f,y)}
    if M(r) for r
    using that Powapply_rel_closed
      Replace_abs[of x r λy z. z = PowapplyM(f,y)] transM[of _ x]
    by simp
  moreover
    have is_Replace(M,x,is_Powapply(M,f),r) ⟷
      is_Replace(M,x,λy z. z = PowapplyM(f,y),r) if M(r) for r
    using assms that is_Powapply_iff is_Replace_cong
    by simp
  ultimately
    have is_Replace(M,x,is_Powapply(M,f),r) ⟷ r = {z . y∈x, z = PowapplyM(f,y)}
    if M(r) for r
    using that assms
    by simp
  moreover
    have M({z . y ∈ x, z = PowapplyM(f,y)})
    using assms strong_replacement_closed[OF Powapply_replacement]
      Powapply_rel_closed transM[of _ x]
    by simp
  moreover from assms
  — intermediate step for body of Replace
    have {z . y ∈ x, z = PowapplyM(f,y)} = {y . w ∈ x, M(y) ∧ M(w) ∧ y =
    PowapplyM(f,w)}
    by (auto dest:transM)
  ultimately
  show ?thesis
    using assms
  unfolding is_HVfrom_def HVfrom_rel_def

```

```

    by (auto dest:transM)
qed

rel_closed for HVfrom
proof -
  assume assms: M(A) M(x) M(f)
  have M({z . y ∈ x, z = PowapplyM(f,y)})
    using assms strong_replacement_closed[OF Powapply_replacement]
    Powapply_rel_closed transM[of _ x]
    by simp
  then
  have M(A ∪ ⋃({z . y ∈ x, z = PowapplyM(f,y)}))
    using assms
    by simp
  moreover from assms
  — intermediate step for body of Replace
  have {z . y ∈ x, z = PowapplyM(f,y)} = {y . w ∈ x, M(y) ∧ M(w) ∧ y =
PowapplyM(f,w)}
    by (auto dest:transM)
  ultimately
  show ?thesis
    using assms
    unfolding HVfrom_rel_def
    by simp
qed

end — M_HVfrom

definition
  Vfrom_rel :: [i ⇒ o, i, i] ⇒ i (‹Vfrom—'(_,_)'›) where
  VfromM(A, i) = transrec(i, HVfrom_rel(M, A))

definition
  is_Vfrom :: [i ⇒ o, i, i, i] ⇒ o where
  is_Vfrom(M, A, i, z) ≡ is_transrec(M, is_HVfrom(M, A), i, z)

definition
  Hrank :: [i, i] ⇒ i where
  Hrank(x, f) ≡ (⋃ y ∈ x. succ(f'y))

definition
  rrank :: i ⇒ i where
  rrank(a) ≡ Memrel(eclose({a}))+

relativize functional Hrank Hrank_rel
relationalize Hrank_rel is_Hrank
synthesize is_Hrank from_definition assuming nonempty
arity_theorem for is_Hrank_fm

```

```

locale  $M\_Vfrom = M\_HVfrom +$ 
  assumes
     $trepl\_HVfrom : \llbracket M(A); M(i) \rrbracket \implies transrec\_replacement(M, is\_HVfrom(M, A), i)$ 
  and
     $Hrank\_replacement : M(f) \implies strong\_replacement(M, \lambda x y . y = succ(f'x))$ 
  and
     $is\_Hrank\_replacement : M(x) \implies wfrec\_replacement(M, is\_Hrank(M), rrank(x))$ 
  and
     $HVfrom\_replacement : \llbracket M(i) ; M(A) \rrbracket \implies$ 
       $transrec\_replacement(M, is\_HVfrom(M, A), i)$ 

begin

lemma  $Vfrom\_rel\_iff :$ 
  assumes  $M(A) \ M(i) \ M(z) \ Ord(i)$ 
  shows  $is\_Vfrom(M, A, i, z) \longleftrightarrow z = Vfrom^M(A, i)$ 
proof -
  have  $relation2(M, is\_HVfrom(M, A), HVfrom\_rel(M, A))$ 
  using  $assms \ is\_HVfrom\_iff$ 
  unfolding  $relation2\_def$ 
  by  $simp$ 
  then
  show  $?thesis$ 
  using  $assms \ HVfrom\_rel\_closed \ trepl\_HVfrom$ 
     $transrec\_abs[of \ is\_HVfrom(M, A) \ i \ HVfrom\_rel(M, A) \ z]$ 
  unfolding  $is\_Vfrom\_def \ Vfrom\_rel\_def$ 
  by  $simp$ 
qed

lemma  $relation2\_HVfrom : M(A) \implies relation2(M, is\_HVfrom(M, A), HVfrom\_rel(M, A))$ 
  using  $is\_HVfrom\_iff$ 
  unfolding  $relation2\_def$ 
  by  $auto$ 

lemma  $HVfrom\_closed :$ 
   $M(A) \implies \forall x[M]. \forall g[M]. function(g) \longrightarrow M(HVfrom\_rel(M, A, x, g))$ 
  using  $HVfrom\_rel\_closed$  by  $simp$ 

lemma  $Vfrom\_rel\_closed :$ 
  assumes  $M(A) \ M(i) \ Ord(i)$ 
  shows  $M(transrec(i, HVfrom\_rel(M, A)))$ 
  using  $is\_HVfrom\_iff \ HVfrom\_closed \ HVfrom\_replacement \ assms \ trepl\_HVfrom$ 
     $relation2\_HVfrom$ 
     $transrec\_closed[of \ is\_HVfrom(M, A) \ i \ HVfrom\_rel(M, A)]$ 
  by  $simp$ 

lemma  $transrec\_HVfrom :$ 
  assumes  $M(A)$ 
  shows  $Ord(i) \implies M(i) \implies \{x \in Vfrom(A, i). M(x)\} = transrec(i, HVfrom\_rel(M, A))$ 

```



```

proof (induct rule:trans_induct)
  have eq:( $\bigcup x \in i. \{x \in \text{Pow}(\text{transrec}(x, \text{HVfrom\_rel}(M, A))) . M(x)\} = \bigcup \{y .$ 
 $x \in i, y = \text{Pow}^M(\text{transrec}(x, \text{HVfrom\_rel}(M, A)))\}$ 
    if  $\text{Ord}(i)$  M( $i$ ) for  $i$ 
    using assms  $\text{Pow\_rel\_char}[OF \text{Vfrom\_rel\_closed}[OF \langle M(A) \rangle \text{transM}[of\_ i]]]$ 
 $\text{Ord\_in\_Ord'}$  that
    by auto
  case (step  $i$ )
  then
    have 0:  $M(\text{Pow}^M(\text{transrec}(x, \text{HVfrom\_rel}(M, A))))$  if  $x \in i$  for  $x$ 
    using assms that  $\text{transM}[of\_ i]$   $\text{Ord\_in\_Ord}$ 
       $\text{transrec\_closed}[of \text{is\_HVfrom}(M, A) \text{ \_ HVfrom\_rel}(M, A)]$  trepl\_HVfrom
 $\text{relation2\_HVfrom}$ 
    by auto
    have  $\text{Vfrom}(A, i) = A \cup (\bigcup y \in i. \text{Pow}((\lambda x \in i. \text{Vfrom}(A, x)) \text{ ` } y))$ 
    using  $\text{def\_transrec}[OF \text{Vfrom\_def}, of \text{A } i]$  by simp
    then
    have  $\text{Vfrom}(A, i) = A \cup (\bigcup y \in i. \text{Pow}(\text{Vfrom}(A, y)))$ 
    by simp
    then
    have  $\{x \in \text{Vfrom}(A, i). M(x)\} = \{x \in A. M(x)\} \cup (\bigcup y \in i. \{x \in \text{Pow}(\text{Vfrom}(A, y)).$ 
 $M(x)\})$ 
    by auto
    with  $\langle M(A) \rangle$ 
    have  $\{x \in \text{Vfrom}(A, i). M(x)\} = A \cup (\bigcup y \in i. \{x \in \text{Pow}(\text{Vfrom}(A, y)). M(x)\})$ 
    by (auto intro:transM)
    also
    have  $\dots = A \cup (\bigcup y \in i. \{x \in \text{Pow}(\{z \in \text{Vfrom}(A, y). M(z)\}). M(x)\})$ 
    proof -
    have  $\{x \in \text{Pow}(\text{Vfrom}(A, y)). M(x)\} = \{x \in \text{Pow}(\{z \in \text{Vfrom}(A, y). M(z)\}). M(x)\}$ 
    if  $y \in i$  for  $y$  by (auto intro:transM)
    then
    show ?thesis by simp
  qed
  also from step
  have  $\dots = A \cup (\bigcup y \in i. \{x \in \text{Pow}(\text{transrec}(y, \text{HVfrom\_rel}(M, A))). M(x)\})$ 
    using  $\text{transM}[of\_ i]$ 
    by auto
  also
  have  $\dots = \text{transrec}(i, \text{HVfrom\_rel}(M, A))$ 
    using 0 step assms  $\text{transM}[of\_ i]$  eq
     $\text{def\_transrec}[of \lambda y. \text{transrec}(y, \text{HVfrom\_rel}(M, A)) \text{HVfrom\_rel}(M, A) i]$ 
    unfolding  $\text{Powapply\_rel\_def}$   $\text{HVfrom\_rel\_def}$ 
    by auto
  finally
  show ?case .
qed

```

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

```

V = {x ∈ Vfrom(A,i). M(x)}
unfolding is_Vfrom_def
using is_HVfrom_iff
transrec_abs[of is_HVfrom(M,A) i HVfrom_rel(M,A)] trepl_HVfrom relation2_HVfrom
transrec_HVfrom
by simp

lemma Vfrom_closed:  $\llbracket M(A); M(i); \text{Ord}(i) \rrbracket \implies M(\{x \in Vfrom(A,i). M(x)\})$ 
unfolding is_Vfrom_def
using is_HVfrom_iff HVfrom_closed HVfrom_replacement
transrec_closed[of is_HVfrom(M,A) i HVfrom_rel(M,A)] trepl_HVfrom relation2_HVfrom
transrec_HVfrom
by simp

end — M_Vfrom

```

12.1 Formula synthesis

```

context M_Vfrom
begin

rel_closed for Hrank
unfolding Hrank_rel_def
using Hrank_replacement
by simp

is_iff_rel for Hrank
proof -
assume M(f) M(x) M(res)
moreover from this
have {a . y ∈ x, M(a) ∧ M(y) ∧ a = succ(f ‘ y)} = {a . y ∈ x, a = succ(f ‘ y)}
using transM[of _ x]
by auto
ultimately
show ?thesis
unfolding is_Hrank_def Hrank_rel_def
using Replace_abs transM[of _ x] Hrank_replacement
by auto
qed

lemma relation2_Hrank :
relation2(M, is_Hrank(M), Hrank)
unfolding relation2_def
proof(clarify)
fix x f res
assume M(x) M(f) M(res)
moreover from this
have {a . y ∈ x, M(a) ∧ M(y) ∧ a = succ(f ‘ y)} = {a . y ∈ x, a = succ(f ‘ y)}
using transM[of _ x]

```

```

    by auto
  ultimately
  show is_Hrank(M, x, f, res)  $\longleftrightarrow$  res = Hrank(x, f)
    unfolding Hrank_def relation2_def
    using is_Hrank_iff[unfolded Hrank_rel_def]
    by auto
qed

lemma Hrank_closed :
   $\forall x[M]. \forall g[M]. \text{function}(g) \longrightarrow M(\text{Hrank}(x, g))$ 
proof(clarify)
  fix x g
  assume M(x) M(g)
  then
  show M(Hrank(x, g))
    using RepFun_closed[OF Hrank_replacement] transM[of _ x]
    unfolding Hrank_def
    by auto
qed

end — M_basic

context M_eclose
begin

lemma wf_rrank :  $M(x) \implies \text{wf}(\text{rrank}(x))$ 
  unfolding rrank_def using wf_transcl[OF wf_Memrel] .

lemma trans_rrank :  $M(x) \implies \text{trans}(\text{rrank}(x))$ 
  unfolding rrank_def using trans_transcl .

lemma relation_rrank :  $M(x) \implies \text{relation}(\text{rrank}(x))$ 
  unfolding rrank_def using relation_transcl .

lemma rrank_in_M :  $M(x) \implies M(\text{rrank}(x))$ 
  unfolding rrank_def by simp

end — M_eclose

lemma Hrank_transcl:  $\text{Hrank}(y, \text{restrict}(f, \text{Memrel}(\text{eclose}(\{x\})) - \{\{y\}\}))$ 
   $= \text{Hrank}(y, \text{restrict}(f, (\text{Memrel}(\text{eclose}(\{x\})) \hat{+}) - \{\{y\}\}))$ 
  unfolding Hrank_def
  using restrict_trans_eq by simp

lemma rank_transcl:  $\text{rank}(x) = \text{wfrec}(\text{rrank}(x), x, \text{Hrank})$ 
proof -
  have  $\text{rank}(x) = \text{wfrec}(\text{Memrel}(\text{eclose}(\{x\})), x, \text{Hrank})$ 
    (is _ =  $\text{wfrec}(\text{?r}, \_, \_)$ )
    unfolding rank_def transrec_def Hrank_def by simp

```

```

also
have ... = wftrec(?r^+, x,  $\lambda y f. Hrank(y, restrict(f, ?r-^+\{y\}))$ )
  unfolding wfrec_def ..
also
have ... = wftrec(?r^+, x,  $\lambda y f. Hrank(y, restrict(f, (?r^+)^-\{y\}))$ )
  using Hrank_trancl by simp
also
have ... = wfrec(?r^+, x, Hrank)
  unfolding wfrec_def using trancl_eq_r[OF relation_trancl trans_trancl] by
simp
  finally
show ?thesis unfolding rrank_def .
qed

```

definition

```

Vset' :: [i]  $\Rightarrow$  i where
Vset'(A)  $\equiv$  Vfrom(0, A)

```

```

reldb_add relational Vfrom is_Vfrom
reldb_add functional Vfrom Vfrom_rel
relativize functional Vset' Vset_rel
relationalize Vset_rel is_Vset
reldb_rem relational Vset
reldb_rem functional Vset_rel
reldb_add relational Vset is_Vset
reldb_add functional Vset Vset_rel

```

schematic_goal sats_is_Vset_fm_auto:

assumes

```

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

```

shows

```

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

```

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

context M_Vfrom

begin

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

using Vfrom_abs **unfolding** is_Vset_def **by** simp

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

using Vfrom_closed **unfolding** is_Vset_def **by** simp

```

lemma rank_closed:  $M(a) \implies M(\text{rank}(a))$ 
  unfolding rank_trancl
  using Hrank_closed is_Hrank_replacement relation2_Hrank
    wf_rrank relation_rrank trans_rrank rrank_in_M
    trans_wfrec_closed[of rrank(a) a is_Hrank(M)]
    transM[of _ a]
  by auto

lemma M_into_Vset:
  assumes  $M(a)$ 
  shows  $\exists i[M]. \exists V[M]. \text{ordinal}(M,i) \wedge \text{is\_Vset}(M,i,V) \wedge a \in V$ 
proof -
  let  $?i = \text{succ}(\text{rank}(a))$ 
  from assms
  have  $a \in \{x \in V \text{from}(0, ?i). M(x)\}$  (is  $a \in ?V$ )
    using Vset_Ord_rank_iff by simp
  moreover from assms
  have  $M(?i)$ 
    using rank_closed by simp
  moreover
  note  $\langle M(a) \rangle$ 
  moreover from calculation
  have  $M(?V)$ 
    using Vfrom_closed by simp
  moreover from calculation
  have  $\text{ordinal}(M, ?i) \wedge \text{is\_Vfrom}(M, 0, ?i, ?V) \wedge a \in ?V$ 
    using Ord_rank Vfrom_abs by simp
  ultimately
  show ?thesis
    using nonempty_empty_abs
    unfolding is_Vset_def
    by blast
qed

end — M_HVfrom

end

```

13 Replacements using Lambdas

```

theory Lambda_Replacement
  imports
    Discipline_Function
  begin

```

In this theory we prove several instances of separation and replacement in *M_basic*. Moreover by assuming a seven instances of separation and ten instances of "lambda" replacements we prove a bunch of other instances.

definition

$lam_replacement :: [i \Rightarrow o, i \Rightarrow i] \Rightarrow o$ **where**
 $lam_replacement(M, b) \equiv strong_replacement(M, \lambda x y. y = \langle x, b(x) \rangle)$

lemma *separation_univ* :

shows $separation(M, M)$

unfolding *separation_def* **by** *auto*

context *M_basic*

begin

lemma *separation_iff'*:

assumes $separation(M, \lambda x. P(x))$ $separation(M, \lambda x. Q(x))$

shows $separation(M, \lambda x. P(x) \longleftrightarrow Q(x))$

using *assms separation_conj separation_imp iff_def*

by *auto*

lemma *separation_in_constant* :

assumes $M(a)$

shows $separation(M, \lambda x. x \in a)$

proof -

have $\{x \in A . x \in a\} = A \cap a$ **for** A **by** *auto*

with $\langle M(a) \rangle$

show *?thesis* **using** *separation_iff Collect_abs*

by *simp*

qed

lemma *separation_equal* :

shows $separation(M, \lambda x. x = a)$

proof -

have $\{x \in A . x = a\} = (if\ a \in A\ then\ \{a\}\ else\ 0)$ **for** A

by *auto*

then

have $M(\{x \in A . x = a\})$ **if** $M(A)$ **for** A

using $transM[OF_ \langle M(A) \rangle]$ **by** *simp*

then

show *?thesis* **using** *separation_iff Collect_abs*

by *simp*

qed

lemma (**in** *M_basic*) *separation_in_rev*:

assumes $(M)(a)$

shows $separation(M, \lambda x. a \in x)$

proof -

have $eq: \{x \in A. a \in x\} = Memrel(A \cup \{a\})$ “ $\{a\}$ **for** A

unfolding *ZF_Base.image_def*

by (*intro equalityI, auto simp: mem_not_refl*)

moreover from *assms*

```

have  $M(\text{Memrel}(A \cup \{a\}) \text{ “ } \{a\})$  if  $M(A)$  for  $A$ 
  using that by simp
ultimately
show ?thesis
  using separation_iff Collect_abs
  by simp
qed

```

```

lemma lam_replacement_iff_lam_closed:
  assumes  $\forall x[M]. M(b(x))$ 
  shows  $\text{lam\_replacement}(M, b) \longleftrightarrow (\forall A[M]. M(\lambda x \in A. b(x)))$ 
  using assms lam_closed lam_funtype[of _ b, THEN Pi_memberD]
  unfolding lam_replacement_def strong_replacement_def
  by (auto intro: lamI dest: transM)
  (rule lam_closed, auto simp add: strong_replacement_def dest: transM)

```

```

lemma lam_replacement_imp_lam_closed:
  assumes  $\text{lam\_replacement}(M, b) \ M(A) \ \forall x \in A. M(b(x))$ 
  shows  $M(\lambda x \in A. b(x))$ 
  using assms unfolding lam_replacement_def
  by (rule_tac lam_closed, auto simp add: strong_replacement_def dest: transM)

```

```

lemma lam_replacement_cong:
  assumes  $\text{lam\_replacement}(M, f) \ \forall x[M]. f(x) = g(x) \ \forall x[M]. M(f(x))$ 
  shows  $\text{lam\_replacement}(M, g)$ 
proof -
  note assms
  moreover from this
  have  $\forall A[M]. M(\lambda x \in A. f(x))$ 
    using lam_replacement_iff_lam_closed
    by simp
  moreover from calculation
  have  $(\lambda x \in A. f(x)) = (\lambda x \in A. g(x))$  if  $M(A)$  for  $A$ 
    using lam_cong[OF refl, of A f g] transM[OF _ that]
    by simp
  ultimately
  show ?thesis
    using lam_replacement_iff_lam_closed
    by simp
qed

```

```

lemma converse_subset :  $\text{converse}(r) \subseteq \{\langle \text{snd}(x), \text{fst}(x) \rangle . x \in r\}$ 
  unfolding converse_def
proof(intro subsetI, auto)
  fix u v
  assume  $\langle u, v \rangle \in r$  (is ?z ∈ r)
  moreover
  have  $v = \text{snd}(\text{?z}) \ u = \text{fst}(\text{?z})$  by simp_all
  ultimately

```

```

  show  $\exists z \in r. v = \text{snd}(z) \wedge u = \text{fst}(z)$ 
  using rexI[where  $x = \langle u, v \rangle$ ] by force
qed

```

```

lemma converse_eq_aux :
  assumes  $\langle 0, 0 \rangle \in r$ 
  shows  $\text{converse}(r) = \{ \langle \text{snd}(x), \text{fst}(x) \rangle . x \in r \}$ 
  using converse_subset
proof(intro equalityI subsetI, auto)
  fix z
  assume  $z \in r$ 
  then show  $\langle \text{fst}(z), \text{snd}(z) \rangle \in r$ 
  proof(cases  $\exists a b . z = \langle a, b \rangle$ )
    case True
    with  $\langle z \in r \rangle$ 
    show ?thesis by auto
  next
    case False
    then
    have  $\text{fst}(z) = 0 \wedge \text{snd}(z) = 0$ 
      unfolding fst_def snd_def by auto
    with  $\langle z \in r \rangle$  assms
    show ?thesis by auto
  qed
qed

```

```

lemma converse_eq_aux' :
  assumes  $\langle 0, 0 \rangle \notin r$ 
  shows  $\text{converse}(r) = \{ \langle \text{snd}(x), \text{fst}(x) \rangle . x \in r \} - \{ \langle 0, 0 \rangle \}$ 
  using converse_subset assms
proof(intro equalityI subsetI, auto)
  fix z
  assume  $z \in r \wedge \text{snd}(z) \neq 0$ 
  then
  obtain a b where  $z = \langle a, b \rangle$  unfolding snd_def by force
  with  $\langle z \in r \rangle$ 
  show  $\langle \text{fst}(z), \text{snd}(z) \rangle \in r$ 
    by auto
  next
  fix z
  assume  $z \in r \wedge \text{fst}(z) \neq 0$ 
  then
  obtain a b where  $z = \langle a, b \rangle$  unfolding fst_def by force
  with  $\langle z \in r \rangle$ 
  show  $\langle \text{fst}(z), \text{snd}(z) \rangle \in r$ 
    by auto
  qed

```

```

lemma diff_un :  $b \subseteq a \implies (a - b) \cup b = a$ 

```



```

by auto

lemma converse_eq: converse(r) = ({⟨snd(x),fst(x)⟩ . x∈r} - {<0,0>}) ∪ (r ∩ {<0,0>})
proof(cases <0,0>∈r)
  case True
  then
  have converse(r) = {⟨snd(x),fst(x)⟩ . x∈r}
    using converse_eq_aux by auto
  moreover
  from True
  have r ∩ {<0,0>} = {<0,0>} {<0,0>} ⊆ {⟨snd(x),fst(x)⟩ . x∈r}
    using converse_subset by auto
  moreover from this True
  have {⟨snd(x),fst(x)⟩ . x∈r} = ({⟨snd(x),fst(x)⟩ . x∈r} - {<0,0>}) ∪ ({<0,0>})
    using diff_un[of {<0,0>},symmetric] converse_eq_aux by auto
  ultimately
  show ?thesis
    by simp
next
  case False
  then
  have r ∩ {<0,0>} = 0 by auto
  then
  have ({⟨snd(x),fst(x)⟩ . x∈r} - {<0,0>}) ∪ (r ∩ {<0,0>}) = ({⟨snd(x),fst(x)⟩ .
x∈r} - {<0,0>})
    by simp
  with False
  show ?thesis
    using converse_eq_aux' by auto
qed

lemma range_subset : range(r) ⊆ {snd(x). x∈r}
  unfolding range_def domain_def converse_def
proof(intro subsetI, auto)
  fix u v
  assume ⟨u,v⟩∈r (is ?z∈r)
  moreover
  have v=snd(?z) u=fst(?z) by simp_all
  ultimately
  show ∃ z∈r. v=snd(z)
    using rexI[where x=v] by force
qed

lemma lam_replacement_imp_strong_replacement_aux:
  assumes lam_replacement(M, b) ∀ x[M]. M(b(x))
  shows strong_replacement(M, λx y. y = b(x))
proof -
  {
    fix A

```

```

    note assms
    moreover
    assume  $M(A)$ 
    moreover from calculation
    have  $M(\lambda x \in A. b(x))$  using lam_replacement_iff_lam_closed by auto
    ultimately
    have  $M((\lambda x \in A. b(x)) \text{``} A \ \forall z[M]. z \in (\lambda x \in A. b(x)) \text{``} A \longleftrightarrow (\exists x \in A. z = b(x)))$ 
      by (auto simp: lam_def)
  }
  then
  show ?thesis unfolding strong_replacement_def
    by clarsimp (rule_tac  $x = (\lambda x \in A. b(x)) \text{``} A$  in rexI, auto)
qed

lemma lam_replacement_imp_RepFun_Lam:
  assumes lam_replacement( $M, f$ )  $M(A)$ 
  shows  $M(\{y . x \in A, M(y) \wedge y = \langle x, f(x) \rangle\})$ 
proof -
  from assms
  obtain  $Y$  where  $1: M(Y) \ \forall b[M]. b \in Y \longleftrightarrow (\exists x[M]. x \in A \wedge b = \langle x, f(x) \rangle)$ 
    unfolding lam_replacement_def strong_replacement_def
    by auto
  moreover from calculation
  have  $Y = \{y . x \in A, M(y) \wedge y = \langle x, f(x) \rangle\}$  (is  $Y = ?R$ )
  proof (intro equalityI subsetI)
    fix  $y$ 
    assume  $y \in Y$ 
    moreover from this 1
    obtain  $x$  where  $x \in A \ y = \langle x, f(x) \rangle \ M(y)$ 
      using transM[OF _ ‹M(Y)›] by auto
    ultimately
    show  $y \in ?R$ 
      by auto
  next
    fix  $z$ 
    assume  $z \in ?R$ 
    moreover from this
    obtain  $a$  where  $a \in A \ z = \langle a, f(a) \rangle \ M(a) \ M(f(a))$ 
      using transM[OF _ ‹M(A)›]
      by auto
    ultimately
    show  $z \in Y$  using 1 by simp
  qed
  ultimately
  show ?thesis by auto
qed

lemma lam_closed_imp_closed:
  assumes  $\forall A[M]. M(\lambda x \in A. f(x))$ 

```

```

shows  $\forall x[M]. M(f(x))$ 
proof
  fix x
  assume  $M(x)$ 
  moreover from this and assms
  have  $M(\lambda x \in \{x\}. f(x))$  by simp
  ultimately
  show  $M(f(x))$ 
    using image_lam[of  $\{x\}$   $\{x\}$   $f$ ]
    image_closed[of  $\{x\}$   $(\lambda x \in \{x\}. f(x))$ ] by (auto dest:transM)
qed

lemma lam_replacement_if:
  assumes lam_replacement( $M, f$ ) lam_replacement( $M, g$ ) separation( $M, b$ )
     $\forall x[M]. M(f(x)) \forall x[M]. M(g(x))$ 
  shows lam_replacement( $M, \lambda x. \text{if } b(x) \text{ then } f(x) \text{ else } g(x)$ )
proof -
  let  $?G = \lambda x. \text{if } b(x) \text{ then } f(x) \text{ else } g(x)$ 
  let  $?b = \lambda A. \{x \in A. b(x)\}$  and  $?b' = \lambda A. \{x \in A. \neg b(x)\}$ 
  have eq:  $(\lambda x \in A. ?G(x)) = (\lambda x \in ?b(A). f(x)) \cup (\lambda x \in ?b'(A). g(x))$  for  $A$ 
    unfolding lam_def by auto
  have  $?b'(A) = A - ?b(A)$  for  $A$  by auto
  moreover
  have  $M(?b(A))$  if  $M(A)$  for  $A$  using assms that by simp
  moreover from calculation
  have  $M(?b'(A))$  if  $M(A)$  for  $A$  using that by simp
  moreover from calculation assms
  have  $M(\lambda x \in ?b(A). f(x))$   $M(\lambda x \in ?b'(A). g(x))$  if  $M(A)$  for  $A$ 
    using lam_replacement_iff_lam_closed that
    by simp_all
  moreover from this
  have  $M((\lambda x \in ?b(A). f(x)) \cup (\lambda x \in ?b'(A). g(x)))$  if  $M(A)$  for  $A$ 
    using that by simp
  ultimately
  have  $M(\lambda x \in A. \text{if } b(x) \text{ then } f(x) \text{ else } g(x))$  if  $M(A)$  for  $A$ 
    using that eq by simp
  with assms
  show thesis using lam_replacement_iff_lam_closed by simp
qed

lemma lam_replacement_constant:  $M(b) \implies \text{lam\_replacement}(M, \lambda \_. b)$ 
  unfolding lam_replacement_def strong_replacement_def
  by safe (rule_tac  $x = \_ \times \{b\}$  in rexI; blast)

```

13.1 Replacement instances obtained through Powerset

The next few lemmas provide bounds for certain constructions.

```

lemma not_functional_Replace_0:
  assumes  $\neg(\forall y y'. P(y) \wedge P(y') \longrightarrow y=y')$ 

```

```

shows  $\{y . x \in A, P(y)\} = 0$ 
using assms by (blast elim!: ReplaceE)

lemma Replace_in_Pow_rel:
  assumes  $\bigwedge x b. x \in A \implies P(x,b) \implies b \in U \ \forall x \in A. \forall y y'. P(x,y) \wedge P(x,y') \longrightarrow$ 
 $y=y'$ 
    separation(M,  $\lambda y. \exists x[M]. x \in A \wedge P(x, y)$ )
    M(U) M(A)
  shows  $\{y . x \in A, P(x, y)\} \in \text{Pow}^M(U)$ 
proof -
  from assms
  have  $\{y . x \in A, P(x, y)\} \subseteq U$ 
     $z \in \{y . x \in A, P(x, y)\} \implies M(z)$  for z
    by (auto dest:transM)
  with assms
  have  $\{y . x \in A, P(x, y)\} = \{y \in U . \exists x[M]. x \in A \wedge P(x,y)\}$ 
    by (intro equalityI) (auto, blast)
  with assms
  have M( $\{y . x \in A, P(x, y)\}$ )
    by simp
  with assms
  show ?thesis
    using mem_Pow_rel_abs by auto
qed

lemma Replace_sing_0_in_Pow_rel:
  assumes  $\bigwedge b. P(b) \implies b \in U$ 
    separation(M,  $\lambda y. P(y)$ ) M(U)
  shows  $\{y . x \in \{0\}, P(y)\} \in \text{Pow}^M(U)$ 
proof (cases  $\forall y y'. P(y) \wedge P(y') \longrightarrow y=y'$ )
  case True
  with assms
  show ?thesis by (rule_tac Replace_in_Pow_rel) auto
next
  case False
  with assms
  show ?thesis
    using nonempty not_functional_Replace_0[of P {0}] Pow_rel_char by auto
qed

lemma The_in_Pow_rel_Union:
  assumes  $\bigwedge b. P(b) \implies b \in U$  separation(M,  $\lambda y. P(y)$ ) M(U)
  shows  $(\text{THE } i. P(i)) \in \text{Pow}^M(\bigcup U)$ 
proof -
  note assms
  moreover from this
  have  $(\text{THE } i. P(i)) \in \text{Pow}(\bigcup U)$ 
    unfolding the_def by auto
  moreover from assms

```

```

have M(THE i. P(i))
  using Replace_sing_0_in_Pow_rel[of P U] unfolding the_def
  by (auto dest:transM)
ultimately
show ?thesis
  using Pow_rel_char by auto
qed

lemma separation_least: separation(M,  $\lambda y. \text{Ord}(y) \wedge P(y) \wedge (\forall j. j < y \longrightarrow \neg P(j))$ )
unfolding separation_def
proof
  fix z
  assume M(z)
  have M( $\{x \in z. x \in z \wedge \text{Ord}(x) \wedge P(x) \wedge (\forall j. j < x \longrightarrow \neg P(j))\}$ )
    (is M(?y))
  proof (cases  $\exists x \in z. \text{Ord}(x) \wedge P(x) \wedge (\forall j. j < x \longrightarrow \neg P(j))$ )
    case True
      with  $\langle M(z) \rangle$ 
      have  $\exists x[M]. ?y = \{x\}$ 
        by (safe, rename_tac x, rule_tac x=x in rexI)
        (auto dest:transM, intro equalityI, auto elim:Ord_linear_lt)
      then
      show ?thesis
        by auto
    next
      case False
      then
      have  $\{x \in z. x \in z \wedge \text{Ord}(x) \wedge P(x) \wedge (\forall j. j < x \longrightarrow \neg P(j))\} = 0$ 
        by auto
      then
      show ?thesis by auto
  qed
moreover from this
  have  $\forall x[M]. x \in ?y \longleftrightarrow x \in z \wedge \text{Ord}(x) \wedge P(x) \wedge (\forall j. j < x \longrightarrow \neg P(j))$  by
    simp
  ultimately
  show  $\exists y[M]. \forall x[M]. x \in y \longleftrightarrow x \in z \wedge \text{Ord}(x) \wedge P(x) \wedge (\forall j. j < x \longrightarrow \neg P(j))$ 
    by blast
qed

lemma Least_in_Pow_rel_Union:
  assumes  $\bigwedge b. P(b) \implies b \in U$ 
  M(U)
  shows  $(\mu i. P(i)) \in \text{Pow}^M(\bigcup U)$ 
  using assms separation_least unfolding Least_def
  by (rule_tac The_in_Pow_rel_Union) simp

```

```

lemma bounded_lam_replacement:
  fixes  $U$ 
  assumes  $\forall X[M]. \forall x \in X. f(x) \in U(X)$ 
    and separation_f:  $\forall A[M]. \text{separation}(M, \lambda y. \exists x[M]. x \in A \wedge y = \langle x, f(x) \rangle)$ 
    and U_closed [intro, simp]:  $\bigwedge X. M(X) \implies M(U(X))$ 
  shows lam_replacement( $M, f$ )
proof -
  have  $M(\lambda x \in A. f(x))$  if  $M(A)$  for  $A$ 
  proof -
    have  $(\lambda x \in A. f(x)) = \{y \in \text{Pow}^M(\text{Pow}^M(A \cup U(A))). \exists x[M]. x \in A \wedge y = \langle x, f(x) \rangle\}$ 
    using  $\langle M(A) \rangle$  unfolding lam_def
  proof (intro equalityI, auto)
    fix  $x$ 
    assume  $x \in A$ 
    moreover
    note  $\langle M(A) \rangle$ 
    moreover from calculation assms
    have  $f(x) \in U(A)$  by simp
    moreover from calculation
    have  $\{x, f(x)\} \in \text{Pow}^M(A \cup U(A))$   $\{x, x\} \in \text{Pow}^M(A \cup U(A))$ 
      using Pow_rel_char[of  $A \cup U(A)$ ] by (auto dest:transM)
    ultimately
    show  $\langle x, f(x) \rangle \in \text{Pow}^M(\text{Pow}^M(A \cup U(A)))$ 
      using Pow_rel_char[of  $\text{Pow}^M(A \cup U(A))$ ] unfolding Pair_def
      by (auto dest:transM)
  qed
  moreover from  $\langle M(A) \rangle$ 
  have  $M(\{y \in \text{Pow}^M(\text{Pow}^M(A \cup U(A))). \exists x[M]. x \in A \wedge y = \langle x, f(x) \rangle\})$ 
    using separation_f
    by (rule_tac separation_closed) simp_all
  ultimately
  show ?thesis
    by simp
qed
  moreover from this
  have  $\forall x[M]. M(f(x))$ 
    using lam_closed_imp_closed by simp
  ultimately
  show ?thesis
    using assms
    by (rule_tac lam_replacement_iff_lam_closed[THEN iffD2]) simp_all
qed

lemma lam_replacement_domain':
  assumes  $\forall A[M]. \text{separation}(M, \lambda y. \exists x \in A. y = \langle x, \text{domain}(x) \rangle)$ 
  shows lam_replacement( $M, \text{domain}$ )
proof -
  have  $\forall x \in X. \text{domain}(x) \in \text{Pow}^M(\bigcup \bigcup X)$  if  $M(X)$  for  $X$ 

```

```

proof
  fix  $x$ 
  assume  $x \in X$ 
  moreover
  note  $\langle M(X) \rangle$ 
  moreover from calculation
  have  $M(x)$  by (auto dest:transM)
  ultimately
  show  $\text{domain}(x) \in \text{Pow}^M(\bigcup \bigcup \bigcup X)$ 
    by(rule_tac mem_Pow_rel_abs[of domain(x)  $\bigcup \bigcup \bigcup X$ , THEN iffD2], auto
simp:Pair_def,force)
  qed
  with assms
  show ?thesis
    using bounded_lam_replacement[of domain  $\lambda X. \text{Pow}^M(\bigcup \bigcup \bigcup X)$ ] by simp
qed

```

— Below we assume the replacement instance for *fst*. Alternatively it follows from the instance of separation assumed in this lemma.

```

lemma lam_replacement_fst':
  assumes  $\forall A[M]. \text{separation}(M, \lambda y. \exists x \in A. y = \langle x, \text{fst}(x) \rangle)$ 
  shows lam_replacement(M,fst)
proof -
  have  $\forall x \in X. \text{fst}(x) \in \{0\} \cup \bigcup \bigcup X$  if  $M(X)$  for  $X$ 
  proof
    fix  $x$ 
    assume  $x \in X$ 
    moreover
    note  $\langle M(X) \rangle$ 
    moreover from calculation
    have  $M(x)$  by (auto dest:transM)
    ultimately
    show  $\text{fst}(x) \in \{0\} \cup \bigcup \bigcup X$  unfolding fst_def Pair_def
      by (auto, rule_tac [1] the_0) force— tricky! And slow. It doesn't work for
snd
  qed
  with assms
  show ?thesis
    using bounded_lam_replacement[of fst  $\lambda X. \{0\} \cup \bigcup \bigcup X$ ] by simp
qed

```

```

lemma lam_replacement_restrict:
  assumes  $\forall A[M]. \text{separation}(M, \lambda y. \exists x \in A. y = \langle x, \text{restrict}(x,B) \rangle)$   $M(B)$ 
  shows lam_replacement(M,  $\lambda r. \text{restrict}(r,B)$ )
proof -
  have  $\forall r \in R. \text{restrict}(r,B) \in \text{Pow}^M(\bigcup R)$  if  $M(R)$  for  $R$ 
  proof -
    {
      fix  $r$ 

```

```

    assume  $r \in R$ 
    with  $\langle M(B) \rangle$ 
    have  $\text{restrict}(r, B) \in \text{Pow}(\bigcup R) \ M(\text{restrict}(r, B))$ 
      using  $\text{Union\_upper\_subset\_Pow\_Union\_subset\_trans}[OF \ \text{restrict\_subset}]$ 
      transM[ $OF \ \langle M(R) \rangle$ ]
      by simp_all
  } then show ?thesis
    using Pow_rel_char that by simp
qed
with assms
show ?thesis
  using bounded_lam_replacement[of  $\lambda r . \text{restrict}(r, B) \ \lambda X. \text{Pow}^M(\bigcup X)$ ]
  by simp
qed

end —  $M\_basic$ 

locale  $M\_replacement = M\_basic +$ 
  assumes
    lam_replacement_domain:  $\text{lam\_replacement}(M, \text{domain})$ 
  and
    lam_replacement_fst:  $\text{lam\_replacement}(M, \text{fst})$ 
  and
    lam_replacement_snd:  $\text{lam\_replacement}(M, \text{snd})$ 
  and
    lam_replacement_Union:  $\text{lam\_replacement}(M, \text{Union})$ 
  and
    middle_separation:  $\text{separation}(M, \lambda x. \text{snd}(\text{fst}(x)) = \text{fst}(\text{snd}(x)))$ 
  and
    middle_del_replacement:  $\text{strong\_replacement}(M, \lambda x y. y = \langle \text{fst}(\text{fst}(x)), \text{snd}(\text{snd}(x)) \rangle)$ 
  and
    product_replacement:
       $\text{strong\_replacement}(M, \lambda x y. y = \langle \text{snd}(\text{fst}(x)), \langle \text{fst}(\text{fst}(x)), \text{snd}(\text{snd}(x)) \rangle \rangle)$ 
  and
    lam_replacement_Upair:  $\text{lam\_replacement}(M, \lambda p. \text{Upair}(\text{fst}(p), \text{snd}(p)))$ 
  and
    lam_replacement_Diff:  $\text{lam\_replacement}(M, \lambda p. \text{fst}(p) - \text{snd}(p))$ 
  and
    lam_replacement_Image:  $\text{lam\_replacement}(M, \lambda p. \text{fst}(p) \text{ `` } \text{snd}(p))$ 
  and
    separation_fst_in_snd:  $\text{separation}(M, \lambda y. \text{fst}(\text{snd}(y)) \in \text{snd}(\text{snd}(y)))$ 
  and
    lam_replacement_converse :  $\text{lam\_replacement}(M, \text{converse})$ 
  and
    lam_replacement_comp:  $\text{lam\_replacement}(M, \lambda x. \text{fst}(x) \ O \ \text{snd}(x))$ 
begin

lemma  $\text{lam\_replacement\_imp\_strong\_replacement}$ :
  assumes  $\text{lam\_replacement}(M, f)$ 

```



```

shows strong_replacement(M,  $\lambda x y. y = f(x)$ )
proof -
{
  fix A
  assume M(A)
  moreover from calculation assms
  obtain Y where 1:M(Y)  $\forall b[M]. b \in Y \longleftrightarrow (\exists x[M]. x \in A \wedge b = \langle x, f(x) \rangle)$ 
    unfolding lam_replacement_def strong_replacement_def
    by auto
  moreover from this
  have M( $\{snd(b) . b \in Y\}$ )
  using transM[OF_  $\langle M(Y) \rangle$ ] lam_replacement_snd lam_replacement_imp_strong_replacement_aux
    RepFun_closed by simp
  moreover
  have  $\{snd(b) . b \in Y\} = \{y . x \in A, M(f(x)) \wedge y = f(x)\}$  (is ?L=?R)
  proof(intro equalityI subsetI)
    fix x
    assume  $x \in ?L$ 
    moreover from this
    obtain b where  $b \in Y$   $x = snd(b)$  M(b)
      using transM[OF_  $\langle M(Y) \rangle$ ] by auto
    moreover from this 1
    obtain a where  $a \in A$   $b = \langle a, f(a) \rangle$  by auto
    moreover from calculation
    have  $x = f(a)$  by simp
    ultimately show  $x \in ?R$ 
      by auto
  next
    fix z
    assume  $z \in ?R$ 
    moreover from this
    obtain a where  $a \in A$   $z = f(a)$  M(a) M(f(a))
      using transM[OF_  $\langle M(A) \rangle$ ]
      by auto
    moreover from calculation this 1
    have  $z = snd(\langle a, f(a) \rangle)$   $\langle a, f(a) \rangle \in Y$  by auto
    ultimately
    show  $z \in ?L$  by force
  qed
  ultimately
  have  $\exists Z[M]. \forall z[M]. z \in Z \longleftrightarrow (\exists a[M]. a \in A \wedge z = f(a))$ 
    by (rule_tac rexI[where  $x = \{snd(b) . b \in Y\}$ ], auto)
}
then
show ?thesis unfolding strong_replacement_def by simp
qed

```

lemma *Collect_middle*: $\{p \in (\lambda x \in A. f(x)) \times (\lambda x \in \{f(x) . x \in A\}. g(x)) . snd(fst(p)) = fst(snd(p))\}$
 $= \{ \langle \langle x, f(x) \rangle, \langle f(x), g(f(x)) \rangle \rangle . x \in A \}$

```

by (intro equalityI; auto simp:lam_def)

lemma RepFun_middle_del: { ⟨fst(fst(p)),snd(snd(p))⟩ . p ∈ { ⟨⟨x,f(x)⟩,f(x),g(f(x))⟩⟩
. x∈A } }
    = { ⟨x,g(f(x))⟩ . x∈A }
by auto

lemma lam_replacement_imp_RepFun:
  assumes lam_replacement(M, f) M(A)
  shows M({y . x∈A , M(y) ∧ y=f(x)})
proof -
  from assms
  obtain Y where 1:M(Y) ∀ b[M]. b ∈ Y ⟷ (∃ x[M]. x ∈ A ∧ b = ⟨x,f(x)⟩)
    unfolding lam_replacement_def strong_replacement_def
    by auto
  moreover from this
  have M({snd(b) . b ∈ Y})
    using transM[OF _ ⟨M(Y)⟩] lam_replacement_snd lam_replacement_imp_strong_replacement_aux
    RepFun_closed by simp
  moreover
  have {snd(b) . b ∈ Y} = {y . x∈A , M(y) ∧ y=f(x)} (is ?L=?R)
  proof(intro equalityI subsetI)
    fix x
    assume x∈?L
    moreover from this
    obtain b where b∈Y x=snd(b) M(b)
      using transM[OF _ ⟨M(Y)⟩] by auto
    moreover from this 1
    obtain a where a∈A b=⟨a,f(a)⟩ by auto
    moreover from calculation
    have x=f(a) by simp
    ultimately show x∈?R
      by auto
  next
    fix z
    assume z∈?R
    moreover from this
    obtain a where a∈A z=f(a) M(a) M(f(a))
      using transM[OF _ ⟨M(A)⟩]
      by auto
    moreover from calculation this 1
    have z=snd(⟨a,f(a)⟩) ⟨a,f(a)⟩ ∈ Y by auto
    ultimately
    show z∈?L by force
  qed
  ultimately
  show ?thesis by simp
qed

```

```

lemma lam_replacement_product:
  assumes lam_replacement(M,f) lam_replacement(M,g)
  shows lam_replacement(M,  $\lambda x. \langle f(x), g(x) \rangle$ )
proof -
  {
    fix A
    let ?Y = {y . x ∈ A , M(y) ∧ y = f(x)}
    let ?Y' = {y . x ∈ A , M(y) ∧ y = ⟨x, f(x)⟩}
    let ?Z = {y . x ∈ A , M(y) ∧ y = g(x)}
    let ?Z' = {y . x ∈ A , M(y) ∧ y = ⟨x, g(x)⟩}
    have x ∈ C ⇒ y ∈ C ⇒ fst(x) = fst(y) → M(fst(y)) ∧ M(snd(x)) ∧ M(snd(y))
  if M(C) for C y x
    using transM[OF _ that] by auto
    moreover
    note assms
    moreover
    assume M(A)
    moreover from ⟨M(A)⟩ assms(1)
    have M(converse(?Y)) M(?Y)
      using lam_replacement_imp_RepFun_Lam lam_replacement_imp_RepFun
  by auto
    moreover from calculation
    have M(?Z) M(?Z')
      using lam_replacement_imp_RepFun_Lam lam_replacement_imp_RepFun
  by auto
    moreover from calculation
    have M(converse(?Y') × ?Z')
      by simp
    moreover from this
    have M({p ∈ converse(?Y') × ?Z' . snd(fst(p)) = fst(snd(p))}) (is M(?P))
      using middle_separation by simp
    moreover from calculation
    have M({ ⟨snd(fst(p)), fst(fst(p)), snd(snd(p))⟩ . p ∈ ?P }) (is M(?R))
      using RepFun_closed[OF product_replacement ⟨M(?P)⟩] by simp
    ultimately
    have b ∈ ?R ⇔ (∃ x[M]. x ∈ A ∧ b = ⟨x, ⟨f(x), g(x)⟩⟩) if M(b) for b
      using that
      apply(intro iffI) apply(auto)[1]
  proof -
    assume ∃ x[M]. x ∈ A ∧ b = ⟨x, f(x), g(x)⟩
    moreover from this
    obtain x where M(x) x ∈ A b = ⟨x, ⟨f(x), g(x)⟩⟩
      by auto
    moreover from calculation that
    have M(⟨x, f(x)⟩) M(⟨x, g(x)⟩) by auto
    moreover from calculation
    have ⟨f(x), x⟩ ∈ converse(?Y') ⟨x, g(x)⟩ ∈ ?Z' by auto
    moreover from calculation
    have ⟨⟨f(x), x⟩, ⟨x, g(x)⟩⟩ ∈ converse(?Y') × ?Z' by auto
  
```

```

moreover from calculation
have  $\langle \langle f(x), x \rangle, \langle x, g(x) \rangle \rangle \in ?P$ 
  (is  $?p \in ?P$ )
  by auto
moreover from calculation
have  $b = \langle \text{snd}(\text{fst}(?p)), \langle \text{fst}(\text{fst}(?p)), \text{snd}(\text{snd}(?p)) \rangle \rangle$  by auto
moreover from calculation
have  $\langle \text{snd}(\text{fst}(?p)), \langle \text{fst}(\text{fst}(?p)), \text{snd}(\text{snd}(?p)) \rangle \rangle \in ?R$ 
  by (rule_tac RepFunI [of  $?p$   $?P$ ], simp)
ultimately show  $b \in ?R$  by simp
qed
with  $\langle M(?R) \rangle$ 
have  $\exists Y[M]. \forall b[M]. b \in Y \longleftrightarrow (\exists x[M]. x \in A \wedge b = \langle x, \langle f(x), g(x) \rangle \rangle)$ 
  by (rule_tac rexI [where  $x = ?R$ ], simp_all)
}
with assms
show ?thesis using lam_replacement_def strong_replacement_def by simp
qed

lemma lam_replacement_hcomp:
  assumes lam_replacement(M, f) lam_replacement(M, g)  $\forall x[M]. M(f(x))$ 
  shows lam_replacement(M,  $\lambda x. g(f(x))$ )
proof -
{
  fix A
  let  $?Y = \{y . x \in A, y = f(x)\}$ 
  let  $?Y' = \{y . x \in A, y = \langle x, f(x) \rangle\}$ 
  have  $\forall x \in C. M(\langle \text{fst}(\text{fst}(x)), \text{snd}(\text{snd}(x)) \rangle)$  if  $M(C)$  for C
    using transM [OF  $\_\text{that}$ ] by auto
  moreover
  note assms
  moreover
  assume  $M(A)$ 
  moreover from assms
  have  $\text{eq}: ?Y = \{y . x \in A, M(y) \wedge y = f(x)\}$   $?Y' = \{y . x \in A, M(y) \wedge y = \langle x, f(x) \rangle\}$ 
    using transM [OF  $\_\langle M(A) \rangle$ ] by auto
  moreover from  $\langle M(A) \rangle$  assms(1)
  have  $M(?Y') \ M(?Y)$ 
    using lam_replacement_imp_RepFun_Lam lam_replacement_imp_RepFun
  eq by auto
  moreover from calculation
  have  $M(\{z . y \in ?Y, M(z) \wedge z = \langle y, g(y) \rangle\})$  (is  $M(?Z)$ )
    using lam_replacement_imp_RepFun_Lam by auto
  moreover from calculation
  have  $M(?Y' \times ?Z)$ 
    by simp
  moreover from this
  have  $M(\{p \in ?Y' \times ?Z . \text{snd}(\text{fst}(p)) = \text{fst}(\text{snd}(p))\})$  (is  $M(?P)$ )
    using middle_separation by simp

```

```

moreover from calculation
have  $M(\{ \langle fst(fst(p)), snd(snd(p)) \rangle . p \in ?P \})$  (is  $M(?R)$ )
  using  $RepFun\_closed[OF middle\_del\_replacement \langle M(?P) \rangle]$  by simp
ultimately
have  $b \in ?R \longleftrightarrow (\exists x[M]. x \in A \wedge b = \langle x, g(f(x)) \rangle)$  if  $M(b)$  for  $b$ 
  using  $that\ assms(\mathcal{I})$ 
  apply( $intro\ iffI$ ) apply( $auto$ )[1]
proof -
  assume  $\exists x[M]. x \in A \wedge b = \langle x, g(f(x)) \rangle$ 
  moreover from this
  obtain  $x$  where  $M(x) \ x \in A \ b = \langle x, g(f(x)) \rangle$ 
    by auto
  moreover from calculation that assms( $\mathcal{I}$ )
  have  $M(f(x)) \ M(g(f(x)))$  by auto
  moreover from calculation
  have  $\langle x, f(x) \rangle \in ?Y'$  by auto
  moreover from calculation
  have  $\langle f(x), g(f(x)) \rangle \in ?Z$  by auto
  moreover from calculation
  have  $\langle \langle x, f(x) \rangle, \langle f(x), g(f(x)) \rangle \rangle \in ?P$ 
    (is  $?p \in ?P$ )
    by auto
  moreover from calculation
  have  $b = \langle fst(fst(?p)), snd(snd(?p)) \rangle$  by auto
  moreover from calculation
  have  $\langle fst(fst(?p)), snd(snd(?p)) \rangle \in ?R$ 
    by( $rule\_tac\ RepFunI[of\ ?p\ ?P],\ simp$ )
  ultimately show  $b \in ?R$  by simp
qed
with  $\langle M(?R) \rangle$ 
have  $\exists Y[M]. \forall b[M]. b \in Y \longleftrightarrow (\exists x[M]. x \in A \wedge b = \langle x, g(f(x)) \rangle)$ 
  by ( $rule\_tac\ rexI[where\ x=?R],\ simp\_all$ )
}
with assms
show  $?thesis$  using  $lam\_replacement\_def\ strong\_replacement\_def$  by simp
qed

```

```

lemma  $lam\_replacement\_Collect :$ 
  assumes  $M(A) \ \forall x[M].\ separation(M, F(x))$ 
     $separation(M, \lambda p . \forall x \in A. x \in snd(p) \longleftrightarrow F(fst(p), x))$ 
  shows  $lam\_replacement(M, \lambda x. \{y \in A . F(x, y)\})$ 
proof -
  {
    fix  $Z$ 
    let  $?Y = \lambda z. \{x \in A . F(z, x)\}$ 
    assume  $M(Z)$ 
    moreover from this
    have  $M(?Y(z))$  if  $z \in Z$  for  $z$ 
      using  $assms\ that\ transM[of\ _\ Z]$  by simp
  }

```

```

moreover from this
have  $?Y(z) \in Pow^M(A)$  if  $z \in Z$  for  $z$ 
  using Pow_rel_char that assms by auto
moreover from calculation  $\langle M(A) \rangle$ 
have  $M(Z \times Pow^M(A))$  by simp
moreover from this
have  $M(\{p \in Z \times Pow^M(A) . \forall x \in A. x \in snd(p) \longleftrightarrow F(fst(p), x)\})$  (is  $M(?P)$ )
  using assms by simp
ultimately
have  $b \in ?P \longleftrightarrow (\exists z[M]. z \in Z \wedge b = \langle z, ?Y(z) \rangle)$  if  $M(b)$  for  $b$ 
  using assms(1) Pow_rel_char[OF  $\langle M(A) \rangle$ ] that
  by(intro iffI,auto,intro equalityI,auto)
with  $\langle M(?P) \rangle$ 
have  $\exists Y[M]. \forall b[M]. b \in Y \longleftrightarrow (\exists z[M]. z \in Z \wedge b = \langle z, ?Y(z) \rangle)$ 
  by (rule_tac rexI[where  $x = ?P$ ],simp_all)
}
then
show ?thesis
  unfolding lam_replacement_def strong_replacement_def
  by simp
qed

```

```

lemma lam_replacement_hcomp2:
assumes lam_replacement( $M, f$ ) lam_replacement( $M, g$ )
   $\forall x[M]. M(f(x)) \forall x[M]. M(g(x))$ 
  lam_replacement( $M, \lambda p. h(fst(p), snd(p))$ )
   $\forall x[M]. \forall y[M]. M(h(x, y))$ 
shows lam_replacement( $M, \lambda x. h(f(x), g(x))$ )
using assms lam_replacement_product[of  $f\ g$ ]
  lam_replacement_hcomp[of  $\lambda x. \langle f(x), g(x) \rangle \lambda \langle x, y \rangle. h(x, y)$ ]
unfolding split_def by simp

```

```

lemma lam_replacement_identity: lam_replacement( $M, \lambda x. x$ )
proof -
{
  fix  $A$ 
  assume  $M(A)$ 
  moreover from this
  have  $id(A) = \{\langle snd(fst(z)), fst(snd(z)) \rangle . z \in \{z \in (A \times A) \times (A \times A). snd(fst(z)) = fst(snd(z))\}\}$ 
  unfolding id_def lam_def
  by(intro equalityI subsetI,simp_all,auto)
  moreover from calculation
  have  $M(\{z \in (A \times A) \times (A \times A). snd(fst(z)) = fst(snd(z))\})$  (is  $M(?A')$ )
  using middle_separation by simp
  moreover from calculation
  have  $M(\{\langle snd(fst(z)), fst(snd(z)) \rangle . z \in ?A'\})$ 
  using transM[of  $\_ A$ ]
  lam_replacement_product lam_replacement_hcomp lam_replacement_fst

```

```

lam_replacement_snd
  lam_replacement_imp_strong_replacement[THEN RepFun_closed]
  by simp_all
ultimately
  have  $M(id(A))$  by simp
}
then
  show ?thesis using lam_replacement_iff_lam_closed
  unfolding id_def by simp
qed

lemma lam_replacement_vimage :
  shows lam_replacement( $M, \lambda x. fst(x) \cdot "snd(x)"$ )
  unfolding vimage_def using
    lam_replacement_hcomp2[OF
      lam_replacement_hcomp[OF lam_replacement_fst lam_replacement_converse]
    ]
  lam_replacement_snd
  — — lam_replacement_Image]
  by auto

lemma strong_replacement_separation_aux :
  assumes strong_replacement( $M, \lambda x y. y=f(x)$ ) separation( $M, P$ )
  shows strong_replacement( $M, \lambda x y. P(x) \wedge y=f(x)$ )
proof -
  {
    fix A
    let ?Q= $\lambda X. \forall b[M]. b \in X \longleftrightarrow (\exists x[M]. x \in A \wedge P(x) \wedge b = f(x))$ 
    assume  $M(A)$ 
    moreover from this
    have  $M(\{x \in A . P(x)\})$  (is  $M(?B)$ ) using assms by simp
    moreover from calculation assms
    obtain Y where  $M(Y) \forall b[M]. b \in Y \longleftrightarrow (\exists x[M]. x \in ?B \wedge b = f(x))$ 
    unfolding strong_replacement_def by auto
    then
    have  $\exists Y[M]. \forall b[M]. b \in Y \longleftrightarrow (\exists x[M]. x \in A \wedge P(x) \wedge b = f(x))$ 
    using rexI[of ?Q _ M] by simp
  }
  then
  show ?thesis
  unfolding strong_replacement_def by simp
qed

lemma separation_in:
  assumes  $\forall x[M]. M(f(x))$  lam_replacement( $M, f$ )
   $\forall x[M]. M(g(x))$  lam_replacement( $M, g$ )
  shows separation( $M, \lambda x. f(x) \in g(x)$ )
proof -
  let ?Z= $\lambda A. \{\langle x, \langle f(x), g(x) \rangle \rangle. x \in A\}$ 
  have  $M(?Z(A))$  if  $M(A)$  for A

```

```

using assms lam_replacement_iff_lam_closed that
  lam_replacement_product[of f g]
unfolding lam_def
by auto
then
have  $M(\{u \in ?Z(A) . \text{fst}(\text{snd}(u)) \in \text{snd}(\text{snd}(u))\})$  (is  $M(?W(A))$ ) if  $M(A)$  for  $A$ 
  using that separationfst_in_snd assms
  by auto
then
have  $M(\{\text{fst}(u) . u \in ?W(A)\})$  if  $M(A)$  for  $A$ 
  using that lam_replacement_imp_strong_replacement[OF lam_replacement_fst, THEN
    RepFun_closed] fst_closed[OF transM]
  by auto
moreover
have  $\{x \in A. f(x) \in g(x)\} = \{\text{fst}(u) . u \in ?W(A)\}$  for  $A$ 
  by auto
ultimately
show ?thesis
  using separation_iff
  by auto
qed

lemma lam_replacement_swap:  $\text{lam\_replacement}(M, \lambda x. \langle \text{snd}(x), \text{fst}(x) \rangle)$ 
  using lam_replacement_fst lam_replacement_snd
  lam_replacement_product[of snd fst] by simp

lemma lam_replacement_range :  $\text{lam\_replacement}(M, \text{range})$ 
  unfolding range_def
  using lam_replacement_hcomp[OF lam_replacement_converse lam_replacement_domain]
  by auto

lemma separation_in_range :  $M(a) \implies \text{separation}(M, \lambda x. a \in \text{range}(x))$ 
  using lam_replacement_range lam_replacement_constant separation_in
  by auto

lemma separation_in_domain :  $M(a) \implies \text{separation}(M, \lambda x. a \in \text{domain}(x))$ 
  using lam_replacement_domain lam_replacement_constant separation_in
  by auto

lemma lam_replacement_separation :
  assumes  $\text{lam\_replacement}(M, f)$   $\text{separation}(M, P)$ 
  shows  $\text{strong\_replacement}(M, \lambda x y . P(x) \wedge y = \langle x, f(x) \rangle)$ 
  using strong_replacement_separation_aux assms
  unfolding lam_replacement_def
  by simp

lemmas strong_replacement_separation =
  strong_replacement_separation_aux[OF lam_replacement_imp_strong_replacement]

```



```

lemma id_closed:  $M(A) \implies M(id(A))$ 
  using lam_replacement_identity lam_replacement_iff_lam_closed
  unfolding id_def by simp

lemma relation_separation:  $separation(M, \lambda z. \exists x y. z = \langle x, y \rangle)$ 
  unfolding separation_def
proof (clarify)
  fix A
  assume  $M(A)$ 
  moreover from this
  have  $\{z \in A. \exists x y. z = \langle x, y \rangle\} = \{z \in A. \exists x \in domain(A). \exists y \in range(A). pair(M,$ 
 $x, y, z)\}$ 
    (is  $?rel = \_$ )
    by (intro equalityI, auto dest:transM)
    (intro bexI, auto dest:transM simp:Pair_def)
  moreover from calculation
  have  $M(?rel)$ 
    using cartprod_separation[THEN separation_closed, of domain(A) range(A)
 $A]$ 
    by simp
  ultimately
  show  $\exists y[M]. \forall x[M]. x \in y \longleftrightarrow x \in A \wedge (\exists w y. x = \langle w, y \rangle)$ 
    by (rule_tac  $x = \{z \in A. \exists x y. z = \langle x, y \rangle\}$  in rexI) auto
qed

lemma separation_pair:
  assumes  $separation(M, \lambda y. P(fst(y), snd(y)))$ 
  shows  $separation(M, \lambda y. \exists u v. y = \langle u, v \rangle \wedge P(u, v))$ 
  unfolding separation_def
proof(clarify)
  fix A
  assume  $M(A)$ 
  moreover from this
  have  $M(\{z \in A. \exists x y. z = \langle x, y \rangle\})$  (is  $M(?P)$ )
    using relation_separation by simp
  moreover from this assms
  have  $M(\{z \in ?P. P(fst(z), snd(z))\})$ 
    by(rule_tac separation_closed, simp_all)
  moreover
  have  $\{y \in A. \exists u v. y = \langle u, v \rangle \wedge P(u, v)\} = \{z \in ?P. P(fst(z), snd(z))\}$ 
    by(rule equalityI subsetI, auto)
  moreover from calculation
  have  $M(\{y \in A. \exists u v. y = \langle u, v \rangle \wedge P(u, v)\})$ 
    by simp
  ultimately
  show  $\exists y[M]. \forall x[M]. x \in y \longleftrightarrow x \in A \wedge (\exists w y. x = \langle w, y \rangle \wedge P(w, y))$ 
    by (rule_tac  $x = \{z \in A. \exists x y. z = \langle x, y \rangle \wedge P(x, y)\}$  in rexI) auto
qed

```

```

lemma lam_replacement_Pair:
  shows lam_replacement(M,  $\lambda x. \langle \text{fst}(x), \text{snd}(x) \rangle$ )
  unfolding lam_replacement_def strong_replacement_def
proof (clarsimp)
  fix A
  assume M(A)
  then
  show  $\exists Y[M]. \forall b[M]. b \in Y \longleftrightarrow (\exists x \in A. b = \langle x, \text{fst}(x), \text{snd}(x) \rangle)$ 
    unfolding lam_replacement_def strong_replacement_def
  proof (cases relation(A))
    case True
    with  $\langle M(A) \rangle$ 
    show ?thesis
      using id_closed unfolding relation_def
      by (rule_tac x=id(A) in rexI) auto
    next
    case False
    moreover
    note  $\langle M(A) \rangle$ 
    moreover from this
    have  $M(\{z \in A. \exists x y. z = \langle x, y \rangle\})$  (is  $M(?rel)$ )
      using relation_separation by auto
    moreover
    have  $z = \langle \text{fst}(z), \text{snd}(z) \rangle$  if  $\text{fst}(z) \neq 0 \vee \text{snd}(z) \neq 0$  for  $z$ 
      using that
      by (cases  $\exists a b. z = \langle a, b \rangle$ ) (auto simp add: the_0 fst_def snd_def)
    ultimately
    show ?thesis
      using id_closed unfolding relation_def
      by (rule_tac x=id(?rel)  $\cup (A - ?rel) \times \{0\} \times \{0\}$  in rexI)
        (force simp:fst_def snd_def)+
  qed
qed

lemma lam_replacement_Un: lam_replacement(M,  $\lambda p. \text{fst}(p) \cup \text{snd}(p)$ )
  using lam_replacement_Upair lam_replacement_Union
  lam_replacement_hcomp[where  $g = \text{Union}$  and  $f = \lambda p. \text{Upair}(\text{fst}(p), \text{snd}(p))$ ]
  unfolding Un_def by simp

lemma lam_replacement_cons: lam_replacement(M,  $\lambda p. \text{cons}(\text{fst}(p), \text{snd}(p))$ )
  using lam_replacement_Upair
  lam_replacement_hcomp2[of _ _ ( $\cup$ )]
  lam_replacement_hcomp2[of fst fst Upair]
  lam_replacement_Un lam_replacement_fst lam_replacement_snd
  unfolding cons_def
  by auto

lemma lam_replacement_sing: lam_replacement(M,  $\lambda x. \{x\}$ )
  using lam_replacement_constant lam_replacement_cons

```

```

    lam_replacement_hcomp2[of  $\lambda x. x \lambda_. 0 \text{ cons}$ ]
  by (force intro: lam_replacement_identity)

lemmas tag_replacement = lam_replacement_constant[unfolded lam_replacement_def]

lemma lam_replacement_id2: lam_replacement(M,  $\lambda x. \langle x, x \rangle$ )
  using lam_replacement_identity lam_replacement_product[of  $\lambda x. x \lambda x. x$ ]
  by simp

lemmas id_replacement = lam_replacement_id2[unfolded lam_replacement_def]

lemma lam_replacement_apply2: lam_replacement(M,  $\lambda p. \text{fst}(p) \text{ ' snd}(p)$ )
  using lam_replacement_sing lam_replacement_fst lam_replacement_snd
    lam_replacement_Image lam_replacement_Union
  unfolding apply_def
  by (rule_tac lam_replacement_hcomp[of _ Union],
    rule_tac lam_replacement_hcomp2[of _ (' ')]
    (force intro: lam_replacement_hcomp)+

definition map_snd where
  map_snd(X) = {snd(z) . z ∈ X}

lemma map_sndE:  $y \in \text{map\_snd}(X) \implies \exists p \in X. y = \text{snd}(p)$ 
  unfolding map_snd_def by auto

lemma map_sndI :  $\exists p \in X. y = \text{snd}(p) \implies y \in \text{map\_snd}(X)$ 
  unfolding map_snd_def by auto

lemma map_snd_closed:  $M(x) \implies M(\text{map\_snd}(x))$ 
  unfolding map_snd_def
  using lam_replacement_imp_strong_replacement[OF lam_replacement_snd]
    RepFun_closed snd_closed[OF transM[of _ x]]
  by simp

lemma lam_replacement_imp_lam_replacement_RepFun:
  assumes lam_replacement(M, f)  $\forall x[M]. M(f(x))$ 
    separation(M,  $\lambda x. ((\forall y \in \text{snd}(x). \text{fst}(y) \in \text{fst}(x)) \wedge (\forall y \in \text{fst}(x). \exists u \in \text{snd}(x). y = \text{fst}(u))))$ )
  and
    lam_replacement_RepFun_snd: lam_replacement(M, map_snd)
  shows lam_replacement(M,  $\lambda x. \{f(y) . y \in x\}$ )
proof -
  have f_closed:  $M(\langle \text{fst}(z), \text{map\_snd}(\text{snd}(z)) \rangle)$  if  $M(z)$  for  $z$ 
    using pair_in_M_iff fst_closed snd_closed map_snd_closed that
    by simp
  have p_closed:  $M(\langle x, \{f(y) . y \in x\} \rangle)$  if  $M(x)$  for  $x$ 
    using pair_in_M_iff RepFun_closed lam_replacement_imp_strong_replacement
      transM[OF _ that] that assms by auto
  {

```

```

fix A
assume M(A)
then
have M( $\{\langle y, f(y) \rangle . y \in x\}$ ) if  $x \in A$  for  $x$ 
  using lam_replacement_iff_lam_closed assms that transM[of _ A]
  unfolding lam_def by simp
from assms  $\langle M(A) \rangle$ 
have  $\forall x \in \bigcup A. M(f(x))$ 
  using transM[of _  $\bigcup A$ ] by auto
with assms  $\langle M(A) \rangle$ 
have M( $\{\langle y, f(y) \rangle . y \in \bigcup A\}$ ) (is M( $?fUnA$ ))
  using lam_replacement_iff_lam_closed[THEN iffD1, OF assms(2) assms(1)]
  unfolding lam_def
  by simp
with  $\langle M(A) \rangle$ 
have M(Pow_rel(M,  $?fUnA$ )) by simp
with  $\langle M(A) \rangle$ 
have M( $\{z \in A \times \text{Pow\_rel}(M, ?fUnA) . ((\forall y \in \text{snd}(z). \text{fst}(y) \in \text{fst}(z)) \wedge (\forall y \in \text{fst}(z). \exists u \in \text{snd}(z). y = \text{fst}(u)))\}$ ) (is M( $?T$ ))
  using assms(3) by simp
then
have 1: M( $\{\langle \text{fst}(z), \text{map\_snd}(\text{snd}(z)) \rangle . z \in ?T\}$ ) (is M( $?Y$ ))
  using lam_replacement_product[OF lam_replacement_fst
    lam_replacement_hcomp[OF lam_replacement_snd lam_replacement_RepFun_snd]]
    RepFun_closed lam_replacement_imp_strong_replacement
    f_closed[OF transM[OF _  $\langle M(?T) \rangle$ ]]
  by simp
have 2:  $?Y = \{\langle x, \{f(y) . y \in x\} \rangle . x \in A\}$  (is _ =  $?R$ )
proof(intro equalityI subsetI)
fix p
assume  $p \in ?R$ 
with  $\langle M(A) \rangle$ 
obtain  $x$  where  $x \in A$   $p = \langle x, \{f(y) . y \in x\} \rangle$   $M(x)$ 
  using transM[OF _  $\langle M(A) \rangle$ ]
  by auto
moreover from calculation
have M( $\{\langle y, f(y) \rangle . y \in x\}$ ) (is M( $?Ux$ ))
  using lam_replacement_iff_lam_closed assms
  unfolding lam_def by auto
moreover from calculation
have  $?Ux \subseteq ?fUnA$ 
  by auto
moreover from calculation
have  $?Ux \in \text{Pow\_rel}(M, ?fUnA)$ 
  using Pow_rel_char[OF  $\langle M(?fUnA) \rangle$ ] by simp
moreover from calculation
have  $\forall u \in x. \exists w \in ?Ux. u = \text{fst}(w)$ 
  by force
moreover from calculation

```

```

have  $\langle x, ?Ux \rangle \in ?T$  by auto
moreover from calculation
have  $\{f(y).y \in x\} = \text{map\_snd}(?Ux)$ 
  unfolding map_snd_def
  by (intro equalityI, auto)
ultimately
show  $p \in ?Y$ 
by (auto, rule_tac bexI[where x=x], simp_all, rule_tac bexI[where x=?Ux], simp_all)
next
fix  $u$ 
assume  $u \in ?Y$ 
moreover from this
obtain  $z$  where  $z \in ?T$   $u = \langle \text{fst}(z), \text{map\_snd}(\text{snd}(z)) \rangle$ 
  by blast
moreover from calculation
obtain  $x \ U$  where
   $1: x \in A \quad U \in \text{Pow\_rel}(M, ?fUnA) \quad (\forall u \in U. \text{fst}(u) \in x) \wedge (\forall w \in x. \exists v \in U. \\ w = \text{fst}(v)) \quad z = \langle x, U \rangle$ 
  by force
moreover from this
have  $\text{fst}(u) \in \bigcup A \quad \text{snd}(u) = f(\text{fst}(u))$  if  $u \in U$  for  $u$ 
  using that Pow_rel_char[OF  $\langle M(?fUnA) \rangle$ ]
  by auto
moreover from calculation
have  $\text{map\_snd}(U) = \{f(y) . y \in x\}$ 
  unfolding map_snd_def
  by (intro equalityI subsetI, auto)
moreover from calculation
have  $u = \langle x, \text{map\_snd}(U) \rangle$ 
  by simp
ultimately
show  $u \in ?R$ 
  by (auto)
qed
from 1 2
have  $M(\{ \langle x, \{f(y) . y \in x\} \rangle . x \in A \})$ 
  by simp
}
then
have  $\forall A[M]. M(\lambda x \in A. \{f(y) . y \in x\})$ 
  unfolding lam_def by auto
then
show ?thesis
  using lam_replacement_iff_lam_closed[THEN iffD2]  $p\_closed$ 
  by simp
qed

```

lemma *lam_replacement_apply*: $M(S) \implies \text{lam_replacement}(M, \lambda x. S \text{ ` } x)$

```

using lam_replacement_Union lam_replacement_constant lam_replacement_identity
  lam_replacement_Image lam_replacement_cons
  lam_replacement_hcomp2[of _ _ Image] lam_replacement_hcomp2[of  $\lambda x. x$ 
 $\lambda_. 0$  cons]
unfolding apply_def
by (rule_tac lam_replacement_hcomp[of _ Union]) (force intro: lam_replacement_hcomp)+

lemma apply_replacement:  $M(S) \implies \text{strong\_replacement}(M, \lambda x y. y = S \text{ ` } x)$ 
using lam_replacement_apply lam_replacement_imp_strong_replacement by
simp

lemma lam_replacement_id_const:  $M(b) \implies \text{lam\_replacement}(M, \lambda x. \langle x, b \rangle)$ 
using lam_replacement_identity lam_replacement_constant
  lam_replacement_product[of  $\lambda x. x \lambda x. b$ ] by simp

lemmas pospend_replacement = lam_replacement_id_const[unfolded lam_replacement_def]

lemma lam_replacement_const_id:  $M(b) \implies \text{lam\_replacement}(M, \lambda z. \langle b, z \rangle)$ 
using lam_replacement_identity lam_replacement_constant
  lam_replacement_product[of  $\lambda x. b \lambda x. x$ ] by simp

lemmas prepend_replacement = lam_replacement_const_id[unfolded lam_replacement_def]

lemma lam_replacement_apply_const_id:  $M(f) \implies M(z) \implies$ 
   $\text{lam\_replacement}(M, \lambda x. f \text{ ` } \langle z, x \rangle)$ 
using lam_replacement_const_id[of  $z$ ] lam_replacement_apply[of  $f$ ]
  lam_replacement_hcomp[of  $\lambda x. \langle z, x \rangle \lambda x. f \text{ ` } x$ ] by simp

lemmas apply_replacement2 = lam_replacement_apply_const_id[unfolded lam_replacement_def]

lemma lam_replacement_Inl:  $\text{lam\_replacement}(M, \text{Inl})$ 
using lam_replacement_identity lam_replacement_constant
  lam_replacement_product[of  $\lambda x. 0 \lambda x. x$ ]
unfolding Inl_def by simp

lemma lam_replacement_Inr:  $\text{lam\_replacement}(M, \text{Inr})$ 
using lam_replacement_identity lam_replacement_constant
  lam_replacement_product[of  $\lambda x. 1 \lambda x. x$ ]
unfolding Inr_def by simp

lemmas Inl_replacement1 = lam_replacement_Inl[unfolded lam_replacement_def]

lemma lam_replacement_Diff':  $M(X) \implies \text{lam\_replacement}(M, \lambda x. x - X)$ 
using lam_replacement_Diff
by (force intro: lam_replacement_hcomp2 lam_replacement_constant
  lam_replacement_identity)+

lemmas Pair_diff_replacement = lam_replacement_Diff'[unfolded lam_replacement_def]

```

lemma *diff_Pair_replacement*: $M(p) \implies \text{strong_replacement}(M, \lambda x y. y = \langle x, x - \{p\} \rangle)$
using *Pair_diff_replacement* **by** *simp*

lemma *swap_replacement*: $\text{strong_replacement}(M, \lambda x y. y = \langle x, (\lambda \langle x, y \rangle. \langle y, x \rangle)(x))$
using *lam_replacement_swap* **unfolding** *lam_replacement_def split_def* **by** *simp*

lemma *lam_replacement_Un_const*: $M(b) \implies \text{lam_replacement}(M, \lambda x. x \cup b)$
using *lam_replacement_Un* *lam_replacement_hcomp2*[*of* $_ _ (\cup)$]
lam_replacement_constant[*of* *b*] *lam_replacement_identity* **by** *simp*

lemmas *tag_union_replacement* = *lam_replacement_Un_const*[*unfolded* *lam_replacement_def*]

lemma *lam_replacement_csquare*: $\text{lam_replacement}(M, \lambda p. \langle \text{fst}(p) \cup \text{snd}(p), \text{fst}(p), \text{snd}(p) \rangle)$
using *lam_replacement_Un* *lam_replacement_fst* *lam_replacement_snd*
by (*fast intro*: *lam_replacement_product* *lam_replacement_hcomp2*)

lemma *csquare_lam_replacement*: $\text{strong_replacement}(M, \lambda x y. y = \langle x, (\lambda \langle x, y \rangle. \langle x \cup y, x, y \rangle)(x))$
using *lam_replacement_csquare* **unfolding** *split_def* *lam_replacement_def* .

lemma *lam_replacement_assoc*: $\text{lam_replacement}(M, \lambda x. \langle \text{fst}(\text{fst}(x)), \text{snd}(\text{fst}(x)), \text{snd}(x) \rangle)$
using *lam_replacement_fst* *lam_replacement_snd*
by (*force intro*: *lam_replacement_product* *lam_replacement_hcomp*)

lemma *assoc_replacement*: $\text{strong_replacement}(M, \lambda x y. y = \langle x, (\lambda \langle \langle x, y \rangle, z \rangle. \langle x, y, z \rangle)(x))$
using *lam_replacement_assoc* **unfolding** *split_def* *lam_replacement_def* .

lemma *lam_replacement_prod_fun*: $M(f) \implies M(g) \implies \text{lam_replacement}(M, \lambda x. \langle f \text{ ' } \text{fst}(x), g \text{ ' } \text{snd}(x) \rangle)$
using *lam_replacement_fst* *lam_replacement_snd*
by (*force intro*: *lam_replacement_product* *lam_replacement_hcomp* *lam_replacement_apply*)

lemma *prod_fun_replacement*: $M(f) \implies M(g) \implies \text{strong_replacement}(M, \lambda x y. y = \langle x, (\lambda \langle w, y \rangle. \langle f \text{ ' } w, g \text{ ' } y \rangle)(x))$
using *lam_replacement_prod_fun* **unfolding** *split_def* *lam_replacement_def* .

lemma *lam_replacement_vimage_sing*: $\text{lam_replacement}(M, \lambda p. \text{fst}(p) - \{ \text{snd}(p) \})$
using *lam_replacement_hcomp*[*OF* *lam_replacement_snd* *lam_replacement_sing*]
lam_replacement_hcomp2[*OF* *lam_replacement_fst* $_ _ _ _ \text{lam_replacement_vimage}$]
by *simp*

lemma *lam_replacement_vimage_sing_fun*: $M(f) \implies \text{lam_replacement}(M, \lambda x. f - \{ x \})$
using *lam_replacement_hcomp2*[*OF* *lam_replacement_constant*[*of* *f*]
lam_replacement_identity $_ _ \text{lam_replacement_vimage_sing}$]

```

by simp
lemma lam_replacement_image_sing_fun:  $M(f) \implies \text{lam\_replacement}(M, \lambda x. f$ 
  “  $\{x\}$ )
  using lam_replacement_hcomp2[OF lam_replacement_constant[of f]
    lam_replacement_hcomp[OF lam_replacement_identity lam_replacement_sing]
    _ lam_replacement_Image]
by simp

lemma converse_apply_projs:  $\forall x[M]. \bigcup (\text{fst}(x) - \text{“} \{ \text{snd}(x) \} ) = \text{converse}(\text{fst}(x))$ 
  “  $(\text{snd}(x))$ 
  using converse_apply_eq by auto

lemma lam_replacement_converse_app:  $\text{lam\_replacement}(M, \lambda p. \text{converse}(\text{fst}(p))$ 
  “  $\text{snd}(p))$ 
  using lam_replacement_cong[OF _ converse_apply_projs]
    lam_replacement_hcomp[OF lam_replacement_vimage_sing lam_replacement_Union]
by simp

lemmas cardinal_lib_assms4 = lam_replacement_vimage_sing_fun[unfolded lam_replacement_def]

lemma lam_replacement_sing_const_id:
   $M(x) \implies \text{lam\_replacement}(M, \lambda y. \{ \langle x, y \rangle \})$ 
  using lam_replacement_hcomp[OF lam_replacement_const_id[of x]]
    lam_replacement_sing_pair_in_M_iff
by simp

lemma tag_singleton_closed:  $M(x) \implies M(z) \implies M(\{ \{ \langle z, y \rangle \} . y \in x \})$ 
  using RepFun_closed[where  $A=x$  and  $f=\lambda u. \{ \langle z, u \rangle \}$ ]
    lam_replacement_imp_strong_replacement lam_replacement_sing_const_id
    transM[of _ x]
by simp

lemma separation_eq:
  assumes  $\forall x[M]. M(f(x)) \text{ lam\_replacement}(M, f)$ 
   $\forall x[M]. M(g(x)) \text{ lam\_replacement}(M, g)$ 
  shows  $\text{separation}(M, \lambda x. f(x) = g(x))$ 
proof -
  let ?Z =  $\lambda A. \{ \langle x, \langle f(x), \langle g(x), x \rangle \rangle \rangle . x \in A \}$ 
  let ?Y =  $\lambda A. \{ \langle \langle x, f(x) \rangle, \langle g(x), x \rangle \rangle . x \in A \}$ 
  note sndsnd = lam_replacement_hcomp[OF lam_replacement_snd lam_replacement_snd]
  note fstsnd = lam_replacement_hcomp[OF lam_replacement_snd lam_replacement_fst]
  note sndfst = lam_replacement_hcomp[OF lam_replacement_fst lam_replacement_snd]
  have  $M(?Z(A))$  if  $M(A)$  for  $A$ 
  using assms lam_replacement_iff_lam_closed that
    lam_replacement_product[OF assms(2)]
    lam_replacement_product[OF assms(4) lam_replacement_identity]]
  unfolding lam_def
  by auto
moreover

```



```

have ?Y(A) = {⟨fst(x), fst(snd(x))⟩, fst(snd(snd(x))), snd(snd(snd(x)))⟩ . x ∈
?Z(A)} for A
  by auto
moreover from calculation
have M(?Y(A)) if M(A) for A
  using
    lam_replacement_imp_strong_replacement[OF
    lam_replacement_product[OF
    lam_replacement_product[OF lam_replacement_fst fstsnd]
    lam_replacement_product[OF
    lam_replacement_hcomp[OF sndsnd lam_replacement_fst]
    lam_replacement_hcomp[OF lam_replacement_snd sndsnd]
    ]
    ], THEN RepFun_closed,simplified,of ?Z(A)]
    fst_closed[OF transM] snd_closed[OF transM] that
  by auto
then
have M({u ∈ ?Y(A) . snd(fst(u)) = fst(snd(u))}) (is M(?W(A))) if M(A) for A
  using that middle_separation assms
  by auto
then
have M({fst(fst(u)) . u ∈ ?W(A)}) if M(A) for A
  using that lam_replacement_imp_strong_replacement[OF
    lam_replacement_hcomp[OF lam_replacement_fst lam_replacement_fst],
    THEN RepFun_closed]
    fst_closed[OF transM]
  by auto
moreover
have {x ∈ A. f(x) = g(x)} = {fst(fst(u)) . u ∈ ?W(A)} for A
  by auto
ultimately
show ?thesis
  using separation_iff by auto
qed

```

```

lemma separation_subset:
  assumes  $\forall x[M]. M(f(x)) \text{ lam\_replacement}(M,f)$ 
     $\forall x[M]. M(g(x)) \text{ lam\_replacement}(M,g)$ 
  shows separation(M, $\lambda x . f(x) \subseteq g(x)$ )
proof -
  have  $f(x) \subseteq g(x) \longleftrightarrow f(x) \cup g(x) = g(x)$  for x
  using subset_Un_iff by simp
moreover from assms
have separation(M, $\lambda x . f(x) \cup g(x) = g(x)$ )
  using separation_eq lam_replacement_Un lam_replacement_hcomp2
  by simp
ultimately
show ?thesis
  using separation_cong[THEN iffD1] by auto

```

qed

lemma *separation_ball*:

assumes *separation*(*M*, $\lambda y. f(\text{fst}(y), \text{snd}(y))$) *M*(*X*)

shows *separation*(*M*, $\lambda y. \forall u \in X. f(y, u)$)

unfolding *separation_def*

proof(*clarify*)

fix *A*

assume *M*(*A*)

moreover

note $\langle M(X) \rangle$

moreover from *calculation*

have *M*(*A* × *X*)

by *simp*

then

have $M(\{p \in A \times X . f(\text{fst}(p), \text{snd}(p))\})$ (**is** *M*(*?P*))

using *assms*(1)

by *auto*

moreover from *calculation*

have $M(\{a \in A . ?P''\{a\} = X\})$ (**is** *M*(*?A'*))

using *separation_eq lam_replacement_image_sing_fun*[*of ?P*] *lam_replacement_constant*

by *simp*

moreover

have *f*(*a*, *x*) **if** *a* ∈ *?A'* **and** *x* ∈ *X* **for** *a* *x*

proof -

from *that*

have *a* ∈ *A* *?P''*{*a*} = *X*

by *auto*

then

have *x* ∈ *?P''*{*a*}

using *that* **by** *simp*

then

show *?thesis* **using** *image_singleton_iff* **by** *simp*

qed

moreover from *this*

have $\forall a[M]. a \in ?A' \longleftrightarrow a \in A \wedge (\forall x \in X. f(a, x))$

using *image_singleton_iff*

by *auto*

with $\langle M(?A') \rangle$

show $\exists y[M]. \forall a[M]. a \in y \longleftrightarrow a \in A \wedge (\forall x \in X. f(a, x))$

by (*rule_tac* *x* = *?A'* **in** *rexI, simp_all*)

qed

lemma *lam_replacement_twist*: *lam_replacement*(*M*, $\lambda \langle x, y \rangle, z. \langle x, y, z \rangle$)

using *lam_replacement_fst lam_replacement_snd*

lam_replacement_Pair[*THEN* [5] *lam_replacement_hcomp2*,

of $\lambda x. \text{snd}(\text{fst}(x)) \lambda x. \text{snd}(x)$, *THEN* [2] *lam_replacement_Pair*[

THEN [5] *lam_replacement_hcomp2*, *of* $\lambda x. \text{fst}(\text{fst}(x))$]]

lam_replacement_hcomp **unfolding** *split_def* **by** *simp*

```

lemma twist_closed[intro,simp]:  $M(x) \implies M((\lambda \langle x,y \rangle, z). \langle x,y,z \rangle)(x)$ 
  unfolding split_def by simp

lemma lam_replacement_Lambda:
  assumes lam_replacement( $M, \lambda y. b(\text{fst}(y), \text{snd}(y))$ )
     $\forall w[M]. \forall y[M]. M(b(w, y)) \implies M(W)$ 
  shows lam_replacement( $M, \lambda x. \lambda w \in W. b(x, w)$ )
proof (intro lam_replacement_iff_lam_closed[THEN iffD2]; clarify)
  have aux_sep:  $\forall x[M]. \text{separation}(M, \lambda y. \langle \text{fst}(x), y \rangle \in A)$ 
    if  $M(X) \implies M(A)$  for  $X \ A$ 
  using separation_in_lam_replacement_hcomp2[OF lam_replacement_hcomp[OF
lam_replacement_constant lam_replacement_fst
  lam_replacement_identity — lam_replacement_Pair]
  lam_replacement_constant[of A]
  that
  by simp
  have aux_closed:  $\forall x[M]. M(\{y \in X . \langle \text{fst}(x), y \rangle \in A\})$  if  $M(X) \implies M(A)$  for  $X \ A$ 
    using aux_sep that by simp
  have aux_lemma: lam_replacement( $M, \lambda p . \{y \in X . \langle \text{fst}(p), y \rangle \in A\}$ )
    if  $M(X) \implies M(A)$  for  $X \ A$ 
  proof -
    note lr = lam_replacement_Collect[OF  $\langle M(X) \rangle$ ]
    note fst3 = lam_replacement_hcomp[OF lam_replacement_fst
      lam_replacement_hcomp[OF lam_replacement_fst lam_replacement_fst]]
    then show ?thesis
      using lam_replacement_Collect[OF  $\langle M(X) \rangle$  aux_sep separation_ball[OF
separation_iff]]
      separation_in[OF — lam_replacement_snd — lam_replacement_hcomp[OF
lam_replacement_fst lam_replacement_snd]]
      separation_in[OF — lam_replacement_hcomp2[OF fst3 lam_replacement_snd
— lam_replacement_Pair] —
      lam_replacement_constant[of A]] that
    by auto
  qed
from assms
show lbc:  $M(x) \implies M(\lambda w \in W. b(x, w))$  for  $x$ 
  using lam_replacement_constant lam_replacement_identity
  lam_replacement_hcomp2[where  $h=b$ ]
  by (intro lam_replacement_iff_lam_closed[THEN iffD1, rule_format])
    simp_all
fix  $A$ 
assume  $M(A)$ 
moreover from this assms
have  $M(\{b(\text{fst}(x), \text{snd}(x)). x \in A \times W\})$  (is  $M(?RFb)$ ) — RepFun b
  using lam_replacement_imp_strong_replacement transM[of —  $A \times W$ ]
  by (rule_tac RepFun_closed) auto
moreover
  have  $\{\langle x,y \rangle, z \rangle \in (A \times W) \times ?RFb. z = b(x,y)\} = (\lambda \langle x,y \rangle \in A \times W. b(x,y)) \cap$ 

```

```

(A × W) × ?RFb
  (is {⟨⟨x,y⟩,z⟩ ∈ (A × W) × ?B. _} = ?lam)
  unfolding lam_def by auto
  moreover from calculation and assms
  have M(?lam)
    using lam_replacement_iff_lam_closed unfolding split_def by simp
  moreover
  have {⟨⟨x,y⟩,z⟩ ∈ (X × Y) × Z . P(x, y, z)} ⊆ (X × Y) × Z for X Y Z P
    by auto
  then
  have {⟨x,y,z⟩ ∈ X × Y × Z. P(x,y,z)} = (λ⟨⟨x,y⟩,z⟩ ∈ (X × Y) × Z. ⟨x,y,z⟩) “
    {⟨⟨x,y⟩,z⟩ ∈ (X × Y) × Z. P(x,y,z)} (is ?C' = Lambda(?A, ?f) “ ?C)
    for X Y Z P
    using image_lam[of ?C ?A ?f]
    by (intro equalityI) (auto)
  with calculation
  have {⟨x,y,z⟩ ∈ A × W × ?RFb. z = b(x,y)} =
    (λ⟨⟨x,y⟩,z⟩ ∈ (A × W) × ?RFb. ⟨x,y,z⟩) “ ?lam (is ?H = ?G)
    by simp
  with ⟨M(A)⟩ ⟨M(W)⟩ ⟨M(?lam)⟩ ⟨M(?RFb)⟩
  have M(?H)
    using lam_replacement_iff_lam_closed[THEN iffD1, rule_format, OF _
lam_replacement_twist]
    by simp
  moreover from this and ⟨M(A)⟩
  have (λx ∈ A. λw ∈ W. b(x, w)) =
    {⟨x,Z⟩ ∈ A × PowM(range(?H)). Z = {y ∈ W × ?RFb . ⟨x, y⟩ ∈ ?H}}
    unfolding lam_def
    by (intro equalityI; subst Pow_rel_char[of range(?H)])
      (auto dest:transM simp: lbc[unfolded lam_def], force+)
  moreover from calculation and ⟨M(A)⟩ and ⟨M(W)⟩
  have M(A × PowM(range(?H))) M(W × ?RFb)
    by auto
  moreover
  note ⟨M(W)⟩
  moreover from calculation
  have M({⟨x,Z⟩ ∈ A × PowM(range(?H)). Z = {y ∈ W × ?RFb . ⟨x, y⟩ ∈ ?H}})
    using separation_eq[OF _ lam_replacement_snd
aux_closed[OF ⟨M(W × ?RFb)⟩ ⟨M(?H)⟩]
aux_lemma[OF ⟨M(W × ?RFb)⟩ ⟨M(?H)⟩]]
    ⟨M(A × PowM(_))⟩ assms
    unfolding split_def
    by auto
  ultimately
  show M(λx ∈ A. λw ∈ W. b(x, w)) by simp
qed

lemma lam_replacement_apply_Pair:
  assumes M(y)

```

```

shows lam_replacement(M,  $\lambda x. y \text{ ' } \langle \text{fst}(x), \text{snd}(x) \rangle$ )
using assms lam_replacement_constant lam_replacement_Pair
      lam_replacement_apply2[THEN [5] lam_replacement_hcomp2]
by auto

lemma lam_replacement_apply_fst_snd:
shows lam_replacement(M,  $\lambda w. \text{fst}(w) \text{ ' } \text{fst}(\text{snd}(w)) \text{ ' } \text{snd}(\text{snd}(w))$ )
using lam_replacement_fst lam_replacement_snd lam_replacement_hcomp
      lam_replacement_apply2[THEN [5] lam_replacement_hcomp2]
by auto

lemma separation_snd_in_fst: separation(M,  $\lambda x. \text{snd}(x) \in \text{fst}(x)$ )
using separation_in lam_replacement_fst lam_replacement_snd
by auto

lemma lam_replacement_if_mem:
      lam_replacement(M,  $\lambda x. \text{if } \text{snd}(x) \in \text{fst}(x) \text{ then } 1 \text{ else } 0$ )
using separation_snd_in_fst
      lam_replacement_constant lam_replacement_if
by auto

lemma lam_replacement_Lambda_apply_fst_snd:
assumes M(X)
shows lam_replacement(M,  $\lambda x. \lambda w \in X. x \text{ ' } \text{fst}(w) \text{ ' } \text{snd}(w)$ )
using assms lam_replacement_apply_fst_snd lam_replacement_Lambda
by simp

lemma lam_replacement_Lambda_apply_Pair:
assumes M(X) M(y)
shows lam_replacement(M,  $\lambda x. \lambda w \in X. y \text{ ' } \langle x, w \rangle$ )
using assms lam_replacement_apply_Pair lam_replacement_Lambda
by simp

lemma lam_replacement_Lambda_if_mem:
assumes M(X)
shows lam_replacement(M,  $\lambda x. \lambda xa \in X. \text{if } xa \in x \text{ then } 1 \text{ else } 0$ )
using assms lam_replacement_if_mem lam_replacement_Lambda
by simp

lemma lam_replacement_comp':
      M(f)  $\implies$  M(g)  $\implies$  lam_replacement(M,  $\lambda x. f \circ x \circ g$ )
using lam_replacement_comp[THEN [5] lam_replacement_hcomp2,
      OF lam_replacement_constant lam_replacement_comp,
      THEN [5] lam_replacement_hcomp2] lam_replacement_constant
      lam_replacement_identity by simp

lemma separation_bex:
assumes separation(M,  $\lambda y. f(\text{fst}(y), \text{snd}(y))$ ) M(X)
shows separation(M,  $\lambda y. \exists u \in X. f(y, u)$ )

```

```

unfolding separation_def
proof(clarify)
  fix A
  assume M(A)
  moreover
  note  $\langle M(X) \rangle$ 
  moreover from calculation
  have  $M(A \times X)$ 
    by simp
  then
  have  $M(\{p \in A \times X . f(\text{fst}(p), \text{snd}(p))\})$  (is  $M(?P)$ )
    using assms(1)
    by auto
  moreover from calculation
  have  $M(\{a \in A . ?P'\{a\} \neq 0\})$  (is  $M(?A')$ )
    using separation_eq lam_replacement_image_sing_fun[of ?P] lam_replacement_constant
      separation_neg
    by simp
  moreover from this
  have  $\forall a[M]. a \in ?A' \longleftrightarrow a \in A \wedge (\exists x \in X. f(a, x))$ 
    using image_singleton_iff
    by auto
  with  $\langle M(?A') \rangle$ 
  show  $\exists y[M]. \forall a[M]. a \in y \longleftrightarrow a \in A \wedge (\exists x \in X. f(a, x))$ 
    by (rule_tac x=?A' in rexI, simp_all)
qed

```

```

lemma case_closed :
  assumes  $\forall x[M]. M(f(x)) \forall x[M]. M(g(x))$ 
  shows  $\forall x[M]. M(\text{case}(f, g, x))$ 
  unfolding case_def split_def cond_def
  using assms by simp

```

```

lemma separation_fst_equal :  $M(a) \implies \text{separation}(M, \lambda x . \text{fst}(x)=a)$ 
  using separation_eq lam_replacement_fst lam_replacement_constant
  by auto

```

```

lemma lam_replacement_case :
  assumes  $\text{lam\_replacement}(M, f) \text{ lam\_replacement}(M, g)$ 
   $\forall x[M]. M(f(x)) \forall x[M]. M(g(x))$ 
  shows  $\text{lam\_replacement}(M, \lambda x . \text{case}(f, g, x))$ 
  unfolding case_def split_def cond_def
  using lam_replacement_if separation_fst_equal
    lam_replacement_hcomp[of snd g]
    lam_replacement_hcomp[of snd f]
    lam_replacement_snd assms
  by simp

```

```

lemma Pi_replacement1:  $M(x) \implies M(y) \implies \text{strong\_replacement}(M, \lambda y a z. ya$ 

```

```

 $\in y \wedge z = \{\langle x, ya \rangle\}$ 
using lam_replacement_imp_strong_replacement
  strong_replacement_separation[OF lam_replacement_sing_const_id[of x],where
P= $\lambda x . x \in y$ ]
  separation_in_constant
by simp

lemma surj_imp_inj_replacement1:
   $M(f) \implies M(x) \implies \text{strong\_replacement}(M, \lambda y z. y \in f^{-1}\{x\} \wedge z = \{\langle x, y \rangle\})$ 
using Pi_replacement1 vimage_closed singleton_closed
by simp

lemmas domain_replacement = lam_replacement_domain[unfolded lam_replacement_def]

lemma domain_replacement_simp:  $\text{strong\_replacement}(M, \lambda x y. y = \text{domain}(x))$ 
using lam_replacement_domain lam_replacement_imp_strong_replacement by
simp

lemma un_Pair_replacement:  $M(p) \implies \text{strong\_replacement}(M, \lambda x y. y = x \cup \{p\})$ 
using lam_replacement_Un_const[THEN lam_replacement_imp_strong_replacement]
by simp

lemma diff_replacement:  $M(X) \implies \text{strong\_replacement}(M, \lambda x y. y = x - X)$ 
using lam_replacement_Diff'[THEN lam_replacement_imp_strong_replacement]
by simp

lemma lam_replacement_succ:
   $\text{lam\_replacement}(M, \lambda z. \text{succ}(z))$ 
unfolding succ_def
using lam_replacement_hcomp2[of  $\lambda x. x \lambda x. x \text{ cons}$ ]
  lam_replacement_cons lam_replacement_identity
by simp

lemma lam_replacement_hcomp_Least:
assumes  $\text{lam\_replacement}(M, g) \text{ lam\_replacement}(M, \lambda x. \mu i. x \in F(i, x))$ 
 $\forall x[M]. M(g(x)) \wedge x i. M(x) \implies i \in F(i, x) \implies M(i)$ 
shows  $\text{lam\_replacement}(M, \lambda x. \mu i. g(x) \in F(i, g(x)))$ 
using assms
by (rule_tac lam_replacement_hcomp[of  $\lambda x. \mu i. x \in F(i, x)$ ])
  (auto intro:Least_closed')

lemma domain_mem_separation:  $M(A) \implies \text{separation}(M, \lambda x. \text{domain}(x) \in A)$ 
using separation_in lam_replacement_constant lam_replacement_domain
by auto

lemma domain_eq_separation:  $M(p) \implies \text{separation}(M, \lambda x. \text{domain}(x) = p)$ 
using separation_eq lam_replacement_domain lam_replacement_constant
by auto

```

```

lemma lam_replacement_Int:
  shows lam_replacement( $M$ ,  $\lambda x. \text{fst}(x) \cap \text{snd}(x)$ )
proof -
  have  $A \cap B = (A \cup B) - ((A - B) \cup (B - A))$  (is  $\_ = ?f(A, B)$ ) for  $A\ B$ 
    by auto
  then
  show ?thesis
    using lam_replacement_cong
      lam_replacement_Diff[THEN[5] lam_replacement_hcomp2]
      lam_replacement_Un[THEN[5] lam_replacement_hcomp2]
      lam_replacement_fst lam_replacement_snd
    by simp
qed

lemma lam_replacement_CartProd:
  assumes lam_replacement( $M, f$ ) lam_replacement( $M, g$ )
     $\forall x[M]. M(f(x)) \ \forall x[M]. M(g(x))$ 
  shows lam_replacement( $M, \lambda x. f(x) \times g(x)$ )
proof -
  note rep_closed = lam_replacement_imp_strong_replacement[THEN RepFun_closed]
  {
    fix  $A$ 
    assume  $M(A)$ 
    moreover
    note transM[OF  $\_ \langle M(A) \rangle$ ]
    moreover from calculation assms
    have  $M(\{ \langle x, \langle f(x), g(x) \rangle \rangle . x \in A \})$  (is  $M(?A')$ )
    using lam_replacement_product[THEN lam_replacement_imp_lam_closed[unfolded
lam_def]]
    by simp
    moreover from calculation
    have  $M(\bigcup \{ f(x) . x \in A \})$  (is  $M(?F)$ )
    using rep_closed[OF assms(1)] assms(3)
    by simp
    moreover from calculation
    have  $M(\bigcup \{ g(x) . x \in A \})$  (is  $M(?G)$ )
    using rep_closed[OF assms(2)] assms(4)
    by simp
    moreover from calculation
    have  $M(?A' \times (?F \times ?G))$  (is  $M(?T)$ )
    by simp
    moreover from this
    have  $M(\{ t \in ?T . \text{fst}(\text{snd}(t)) \in \text{fst}(\text{snd}(\text{fst}(t))) \wedge \text{snd}(\text{snd}(t)) \in \text{snd}(\text{snd}(\text{fst}(t))) \})$ 
    (is  $M(?Q)$ )
    using
      lam_replacement_hcomp[OF lam_replacement_hcomp[OF lam_replacement_fst
lam_replacement_snd]  $\_$ ]
      lam_replacement_hcomp lam_replacement_identity lam_replacement_fst
      lam_replacement_snd
  }

```



```

      separation_in separation_conj
    by simp
  moreover from this
  have  $M(\{\langle \text{fst}(\text{fst}(t)), \text{snd}(t) \rangle . t \in ?Q\})$  (is  $M(?R)$ )
  using rep_closed lam_replacement_Pair[THEN [5] lam_replacement_hcomp2]
    lam_replacement_hcomp[OF lam_replacement_fst lam_replacement_fst]
lam_replacement_snd
  transM[of _ ?Q]
  by simp
  moreover from calculation
  have  $M(\{\langle x, ?R\text{``}\{x\} \rangle . x \in A\})$ 
  using lam_replacement_imp lam_closed[unfolded lam_def] lam_replacement_sing
    lam_replacement_Image[THEN [5] lam_replacement_hcomp2] lam_replacement_constant[of
?R]
  by simp
  moreover
  have  $?R\text{``}\{x\} = f(x) \times g(x)$  if  $x \in A$  for  $x$ 
  by (rule equalityI subsetI, force, rule subsetI, rule_tac a=x in imageI)
    (auto simp: that, (rule_tac rev_bexI[of x], simp_all add: that) +)
  ultimately
  have  $M(\{\langle x, f(x) \times g(x) \rangle . x \in A\})$  by auto
}
with assms
show ?thesis using lam_replacement_iff_lam_closed[THEN iffD2, unfolded lam_def]
  by simp
qed

```

```

lemma restrict_eq_separation':  $M(B) \implies \forall A[M]. \text{separation}(M, \lambda y. \exists x \in A. y = \langle x, \text{restrict}(x, B) \rangle)$ 
proof (clarify)
  fix A
  have  $\text{restrict}(r, B) = r \cap (B \times \text{range}(r))$  for  $r$ 
  unfolding restrict_def by (rule equalityI subsetI, auto)
  moreover
  assume  $M(A) \ M(B)$ 
  moreover from this
  have  $\text{separation}(M, \lambda y. \exists x \in A. y = \langle x, x \cap (B \times \text{range}(x)) \rangle)$ 
  using lam_replacement_Int[THEN [5] lam_replacement_hcomp2]
    lam_replacement_Pair[THEN [5] lam_replacement_hcomp2]
  using lam_replacement_fst lam_replacement_snd lam_replacement_constant
    lam_replacement_hcomp lam_replacement_range lam_replacement_identity
    lam_replacement_CartProd separation_bex separation_eq
  by simp_all
  ultimately
  show  $\text{separation}(M, \lambda y. \exists x \in A. y = \langle x, \text{restrict}(x, B) \rangle)$ 
  by simp
qed

```

lemmas lam_replacement_restrict' = lam_replacement_restrict[OF restrict_eq_separation']

```

lemma restrict_strong_replacement:  $M(A) \implies \text{strong\_replacement}(M, \lambda x y. y = \text{restrict}(x, A))$ 
  using lam_replacement_restrict restrict_eq_separation'
  lam_replacement_imp_strong_replacement
  by simp

lemma restrict_eq_separation:  $M(r) \implies M(p) \implies \text{separation}(M, \lambda x. \text{restrict}(x, r) = p)$ 
  using separation_eq lam_replacement_restrict' lam_replacement_constant
  by auto

lemma separation_equal_fst2:  $M(a) \implies \text{separation}(M, \lambda x. \text{fst}(\text{fst}(x)) = a)$ 
  using separation_eq lam_replacement_hcomp lam_replacement_fst lam_replacement_constant
  by auto

lemma separation_equal_apply:  $M(f) \implies M(a) \implies \text{separation}(M, \lambda x. f'x = a)$ 
  using separation_eq lam_replacement_apply[of f] lam_replacement_constant
  by auto

lemma lam_apply_replacement:  $M(A) \implies M(f) \implies \text{lam\_replacement}(M, \lambda x. \lambda n \in A. f' \langle x, n \rangle)$ 
  using lam_replacement_Lambda lam_replacement_hcomp[OF lam_replacement_apply[of f]] lam_replacement_Pair
  by simp

end — M_replacement

locale M_replacement_extra = M_replacement +
  assumes
    lam_replacement_minimum:  $\text{lam\_replacement}(M, \lambda p. \text{minimum}(\text{fst}(p), \text{snd}(p)))$ 
  and
    lam_replacement_RepFun_cons:  $\text{lam\_replacement}(M, \lambda p. \text{RepFun}(\text{fst}(p), \lambda x. \{\langle \text{snd}(p), x \rangle\}))$ 
  — This one is too particular: It is for Sigfun. I would like greater modularity here.

begin
lemma lam_replacement_Sigfun:
  assumes  $\text{lam\_replacement}(M, f) \ \forall y[M]. M(f(y))$ 
  shows  $\text{lam\_replacement}(M, \lambda x. \text{Sigfun}(x, f))$ 
  using lam_replacement_Union lam_replacement_identity
    lam_replacement_sing[THEN lam_replacement_imp_strong_replacement]
    lam_replacement_hcomp[of _ Union] assms tag_singleton_closed
    lam_replacement_RepFun_cons[THEN [5] lam_replacement_hcomp2]
  unfolding Sigfun_def
  by (rule_tac lam_replacement_hcomp[of _ Union], simp_all)

```

13.2 Particular instances

lemma *surj_imp_inj_replacement2*:

$M(f) \implies \text{strong_replacement}(M, \lambda x z. z = \text{Sigfun}(x, \lambda y. f - \{y\}))$
using *lam_replacement_imp_strong_replacement lam_replacement_Sigfun*
lam_replacement_vimage_sing_fun
by *simp*

lemma *lam_replacement_minimum_vimage*:

$M(f) \implies M(r) \implies \text{lam_replacement}(M, \lambda x. \text{minimum}(r, f - \{x\}))$
using *lam_replacement_minimum lam_replacement_vimage_sing_fun lam_replacement_constant*
by (*rule_tac lam_replacement_hcomp2[of _ minimum]*)
(force intro: lam_replacement_identity)+

lemmas *surj_imp_inj_replacement4* = *lam_replacement_minimum_vimage[unfolded lam_replacement_def]*

lemma *lam_replacement_Pi*: $M(y) \implies \text{lam_replacement}(M, \lambda x. \bigcup_{xa \in y} \{ \langle x, xa \rangle \})$

using *lam_replacement_Union lam_replacement_identity lam_replacement_constant*
lam_replacement_RepFun_cons[THEN [5] lam_replacement_hcomp2] tag_singleton_closed
by (*rule_tac lam_replacement_hcomp[of _ Union], simp_all*)

lemma *Pi_replacement2*: $M(y) \implies \text{strong_replacement}(M, \lambda x z. z = (\bigcup_{xa \in y} \{ \langle x, xa \rangle \}))$

using *lam_replacement_Pi[THEN lam_replacement_imp_strong_replacement, of y]*

proof -

assume $M(y)$
then
have $M(x) \implies M(\bigcup_{xa \in y} \{ \langle x, xa \rangle \})$ **for** x
using *tag_singleton_closed*
by (*rule_tac Union_closed RepFun_closed*)
with $\langle M(y) \rangle$
show *?thesis*

using *lam_replacement_Pi[THEN lam_replacement_imp_strong_replacement, of y]*

by *blast*

qed

lemma *if_then_Inj_replacement*:

shows $M(A) \implies \text{strong_replacement}(M, \lambda x y. y = \langle x, \text{if } x \in A \text{ then } \text{Inl}(x) \text{ else } \text{Inr}(x) \rangle)$

using *lam_replacement_if lam_replacement_Inl lam_replacement_Inr separation_in_constant*
unfolding *lam_replacement_def*

by *simp*

lemma *lam_if_then_replacement*:

$M(b) \implies$

$M(a) \implies M(f) \implies \text{strong_replacement}(M, \lambda y ya. ya = \langle y, \text{if } y = a \text{ then } b$

```

else f ‘ y))
  using lam_replacement_if lam_replacement_apply lam_replacement_constant
    separation_equal
  unfolding lam_replacement_def
  by simp

lemma if_then_replacement:
  M(A)  $\implies$  M(f)  $\implies$  M(g)  $\implies$  strong_replacement(M,  $\lambda x y. y = \langle x, \text{if } x \in A \text{ then } f \text{ ‘ } x \text{ else } g \text{ ‘ } x \rangle$ )
  using lam_replacement_if lam_replacement_apply[of f] lam_replacement_apply[of g]
    separation_in_constant
  unfolding lam_replacement_def
  by simp

lemma ifx_replacement:
  M(f)  $\implies$ 
  M(b)  $\implies$  strong_replacement(M,  $\lambda x y. y = \langle x, \text{if } x \in \text{range}(f) \text{ then } \text{converse}(f) \text{ ‘ } x \text{ else } b \rangle$ )
  using lam_replacement_if lam_replacement_apply lam_replacement_constant
    separation_in_constant
  unfolding lam_replacement_def
  by simp

lemma if_then_range_replacement2:
  M(A)  $\implies$  M(C)  $\implies$  strong_replacement(M,  $\lambda x y. y = \langle x, \text{if } x = \text{Inl}(A) \text{ then } C \text{ else } x \rangle$ )
  using lam_replacement_if lam_replacement_constant lam_replacement_identity
    separation_equal
  unfolding lam_replacement_def
  by simp

lemma if_then_range_replacement:
  M(u)  $\implies$ 
  M(f)  $\implies$ 
  strong_replacement
  (M,
     $\lambda z y. y = \langle z, \text{if } z = u \text{ then } f \text{ ‘ } 0 \text{ else if } z \in \text{range}(f) \text{ then } f \text{ ‘ } \text{succ}(\text{converse}(f) \text{ ‘ } z) \text{ else } z \rangle$ )
  using lam_replacement_if separation_equal separation_in_constant
    lam_replacement_constant lam_replacement_identity
    lam_replacement_succ lam_replacement_apply
    lam_replacement_hcomp[of  $\lambda x. \text{converse}(f) \text{ ‘ } x \text{ succ}$ ]
    lam_replacement_hcomp[of  $\lambda x. \text{succ}(\text{converse}(f) \text{ ‘ } x) \lambda x. f \text{ ‘ } x$ ]
  unfolding lam_replacement_def
  by simp

lemma Inl_replacement2:
  M(A)  $\implies$ 

```

$strong_replacement(M, \lambda x y. y = \langle x, if\ fst(x) = A\ then\ Inl(snd(x))\ else\ Inr(x) \rangle)$
using $lam_replacement_if\ separation_fst_equal$
 $lam_replacement_hcomp[of\ snd\ Inl]$
 $lam_replacement_Inl\ lam_replacement_Inr\ lam_replacement_snd$
unfolding $lam_replacement_def$
by $simp$

lemma $case_replacement1$:
 $strong_replacement(M, \lambda z y. y = \langle z, case(Inr, Inl, z) \rangle)$
using $lam_replacement_case\ lam_replacement_Inl\ lam_replacement_Inr$
unfolding $lam_replacement_def$
by $simp$

lemma $case_replacement2$:
 $strong_replacement(M, \lambda z y. y = \langle z, case(case(Inl, \lambda y. Inr(Inl(y))), \lambda y. Inr(Inr(y))), z) \rangle)$
using $lam_replacement_case\ lam_replacement_hcomp$
 $case_closed[of\ Inl\ \lambda x. Inr(Inl(x))]$
 $lam_replacement_Inl\ lam_replacement_Inr$
unfolding $lam_replacement_def$
by $simp$

lemma $case_replacement4$:
 $M(f) \implies M(g) \implies strong_replacement(M, \lambda z y. y = \langle z, case(\lambda w. Inl(f\ ' \ w), \lambda y. Inr(g\ ' \ y), z) \rangle)$
using $lam_replacement_case\ lam_replacement_hcomp$
 $lam_replacement_Inl\ lam_replacement_Inr\ lam_replacement_apply$
unfolding $lam_replacement_def$
by $simp$

lemma $case_replacement5$:
 $strong_replacement(M, \lambda x y. y = \langle x, (\lambda \langle x, z \rangle. case(\lambda y. Inl(\langle y, z \rangle), \lambda y. Inr(\langle y, z \rangle), x))(x) \rangle)$
unfolding $split_def\ case_def\ cond_def$
using $lam_replacement_if\ separation_equal_fst2$
 $lam_replacement_snd\ lam_replacement_Inl\ lam_replacement_Inr$
 $lam_replacement_hcomp[OF$
 $lam_replacement_product[OF$
 $lam_replacement_hcomp[OF\ lam_replacement_fst\ lam_replacement_snd]]]$
unfolding $lam_replacement_def$
by $simp$

end — $M_replacement_extra$

— To be used in the relativized treatment of Cohen posets

definition

— "domain collect F"

$dC_F :: i \Rightarrow i \Rightarrow i$ **where**

$dC_F(A, d) \equiv \{p \in A. domain(p) = d\}$

definition

— "domain restrict SepReplace Y"
 $drSR_Y :: i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow i$ **where**
 $drSR_Y(B,D,A,x) \equiv \{y . r \in A , restrict(r,B) = x \wedge y = domain(r) \wedge domain(r) \in D\}$

lemma $drSR_Y_equality$: $drSR_Y(B,D,A,x) = \{ dr \in D . (\exists r \in A . restrict(r,B) = x \wedge dr = domain(r)) \}$
unfolding $drSR_Y_def$ **by** *auto*

context $M_replacement_extra$
begin

lemma $separation_restrict_eq_dom_eq$: $\forall x[M]. separation(M, \lambda dr. \exists r \in A . restrict(r,B) = x \wedge dr = domain(r))$
if $M(A)$ **and** $M(B)$ **for** $A B$
using *that*
 $separation_eq[OF _$
 $lam_replacement_fst _$
 $lam_replacement_hcomp[OF lam_replacement_snd lam_replacement_domain$
 $]]$
 $separation_eq[OF _$
 $lam_replacement_hcomp[OF lam_replacement_snd lam_replacement_restrict']$
 $_$
 $lam_replacement_constant]$
by (*clarify*, *rule_tac separation_bex* $[OF _ \langle M(A) \rangle]$, *rule_tac separation_conj*, *simp_all*)

lemma $separation_is_insnd_restrict_eq_dom$: $separation(M, \lambda p. \forall x \in D. x \in snd(p) \longleftrightarrow (\exists r \in A. restrict(r, B) = fst(p) \wedge x = domain(r)))$
if $M(B)$ $M(D)$ $M(A)$ **for** $A B D$
using *that* $lam_replacement_fst$ $lam_replacement_hcomp$ $lam_replacement_snd$ $separation_in$
 $separation_eq[OF _$
 $lam_replacement_hcomp[OF lam_replacement_fst lam_replacement_snd] _$
 $lam_replacement_hcomp[OF lam_replacement_snd lam_replacement_domain]]$
 $separation_eq$ $separation_restrict_eq_dom_eq$
 $lam_replacement_hcomp[OF lam_replacement_snd lam_replacement_restrict']$
 $lam_replacement_hcomp[OF lam_replacement_fst$
 $lam_replacement_hcomp[OF lam_replacement_fst lam_replacement_fst]]$
by (*rule_tac separation_ball*, *rule_tac separation_iff'*, *simp_all*,
 $rule_tac separation_bex$ $[OF _ \langle M(A) \rangle]$, *rule_tac separation_conj*, *simp_all*)

lemma $lam_replacement_drSR_Y$:

assumes

$M(B)$ $M(D)$ $M(A)$

shows $lam_replacement(M, drSR_Y(B,D,A))$

using $lam_replacement_cong$ $lam_replacement_Collect$ $[OF _ \langle M(D) \rangle separation_restrict_eq_dom_eq$ *of*

```

A B]]
  assms drSR_Y_equality separation_is_insnd_restrict_eq_dom separation_restrict_eq_dom_eq
  by simp

lemma drSR_Y_closed:
  assumes
    M(B) M(D) M(A) M(f)
  shows M(drSR_Y(B,D,A,f))
  using assms drSR_Y_equality lam_replacement_Collect[OF ‹M(D)› separation_restrict_eq_dom_eq[of
A B]]
  assms drSR_Y_equality separation_is_insnd_restrict_eq_dom separation_restrict_eq_dom_eq
  by simp

lemma lam_if_then_apply_replacement: M(f)  $\implies$  M(v)  $\implies$  M(u)  $\implies$ 
  lam_replacement(M,  $\lambda x.$  if f ' x = v then f ' u else f ' x)
  using lam_replacement_if separation_equal_apply lam_replacement_constant
  lam_replacement_apply
  by simp

lemma lam_if_then_apply_replacement2: M(f)  $\implies$  M(m)  $\implies$  M(y)  $\implies$ 
  lam_replacement(M,  $\lambda z.$  if f ' z = m then y else f ' z)
  using lam_replacement_if separation_equal_apply lam_replacement_constant
  lam_replacement_apply
  by simp

lemma lam_if_then_replacement2: M(A)  $\implies$  M(f)  $\implies$ 
  lam_replacement(M,  $\lambda x.$  if x  $\in$  A then f ' x else x)
  using lam_replacement_if separation_in_constant lam_replacement_identity
  lam_replacement_apply
  by simp

lemma lam_if_then_replacement_apply: M(G)  $\implies$  lam_replacement(M,  $\lambda x.$  if
M(x) then G ' x else 0)
  using lam_replacement_if separation_in_constant lam_replacement_identity
  lam_replacement_apply
  lam_replacement_constant[of 0] separation_univ
  by simp

lemma lam_replacement_dC_F:
  assumes M(A)
  shows lam_replacement(M, dC_F(A))
proof -
  have separation(M,  $\lambda p.$   $\forall x \in A. x \in \text{snd}(p) \longleftrightarrow \text{domain}(x) = \text{fst}(p)$ ) if M(A)
  for A
  using separation_ball separation_iff'
  lam_replacement_hcomp lam_replacement_fst lam_replacement_snd lam_replacement_domain
  separation_in separation_eq that
  by simp_all
then

```

```

show ?thesis
  unfolding dC_F_def
  using assms lam_replacement_Collect[of A  $\lambda d x . \text{domain}(x) = d$ ]
    separation_eq[OF lam_replacement_domain lam_replacement_constant]
  by simp
qed

lemma dCF_closed:
  assumes M(A) M(f)
  shows M(dC_F(A,f))
  unfolding dC_F_def
  using assms lam_replacement_Collect[of A  $\lambda d x . \text{domain}(x) = d$ ]
    separation_eq[OF lam_replacement_domain lam_replacement_constant]
  by simp

lemma lam_replacement_min:  $M(f) \implies M(r) \implies \text{lam\_replacement}(M, \lambda x .$ 
   $\text{minimum}(r, f - \text{“}\{x\}\text{”}))$ 
  using lam_replacement_hcomp2[OF lam_replacement_constant[of r] lam_replacement_vimageSingFun]
    lam_replacement_minimum
  by simp

lemma lam_replacement_Collect_ball_Pair:
  assumes separation(M,  $\lambda p. \forall x \in G. x \in \text{snd}(p) \longleftrightarrow (\forall s \in \text{fst}(p). \langle s, x \rangle \in Q)$ )
   $M(G) \ M(Q)$ 
  shows  $\text{lam\_replacement}(M, \lambda x . \{a \in G . \forall s \in x. \langle s, a \rangle \in Q\})$ 
proof -
  have  $1: \forall x[M]. \text{separation}(M, \lambda a . \forall s \in x. \langle s, a \rangle \in Q) \text{ if } M(Q) \text{ for } Q$ 
  using separation_in lam_replacement_hcomp2[OF lam_replacement_Pair]
    lam_replacement_constant separation_ball
    lam_replacement_hcomp lam_replacement_fst lam_replacement_snd that
  by simp
  then
  show ?thesis
    using assms lam_replacement_Collect
    by simp_all
qed

lemma surj_imp_inj_replacement3:
   $(\bigwedge x. M(x) \implies \text{separation}(M, \lambda y. \forall s \in x. \langle s, y \rangle \in Q)) \implies M(G) \implies M(Q) \implies$ 
 $M(x) \implies$ 
   $\text{strong\_replacement}(M, \lambda y z. y \in \{a \in G . \forall s \in x. \langle s, a \rangle \in Q\} \wedge z = \{\langle x, y \rangle\})$ 
  using lam_replacement_imp_strong_replacement
  using lam_replacement_sing_const_id[THEN lam_replacement_imp_strong_replacement,
of x]
  unfolding strong_replacement_def
  by (simp, safe, drule_tac  $x=A \cap \{a \in G . \forall s \in x. \langle s, a \rangle \in Q\}$  in rspec,
simp, erule_tac  $\text{rex}E$ , rule_tac  $x=Y$  in rexI) auto

lemmas replacements = Pair_diff_replacement id_replacement tag_replacement

```



```

    pospend_replacement prepend_replacement
    Inl_replacement1 diff_Pair_replacement
    swap_replacement tag_union_replacement csquare_lam_replacement
    assoc_replacement prod_fun_replacement
    cardinal_lib_assms4 domain_replacement
    apply_replacement
    un_Pair_replacement restrict_strong_replacement diff_replacement
    if_then_Inj_replacement lam_if_then_replacement if_then_replacement
    ifx_replacement if_then_range_replacement2 if_then_range_replacement
    Inl_replacement2
    case_replacement1 case_replacement2 case_replacement4 case_replacement5

end — M_replacement_extra

end

```

14 Relative, Choice-less Cardinal Numbers

theory *Cardinal_Relative*

imports

Discipline_Cardinal
Lambda_Replacement
Univ_Relative

begin

hide_const (open) *L*

definition

Finite_rel :: $[i \Rightarrow o, i] \Rightarrow o$ **where**
Finite_rel(*M*, *A*) $\equiv \exists om[M]. \exists n[M]. \omega(M, om) \wedge n \in om \wedge eqpoll_rel(M, A, n)$

definition

banach_functor :: $[i, i, i, i, i] \Rightarrow i$ **where**
banach_functor(*X*, *Y*, *f*, *g*, *W*) $\equiv X - g^{“}(Y - f^{“}W)$

definition

is_banach_functor :: $[i \Rightarrow o, i, i, i, i, i] \Rightarrow o$ **where**
is_banach_functor(*M*, *X*, *Y*, *f*, *g*, *W*, *b*) \equiv
 $\exists fW[M]. \exists YfW[M]. \exists gYfW[M]. image(M, f, W, fW) \wedge setdiff(M, Y, fW, YfW)$
 \wedge
 $image(M, g, YfW, gYfW) \wedge setdiff(M, X, gYfW, b)$

lemma (in *M_basic*) *banach_functor_abs* :

assumes *M*(*X*) *M*(*Y*) *M*(*f*) *M*(*g*)

shows *relation1*(*M*, *is_banach_functor*(*M*, *X*, *Y*, *f*, *g*), *banach_functor*(*X*, *Y*, *f*, *g*))

unfolding *relation1_def is_banach_functor_def banach_functor_def*

using *assms*

by *simp*

```

lemma (in M_basic) banach_functor_closed:
  assumes  $M(X) \ M(Y) \ M(f) \ M(g)$ 
  shows  $\forall W[M]. \ M(\text{banach\_functor}(X, Y, f, g, W))$ 
  unfolding banach_functor_def using assms image_closed
  by simp

locale M_cardinals = M_ordertype + M_trancl + M_Perm + M_replacement_extra
+
assumes
  radd_separation:  $M(R) \implies M(S) \implies$ 
    separation(M,  $\lambda z.$ 
       $(\exists x \ y. \ z = \langle \text{Inl}(x), \text{Inr}(y) \rangle) \vee$ 
       $(\exists x' \ x. \ z = \langle \text{Inl}(x'), \text{Inl}(x) \rangle \wedge \langle x', x \rangle \in R) \vee$ 
       $(\exists y' \ y. \ z = \langle \text{Inr}(y'), \text{Inr}(y) \rangle \wedge \langle y', y \rangle \in S))$ 
    )
  and
  rmult_separation:  $M(b) \implies M(d) \implies \text{separation}(M,$ 
     $\lambda z. \exists x' \ y' \ x \ y. \ z = \langle \langle x', y' \rangle, x, y \rangle \wedge (\langle x', x \rangle \in b \vee x' = x \wedge \langle y', y \rangle \in d))$ 
  )
  and
  banach_repl_iter:  $M(X) \implies M(Y) \implies M(f) \implies M(g) \implies$ 
    strong_replacement(M,  $\lambda x \ y. \ x \in \text{nat} \wedge y = \text{banach\_functor}(X, Y, f,$ 
       $g) \hat{\ }^x (0)$ 
    )
begin

lemma rvimage_separation:  $M(f) \implies M(r) \implies$ 
  separation(M,  $\lambda z. \exists x \ y. \ z = \langle x, y \rangle \wedge \langle f \ ' \ x, f \ ' \ y \rangle \in r$ )
  using separation_pair separation_in
    lam_replacement_Pair[THEN[5] lam_replacement_hcomp2]
    lam_replacement_constant lam_replacement_apply2[THEN[5] lam_replacement_hcomp2, OF
lam_replacement_constant[of f]]
    lam_replacement_fst lam_replacement_snd
    lam_replacement_identity lam_replacement_hcomp
  by(simp_all)

lemma radd_closed[intro, simp]:  $M(a) \implies M(b) \implies M(c) \implies M(d) \implies M(\text{radd}(a, b, c, d))$ 
  using radd_separation by (auto simp add: radd_def)

lemma rmult_closed[intro, simp]:  $M(a) \implies M(b) \implies M(c) \implies M(d) \implies M(\text{rmult}(a, b, c, d))$ 
  using rmult_separation by (auto simp add: rmult_def)

end — M_cardinals

lemma (in M_cardinals) is_cardinal_iff_Least:
  assumes  $M(A) \ M(\kappa)$ 
  shows  $\text{is\_cardinal}(M, A, \kappa) \longleftrightarrow \kappa = (\mu \ i. \ M(i) \wedge i \approx^M A)$ 
  using is_cardinal_iff assms
  unfolding cardinal_rel_def by simp

```

14.1 The Schroeder-Bernstein Theorem

See Davey and Priestly, page 106

context $M_cardinals$
begin

lemma $bnd_mono_banach_functor$: $bnd_mono(X, banach_functor(X, Y, f, g))$
unfolding bnd_mono_def $banach_functor_def$
by $blast$

lemma inj_Inter :
assumes $g \in inj(Y, X)$ $A \neq 0$ $\forall a \in A. a \subseteq Y$
shows $g'(\bigcap A) = (\bigcap a \in A. g'a)$
proof ($intro$ $equalityI$ $subsetI$)
fix x
from $assms$
obtain a **where** $a \in A$ **by** $blast$
moreover
assume $x \in (\bigcap a \in A. g'a)$
ultimately
have x_in_im : $x \in g'y$ **if** $y \in A$ **for** y
using $that$ **by** $auto$
have $exists$: $\exists z \in y. x = g'z$ **if** $y \in A$ **for** y
proof -
note $that$
moreover **from** $this$ **and** x_in_im
have $x \in g'y$ **by** $simp$
moreover **from** $calculation$
have $x \in g'y$ **by** $simp$
moreover
note $assms$
ultimately
show $?thesis$
using $image_fun[OF inj_is_fun]$ **by** $auto$
qed
with $\langle a \in A \rangle$
obtain z **where** $z \in a$ $x = g'z$ **by** $auto$
moreover
have $z \in y$ **if** $y \in A$ **for** y
proof -
from $that$ **and** $exists$
obtain w **where** $w \in y$ $x = g'w$ **by** $auto$
moreover **from** $this$ $\langle x = g'z \rangle$ $assms$ **that** $\langle a \in A \rangle$ $\langle z \in a \rangle$
have $z = w$ **unfolding** inj_def **by** $blast$
ultimately
show $?thesis$ **by** $simp$
qed

```

moreover
note assms
moreover from calculation
have  $z \in \bigcap A$  by auto
moreover from calculation
have  $z \in Y$  by blast
ultimately
show  $x \in g$  “  $(\bigcap A)$ 
  using inj_is_fun[THEN funcI, of g] by fast
qed auto

```

```

lemma contin_banach_functor:
  assumes  $g \in \text{inj}(Y, X)$ 
  shows  $\text{contin}(\text{banach\_functor}(X, Y, f, g))$ 
  unfolding contin_def
proof (intro allI impI)
  fix  $A$ 
  assume  $\text{directed}(A)$ 
  then
  have  $A \neq 0$ 
    unfolding directed_def ..
  have  $\text{banach\_functor}(X, Y, f, g, \bigcup A) = X - g$ “(  $Y - f$ “(  $\bigcup A$ ) )
    unfolding banach_functor_def ..
  also
  have  $\dots = X - g$ “(  $Y - (\bigcup a \in A. f$ “(  $a$ ))
    by auto
  also from  $\langle A \neq 0 \rangle$ 
  have  $\dots = X - g$ “(  $(\bigcap a \in A. Y - f$ “(  $a$ ))
    by auto
  also from  $\langle A \neq 0 \rangle$  and assms
  have  $\dots = X - (\bigcap a \in A. g$ “(  $Y - f$ “(  $a$ ))
    using inj_Inter[of g Y X {Y-f“(  $a. a \in A$  } ] by fastforce
  also from  $\langle A \neq 0 \rangle$ 
  have  $\dots = (\bigcup a \in A. X - g$ “(  $Y - f$ “(  $a$ )) by simp
  also
  have  $\dots = (\bigcup a \in A. \text{banach\_functor}(X, Y, f, g, a))$ 
    unfolding banach_functor_def ..
  finally
  show  $\text{banach\_functor}(X, Y, f, g, \bigcup A) = (\bigcup a \in A. \text{banach\_functor}(X, Y, f, g, a))$  .
qed

```

```

lemma lfp_banach_functor:
  assumes  $g \in \text{inj}(Y, X)$ 
  shows  $\text{lfp}(X, \text{banach\_functor}(X, Y, f, g)) =$ 
     $(\bigcup n \in \text{nat}. \text{banach\_functor}(X, Y, f, g)^n (0))$ 
  using assms lfp_eq_Union bnd_mono_banach_functor contin_banach_functor
  by simp

```

```

lemma lfp_banach_functor_closed:

```

```

assumes  $M(g) \ M(X) \ M(Y) \ M(f) \ g \in \text{inj}(Y, X)$ 
shows  $M(\text{lfp}(X, \text{banach\_functor}(X, Y, f, g)))$ 
proof -
  from assms
  have  $M(\text{banach\_functor}(X, Y, f, g)^{\wedge n} (0))$  if  $n \in \text{nat}$  for  $n$ 
    by (rule_tac nat_induct [OF that], simp_all add: banach_functor_closed)
  with assms
  show ?thesis
    using family_union_closed' [OF banach_repl_iter M_nat] lfp_banach_functor
    by simp
qed

```

```

lemma banach_decomposition_rel:
  [|  $M(f); M(g); M(X); M(Y); f \in X \rightarrow Y; g \in \text{inj}(Y, X)$  |] ==>
     $\exists XA[M]. \exists XB[M]. \exists YA[M]. \exists YB[M].$ 
     $(XA \cap XB = 0) \ \& \ (XA \cup XB = X) \ \&$ 
     $(YA \cap YB = 0) \ \& \ (YA \cup YB = Y) \ \&$ 
     $f''XA = YA \ \& \ g''YB = XB$ 
  apply (intro rexI conjI)
    apply (rule_tac [6] Banach_last_equation)
    apply (rule_tac [5] refl)
    apply (assumption |
      rule inj_is_fun Diff_disjoint Diff_partition fun_is_rel
      image_subset lfp_subset) +
  using lfp_banach_functor_closed [of g X Y f]
  unfolding banach_functor_def by simp_all

```

```

lemma schroeder_bernstein_closed:
  [|  $M(f); M(g); M(X); M(Y); f \in \text{inj}(X, Y); g \in \text{inj}(Y, X)$  |] ==>  $\exists h[M]. h \in \text{bij}(X, Y)$ 
  apply (insert banach_decomposition_rel [of f g X Y])
  apply (simp add: inj_is_fun)
  apply (auto)
  apply (rule_tac  $x = \text{restrict}(f, XA) \cup \text{converse}(\text{restrict}(g, YB))$  in rexI)
  apply (auto intro!: restrict_bij bij_disjoint_Un intro: bij_converse_bij)
done

```

```

lemma mem_Pow_rel:  $M(r) \implies a \in \text{Pow\_rel}(M, r) \implies a \in \text{Pow}(r) \wedge M(a)$ 
  using Pow_rel_char by simp

```

```

lemma mem_bij_abs[simp]:  $\llbracket M(f); M(A); M(B) \rrbracket \implies f \in \text{bij}^M(A, B) \longleftrightarrow f \in \text{bij}(A, B)$ 
  using bij_rel_char by simp

```

```

lemma mem_inj_abs[simp]:  $\llbracket M(f); M(A); M(B) \rrbracket \implies f \in \text{inj}^M(A, B) \longleftrightarrow f \in \text{inj}(A, B)$ 
  using inj_rel_char by simp

```

```

lemma mem_surj_abs:  $\llbracket M(f); M(A); M(B) \rrbracket \implies f \in \text{surj}^M(A, B) \longleftrightarrow f \in \text{surj}(A, B)$ 

```

```

using surj_rel_char by simp

lemma bij_imp_eqpoll_rel:
  assumes  $f \in \text{bij}(A, B)$   $M(f)$   $M(A)$   $M(B)$ 
  shows  $A \approx^M B$ 
  using assms by (auto simp add: def_eqpoll_rel)

lemma eqpoll_rel_refl:  $M(A) \implies A \approx^M A$ 
  using bij_imp_eqpoll_rel[OF id_bij, OF id_closed] .

lemma eqpoll_rel_sym:  $X \approx^M Y \implies M(X) \implies M(Y) \implies Y \approx^M X$ 
  unfolding def_eqpoll_rel using converse_closed
  by (auto intro: bij_converse_bij)

lemma eqpoll_rel_trans [trans]:
  [|  $X \approx^M Y$ ;  $Y \approx^M Z$ ;  $M(X)$ ;  $M(Y)$ ;  $M(Z)$  |] ==>  $X \approx^M Z$ 
  unfolding def_eqpoll_rel by (auto intro: comp_bij)

lemma subset_imp_lepoll_rel:  $X \subseteq Y \implies M(X) \implies M(Y) \implies X \lesssim^M Y$ 
  unfolding def_lepoll_rel using id_subset_inj id_closed
  by simp blast

lemmas lepoll_rel_refl = subset_refl [THEN subset_imp_lepoll_rel, simp]

lemmas le_imp_lepoll_rel = le_imp_subset [THEN subset_imp_lepoll_rel]

lemma eqpoll_rel_imp_lepoll_rel:  $X \approx^M Y \implies M(X) \implies M(Y) \implies X \lesssim^M Y$ 
  unfolding def_eqpoll_rel bij_def def_lepoll_rel using bij_is_inj
  by (auto)

lemma lepoll_rel_trans [trans]:
  assumes
     $X \lesssim^M Y$   $Y \lesssim^M Z$   $M(X)$   $M(Y)$   $M(Z)$ 
  shows
     $X \lesssim^M Z$ 
  using assms def_lepoll_rel
  by (auto intro: comp_inj)

lemma eq_lepoll_rel_trans [trans]:
  assumes
     $X \approx^M Y$   $Y \lesssim^M Z$   $M(X)$   $M(Y)$   $M(Z)$ 
  shows
     $X \lesssim^M Z$ 
  using assms
  by (blast intro: eqpoll_rel_imp_lepoll_rel lepoll_rel_trans)

```

lemma *lepoll_rel_eq_trans* [*trans*]:
assumes $X \lesssim^M Y$ $Y \approx^M Z$ $M(X)$ $M(Y)$ $M(Z)$
shows $X \lesssim^M Z$
using *assms*
eqpoll_rel_imp_lepoll_rel[*of* Y Z] *lepoll_rel_trans*[*of* X Y Z]
by *simp*

lemma *eqpoll_relI*: $\llbracket X \lesssim^M Y; Y \lesssim^M X; M(X); M(Y) \rrbracket \implies X \approx^M Y$
unfolding *def_lepoll_rel* *def_eqpoll_rel* **using** *schroeder_bernstein_closed*
by *auto*

lemma *eqpoll_relE*:
 $\llbracket X \approx^M Y; \llbracket X \lesssim^M Y; Y \lesssim^M X \rrbracket \implies P; M(X); M(Y) \rrbracket \implies P$
by (*blast intro: eqpoll_rel_imp_lepoll_rel eqpoll_rel_sym*)

lemma *eqpoll_rel_iff*: $M(X) \implies M(Y) \implies X \approx^M Y \longleftrightarrow X \lesssim^M Y \ \& \ Y \lesssim^M X$
by (*blast intro: eqpoll_relI elim: eqpoll_relE*)

lemma *lepoll_rel_0_is_0*: $A \lesssim^M 0 \implies M(A) \implies A = 0$
using *def_lepoll_rel*
by (*cases A=0*) (*auto simp add: inj_def*)

lemmas *empty_lepoll_relI* = *empty_subsetI* [*THEN subset_imp_lepoll_rel, OF nonempty*]

lemma *lepoll_rel_0_iff*: $M(A) \implies A \lesssim^M 0 \longleftrightarrow A=0$
by (*blast intro: lepoll_rel_0_is_0 lepoll_rel_refl*)

lemma *Un_lepoll_rel_Un*:
 $\llbracket A \lesssim^M B; C \lesssim^M D; B \cap D = 0; M(A); M(B); M(C); M(D) \rrbracket \implies A \cup C \lesssim^M B \cup D$
using *def_lepoll_rel* **using** *inj_disjoint_Un*[*of* A B C D] *if_then_replacement*
apply (*auto*)
apply (*rule, assumption*)
apply (*auto intro!: lam_closed elim: transM*) +
done

lemma *eqpoll_rel_0_is_0*: $A \approx^M 0 \implies M(A) \implies A = 0$
using *eqpoll_rel_imp_lepoll_rel* *lepoll_rel_0_is_0* *nonempty*
by *blast*

lemma *eqpoll_rel_0_iff*: $M(A) \implies A \approx^M 0 \longleftrightarrow A=0$
by (*blast intro: eqpoll_rel_0_is_0 eqpoll_rel_refl*)

lemma *eqpoll_rel_disjoint_Un*:
 $\llbracket A \approx^M B; C \approx^M D; A \cap C = 0; B \cap D = 0; M(A); M(B); M(C); M(D) \rrbracket \implies A \cup C \approx^M B \cup D$
by (*auto intro: bij_disjoint_Un simp add: def_eqpoll_rel*)

14.2 lesspoll_rel: contributions by Krzysztof Grabczewski

lemma *lesspoll_rel_not_refl*: $M(i) \implies \sim (i \prec^M i)$
by (*simp add: lesspoll_rel_def eqpoll_rel_refl*)

lemma *lesspoll_rel_irrefl*: $i \prec^M i \implies M(i) \implies P$
by (*simp add: lesspoll_rel_def eqpoll_rel_refl*)

lemma *lesspoll_rel_imp_lepoll_rel*: $\llbracket A \prec^M B; M(A); M(B) \rrbracket \implies A \lesssim^M B$
by (*unfold lesspoll_rel_def, blast*)

lemma *rvimage_closed* [*intro, simp*]:
assumes
 $M(A) \ M(f) \ M(r)$
shows
 $M(rvimage(A, f, r))$
unfolding *rvimage_def* **using** *assms rvimage_separation* **by** *auto*

lemma *lepoll_rel_well_ord*: $\llbracket A \lesssim^M B; well_ord(B, r); M(A); M(B); M(r) \rrbracket$
 $\implies \exists s[M]. well_ord(A, s)$
unfolding *def_lepoll_rel* **by** (*auto intro: well_ord_rvimage*)

lemma *lepoll_rel_iff_leqpoll_rel*: $\llbracket M(A); M(B) \rrbracket \implies A \lesssim^M B \longleftrightarrow A \prec^M B \mid A \approx^M B$
apply (*unfold lesspoll_rel_def*)
apply (*blast intro: eqpoll_relI elim: eqpoll_relE*)
done

end — *M_cardinals*

context *M_cardinals*
begin

lemma *inj_rel_is_fun_M*: $f \in inj^M(A, B) \implies M(f) \implies M(A) \implies M(B) \implies f$
 $\in A \rightarrow^M B$
using *inj_is_fun function_space_rel_char* **by** *simp*

— In porting the following theorem, I tried to follow the Discipline strictly, though finally only an approach maximizing the use of absoluteness results ($\llbracket M(?A); M(?y) \rrbracket \implies ?A \rightarrow^M ?y = \{f \in ?A \rightarrow ?y \mid M(f)\}$
 $\llbracket M(?A); M(?B) \rrbracket \implies inj^M(?A, ?B) = \{f \in inj(?A, ?B) \mid M(f)\}$) was the one paying dividends.

lemma *inj_rel_not_surj_rel_succ*:
notes *mem_inj_abs* [*simp del*]
assumes *fi*: $f \in inj^M(A, succ(m))$ **and** *fns*: $f \notin surj^M(A, succ(m))$
and types: $M(f) \ M(A) \ M(m)$
shows $\exists f[M]. f \in inj^M(A, m)$

proof -

from *fi* [*THEN inj_rel_is_fun_M*] *fns types*
obtain *y* **where** *y*: $y \in succ(m) \wedge x \in A \implies f \cdot x \neq y \ M(y)$


```

    by (auto simp add: def_surj_rel)
show ?thesis
proof
  from types and  $\langle M(y) \rangle$ 
  show  $M(\lambda z \in A. \text{ if } f'z = m \text{ then } y \text{ else } f'z)$ 
  using transM[OF  $\langle M(A) \rangle$ ] lam_if_then_apply_replacement2 lam_replacement_iff lam_closed
    by (auto)
  with types y fi
  have  $(\lambda z \in A. \text{ if } f'z = m \text{ then } y \text{ else } f'z) \in A \rightarrow^M m$ 
    using function_space_rel_char inj_rel_char inj_is_fun[of f A succ(m)]
    by (auto intro!: if_type [THEN lam_type] dest: apply_funtype)
  with types y fi
  show  $(\lambda z \in A. \text{ if } f'z = m \text{ then } y \text{ else } f'z) \in \text{inj}^M(A, m)$ 
    by (simp add: def_inj_rel) blast
qed
qed

```

```

lemma lesspoll_rel_trans [trans]:
  [|  $X \prec^M Y$ ;  $Y \prec^M Z$ ;  $M(X)$ ;  $M(Y)$ ;  $M(Z)$  |] ==>  $X \prec^M Z$ 
  apply (unfold lesspoll_rel_def)
  apply (blast elim: eqpoll_relE intro: eqpoll_relI lepoll_rel_trans)
  done

```

```

lemma lesspoll_rel_trans1 [trans]:
  [|  $X \lesssim^M Y$ ;  $Y \prec^M Z$ ;  $M(X)$ ;  $M(Y)$ ;  $M(Z)$  |] ==>  $X \prec^M Z$ 
  apply (unfold lesspoll_rel_def)
  apply (blast elim: eqpoll_relE intro: eqpoll_relI lepoll_rel_trans)
  done

```

```

lemma lesspoll_rel_trans2 [trans]:
  [|  $X \prec^M Y$ ;  $Y \lesssim^M Z$ ;  $M(X)$ ;  $M(Y)$ ;  $M(Z)$  |] ==>  $X \prec^M Z$ 
  apply (unfold lesspoll_rel_def)
  apply (blast elim: eqpoll_relE intro: eqpoll_relI lepoll_rel_trans)
  done

```

```

lemma eq_lespoll_rel_trans [trans]:
  [|  $X \approx^M Y$ ;  $Y \prec^M Z$ ;  $M(X)$ ;  $M(Y)$ ;  $M(Z)$  |] ==>  $X \prec^M Z$ 
  by (blast intro: eqpoll_rel_imp_lepoll_rel lesspoll_rel_trans1)

```

```

lemma lesspoll_rel_eq_trans [trans]:
  [|  $X \prec^M Y$ ;  $Y \approx^M Z$ ;  $M(X)$ ;  $M(Y)$ ;  $M(Z)$  |] ==>  $X \prec^M Z$ 
  by (blast intro: eqpoll_rel_imp_lepoll_rel lesspoll_rel_trans2)

```

```

lemma is_cardinal_cong:
  assumes  $X \approx^M Y$   $M(X)$   $M(Y)$ 
  shows  $\exists \kappa[M]. \text{is\_cardinal}(M, X, \kappa) \wedge \text{is\_cardinal}(M, Y, \kappa)$ 
proof -

```

```

from assms
have  $(\mu i. M(i) \wedge i \approx^M X) = (\mu i. M(i) \wedge i \approx^M Y)$ 
by (intro Least_cong) (auto intro: comp_bij bij_converse_bij simp add: def_eqpoll_rel)
moreover from assms
have  $M(\mu i. M(i) \wedge i \approx^M X)$ 
using Least_closed' by fastforce
moreover
note assms
ultimately
show ?thesis
using is_cardinal_iff_Least
by auto
qed

```

— ported from Cardinal

```

lemma cardinal_rel_cong:  $X \approx^M Y \implies M(X) \implies M(Y) \implies |X|^M = |Y|^M$ 
apply (simp add: def_eqpoll_rel cardinal_rel_def)
apply (rule Least_cong)
apply (auto intro: comp_bij bij_converse_bij)
done

```

```

lemma well_ord_is_cardinal_eqpoll_rel:
  assumes well_ord(A,r) shows  $\text{is\_cardinal}(M, A, \kappa) \implies M(A) \implies M(\kappa) \implies$ 
 $M(r) \implies \kappa \approx^M A$ 
proof (subst is_cardinal_iff_Least[THEN iffD1, of A  $\kappa$ ])
  assume  $M(A) \ M(\kappa) \ M(r) \ \text{is\_cardinal}(M, A, \kappa)$ 
  moreover from assms and calculation
  obtain f i where  $M(f) \ \text{Ord}(i) \ M(i) \ f \in \text{bij}(A, i)$ 
    using ordertype_exists[of A r] ord_iso_is_bij by auto
  moreover
  have  $M(\mu i. M(i) \wedge i \approx^M A)$ 
    using Least_closed' by fastforce
  ultimately
  show  $(\mu i. M(i) \wedge i \approx^M A) \approx^M A$ 
  using assms[THEN well_ord_imp_relativized]
    LeastI[of  $\lambda i. M(i) \wedge i \approx^M A \ i$ ] Ord_ordertype[OF assms]
    bij_converse_bij[THEN bij_imp_eqpoll_rel, of f] by simp
qed

```

```

lemmas Ord_is_cardinal_eqpoll_rel = well_ord_Memrel[THEN well_ord_is_cardinal_eqpoll_rel]

```

15 Porting from *ZF.Cardinal*

The following results were ported more or less directly from *ZF.Cardinal*

```

lemma well_ord_cardinal_rel_eqpoll_rel:
  assumes r: well_ord(A,r) and  $M(A) \ M(r)$  shows  $|A|^M \approx^M A$ 
  using assms well_ord_is_cardinal_eqpoll_rel is_cardinal_iff
  by blast

```

lemmas $Ord_cardinal_rel_eqpoll_rel = well_ord_Memrel[THEN well_ord_cardinal_rel_eqpoll_rel]$

lemma $Ord_cardinal_rel_idem: Ord(A) \implies M(A) \implies ||A|^M|^M = |A|^M$
by (rule_tac $Ord_cardinal_rel_eqpoll_rel$ [THEN $cardinal_rel_cong$]) *auto*

lemma $well_ord_cardinal_rel_eqE$:
assumes $woX: well_ord(X,r)$ **and** $woY: well_ord(Y,s)$ **and** $eq: |X|^M = |Y|^M$
and types: $M(X) M(r) M(Y) M(s)$
shows $X \approx^M Y$
proof -
from *types*
have $X \approx^M |X|^M$ **by** (blast intro: $well_ord_cardinal_rel_eqpoll_rel$ [OF woX]
 $eqpoll_rel_sym$)
also
have $\dots = |Y|^M$ **by** (rule eq)
also from *types*
have $\dots \approx^M Y$ **by** (rule_tac $well_ord_cardinal_rel_eqpoll_rel$ [OF woY])
finally show $?thesis$ **by** (simp add: *types*)
qed

lemma $well_ord_cardinal_rel_eqpoll_rel_iff$:
 $[| well_ord(X,r); well_ord(Y,s); M(X); M(r); M(Y); M(s) |] \implies |X|^M = |Y|^M \iff X \approx^M Y$
by (blast intro: $cardinal_rel_cong well_ord_cardinal_rel_eqE$)

lemma $Ord_cardinal_rel_le: Ord(i) \implies M(i) \implies |i|^M \leq i$
unfolding $cardinal_rel_def$
using $eqpoll_rel_refl Least_le$ **by** *simp*

lemma $Card_rel_cardinal_rel_eq: Card^M(K) \implies M(K) \implies |K|^M = K$
apply (unfold $Card_rel_def$)
apply (erule *sym*)
done

lemma $Card_relI: [| Ord(i); !!j. j < i \implies M(j) \implies \sim(j \approx^M i); M(i) |] \implies Card^M(i)$
apply (unfold $Card_rel_def$ $cardinal_rel_def$)
apply (subst $Least_equality$)
apply (blast intro: $eqpoll_rel_refl$)
done

lemma $Card_rel_is_Ord: Card^M(i) \implies M(i) \implies Ord(i)$
apply (unfold $Card_rel_def$ $cardinal_rel_def$)
apply (erule *ssubst*)
apply (rule Ord_Least)
done

lemma $Card_rel_cardinal_rel_le: Card^M(K) \implies M(K) \implies K \leq |K|^M$

```

apply (simp (no_asm_simp) add: Card_rel_is_Ord Card_rel_cardinal_rel_eq)
done

lemma Ord_cardinal_rel [simp,intro!]:  $M(A) \implies \text{Ord}(|A|^M)$ 
apply (unfold cardinal_rel_def)
apply (rule Ord_Least)
done

lemma Card_rel_iff_initial: assumes types:M(K)
shows  $\text{Card}^M(K) \longleftrightarrow \text{Ord}(K) \ \& \ (\forall j[M]. j < K \longrightarrow \sim (j \approx^M K))$ 
proof -
  { fix j
    assume K:  $\text{Card}^M(K) \ M(j) \wedge j \approx^M K$ 
    assume  $j < K$ 
    also have ... =  $(\mu i. M(i) \wedge i \approx^M K)$  using K
    by (simp add: Card_rel_def cardinal_rel_def types)
    finally have  $j < (\mu i. M(i) \wedge i \approx^M K)$  .
    then have False using K
    by (best intro: less_LeastE[of  $\lambda j. M(j) \wedge j \approx^M K$ ])
  }
  with types
  show ?thesis
  by (blast intro: Card_relI Card_rel_is_Ord)
qed

lemma lt_Card_rel_imp_lesspoll_rel:  $[[ \text{Card}^M(a); i < a; M(a); M(i) ] \implies i \prec^M a$ 
apply (unfold lesspoll_rel_def)
apply (frule Card_rel_iff_initial [THEN iffD1], assumption)
apply (blast intro!: leI [THEN le_imp_lepoll_rel])
done

lemma Card_rel_0:  $\text{Card}^M(0)$ 
apply (rule Ord_0 [THEN Card_relI])
apply (auto elim!: ltE)
done

lemma Card_rel_Un:  $[[ \text{Card}^M(K); \text{Card}^M(L); M(K); M(L) ] \implies \text{Card}^M(K \cup L)$ 
apply (rule Ord_linear_le [of K L])
apply (simp_all add: subset_Un_iff [THEN iffD1] Card_rel_is_Ord le_imp_subset
  subset_Un_iff2 [THEN iffD1])
done

lemma Card_rel_cardinal_rel [iff]: assumes types:M(A) shows  $\text{Card}^M(|A|^M)$ 
using assms
proof (unfold cardinal_rel_def)
show  $\text{Card}^M(\mu i. M(i) \wedge i \approx^M A)$ 
proof (cases  $\exists i[M]. \text{Ord}(i) \wedge i \approx^M A$ )

```

```

case False thus ?thesis — degenerate case
  using Least_0[of  $\lambda i. M(i) \wedge i \approx^M A$ ] Card_rel_0
  by fastforce
next
case True — real case: A is isomorphic to some ordinal
then obtain i where  $i: \text{Ord}(i) \ i \approx^M A \ M(i)$  by blast
show ?thesis
proof (rule Card_relI [OF Ord_Least], rule notI)
  fix j
  assume  $j: j < (\mu i. M(i) \wedge i \approx^M A)$  and  $M(j)$ 
  assume  $j \approx^M (\mu i. M(i) \wedge i \approx^M A)$ 
  also have  $\dots \approx^M A$  using i LeastI[of  $\lambda i. M(i) \wedge i \approx^M A$ ] by (auto)
  finally have  $j \approx^M A$ 
    using Least_closed'[of  $\lambda i. M(i) \wedge i \approx^M A$ ] by (simp add:  $\langle M(j) \rangle$  types)
  thus False
    using  $\langle M(j) \rangle$  by (blast intro:less_LeastE [OF  $\_ j$ ])
  qed (auto intro:Least_closed)
qed
qed

lemma cardinal_rel_eq_lemma:
  assumes  $i: |i|^M \leq j$  and  $j: j \leq i$  and types:  $M(i) \ M(j)$ 
  shows  $|j|^M = |i|^M$ 
proof (rule eqpoll_relI [THEN cardinal_rel_cong])
  show  $j \lesssim^M i$  by (rule le_imp_lepoll_rel [OF j]) (simp_all add:types)
next
  have Oi:  $\text{Ord}(i)$  using j by (rule le_Ord2)
  with types
  have  $i \approx^M |i|^M$ 
    by (blast intro: Ord_cardinal_rel_eqpoll_rel eqpoll_rel_sym)
  also from types
  have  $\dots \lesssim^M j$ 
    by (blast intro: le_imp_lepoll_rel i)
  finally show  $i \lesssim^M j$  by (simp_all add:types)
qed (simp_all add:types)

lemma cardinal_rel_mono:
  assumes ij:  $i \leq j$  and types:  $M(i) \ M(j)$  shows  $|i|^M \leq |j|^M$ 
  using Ord_cardinal_rel [OF  $\langle M(i) \rangle$ ] Ord_cardinal_rel [OF  $\langle M(j) \rangle$ ]
proof (cases rule: Ord_linear_le)
  case le then show ?thesis .
next
case ge
  have i:  $\text{Ord}(i)$  using ij
    by (simp add: lt_Ord)
  have ci:  $|i|^M \leq j$ 
    by (blast intro: Ord_cardinal_rel_le ij le_trans i types)
  have  $|i|^M = ||i|^M|^M$ 
    by (auto simp add: Ord_cardinal_rel_idem i types)

```

```

also have ... =  $|j|^M$ 
  by (rule cardinal_rel_eq_lemma [OF ge ci]) (simp_all add:types)
finally have  $|i|^M = |j|^M$  .
thus ?thesis by (simp add:types)
qed

lemma cardinal_rel_lt_imp_lt: [|  $|i|^M < |j|^M$ ; Ord( $i$ ); Ord( $j$ );  $M(i)$ ;  $M(j)$  |]
==>  $i < j$ 
  apply (rule Ord_linear2 [of  $i$   $j$ ], assumption+)
  apply (erule lt_trans2 [THEN lt_irrefl])
  apply (erule cardinal_rel_mono, assumption+)
  done

lemma Card_rel_lt_imp_lt: [|  $|i|^M < K$ ; Ord( $i$ );  $\text{Card}^M(K)$ ;  $M(i)$ ;  $M(K)$  |]
==>  $i < K$ 
  by (simp (no_asm_simp) add: cardinal_rel_lt_imp_lt Card_rel_is_Ord Card_rel_cardinal_rel_eq)

lemma Card_rel_lt_iff: [| Ord( $i$ );  $\text{Card}^M(K)$ ;  $M(i)$ ;  $M(K)$  |] ==> ( $|i|^M < K$ )
<=> ( $i < K$ )
  by (blast intro: Card_rel_lt_imp_lt Ord_cardinal_rel_le [THEN lt_trans1])

lemma Card_rel_le_iff: [| Ord( $i$ );  $\text{Card}^M(K)$ ;  $M(i)$ ;  $M(K)$  |] ==> ( $K \leq |i|^M$ )
<=> ( $K \leq i$ )
  by (simp add: Card_rel_lt_iff Card_rel_is_Ord not_lt_iff_le [THEN iff_sym])

lemma well_ord_lepoll_rel_imp_cardinal_rel_le:
  assumes  $wB$ : well_ord( $B, r$ ) and  $AB$ :  $A \lesssim^M B$ 
  and
    types:  $M(B)$   $M(r)$   $M(A)$ 
  shows  $|A|^M \leq |B|^M$ 
  using Ord_cardinal_rel [OF  $\langle M(A) \rangle$ ] Ord_cardinal_rel [OF  $\langle M(B) \rangle$ ]
proof (cases rule: Ord_linear_le)
  case le thus ?thesis .
next
  case ge
  from lepoll_rel_well_ord [OF  $AB$   $wB$ ]
  obtain  $s$  where  $s$ : well_ord( $A, s$ )  $M(s)$  by (blast intro:types)
  have  $B \approx^M |B|^M$  by (blast intro:  $wB$  eqpoll_rel_sym well_ord_cardinal_rel_eqpoll_rel
types)
  also have ...  $\lesssim^M |A|^M$  by (rule le_imp_lepoll_rel [OF ge]) (simp_all add:types)
  also have ...  $\approx^M A$  by (rule well_ord_cardinal_rel_eqpoll_rel [OF  $s(1)$  _  $s(2)$ ])
(simp_all add:types)
  finally have  $B \lesssim^M A$  by (simp_all add:types)
  hence  $A \approx^M B$  by (blast intro: eqpoll_relI  $AB$  types)
  hence  $|A|^M = |B|^M$  by (rule cardinal_rel_cong) (simp_all add:types)
  thus ?thesis by (simp_all add:types)
qed

lemma lepoll_rel_cardinal_rel_le: [|  $A \lesssim^M i$ ; Ord( $i$ );  $M(A)$ ;  $M(i)$  |] ==>  $|A|^M$ 

```

```

≤ i
  using Memrel_closed
  apply (rule_tac le_trans)
  apply (erule well_ord_Memrel [THEN well_ord_lepoll_rel_imp_cardinal_rel_le],
  assumption+)
  apply (erule Ord_cardinal_rel_le, assumption)
  done

lemma lepoll_rel_Ord_imp_eqpoll_rel: [| A ≲M i; Ord(i); M(A); M(i) |] ==>
|A|M ≈M A
  by (blast intro: lepoll_rel_cardinal_rel_le well_ord_Memrel well_ord_cardinal_rel_eqpoll_rel
  dest!: lepoll_rel_well_ord)

lemma lesspoll_rel_imp_eqpoll_rel: [| A ≲M i; Ord(i); M(A); M(i) |] ==> |A|M
≈M A
  using lepoll_rel_Ord_imp_eqpoll_rel[OF lesspoll_rel_imp_lepoll_rel] .

lemma lesspoll_cardinal_lt_rel:
  shows [| A ≲M i; Ord(i); M(i); M(A) |] ==> |A|M < i
proof -
  assume assms: A ≲M i ⟨Ord(i)⟩ ⟨M(i)⟩ ⟨M(A)⟩
  then
  have A: Ord(|A|M) |A|M ≈M A M(|A|M)
    using Ord_cardinal_rel lesspoll_rel_imp_eqpoll_rel
    by simp_all
  with assms
  have |A|M ≲M i
    using eq_lesspoll_rel_trans by auto
  consider |A|M ∈ i | |A|M = i | i ∈ |A|M
    using Ord_linear[OF ⟨Ord(i)⟩ ⟨Ord(|A|M)⟩] by auto
  then
  have |A|M < i
  proof(cases)
    case 1
    then show ?thesis using ltI ⟨Ord(i)⟩ by simp
  next
    case 2
    with ⟨|A|M ≲M i⟩ ⟨M(i)⟩
    show ?thesis using lesspoll_rel_irrefl by simp
  next
    case 3
    with ⟨Ord(|A|M)⟩
    have i < |A|M using ltI by simp
    with ⟨M(A)⟩ A ⟨M(i)⟩
    have i ≲M |A|M
    using lt_Card_rel_imp_lesspoll_rel Card_rel_cardinal_rel by simp
    with ⟨M(|A|M)⟩ ⟨M(i)⟩
    show ?thesis
    using lesspoll_rel_irrefl lesspoll_rel_trans[OF ⟨|A|M ≲M i⟩ ⟨i ≲M _⟩]

```

```

      by simp
    qed
  then show ?thesis by simp
qed

```

```

lemma cardinal_rel_subset_Ord: [|A<=i; Ord(i); M(A); M(i)|] ==> |A|^M ⊆ i
  apply (drule subset_imp_lepoll_rel [THEN lepoll_rel_cardinal_rel_le])
  apply (auto simp add: lt_def)
  apply (blast intro: Ord_trans)
done

```

— The next lemma is the first with several porting issues

```

lemma cons_lepoll_rel_consD:
  [| cons(u,A) ≲M cons(v,B); u∉A; v∉B; M(u); M(A); M(v); M(B) |] ==> A
  ≲M B
  apply (simp add: def_lepoll_rel, unfold inj_def, safe)
  apply (rule_tac x = λx∈A. if f'x=v then f'u else f'x in rexI)
  apply (rule CollectI)

  apply (rule if_type [THEN lam_type])
  apply (blast dest: apply_funtype)
  apply (blast elim!: mem_irrefl dest: apply_funtype)

```

```

    apply (auto simp add:transM[of _ A])
  using lam_replacement_iff_lam_closed lam_if_then_apply_replacement
  by simp

```

```

lemma cons_eqpoll_rel_consD: [| cons(u,A) ≈M cons(v,B); u∉A; v∉B; M(u);
M(A); M(v); M(B) |] ==> A ≈M B
  apply (simp add: eqpoll_rel_iff)
  apply (blast intro: cons_lepoll_rel_consD)
done

```

```

lemma succ_lepoll_rel_succD: succ(m) ≲M succ(n) ==> M(m) ==> M(n) ==>
m ≲M n
  apply (unfold succ_def)
  apply (erule cons_lepoll_rel_consD)
  apply (rule mem_not_refl)+
  apply assumption+
done

```

```

lemma nat_lepoll_rel_imp_le:
  m ∈ nat ==> n ∈ nat ==> m ≲M n ==> M(m) ==> M(n) ==> m ≤ n
proof (induct m arbitrary: n rule: nat_induct)
  case 0 thus ?case by (blast intro!: nat_0_le)
next
  case (succ m)
  show ?case using ⟨n ∈ nat⟩
  proof (cases rule: natE)

```



```

    case 0 thus ?thesis using succ
    by (simp add: def_lepoll_rel inj_def)
next
    case (succ n') thus ?thesis using succ.hyps ‹ succ(m)  $\lesssim^M$  n ›
    by (blast dest!: succ_lepoll_rel_succD)
qed
qed

lemma nat_eqpoll_rel_iff: [| m ∈ nat; n ∈ nat; M(m); M(n) |] ==> m  $\approx^M$  n
<=> m = n
  apply (rule iffI)
  apply (blast intro: nat_lepoll_rel_imp_le le_anti_sym elim!: eqpoll_relE)
  apply (simp add: eqpoll_rel_refl)
  done

lemma nat_into_Card_rel:
  assumes n: n ∈ nat and types: M(n) shows CardM(n)
  using types
  apply (subst Card_rel_def)
proof (unfold cardinal_rel_def, rule sym)
  have Ord(n) using n by auto
  moreover
  { fix i
    assume i < n M(i) i  $\approx^M$  n
    hence False using n
    by (auto simp add: lt_nat_in_nat [THEN nat_eqpoll_rel_iff] types)
  }
  ultimately show (μ i. M(i) ∧ i  $\approx^M$  n) = n by (auto intro!: Least_equality
types eqpoll_rel_refl)
qed

lemmas cardinal_rel_0 = nat_0I [THEN nat_into_Card_rel, THEN Card_rel_cardinal_rel_eq,
simplified, iff]
lemmas cardinal_rel_1 = nat_1I [THEN nat_into_Card_rel, THEN Card_rel_cardinal_rel_eq,
simplified, iff]

lemma succ_lepoll_rel_natE: [| succ(n)  $\lesssim^M$  n; n ∈ nat |] ==> P
  by (rule nat_lepoll_rel_imp_le [THEN lt_irrefl], auto)

lemma nat_lepoll_rel_imp_ex_eqpoll_rel_n:
  [| n ∈ nat; nat  $\lesssim^M$  X ; M(n); M(X) |] ==> ∃ Y[M]. Y ⊆ X & n  $\approx^M$  Y
  apply (simp add: def_lepoll_rel def_eqpoll_rel)
  apply (fast del: subsetI subsetCE
    intro!: subset_SIs
    dest!: Ord_nat [THEN [2] OrdmemD, THEN [2] restrict_inj]
    elim!: restrict_bij
    inj_is_fun [THEN fun_is_rel, THEN image_subset])
  done

```

```

lemma lepoll_rel_succ:  $M(i) \implies i \lesssim^M \text{succ}(i)$ 
  by (blast intro: subset_imp_lepoll_rel)

lemma lepoll_rel_imp_lespoll_rel_succ:
  assumes  $A: A \lesssim^M m$  and  $m: m \in \text{nat}$ 
  and types:  $M(A) \ M(m)$ 
  shows  $A \prec^M \text{succ}(m)$ 
proof -
  { assume  $A \approx^M \text{succ}(m)$ 
    hence  $\text{succ}(m) \approx^M A$  by (rule eqpoll_rel_sym) (auto simp add:types)
    also have  $\dots \lesssim^M m$  by (rule A)
    finally have  $\text{succ}(m) \lesssim^M m$  by (auto simp add:types)
    hence False by (rule succ_lepoll_rel_natE) (rule m) }
  moreover have  $A \lesssim^M \text{succ}(m)$  by (blast intro: lepoll_rel_trans A lepoll_rel_succ
types)
  ultimately show ?thesis by (auto simp add: types lesspoll_rel_def)
qed

lemma lesspoll_rel_succ_imp_lepoll_rel:
   $\llbracket A \prec^M \text{succ}(m); m \in \text{nat}; M(A); M(m) \rrbracket \implies A \lesssim^M m$ 
proof -
  {
    assume  $m \in \text{nat} \ M(A) \ M(m) \ A \lesssim^M \text{succ}(m)$ 
     $\forall f \in \text{inj}^M(A, \text{succ}(m)). f \notin \text{surj}^{\tilde{M}}(A, \text{succ}(m))$ 
    moreover from this
    obtain  $f$  where  $M(f) \ f \in \text{inj}^M(A, \text{succ}(m))$ 
    using def_lepoll_rel by auto
    moreover from calculation
    have  $f \notin \text{surj}^{\tilde{M}}(A, \text{succ}(m))$  by simp
    ultimately
    have  $\exists f[M]. f \in \text{inj}^M(A, m)$ 
    using inj_rel_not_surj_rel_succ by auto
  }
  from this
  show  $\llbracket A \prec^M \text{succ}(m); m \in \text{nat}; M(A); M(m) \rrbracket \implies A \lesssim^M m$ 
  unfolding lepoll_rel_def eqpoll_rel_def bij_rel_def lesspoll_rel_def
  by (simp del:mem_inj_abs)
qed

lemma lesspoll_rel_succ_iff:  $m \in \text{nat} \implies M(A) \implies A \prec^M \text{succ}(m) \longleftrightarrow A \lesssim^M m$ 
  by (blast intro!: lepoll_rel_imp_lespoll_rel_succ lesspoll_rel_succ_imp_lepoll_rel)

lemma lepoll_rel_succ_disj:  $\llbracket A \lesssim^M \text{succ}(m); m \in \text{nat}; M(A) ; M(m) \rrbracket \implies$ 
 $A \lesssim^M m \mid A \approx^M \text{succ}(m)$ 
  apply (rule disjCI)
  apply (rule lesspoll_rel_succ_imp_lepoll_rel)
  prefer 2 apply assumption
  apply (simp (no_asm_simp) add: lesspoll_rel_def, assumption+)

```

```

done

lemma lesspoll_rel_cardinal_rel_lt: [| A <M i; Ord(i); M(A); M(i) |] ==> |A|M
< i
  apply (unfold lesspoll_rel_def, clarify)
  apply (frule lepoll_rel_cardinal_rel_le, assumption+) — because of types
  apply (blast intro: well_ord_Memrel well_ord_cardinal_rel_eqpoll_rel [THEN
eqpoll_rel_sym]
    dest: lepoll_rel_well_ord elim!: leE)
done

lemma lt_not_lepoll_rel:
  assumes n: n < i n ∈ nat
  and types: M(n) M(i) shows ~ i ≲M n
proof -
  { assume i: i ≲M n
    have succ(n) ≲M i using n
    by (elim ltE, blast intro: Ord_succ_subsetI [THEN subset_imp_lepoll_rel]
types)
    also have ... ≲M n by (rule i)
    finally have succ(n) ≲M n by (simp add: types)
    hence False by (rule succ_lepoll_rel_natE) (rule n) }
  thus ?thesis by auto
qed

A slightly weaker version of nat_eqpoll_rel_iff

lemma Ord_nat_eqpoll_rel_iff:
  assumes i: Ord(i) and n: n ∈ nat
  and types: M(i) M(n)
  shows i ≈M n ⟷ i = n
  using i nat_into_Ord [OF n]
proof (cases rule: Ord_linear_lt)
  case lt
  hence i ∈ nat by (rule lt_nat_in_nat) (rule n)
  thus ?thesis by (simp add: nat_eqpoll_rel_iff n types)
next
  case eq
  thus ?thesis by (simp add: eqpoll_rel_refl types)
next
  case gt
  hence ~ i ≲M n using n by (rule lt_not_lepoll_rel) (simp_all add: types)
  hence ~ i ≈M n using n by (blast intro: eqpoll_rel_imp_lepoll_rel types)
  moreover have i ≠ n using <n<i> by auto
  ultimately show ?thesis by blast
qed

lemma Card_rel_nat: CardM(nat)
proof -

```

```

{ fix i
  assume i: i < nat i  $\approx^M$  nat M(i)
  hence  $\sim$  nat  $\lesssim^M$  i
    by (simp add: lt_def lt_not_lepoll_rel)
  hence False using i
    by (simp add: eqpoll_rel_iff)
}
hence ( $\mu$  i. M(i)  $\wedge$  i  $\approx^M$  nat) = nat by (blast intro: Least_equality eqpoll_rel_refl)
thus ?thesis
  by (auto simp add: Card_rel_def cardinal_rel_def)
qed

```

```

lemma nat_le_cardinal_rel: nat < i  $\implies$  M(i)  $\implies$  nat < |i|^M
  apply (rule Card_rel_nat [THEN Card_rel_cardinal_rel_eq, THEN subst],
  simp_all)
  apply (erule cardinal_rel_mono, simp_all)
  done

```

```

lemma n_lesspoll_rel_nat: n  $\in$  nat  $\implies$  n  $\prec^M$  nat
  by (blast intro: Card_rel_nat ltI lt_Card_rel_imp_lesspoll_rel)

```

```

lemma cons_lepoll_rel_cong:
  [| A  $\lesssim^M$  B; b  $\notin$  B; M(A); M(B); M(b); M(a) |]  $\implies$  cons(a,A)  $\lesssim^M$  cons(b,B)
  apply (subst (asm) def_lepoll_rel, simp_all, subst def_lepoll_rel, simp_all,
  safe)
  apply (rule_tac x =  $\lambda y \in \text{cons } (a,A) . \text{if } y=a \text{ then } b \text{ else } f'y$  in rexI)
  apply (rule_tac d =  $\%z . \text{if } z \in B \text{ then } \text{converse } (f) 'z \text{ else } a$  in lam_injective)
  apply (safe elim!: consE')
  apply simp_all
  apply (blast intro: inj_is_fun [THEN apply_type])+
  apply (auto intro: lam_closed lam_if_then_replacement simp add: transM[of _
  A])
  done

```

```

lemma cons_eqpoll_rel_cong:
  [| A  $\approx^M$  B; a  $\notin$  A; b  $\notin$  B; M(A); M(B); M(a) ; M(b) |]  $\implies$  cons(a,A)  $\approx^M$ 
  cons(b,B)
  by (simp add: eqpoll_rel_iff cons_lepoll_rel_cong)

```

```

lemma cons_lepoll_rel_cons_iff:
  [| a  $\notin$  A; b  $\notin$  B; M(a); M(A); M(b); M(B) |]  $\implies$  cons(a,A)  $\lesssim^M$  cons(b,B)
 $\longleftrightarrow$  A  $\lesssim^M$  B
  by (blast intro: cons_lepoll_rel_cong cons_lepoll_rel_consD)

```

```

lemma cons_eqpoll_rel_cons_iff:
  [| a  $\notin$  A; b  $\notin$  B; M(a); M(A); M(b); M(B) |]  $\implies$  cons(a,A)  $\approx^M$  cons(b,B)
 $\longleftrightarrow$  A  $\approx^M$  B
  by (blast intro: cons_eqpoll_rel_cong cons_eqpoll_rel_consD)

```

```

lemma singleton_eqpoll_rel_1:  $M(a) \implies \{a\} \approx^M 1$ 
  apply (unfold succ_def)
  apply (blast intro!: eqpoll_rel_refl [THEN cons_eqpoll_rel_cong])
  done

lemma cardinal_rel_singleton:  $M(a) \implies |\{a\}|^M = 1$ 
  apply (rule singleton_eqpoll_rel_1 [THEN cardinal_rel_cong, THEN trans])
  apply (simp (no_asm) add: nat_into_Card_rel [THEN Card_rel_cardinal_rel_eq])
  apply auto
  done

lemma not_0_is_lepoll_rel_1:  $A \neq 0 \implies M(A) \implies 1 \lesssim^M A$ 
  apply (erule not_emptyE)
  apply (rule_tac a = cons (x, A - {x}) in subst)
  apply (rule_tac [2] a = cons(0,0) and P =  $\%y. y \lesssim^M \text{cons}(x, A - \{x\})$  in subst)
  apply auto
proof -
  fix x
  assume M(A)
  then
  show  $x \in A \implies \{0\} \lesssim^M \text{cons}(x, A - \{x\})$ 
  by (auto intro: cons_lepoll_rel_cong transM[OF_  $\langle M(A) \rangle$ ] subset_imp_lepoll_rel)
qed

```

```

lemma succ_eqpoll_rel_cong:  $A \approx^M B \implies M(A) \implies M(B) \implies \text{succ}(A) \approx^M \text{succ}(B)$ 
  apply (unfold succ_def)
  apply (simp add: cons_eqpoll_rel_cong mem_not_refl)
  done

```

The next result was not straightforward to port, and even a different statement was needed.

```

lemma sum_bij_rel:
  [|  $f \in \text{bij}^M(A, C); g \in \text{bij}^M(B, D); M(f); M(A); M(C); M(g); M(B); M(D)$  |]
   $\implies (\lambda z \in A+B. \text{case}(\%x. \text{Inl}(f'x), \%y. \text{Inr}(g'y), z)) \in \text{bij}^M(A+B, C+D)$ 
proof -
  assume asm:  $f \in \text{bij}^M(A, C) \ g \in \text{bij}^M(B, D) \ M(f) \ M(A) \ M(C) \ M(g) \ M(B) \ M(D)$ 
  then
  have  $M(\lambda z \in A+B. \text{case}(\%x. \text{Inl}(f'x), \%y. \text{Inr}(g'y), z))$ 
    using transM[OF_  $\langle M(A) \rangle$ ] transM[OF_  $\langle M(B) \rangle$ ]
    by (auto intro: case_replacement4 [THEN lam_closed])
  with asm
  show ?thesis
  apply simp
  apply (rule_tac d = case (%x. Inl (converse(f)'x), %y. Inr (converse(g)'y))
    in lam_bijective)
  apply (typecheck add: bij_is_inj inj_is_fun)
  apply (auto simp add: left_inverse_bij right_inverse_bij)

```

done
qed

lemma *sum_bij_rel'*:
assumes $f \in \text{bij}^M(A, C)$ $g \in \text{bij}^M(B, D)$ $M(f)$
 $M(A)$ $M(C)$ $M(g)$ $M(B)$ $M(D)$
shows
 $(\lambda z \in A+B. \text{case}(\lambda x. \text{Inl}(f'x), \lambda y. \text{Inr}(g'y), z)) \in \text{bij}(A+B, C+D)$
 $M(\lambda z \in A+B. \text{case}(\lambda x. \text{Inl}(f'x), \lambda y. \text{Inr}(g'y), z))$
proof -
from *assms*
show $M(\lambda z \in A+B. \text{case}(\lambda x. \text{Inl}(f'x), \lambda y. \text{Inr}(g'y), z))$
using $\text{transM}[OF _ \langle M(A) \rangle]$ $\text{transM}[OF _ \langle M(B) \rangle]$
by (*auto intro: case_replacement4[THEN lam_closed]*)
with *assms*
show $(\lambda z \in A+B. \text{case}(\lambda x. \text{Inl}(f'x), \lambda y. \text{Inr}(g'y), z)) \in \text{bij}(A+B, C+D)$
apply *simp*
apply (*rule_tac d = case* ($\%x. \text{Inl}(\text{converse}(f)'x)$, $\%y. \text{Inr}(\text{converse}(g)'y)$)
in *lam_bijective*)
apply (*typecheck add: bij_is_inj inj_is_fun*)
apply (*auto simp add: left_inverse_bij right_inverse_bij*)
done
qed

lemma *sum_eqpoll_rel_cong*:
assumes $A \approx^M C$ $B \approx^M D$ $M(A)$ $M(C)$ $M(B)$ $M(D)$
shows $A+B \approx^M C+D$
using *assms*
proof (*simp add: def_eqpoll_rel, safe, rename_tac g*)
fix f g
assume $M(f)$ $f \in \text{bij}(A, C)$ $M(g)$ $g \in \text{bij}(B, D)$
with *assms*
obtain h **where** $h \in \text{bij}(A+B, C+D)$ $M(h)$
using *sum_bij_rel'[of f A C g B D]* **by** *simp*
then
show $\exists f[M]. f \in \text{bij}(A+B, C+D)$
by *auto*
qed

lemma *prod_bij_rel'*:
assumes $f \in \text{bij}^M(A, C)$ $g \in \text{bij}^M(B, D)$ $M(f)$
 $M(A)$ $M(C)$ $M(g)$ $M(B)$ $M(D)$
shows
 $(\lambda \langle x, y \rangle \in A*B. \langle f'x, g'y \rangle) \in \text{bij}(A*B, C*D)$
 $M(\lambda \langle x, y \rangle \in A*B. \langle f'x, g'y \rangle)$
proof -
from *assms*
show $M((\lambda \langle x, y \rangle \in A*B. \langle f'x, g'y \rangle))$
using $\text{transM}[OF _ \langle M(A) \rangle]$ $\text{transM}[OF _ \langle M(B) \rangle]$

```

      transM[OF _ cartprod_closed, of _ A B]
    by (auto intro:prod_fun_replacement[THEN lam_closed, of f g A×B])
  with assms
  show (λ<x,y>∈A*B. <f'x, g'y>) ∈ bij(A*B, C*D)
    apply simp
    apply (rule_tac d = %<x,y>. <converse (f) 'x, converse (g) 'y>
      in lam_bijective)
    apply (typecheck add: bij_is_inj inj_is_fun)
    apply (auto simp add: left_inverse_bij right_inverse_bij)
  done
qed

```

```

lemma prod_eqpoll_rel_cong:
  assumes A ≈M C B ≈M D M(A) M(C) M(B) M(D)
  shows A×B ≈M C×D
  using assms
proof (simp add: def_eqpoll_rel, safe, rename_tac g)
  fix f g
  assume M(f) f ∈ bij(A, C) M(g) g ∈ bij(B, D)
  with assms
  obtain h where h∈bij(A×B, C×D) M(h)
    using prod_bij_rel[of f A C g B D] by simp
  then
  show ∃f[M]. f ∈ bij(A × B, C × D)
    by auto
qed

```

```

lemma inj_rel_disjoint_eqpoll_rel:
  [| f ∈ injM(A,B); A ∩ B = 0; M(f); M(A); M(B) |] ==> A ∪ (B - range(f))
  ≈M B
  apply (simp add: def_eqpoll_rel)
  apply (rule rexI)
  apply (rule_tac c = %x. if x ∈ A then f'x else x
    and d = %y. if y ∈ range (f) then converse (f) 'y else y
    in lam_bijective)
  apply (blast intro!: if_type inj_is_fun [THEN apply_type])
  apply (simp (no_asm_simp) add: inj_converse_fun [THEN apply_funtypel])
  apply (safe elim!: UnE')
  apply (simp_all add: inj_is_fun [THEN apply_rangeI])
  apply (blast intro: inj_converse_fun [THEN apply_type])
proof -
  assume f ∈ inj(A, B) A ∩ B = 0 M(f) M(A) M(B)
  then
  show M(λx∈A ∪ (B - range(f)). if x ∈ A then f 'x else x)
    using transM[OF _ 'M(A)'] transM[OF _ 'M(B)']
    lam_replacement_iff_lam_closed lam_if_then_replacement2
    by auto
qed

```

```

lemma Diff_sing_lepoll_rel:
  [| a ∈ A; A ≲M succ(n); M(a); M(A); M(n) |] ==> A - {a} ≲M n
  apply (unfold succ_def)
  apply (rule cons_lepoll_rel_consD)
  apply (rule_tac [3] mem_not_refl)
  apply (erule cons_Diff [THEN ssubst], simp_all)
done

```

```

lemma lepoll_rel_Diff_sing:
  assumes A: succ(n) ≲M A
  and types: M(n) M(A) M(a)
  shows n ≲M A - {a}
proof -
  have cons(n,n) ≲M A using A
  by (unfold succ_def)
  also from types
  have ... ≲M cons(a, A-{a})
  by (blast intro: subset_imp_lepoll_rel)
  finally have cons(n,n) ≲M cons(a, A-{a}) by (simp_all add:types)
  with types
  show ?thesis
  by (blast intro: cons_lepoll_rel_consD mem_irrefl)
qed

```

```

lemma Diff_sing_eqpoll_rel: [| a ∈ A; A ≈M succ(n); M(a); M(A); M(n) |] ==>
  A - {a} ≈M n
  by (blast intro!: eqpoll_relI
    elim!: eqpoll_relE
    intro: Diff_sing_lepoll_rel lepoll_rel_Diff_sing)

```

```

lemma lepoll_rel_1_is_sing: [| A ≲M 1; a ∈ A ; M(a); M(A) |] ==> A = {a}
  apply (frule Diff_sing_lepoll_rel, assumption+, simp)
  apply (drule lepoll_rel_0_is_0, simp)
  apply (blast elim: equalityE)
done

```

```

lemma Un_lepoll_rel_sum: M(A) ==> M(B) ==> A ∪ B ≲M A+B
  apply (simp add: def_lepoll_rel)
  apply (rule_tac x = λx∈A ∪ B. if x∈A then Inl(x) else Inr(x) in rexI)
  apply (rule_tac d = %z. snd(z) in lam_injective)
  apply force
  apply (simp add: Inl_def Inr_def)
proof -
  assume M(A) M(B)
  then
  show M(λx∈A ∪ B. if x ∈ A then Inl(x) else Inr(x))
    using transM[OF _ ⟨M(A)⟩] transM[OF _ ⟨M(B)⟩] if_then_Inj_replacement
    by (rule_tac lam_closed) auto
qed

```



```

lemma well_ord_Un_M:
  assumes well_ord(X,R) well_ord(Y,S)
  and types: M(X) M(R) M(Y) M(S)
  shows  $\exists T[M]. \text{well\_ord}(X \cup Y, T)$ 
  using assms
  by (erule_tac well_ord_radd [THEN [3] Un_lepoll_rel_sum [THEN lepoll_rel_well_ord]])
    (auto simp add: types)

```

```

lemma disj_Un_eqpoll_rel_sum:  $M(A) \implies M(B) \implies A \cap B = 0 \implies A \cup B \approx^M A + B$ 
  apply (simp add: def_eqpoll_rel)
  apply (rule_tac x =  $\lambda a \in A \cup B. \text{if } a \in A \text{ then } \text{Inl } (a) \text{ else } \text{Inr } (a)$  in rexI)
  apply (rule_tac d =  $\%z. \text{case } (\%x. x, \%x. x, z)$  in lam_bijective)
  apply auto
proof -
  assume M(A) M(B)
  then
  show  $M(\lambda x \in A \cup B. \text{if } x \in A \text{ then } \text{Inl}(x) \text{ else } \text{Inr}(x))$ 
    using transM[OF  $\langle M(A) \rangle$ ] transM[OF  $\langle M(B) \rangle$ ] if_then_Inj_replacement
    by (rule_tac lam_closed) auto
qed

```

```

lemma eqpoll_rel_imp_Finite_rel_iff:  $A \approx^M B \implies M(A) \implies M(B) \implies \text{Finite\_rel}(M,A) \longleftrightarrow \text{Finite\_rel}(M,B)$ 
  apply (unfold Finite_rel_def)
  apply (blast intro: eqpoll_rel_trans eqpoll_rel_sym)
  done

```

— It seems reasonable to have the absoluteness of *Finite* here, and deduce the rest of the results from this.

Perhaps modularize that proof to have absoluteness of injections and bijections of finite sets (cf. $\llbracket ?A \prec^M \text{succ}(?m); ?m \in \text{nat}; M(?A); M(?m) \rrbracket \implies ?A \lesssim^M ?m$).

```

lemma Finite_abs[simp]: assumes M(A) shows  $\text{Finite\_rel}(M,A) \longleftrightarrow \text{Finite}(A)$ 
  unfolding Finite_rel_def Finite_def
proof (simp, intro iffI)
  assume  $\exists n \in \text{nat}. A \approx^M n$ 
  then
  obtain n where  $A \approx^M n$  n  $\in \text{nat}$  by blast
  with assms
  show  $\exists n \in \text{nat}. A \approx n$ 
    unfolding eqpoll_def using nat_into_M by (auto simp add: def_eqpoll_rel)
next
  fix n
  assume  $\exists n \in \text{nat}. A \approx n$ 
  then
  obtain n where  $A \approx n$  n  $\in \text{nat}$  by blast
  moreover from this

```

```

obtain  $f$  where  $f \in \text{bij}(A, n)$  unfolding  $\text{eqpoll\_def}$  by auto
moreover
note  $\text{assms}$ 
moreover from  $\text{calculation}$ 
have  $\text{converse}(f) \in n \rightarrow A$  using  $\text{bij\_is\_fun}$  by  $\text{simp}$ 
moreover from  $\text{calculation}$ 
have  $M(\text{converse}(f))$  using  $\text{transM}[of \_ n \rightarrow A]$  by  $\text{simp}$ 
moreover from  $\text{calculation}$ 
have  $M(f)$  using  $\text{bij\_is\_fun}$ 
   $\text{fun\_is\_rel}[of f A \lambda \_. n, \text{THEN } \text{converse\_converse}]$ 
   $\text{converse\_closed}[of \text{converse}(f)]$  by  $\text{simp}$ 
ultimately
show  $\exists n \in \text{nat}. A \approx^M n$ 
  by  $(\text{force } \text{dest:nat\_into\_M } \text{simp } \text{add:def\_eqpoll\_rel})$ 
qed

```

```

lemma  $\text{lepoll\_rel\_nat\_imp\_Finite\_rel}$ :
  assumes  $A: A \lesssim^M n$  and  $n: n \in \text{nat}$ 
  and  $\text{types}: M(A) \ M(n)$ 
  shows  $\text{Finite\_rel}(M, A)$ 
proof -
  have  $A \lesssim^M n \implies \text{Finite\_rel}(M, A)$  using  $n$ 
  proof (induct  $n$ )
    case 0
    hence  $A = 0$  by  $(\text{rule } \text{lepoll\_rel\_0\_is\_0}, \text{simp\_all } \text{add:types})$ 
    thus ?case by  $\text{simp}$ 
  next
    case  $(\text{succ } n)$ 
    hence  $A \lesssim^M n \vee A \approx^M \text{succ}(n)$  by  $(\text{blast } \text{dest: } \text{lepoll\_rel\_succ\_disj } \text{intro:types})$ 
    thus ?case using  $\text{succ}$  by  $(\text{auto } \text{simp } \text{add: } \text{Finite\_rel\_def } \text{types})$ 
  qed
  thus ?thesis using  $A$  .
qed

```

```

lemma  $\text{lesspoll\_rel\_nat\_is\_Finite\_rel}$ :
   $A \prec^M \text{nat} \implies M(A) \implies \text{Finite\_rel}(M, A)$ 
apply  $(\text{unfold } \text{Finite\_rel\_def})$ 
apply  $(\text{auto } \text{dest: } \text{ltD } \text{lesspoll\_rel\_cardinal\_rel\_lt}$ 
   $\text{lesspoll\_rel\_imp\_eqpoll\_rel } [\text{THEN } \text{eqpoll\_rel\_sym}])$ 
done

```

```

lemma  $\text{lepoll\_rel\_Finite\_rel}$ :
  assumes  $Y: Y \lesssim^M X$  and  $X: \text{Finite\_rel}(M, X)$ 
  and  $\text{types}: M(Y) \ M(X)$ 
  shows  $\text{Finite\_rel}(M, Y)$ 
proof -
  obtain  $n$  where  $n: n \in \text{nat} \ X \approx^M n \ M(n)$  using  $X$ 

```

```

    by (auto simp add: Finite_rel_def)
  have  $Y \lesssim^M X$  by (rule Y)
  also have  $\dots \approx^M n$  by (rule n)
  finally have  $Y \lesssim^M n$  by (simp_all add: types  $\langle M(n) \rangle$ )
  thus ?thesis using n
    by (simp add: lepoll_rel_nat_imp_Finite_rel types  $\langle M(n) \rangle$  del: Finite_abs)
qed

lemma succ_lepoll_rel_imp_not_empty:  $\text{succ}(x) \lesssim^M y \implies M(x) \implies M(y) \implies y \neq 0$ 
  by (fast dest!: lepoll_rel_0_is_0)

lemma eqpoll_rel_succ_imp_not_empty:  $x \approx^M \text{succ}(n) \implies M(x) \implies M(n) \implies x \neq 0$ 
  by (fast elim!: eqpoll_rel_sym [THEN eqpoll_rel_0_is_0, THEN succ_neq_0])

lemma Finite_subset_closed:
  assumes Finite(B)  $B \subseteq A$   $M(A)$ 
  shows  $M(B)$ 
proof -
  from  $\langle \text{Finite}(B) \rangle \langle B \subseteq A \rangle$ 
  show ?thesis
  proof(induct, simp)
    case (cons x D)
    with assms
    have  $M(D)$   $x \in A$ 
    unfolding cons_def by auto
    then
    show ?case using transM[OF  $\_ \langle M(A) \rangle$ ] by simp
  qed
qed

lemma Finite_Pow_abs:
  assumes Finite(A)  $M(A)$ 
  shows  $\text{Pow}(A) = \text{Pow\_rel}(M, A)$ 
  using Finite_subset_closed[OF subset_Finite] assms Pow_rel_char
  by auto

lemma Finite_Pow_rel:
  assumes Finite(A)  $M(A)$ 
  shows Finite(Pow_rel(M, A))
  using Finite_Pow Finite_Pow_abs[symmetric] assms by simp

lemma Pow_rel_0 [simp]:  $\text{Pow\_rel}(M, 0) = \{0\}$ 
  using Finite_Pow_abs[of 0] by simp

lemma eqpoll_rel_imp_Finite:  $A \approx^M B \implies \text{Finite}(A) \implies M(A) \implies M(B) \implies \text{Finite}(B)$ 
proof -

```

```

assume  $A \approx^M B$   $Finite(A)$   $M(A)$   $M(B)$ 
then obtain  $f$   $n$   $g$  where  $f \in bij(A, B)$   $n \in nat$   $g \in bij(A, n)$ 
  unfolding  $Finite\_def$   $eqpoll\_def$   $eqpoll\_rel\_def$ 
  using  $bij\_rel\_char$ 
  by auto
then
  have  $g \circ converse(f) \in bij(B, n)$ 
  using  $bij\_converse\_bij\_comp\_bij$  by simp
  with  $\langle n \in \_ \rangle$ 
  show  $Finite(B)$ 
  unfolding  $Finite\_def$   $eqpoll\_def$  by auto
qed

lemma  $eqpoll\_rel\_imp\_Finite\_iff$ :  $A \approx^M B \implies M(A) \implies M(B) \implies Finite(A)$ 
 $\iff Finite(B)$ 
  using  $eqpoll\_rel\_imp\_Finite$   $eqpoll\_rel\_sym$  by force

end —  $M\_cardinals$ 

end

```

16 Relative, Choice-less Cardinal Arithmetic

```

theory CardinalArith_Relative
  imports
    Cardinal_Relative

begin

relativize functional  $rvimage$   $rvimage\_rel$  external
relationalize  $rvimage\_rel$   $is\_rvimage$ 

definition
   $csquare\_lam :: i \Rightarrow i$  where
     $csquare\_lam(K) \equiv \lambda \langle x, y \rangle \in K \times K. \langle x \cup y, x, y \rangle$ 

  — Can't do the next thing because split is a missing HOC

relativize_tm  $\langle fst(x) \cup snd(x), fst(x), snd(x) \rangle$   $is\_csquare\_lam\_body$ 

definition
   $is\_csquare\_lam :: [i \Rightarrow o, i, i] \Rightarrow o$  where
     $is\_csquare\_lam(M, K, l) \equiv \exists K2[M]. cartprod(M, K, K, K2) \wedge$ 
     $is\_lambda(M, K2, is\_csquare\_lam\_body(M), l)$ 

definition  $jump\_cardinal\_body :: [i \Rightarrow o, i] \Rightarrow i$  where
   $jump\_cardinal\_body(M, X) \equiv$ 

```

$\{z . r \in \text{Pow}^M(X \times X), M(z) \wedge M(r) \wedge \text{well_ord}(X, r) \wedge z = \text{ordertype}(X, r)\}$

lemma (in $M_cardinals$) $\text{csquare_lam_closed}[\text{intro}, \text{simp}]: M(K) \implies M(\text{csquare_lam}(K))$
using $\text{csquare_lam_replacement}$ **unfolding** csquare_lam_def
by (rule lam_closed) (auto dest: transM)

locale $M_pre_cardinal_arith = M_cardinals +$
assumes
 $\text{wfrec_pred_replacement}: M(A) \implies M(r) \implies$
 $\text{wfrec_replacement}(M, \lambda x f z. z = f \text{ `` Order.pred}(A, x, r), r)$
begin

lemma $\text{ord_iso_separation}: M(A) \implies M(r) \implies M(s) \implies$
 $\text{separation}(M, \lambda f. \forall x \in A. \forall y \in A. \langle x, y \rangle \in r \longleftrightarrow \langle f \text{ ` } x, f \text{ ` } y \rangle \in s)$
using
 $\text{lam_replacement_Pair}[\text{THEN}[5] \text{ lam_replacement_hcomp2}]$
 $\text{lam_replacement_hcomp lam_replacement_fst lam_replacement_snd}$
 $\text{separation_in lam_replacement_fst lam_replacement_apply2}[\text{THEN}[5]$
 $\text{lam_replacement_hcomp2}]$
 $\text{lam_replacement_identity lam_replacement_constant}$
by(rule $\text{tac separation_ball}$, rule $\text{tac separation_ball}$, simp all , rule $\text{tac separation_iff'}$, simp all)

end

locale $M_cardinal_arith = M_pre_cardinal_arith +$
assumes
 $\text{ordertype_replacement} :$
 $M(X) \implies \text{strong_replacement}(M, \lambda x z . M(z) \wedge M(x) \wedge x \in \text{Pow_rel}(M, X \times X)$
 $\wedge \text{well_ord}(X, x) \wedge z = \text{ordertype}(X, x))$
and
 $\text{strong_replacement_jc_body} :$
 $\text{strong_replacement}(M, \lambda x z . M(z) \wedge M(x) \wedge z = \text{jump_cardinal_body}(M, x))$

lemmas (in $M_cardinal_arith$) $\text{surj_imp_inj_replacement} =$
 $\text{surj_imp_inj_replacement1 surj_imp_inj_replacement2 surj_imp_inj_replacement4}$
 $\text{lam_replacement_vimage_sing_fun}[\text{THEN lam_replacement_imp_strong_replacement}]$

relativize_tm $\exists x' y' x y. z = \langle \langle x', y' \rangle, x, y \rangle \wedge (\langle x', x \rangle \in r \vee x' = x \wedge \langle y', y \rangle \in s)$
 is_rmultP

relativize functional $\text{rmult rmult_rel external}$
relationalize $\text{rmult_rel is_rmult}$

lemma (in $M_trivial$) $\text{rmultP_abs}[\text{absolut}]: \llbracket M(r); M(s); M(z) \rrbracket \implies \text{is_rmultP}(M, s, r, z)$
 \longleftrightarrow
 $(\exists x' y' x y. z = \langle \langle x', y' \rangle, x, y \rangle \wedge (\langle x', x \rangle \in r \vee x' = x \wedge \langle y', y \rangle \in s))$
unfolding is_rmultP_def **by** (auto dest: transM)

definition

$is_csquare_rel :: [i \Rightarrow o, i, i] \Rightarrow o$ **where**
 $is_csquare_rel(M, K, cs) \equiv \exists K2[M]. \exists la[M]. \exists memK[M].$
 $\exists rmKK[M]. \exists rmKK2[M].$
 $cartprod(M, K, K, K2) \wedge is_csquare_lam(M, K, la) \wedge$
 $membership(M, K, memK) \wedge is_rmult(M, K, memK, K, memK, rmKK) \wedge$
 $is_rmult(M, K, memK, K2, rmKK, rmKK2) \wedge is_rvimage(M, K2, la, rmKK2, cs)$

context M_basic

begin

lemma $rvimage_abs[absolut]$:

assumes $M(A) \ M(f) \ M(r) \ M(z)$
shows $is_rvimage(M, A, f, r, z) \longleftrightarrow z = rvimage(A, f, r)$
using $assms \ transM[OF_ \langle M(A) \rangle]$
unfolding $is_rvimage_def \ rvimage_def$
by $auto$

lemma $rmult_abs \ [absolut]: \llbracket M(A); M(r); M(B); M(s); M(z) \rrbracket \Longrightarrow$

$is_rmult(M, A, r, B, s, z) \longleftrightarrow z = rmult(A, r, B, s)$
using $rmultP_abs \ transM[of_ (A \times B) \times A \times B]$
unfolding $is_rmultP_def \ is_rmult_def \ rmult_def$
by $(auto \ del: \text{iffI})$

lemma $csquare_lam_body_abs[absolut]: M(x) \Longrightarrow M(z) \Longrightarrow$

$is_csquare_lam_body(M, x, z) \longleftrightarrow z = \langle fst(x) \cup snd(x), fst(x), snd(x) \rangle$
unfolding $is_csquare_lam_body_def$ **by** $(simp \ add: absolut)$

lemma $csquare_lam_abs[absolut]: M(K) \Longrightarrow M(l) \Longrightarrow$

$is_csquare_lam(M, K, l) \longleftrightarrow l = (\lambda x \in K \times K. \langle fst(x) \cup snd(x), fst(x), snd(x) \rangle)$
unfolding $is_csquare_lam_def$
using $lambda_abs2[of \ K \times K \ is_csquare_lam_body(M)$
 $\lambda x. \langle fst(x) \cup snd(x), fst(x), snd(x) \rangle]$
unfolding $Relation1_def$ **by** $(simp \ add: absolut)$

lemma $csquare_lam_eq_lam: csquare_lam(K) = (\lambda z \in K \times K. \langle fst(z) \cup snd(z),$
 $fst(z), snd(z) \rangle)$

proof -

have $(\lambda \langle x, y \rangle \in K \times K. \langle x \cup y, x, y \rangle)'z =$
 $(\lambda z \in K \times K. \langle fst(z) \cup snd(z), fst(z), snd(z) \rangle)'z$ **if** $z \in K \times K$ **for** z
using $that$ **by** $auto$
then
show $?thesis$
unfolding $csquare_lam_def$
by $simp$

qed

end — M_basic

```

context M_pre_cardinal_arith
begin

lemma csquare_rel_closed[intro,simp]:  $M(K) \implies M(\text{csquare\_rel}(K))$ 
  using csquare_lam_replacement unfolding csquare_rel_def
  by (intro rvimage_closed lam_closed) (auto dest:transM)

lemma csquare_rel_abs[absolut]:  $\llbracket M(K); M(cs) \rrbracket \implies$ 
   $\text{is\_csquare\_rel}(M,K,cs) \longleftrightarrow cs = \text{csquare\_rel}(K)$ 
  unfolding is_csquare_rel_def csquare_rel_def
  using csquare_lam_closed[unfolded csquare_lam_eq_lam]
  by (simp add:absolut csquare_lam_eq_lam[unfolded csquare_lam_def])

end — M_pre_cardinal_arith

relativize functional csucc csucc_rel external
relationalize csucc_rel is_csucc
synthesize is_csucc from_definition assuming nonempty
arity_theorem for is_csucc_fm

abbreviation
   $\text{csucc\_r} :: [i,i \Rightarrow o] \Rightarrow i \ (\langle'(\_+)' \rangle)$  where
   $\text{csucc\_r}(x,M) \equiv \text{csucc\_rel}(M,x)$ 

abbreviation
   $\text{csucc\_r\_set} :: [i,i] \Rightarrow i \ (\langle'(\_+)' \rangle)$  where
   $\text{csucc\_r\_set}(x,M) \equiv \text{csucc\_rel}(\#\#M,x)$ 

context M_Perm
begin

rel_closed for csucc
  using Least_closed'[of  $\lambda L. M(L) \wedge \text{Card}^M(L) \wedge K < L$ ]
  unfolding csucc_rel_def
  by simp

is_iff_rel for csucc
  using least_abs'[of  $\lambda L. M(L) \wedge \text{Card}^M(L) \wedge K < L$  res]
  is_Card_iff
  unfolding is_csucc_def csucc_rel_def
  by (simp add:absolut)

end — M_Perm

notation csucc_rel ( $\langle' \text{csucc} -' (\_) \rangle$ )

```

```

context  $M\_cardinals$ 
begin

lemma  $Card\_rel\_Union$  [ $simp,intro,TC$ ]:
  assumes  $A: \bigwedge x. x \in A \implies Card^M(x)$  and
     $types:M(A)$ 
  shows  $Card^M(\bigcup(A))$ 
proof ( $rule\ Card\_relI$ )
  show  $Ord(\bigcup A)$  using  $A$ 
    by ( $simp\ add: Card\_rel\_is\_Ord\ types\ transM$ )
next
  fix  $j$ 
  assume  $j: j < \bigcup A$ 
  moreover from  $this$ 
  have  $M(j)$  unfolding  $lt\_def$  by ( $auto\ simp\ add:types\ dest:transM$ )
  from  $j$ 
  have  $\exists c \in A. j \in c \wedge Card^M(c)$  using  $A\ types$ 
    unfolding  $lt\_def$ 
    by ( $simp$ )
  then
  obtain  $c$  where  $c: c \in A\ j < c\ Card^M(c)\ M(c)$ 
    using  $Card\_rel\_is\_Ord\ types$  unfolding  $lt\_def$ 
    by ( $auto\ dest:transM$ )
  with  $\langle M(j) \rangle$ 
  have  $jls: j \prec^M c$ 
    by ( $simp\ add: lt\_Card\_rel\_imp\_lesspoll\_rel\ types$ )
  { assume  $eqp: j \approx^M \bigcup A$ 
    have  $c \lesssim^M \bigcup A$  using  $c$ 
      by ( $blast\ intro: subset\_imp\_lepoll\_rel\ types$ )
    also from  $types\ \langle M(j) \rangle$ 
    have  $\dots \approx^M j$  by ( $rule\_tac\ eqpoll\_rel\_sym\ [OF\ eqp]$ ) ( $simp\_all\ add:types$ )
    also have  $\dots \prec^M c$  by ( $rule\ jls$ )
    finally have  $c \prec^M c$  by ( $simp\_all\ add:\langle M(c) \rangle\ \langle M(j) \rangle\ types$ )
    with  $\langle M(c) \rangle$ 
    have  $False$ 
      by ( $auto\ dest:lesspoll\_rel\_irrefl$ )
  } thus  $\neg j \approx^M \bigcup A$  by  $blast$ 
qed ( $simp\_all\ add:types$ )

```

```

lemma  $in\_Card\_imp\_lesspoll$ :  $[\mid Card^M(K); b \in K; M(K); M(b) \mid] \implies b \prec^M K$ 
apply ( $unfold\ lesspoll\_rel\_def$ )
apply ( $simp\ add: Card\_rel\_iff\_initial$ )
apply ( $fast\ intro!: le\_imp\_lepoll\_rel\ ltI\ leI$ )
done

```


16.1 Cardinal addition

Note (Paulson): Could omit proving the algebraic laws for cardinal addition and multiplication. On finite cardinals these operations coincide with addition and multiplication of natural numbers; on infinite cardinals they coincide with union (maximum). Either way we get most laws for free.

16.1.1 Cardinal addition is commutative

lemma *sum_commute_eqpoll_rel*: $M(A) \implies M(B) \implies A+B \approx^M B+A$

proof (*simp add: def_eqpoll_rel, rule rexI*)

show $(\lambda z \in A+B. \text{case}(Inr, Inl, z)) \in \text{bij}(A+B, B+A)$

by (*auto intro: lam_bijective [where d = case(Inr, Inl)]*)

assume $M(A) \ M(B)$

then

show $M(\lambda z \in A+B. \text{case}(Inr, Inl, z))$

using *case_replacement1*

by (*rule_tac lam_closed*) (*auto dest:transM*)

qed

lemma *cadd_rel_commute*: $M(i) \implies M(j) \implies i \oplus^M j = j \oplus^M i$

apply (*unfold cadd_rel_def*)

apply (*auto intro: sum_commute_eqpoll_rel [THEN cardinal_rel_cong]*)

done

16.1.2 Cardinal addition is associative

lemma *sum_assoc_eqpoll_rel*: $M(A) \implies M(B) \implies M(C) \implies (A+B)+C \approx^M A+(B+C)$

apply (*simp add: def_eqpoll_rel*)

apply (*rule rexI*)

apply (*rule sum_assoc_bij*)

using *case_replacement2*

by (*rule_tac lam_closed*) (*auto dest:transM*)

Unconditional version requires AC

lemma *well_ord_cadd_rel_assoc*:

assumes $i: \text{well_ord}(i, ri)$ **and** $j: \text{well_ord}(j, rj)$ **and** $k: \text{well_ord}(k, rk)$

and

types: $M(i) \ M(ri) \ M(j) \ M(rj) \ M(k) \ M(rk)$

shows $(i \oplus^M j) \oplus^M k = i \oplus^M (j \oplus^M k)$

proof (*simp add: assms cadd_rel_def, rule cardinal_rel_cong*)

from *types*

have $|i + j|^M + k \approx^M (i + j) + k$

by (*auto intro!: sum_eqpoll_rel_cong well_ord_cardinal_rel_eqpoll_rel_eqpoll_rel_refl well_ord_radd i j*)

also have $\dots \approx^M i + (j + k)$

by (*rule sum_assoc_eqpoll_rel*) (*simp_all add:types*)

also

```

have ...  $\approx^M i + |j + k|^M$ 
proof (auto intro!: sum_eqpoll_rel_cong intro: eqpoll_rel_refl simp add: types)
  from types
  have  $|j + k|^M \approx^M j + k$ 
    using well_ord_cardinal_rel_eqpoll_rel[OF well_ord_radd, OF j k]
    by (simp)
  with types
  show  $j + k \approx^M |j + k|^M$ 
    using eqpoll_rel_sym by simp
qed
finally show  $|i + j|^M + k \approx^M i + |j + k|^M$  by (simp_all add: types)
qed (simp_all add: types)

```

16.1.3 0 is the identity for addition

```

lemma case_id_eq:  $x \in \text{sum}(A, B) \implies \text{case}(\lambda z. z, \lambda z. z, x) = \text{snd}(x)$ 
  unfolding case_def cond_def by (auto simp: Inl_def Inr_def)

lemma lam_case_id:  $(\lambda z \in 0 + A. \text{case}(\lambda x. x, \lambda y. y, z)) = (\lambda z \in 0 + A. \text{snd}(z))$ 
  using case_id_eq by simp

```

```

lemma sum_0_eqpoll_rel:  $M(A) \implies 0 + A \approx^M A$ 
  apply (simp add: def_eqpoll_rel)
  apply (rule rexI)
  apply (rule bij_0_sum, subst lam_case_id)
  using lam_replacement_snd[unfolded lam_replacement_def]
  by (rule lam_closed)
  (auto simp add: case_def cond_def Inr_def dest: transM)

```

```

lemma cadd_rel_0 [simp]:  $\text{Card}^M(K) \implies M(K) \implies 0 \oplus^M K = K$ 
  apply (simp add: cadd_rel_def)
  apply (simp add: sum_0_eqpoll_rel [THEN cardinal_rel_cong] Card_rel_cardinal_rel_eq)
  done

```

16.1.4 Addition by another cardinal

```

lemma sum_lepoll_rel_self:  $M(A) \implies M(B) \implies A \lesssim^M A + B$ 
proof (simp add: def_lepoll_rel, rule rexI)
  show  $(\lambda x \in A. \text{Inl}(x)) \in \text{inj}(A, A + B)$ 
    by (simp add: inj_def)
  assume  $M(A) \ M(B)$ 
  then
  show  $M(\lambda x \in A. \text{Inl}(x))$ 
    using Inl_replacement1 transM[OF _  $\langle M(A) \rangle$ ]
    by (rule_tac lam_closed) (auto simp add: Inl_def)
qed

```

```

lemma cadd_rel_le_self:

```

```

assumes  $K: \text{Card}^M(K)$  and  $L: \text{Ord}(L)$  and
   $\text{types}: M(K) \ M(L)$ 
shows  $K \leq (K \oplus^M L)$ 
proof (simp add: types cadd_rel_def)
  have  $K \leq |K|^M$ 
    by (rule Card_rel_cardinal_rel_le [OF K]) (simp add: types)
  moreover have  $|K|^M \leq |K + L|^M$  using  $K \ L$ 
    by (blast intro: well_ord_lepoll_rel_imp_cardinal_rel_le sum_lepoll_rel_self
      well_ord_radd well_ord_Memrel Card_rel_is_Ord types)
  ultimately show  $K \leq |K + L|^M$ 
    by (blast intro: le_trans)
qed

```

16.1.5 Monotonicity of addition

```

lemma sum_lepoll_rel_mono:
   $[[ A \lesssim^M C; B \lesssim^M D; M(A); M(B); M(C); M(D) ]] \implies A + B \lesssim^M C + D$ 
apply (simp add: def_lepoll_rel)
apply (elim rexE)
apply (rule_tac x =  $\lambda z \in A+B. \text{case } (\%w. \text{Inl}(f'w), \%y. \text{Inr}(fa'y), z)$  in rexI)
apply (rule_tac d = case ( $\%w. \text{Inl}(\text{converse}(f) 'w), \%y. \text{Inr}(\text{converse}(fa) 'y)$ )
  in lam_injective)
apply (typecheck add: inj_is_fun, auto)
apply (rule_tac lam_closed, auto dest: transM intro: case_replacement4)
done

```

```

lemma cadd_rel_le_mono:
   $[[ K' \leq K; L' \leq L; M(K'); M(K); M(L'); M(L) ]] \implies (K' \oplus^M L') \leq (K \oplus^M L)$ 
apply (unfold cadd_rel_def)
apply (safe dest!: le_subset_iff [THEN iffD1])
apply (rule well_ord_lepoll_rel_imp_cardinal_rel_le)
apply (blast intro: well_ord_radd well_ord_Memrel)
apply (auto intro: sum_lepoll_rel_mono subset_imp_lepoll_rel)
done

```

16.1.6 Addition of finite cardinals is "ordinary" addition

```

lemma sum_succ_eqpoll_rel:  $M(A) \implies M(B) \implies \text{succ}(A) + B \approx^M \text{succ}(A+B)$ 
apply (simp add: def_eqpoll_rel)
apply (rule rexI)
apply (rule_tac c =  $\%z. \text{if } z = \text{Inl } (A) \text{ then } A+B \text{ else } z$ 
  and d =  $\%z. \text{if } z = A+B \text{ then } \text{Inl } (A) \text{ else } z$  in lam_bijective)
apply simp_all
apply (blast dest: sym [THEN eq_imp_not_mem] elim: mem_irrefl) +
apply (rule_tac lam_closed, auto dest: transM intro: if_then_range_replacement2)
done

```

```

lemma cadd_succ_lemma:
  assumes Ord(m) Ord(n) and
    types: M(m) M(n)
  shows succ(m)  $\oplus^M$  n = |succ(m  $\oplus^M$  n)|M
  using types
proof (simp add: cadd_rel_def)
  have [intro]: m + n  $\approx^M$  |m + n|M using assms
  by (blast intro: eqpoll_rel_sym well_ord_cardinal_rel_eqpoll_rel well_ord_radd
    well_ord_Memrel)

  have |succ(m) + n|M = |succ(m + n)|M
  by (rule sum_succ_eqpoll_rel [THEN cardinal_rel_cong]) (simp_all add: types)
  also have ... = |succ(|m + n|M)|M
  by (blast intro: succ_eqpoll_rel_cong cardinal_rel_cong types)
  finally show |succ(m) + n|M = |succ(|m + n|M)|M.
qed

```

```

lemma nat_cadd_rel_eq_add:
  assumes m: m ∈ nat and [simp]: n ∈ nat shows m  $\oplus^M$  n = m #+ n
  using m
proof (induct m)
  case 0 thus ?case
    using transM[OF M_nat]
    by (auto simp add: nat_into_Card_rel)
next
  case (succ m) thus ?case
    using transM[OF M_nat]
    by (simp add: cadd_succ_lemma nat_into_Card_rel Card_rel_cardinal_rel_eq)
qed

```

16.2 Cardinal multiplication

16.2.1 Cardinal multiplication is commutative

```

lemma prod_commute_eqpoll_rel: M(A)  $\implies$  M(B)  $\implies$  A*B  $\approx^M$  B*A
apply (simp add: def_eqpoll_rel)
apply (rule rexI)
apply (rule_tac c = %<x,y>.<y,x> and d = %<x,y>.<y,x> in lam_bijective,
  auto)
apply (rule_tac lam_closed, auto intro: swap_replacement dest: transM)
done

```

```

lemma cmult_rel_commute: M(i)  $\implies$  M(j)  $\implies$  i  $\otimes^M$  j = j  $\otimes^M$  i
apply (unfold cmult_rel_def)
apply (rule prod_commute_eqpoll_rel [THEN cardinal_rel_cong], simp_all)
done

```

16.2.2 Cardinal multiplication is associative

```

lemma prod_assoc_eqpoll_rel:  $M(A) \implies M(B) \implies M(C) \implies (A*B)*C \approx^M A*(B*C)$ 
apply (simp add: def_eqpoll_rel)
apply (rule rexI)
apply (rule prod_assoc_bij)
  apply(rule_tac lam_closed, auto intro:assoc_replacement dest:transM)
done

```

Unconditional version requires AC

```

lemma well_ord_cmult_rel_assoc:
  assumes  $i$ : well_ord( $i, ri$ ) and  $j$ : well_ord( $j, rj$ ) and  $k$ : well_ord( $k, rk$ )
  and
    types:  $M(i) \ M(ri) \ M(j) \ M(rj) \ M(k) \ M(rk)$ 
  shows  $(i \otimes^M j) \otimes^M k = i \otimes^M (j \otimes^M k)$ 
proof (simp add: asms_cmult_rel_def, rule cardinal_rel_cong)
  have  $|i * j|^M * k \approx^M (i * j) * k$ 
    by (auto intro!: prod_eqpoll_rel_cong
      well_ord_cardinal_rel_eqpoll_rel_eqpoll_rel_refl
      well_ord_rmult  $i \ j$  simp add:types)
  also have  $\dots \approx^M i * (j * k)$ 
    by (rule prod_assoc_eqpoll_rel, simp_all add:types)
  also have  $\dots \approx^M i * |j * k|^M$ 
    by (blast intro: prod_eqpoll_rel_cong well_ord_cardinal_rel_eqpoll_rel
      eqpoll_rel_refl well_ord_rmult  $j \ k$  eqpoll_rel_sym types)
  finally show  $|i * j|^M * k \approx^M i * |j * k|^M$  by (simp add:types)
qed (simp_all add:types)

```

16.2.3 Cardinal multiplication distributes over addition

```

lemma sum_prod_distrib_eqpoll_rel:  $M(A) \implies M(B) \implies M(C) \implies (A+B)*C \approx^M (A*C)+(B*C)$ 
apply (simp add: def_eqpoll_rel)
apply (rule rexI)
  apply (rule sum_prod_distrib_bij)
  apply(rule_tac lam_closed, auto intro:case_replacement5 dest:transM)
done

```

```

lemma well_ord_cadd_cmult_distrib:
  assumes  $i$ : well_ord( $i, ri$ ) and  $j$ : well_ord( $j, rj$ ) and  $k$ : well_ord( $k, rk$ )
  and
    types:  $M(i) \ M(ri) \ M(j) \ M(rj) \ M(k) \ M(rk)$ 
  shows  $(i \oplus^M j) \otimes^M k = (i \otimes^M k) \oplus^M (j \otimes^M k)$ 
proof (simp add: asms_cadd_rel_def cmult_rel_def, rule cardinal_rel_cong)
  have  $|i + j|^M * k \approx^M (i + j) * k$ 
    by (blast intro: prod_eqpoll_rel_cong well_ord_cardinal_rel_eqpoll_rel
      eqpoll_rel_refl well_ord_radd  $i \ j$  types)
  also have  $\dots \approx^M i * k + j * k$ 

```

```

    by (rule sum_prod_distrib_eqpoll_rel) (simp_all add:types)
  also have ...  $\approx^M |i * k|^M + |j * k|^M$ 
    by (blast intro: sum_eqpoll_rel_cong well_ord_cardinal_rel_eqpoll_rel
        well_ord_rmult i j k eqpoll_rel_sym types)
  finally show  $|i + j|^M * k \approx^M |i * k|^M + |j * k|^M$  by (simp add:types)
qed (simp_all add:types)

```

16.2.4 Multiplication by 0 yields 0

```

lemma prod_0_eqpoll_rel:  $M(A) \implies 0 * A \approx^M 0$ 
apply (simp add: def_eqpoll_rel)
apply (rule rexI)
apply (rule lam_bijective, auto)
done

```

```

lemma cmult_rel_0 [simp]:  $M(i) \implies 0 \otimes^M i = 0$ 
by (simp add: cmult_rel_def prod_0_eqpoll_rel [THEN cardinal_rel_cong])

```

16.2.5 1 is the identity for multiplication

```

lemma prod_singleton_eqpoll_rel:  $M(x) \implies M(A) \implies \{x\} * A \approx^M A$ 
apply (simp add: def_eqpoll_rel)
apply (rule rexI)
  apply (rule singleton_prod_bij [THEN bij_converse_bij])
  apply (rule converse_closed)
  apply (rule_tac lam_closed, auto intro:prepend_replacement dest:transM)
done

```

```

lemma cmult_rel_1 [simp]:  $\text{Card}^M(K) \implies M(K) \implies 1 \otimes^M K = K$ 
apply (simp add: cmult_rel_def succ_def)
apply (simp add: prod_singleton_eqpoll_rel [THEN cardinal_rel_cong] Card_rel_cardinal_rel_eq)
done

```

16.3 Some inequalities for multiplication

```

lemma prod_square_lepoll_rel:  $M(A) \implies A \lesssim^M A * A$ 
apply (simp add: def_lepoll_rel inj_def)
apply (rule_tac x =  $\lambda x \in A. \langle x, x \rangle$  in rexI, simp)
  apply (rule_tac lam_closed, auto intro: id_replacement dest:transM)
done

```

```

lemma cmult_rel_square_le:  $\text{Card}^M(K) \implies M(K) \implies K \leq K \otimes^M K$ 
apply (unfold cmult_rel_def)
apply (rule le_trans)
apply (rule_tac [2] well_ord_lepoll_rel_imp_cardinal_rel_le)
  apply (rule_tac [3] prod_square_lepoll_rel)
apply (simp add: le_refl Card_rel_is_Ord Card_rel_cardinal_rel_eq)
  apply (blast intro: well_ord_rmult well_ord_Memrel Card_rel_is_Ord)
  apply simp_all

```

done

16.3.1 Multiplication by a non-zero cardinal

```
lemma prod_lepoll_rel_self:  $b \in B \implies M(b) \implies M(B) \implies M(A) \implies A \lesssim^M A * B$ 
apply (simp add: def_lepoll_rel inj_def)
apply (rule_tac x =  $\lambda x \in A. \langle x, b \rangle$  in rexI, simp)
  apply (rule_tac lam_closed, auto intro: pospend_replacement dest: transM)
done
```

```
lemma cmult_rel_le_self:
  [|  $Card^M(K); Ord(L); 0 < L; M(K); M(L)$  |] ==>  $K \leq (K \otimes^M L)$ 
apply (unfold cmult_rel_def)
apply (rule le_trans [OF Card_cardinal_rel_le well_ord_lepoll_rel_imp_cardinal_rel_le])
  apply assumption apply simp
  apply (blast intro: well_ord_rmult well_ord_Memrel Card_rel_is_Ord)
apply (auto intro: prod_lepoll_rel_self ltD)
done
```

16.3.2 Monotonicity of multiplication

```
lemma prod_lepoll_rel_mono:
  [|  $A \lesssim^M C; B \lesssim^M D; M(A); M(B); M(C); M(D)$  |] ==>  $A * B \lesssim^M C * D$ 
apply (simp add: def_lepoll_rel)
apply (elim rexE)
apply (rule_tac x =  $\text{lam } \langle w, y \rangle : A * B. \langle f'w, fa'y \rangle$  in rexI)
apply (rule_tac d =  $\% \langle w, y \rangle. \langle \text{converse } (f) 'w, \text{converse } (fa) 'y \rangle$ 
  in lam_injective)
  apply (typecheck add: inj_is_fun, auto)
  apply (rule_tac lam_closed, auto intro: prod_fun_replacement dest: transM)
done
```

```
lemma cmult_rel_le_mono:
  [|  $K' \leq K; L' \leq L; M(K'); M(K); M(L'); M(L)$  |] ==>  $(K' \otimes^M L') \leq (K \otimes^M L)$ 
apply (unfold cmult_rel_def)
apply (safe dest!: le_subset_iff [THEN iffD1])
apply (rule well_ord_lepoll_rel_imp_cardinal_rel_le)
  apply (blast intro: well_ord_rmult well_ord_Memrel)
apply (auto intro: prod_lepoll_rel_mono subset_imp_lepoll_rel)
done
```

16.4 Multiplication of finite cardinals is "ordinary" multiplication

```
lemma prod_succ_eqpoll_rel:  $M(A) \implies M(B) \implies \text{succ}(A) * B \approx^M B + A * B$ 
apply (simp add: def_eqpoll_rel)
apply (rule rexI)
apply (rule_tac c =  $\lambda p. \text{if fst}(p) = A \text{ then Inl } (\text{snd}(p)) \text{ else Inr } (p)$ )
```

```

      and d = case (%y. <A,y>, %z. z) in lam_bijective)
apply safe
  apply (simp_all add: succI2 if_type mem_imp_not_eq)
  apply (rule_tac lam_closed, auto intro: Inl_replacement2 dest:transM)
done

lemma cmult_rel_succ_lemma:
  [| Ord(m); Ord(n) ; M(m); M(n) |] ==> succ(m)  $\otimes^M$  n = n  $\oplus^M$  (m  $\otimes^M$  n)
apply (simp add: cmult_rel_def cadd_rel_def)
apply (rule prod_succ_eqpoll_rel [THEN cardinal_rel_cong, THEN trans], simp_all)
apply (rule cardinal_rel_cong [symmetric], simp_all)
apply (rule sum_eqpoll_rel_cong [OF eqpoll_rel_refl well_ord_cardinal_rel_eqpoll_rel],
  assumption)
  apply (blast intro: well_ord_rmult well_ord_Memrel)
  apply simp_all
done

lemma nat_cmult_rel_eq_mult: [| m  $\in$  nat; n  $\in$  nat |] ==> m  $\otimes^M$  n = m#*n
  using transM[OF M_nat]
apply (induct_tac m)
apply (simp_all add: cmult_rel_succ_lemma nat_cadd_rel_eq_add)
done

lemma cmult_rel_2: CardM(n)  $\implies$  M(n)  $\implies$  2  $\otimes^M$  n = n  $\oplus^M$  n
  by (simp add: cmult_rel_succ_lemma Card_rel_is_Ord cadd_rel_commute [of
    _ 0])

lemma sum_lepoll_rel_prod:
  assumes C: 2  $\lesssim^M$  C and
    types:M(C) M(B)
  shows B+B  $\lesssim^M$  C*B
proof -
  have B+B  $\lesssim^M$  2*B
  by (simp add: sum_eq_2_times types)
  also have ...  $\lesssim^M$  C*B
  by (blast intro: prod_lepoll_rel_mono lepoll_rel_refl C types)
  finally show B+B  $\lesssim^M$  C*B by (simp_all add:types)
qed

lemma lepoll_imp_sum_lepoll_prod: [| A  $\lesssim^M$  B; 2  $\lesssim^M$  A; M(A) ;M(B) |] ==>
  A+B  $\lesssim^M$  A*B
by (blast intro: sum_lepoll_rel_mono sum_lepoll_rel_prod lepoll_rel_trans lepoll_rel_refl)

end — M_cardinals

```

16.5 Infinite Cardinals are Limit Ordinals

context M_pre_cardinal_arith

begin

lemma *nat_cons_lepoll_rel*: $\text{nat} \lesssim^M A \implies M(A) \implies M(u) \implies \text{cons}(u, A) \lesssim^M A$

apply (*simp add: def_lepoll_rel*)

apply (*erule rexE*)

apply (*rule_tac x =*

λz ∈ cons (u, A).

if z = u then f'0

else if z ∈ range (f) then f'succ (converse (f) 'z) else z

in *rexI*)

apply (*rule_tac d =*

%y. if y ∈ range(f) then nat_case (u, %z. f'z, converse(f) 'y)

else y

in *lam_injective*)

apply (*fast intro!: if_type apply_type intro: inj_is_fun inj_converse_fun*)

apply (*simp add: inj_is_fun [THEN apply_rangeI]*

inj_converse_fun [THEN apply_rangeI]

inj_converse_fun [THEN apply_funtype])

proof -

fix *f*

assume *M(A) M(f) M(u)*

then

show *M(λz ∈ cons(u, A). if z = u then f ' 0 else if z ∈ range(f) then f ' succ(converse(f) ' z) else z)*

using *if_then_range_replacement transM[OF _ ⟨M(A)⟩]*

by (*rule_tac lam_closed, auto*)

qed

lemma *nat_cons_eqpoll_rel*: $\text{nat} \lesssim^M A \implies M(A) \implies M(u) \implies \text{cons}(u, A) \approx^M A$

apply (*erule nat_cons_lepoll_rel [THEN eqpoll_rel], assumption+*)

apply (*rule subset_consI [THEN subset_imp_lepoll_rel], simp_all*)

done

lemma *nat_succ_eqpoll_rel*: $\text{nat} \subseteq A \implies M(A) \implies \text{succ}(A) \approx^M A$

apply (*unfold succ_def*)

apply (*erule subset_imp_lepoll_rel [THEN nat_cons_eqpoll_rel], simp_all*)

done

lemma *InfCard_rel_nat*: $\text{InfCard}^M(\text{nat})$

apply (*simp add: InfCard_rel_def*)

apply (*blast intro: Card_rel_nat Card_rel_is_Ord*)

done

lemma *InfCard_rel_is_Card_rel*: $M(K) \implies \text{InfCard}^M(K) \implies \text{Card}^M(K)$

apply (*simp add: InfCard_rel_def*)

done

```

lemma InfCard_rel_Un:
  [| InfCardM(K); CardM(L); M(K); M(L) |] ==> InfCardM(K ∪ L)
apply (simp add: InfCard_rel_def)
apply (simp add: Card_rel_Un Un_upper1_le [THEN [2] le_trans] Card_rel_is_Ord)
done

lemma InfCard_rel_is_Limit: InfCardM(K) ==> M(K) ==> Limit(K)
apply (simp add: InfCard_rel_def)
apply (erule conjE)
apply (frule Card_rel_is_Ord, assumption)
apply (rule ltI [THEN non_succ_LimitI])
apply (erule le_imp_subset [THEN subsetD])
apply (safe dest!: Limit_nat [THEN Limit_le_succD])
  apply (unfold Card_rel_def)
  apply (drule trans)
apply (erule le_imp_subset [THEN nat_succ_eqpoll_rel, THEN cardinal_rel_cong],
simp_all)
apply (erule Ord_cardinal_rel_le [THEN lt_trans2, THEN lt_irrefl], assumption)
apply (rule le_eqI) prefer 2
apply (rule Ord_cardinal_rel, assumption+)
done

end — M_pre_cardinal_arith

```

```

lemma (in M_ordertype) ordertype_abs[absolut]:
  [| wellordered(M, A, r); M(A); M(r); M(i) |] ==>
  otype(M, A, r, i) ↔ i = ordertype(A, r)
proof (intro iffI)
  assume wellordered(M, A, r) M(A) M(r) M(i) otype(M, A, r, i)
  moreover from this
  obtain f j where M(f) M(j) Ord(j) f ∈ ⟨A, r⟩ ≅ ⟨j, Memrel(j)⟩
    using ordertype_exists[of A r] by auto
  moreover from calculation
  have ∃ f[M]. f ∈ ⟨A, r⟩ ≅ ⟨j, Memrel(j)⟩ by auto
  moreover
  have ∃ f[M]. f ∈ ⟨A, r⟩ ≅ ⟨i, Memrel(i)⟩
  proof -
    note calculation
    moreover from this
    obtain h where omap(M, A, r, h) M(h)
      using omap_exists by auto
    moreover from calculation
    have h ∈ ⟨A, r⟩ ≅ ⟨i, Memrel(i)⟩
      using omap_ord_iso obase_equals by simp
    moreover from calculation
    have h O converse(f) ∈ ⟨j, Memrel(j)⟩ ≅ ⟨i, Memrel(i)⟩

```

```

    using ord_iso_sym ord_iso_trans by blast
  moreover from calculation
  have i=j
    using Ord_iso_implies_eq[of j i h O converse(f)]
    Ord_otype[OF well_ord_is_trans_on] by simp
  ultimately
  show ?thesis by simp
qed
ultimately
show i = ordertype(A, r)
  by (force intro:ordetypes_are_absolute[of A r i]
    simp add:Ord_otype[OF well_ord_is_trans_on])
next
assume wellordered(M,A, r) i = ordertype(A, r)
  M(i) M(A) M(r)
  moreover from this
  obtain h where omap(M, A, r, h) M(h)
    using omap_exists by auto
  moreover from calculation
  obtain j where otype(M,A,r,j) M(j)
    using otype_exists by auto
  moreover from calculation
  have h ∈ ⟨A, r⟩ ≅ ⟨j, Memrel(j)⟩
    using omap_ord_iso_otype by simp
  moreover from calculation
  obtain f where f ∈ ⟨A, r⟩ ≅ ⟨i, Memrel(i)⟩
    using ordertype_ord_iso by auto
  moreover
  have j=i
  proof -
    note calculation
    moreover from this
    have h O converse(f) ∈ ⟨i, Memrel(i)⟩ ≅ ⟨j, Memrel(j)⟩
      using ord_iso_sym ord_iso_trans by blast
    moreover from calculation
    have Ord(i) using Ord_ordertype by simp
    ultimately
    show j=i
      using Ord_iso_implies_eq[of i j h O converse(f)]
      Ord_otype[OF well_ord_is_trans_on] by simp
  qed
  ultimately
  show otype(M, A, r, i) by simp
qed

lemma (in M_ordertype) ordertype_closed[intro,simp]: [ wellordered(M,A,r);M(A);M(r)]
  ⇒ M(ordertype(A,r))
  using ordetypes_exists ordetypes_are_absolute by blast

```

relationalize *transitive_rel is_transitive* **external**
synthesize *is_transitive* **from_definition** **assuming** *nonempty*

lemma (in *M_trivial*) *is_transitive_iff_transitive_rel*:
 $M(A) \implies M(r) \implies \text{transitive_rel}(M, A, r) \longleftrightarrow \text{is_transitive}(M, A, r)$
unfolding *transitive_rel_def is_transitive_def* **by** *simp*

relationalize *linear_rel is_linear* **external**
synthesize *is_linear* **from_definition** **assuming** *nonempty*

lemma (in *M_trivial*) *is_linear_iff_linear_rel*:
 $M(A) \implies M(r) \implies \text{is_linear}(M, A, r) \longleftrightarrow \text{linear_rel}(M, A, r)$
unfolding *linear_rel_def is_linear_def* **by** *simp*

relationalize *wellfounded_on is_wellfounded_on* **external**
synthesize *is_wellfounded_on* **from_definition** **assuming** *nonempty*

lemma (in *M_trivial*) *is_wellfounded_on_iff_wellfounded_on*:
 $M(A) \implies M(r) \implies \text{is_wellfounded_on}(M, A, r) \longleftrightarrow \text{wellfounded_on}(M, A, r)$
unfolding *wellfounded_on_def is_wellfounded_on_def* **by** *simp*

definition

is_well_ord :: $[i=>o, i, i] => o$ **where**
— linear and wellfounded on *A*
 $\text{is_well_ord}(M, A, r) ==$
 $\text{is_transitive}(M, A, r) \wedge \text{is_linear}(M, A, r) \wedge \text{is_wellfounded_on}(M, A, r)$

lemma (in *M_trivial*) *is_well_ord_iff_wellordered*:
 $M(A) \implies M(r) \implies \text{is_well_ord}(M, A, r) \longleftrightarrow \text{wellordered}(M, A, r)$
using *is_wellfounded_on_iff_wellfounded_on is_linear_iff_linear_rel*
is_transitive_iff_transitive_rel
unfolding *wellordered_def is_well_ord_def* **by** *simp*

reldb_add relational *well_ord is_well_ord*
reldb_add functional *well_ord well_ord*
synthesize *is_well_ord* **from_definition** **assuming** *nonempty*

— One keyword (functional or relational) means going from an absolute term to that kind of term

reldb_add relational *Order.pred pred_set*

— The following form (twice the same argument) is only correct when an “_abs” theorem is available

reldb_add functional *Order.pred Order.pred*

relativize functional *ord_iso ord_iso_rel external*
 — The following corresponds to "relativize functional relational"
relationalize *ord_iso_rel is_ord_iso*

context *M_pre_cardinal_arith*
begin

is_iff_rel **for** *ord_iso*
using *bij_rel_iff*
unfolding *is_ord_iso_def ord_iso_rel_def*
by *simp*

rel_closed **for** *ord_iso*
using *ord_iso_separation* **unfolding** *ord_iso_rel_def*
by *simp*

end — *M_pre_cardinal_arith*

synthesize *is_ord_iso* **from_definition** **assuming** *nonempty*

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 [|*a0 a1 a2 a3 a4*. [|*a0* ∈ ?*A*; *a1* ∈ ?*A*; *a2* ∈ ?*A*; *a3* ∈ ?*A*; *a4* ∈ ?*A*]
 ==> ?*MH(a2, a1, a0)* ↔ ?*A, Cons(a0, Cons(a1, Cons(a2, Cons(a3, Cons(a4, env))))* ⊨ ?*p*; ?*x* ∈ *nat*; ?*y* < *length(?env)*; ?*z* < *length(?env)*; ?*env* ∈ *list(?A)*]
 ==> (?*A, ?env* ⊨ *is_wfrec_fm(?p, ?x, ?y, ?z)*) ↔ *is_wfrec(##?A, ?MH, nth(?x, ?env), nth(?y, ?env), nth(?z, ?env))*, 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*) \longleftrightarrow *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*) \longleftrightarrow *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*) \longleftrightarrow *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*) \longleftrightarrow *Aa, env* ⊨ *is_wfrec_fm*(*p, x, y, z*)

using *assms* *sats_is_wfrec_fm'*[*OF assms*] **unfolding** *is_wfrec_on_def* **by** *simp*

lemma *trans_on_iff_trans*: *trans*[*A*](*r*) \longleftrightarrow *trans*(*r* ∩ *A* × *A*)

unfolding *trans_on_def trans_def* **by** *auto*

lemma *trans_on_subset*: *trans*[*A*](*r*) ==> *B* ⊆ *A* ==> *trans*[*B*](*r*)

unfolding *trans_on_def*

by *auto*

lemma *relation_Int*: *relation*(*r* ∩ *B* × *B*)

unfolding *relation_def*

by *auto*

Discipline for *ordermap*

relativize functional *ordermap ordermap_rel* **external**

relationalize *ordermap_rel is_ordermap*

context *M_pre_cardinal_arith*

begin

lemma *wfrec_on_pred_eq*:

```

assumes  $r \in Pow(A \times A) \ M(A) \ M(r)$ 
shows  $wfrec[A](r, x, \lambda x f. f \text{ `` } Order.pred(A, x, r)) = wfrec(r, x, \lambda x f. f \text{ `` } Order.pred(A, x, r))$ 
proof -
  from  $\langle r \in Pow(A \times A) \rangle$ 
  have  $r \cap A \times A = r$  by auto
  moreover from this
  show ?thesis
  unfolding wfrec_on_def by simp
qed

```

```

lemma wfrec_on_pred_closed:
assumes  $wf[A](r) \ trans[A](r) \ r \in Pow(A \times A) \ M(A) \ M(r) \ x \in A$ 
shows  $M(wfrec(r, x, \lambda x f. f \text{ `` } Order.pred(A, x, r)))$ 
proof -
  from assms
  have  $wfrec[A](r, x, \lambda x f. f \text{ `` } Order.pred(A, x, r)) = wfrec(r, x, \lambda x f. f \text{ `` } Order.pred(A, x, r))$ 
  using wfrec_on_pred_eq by simp
  moreover from assms
  have  $M(wfrec(r, x, \lambda x f. f \text{ `` } Order.pred(A, x, r)))$ 
  using wfrec_pred_replacement wf_on_imp wftrans_on_imp_trans subset_Sigma_imp_relation
  by (rule_tac MH =  $\lambda x f b. \exists a[M]. image(M, f, a, b) \wedge pred\_set(M, A, x, r, a)$ )
in trans_wfrec_closed
  (auto dest: transM simp: relation2_def)
  ultimately
  show ?thesis by simp
qed

```

```

lemma wfrec_on_pred_closed':
assumes  $wf[A](r) \ trans[A](r) \ r \in Pow(A \times A) \ M(A) \ M(r) \ x \in A$ 
shows  $M(wfrec[A](r, x, \lambda x f. f \text{ `` } Order.pred(A, x, r)))$ 
using assms wfrec_on_pred_closed wfrec_on_pred_eq by simp

```

```

lemma ordermap_rel_closed':
assumes  $wf[A](r) \ trans[A](r) \ r \in Pow(A \times A) \ M(A) \ M(r)$ 
shows  $M(ordermap\_rel(M, A, r))$ 
proof -
  from assms
  have  $r \cap A \times A = r$  by auto
  with assms have  $wf(r) \ trans(r) \ relation(r)$ 
  unfolding wf_on_def using trans_on_iff_trans relation_def by auto
  then
  have  $1: \bigwedge x z. M(x) \implies M(z) \implies$ 
    ( $\exists y[M]. pair(M, x, y, z) \wedge is\_wfrec(M, \lambda x f z. z = f \text{ `` } Order.pred(A, x, r), r, x, y)$ )
     $\longleftrightarrow$ 
     $z = \langle x, wfrec(r, x, \lambda x f. f \text{ `` } Order.pred(A, x, r)) \rangle$ 

```

```

using trans_wfrec_abs[of r, where
  H =  $\lambda x f. f \text{ `` } \text{Order.pred}(A, x, r)$  and
  MH =  $\lambda x f z. z = f \text{ `` } \text{Order.pred}(A, x, r), \text{simplified}$ ] assms
  wfrec_pred_replacement unfolding relation2_def
by auto
then
have strong_replacement(M,  $\lambda x z. z = \langle x, wfrec(r, x, \lambda x f. f \text{ `` } \text{Order.pred}(A, x,$ 
r)) \rangle)
  using strong_replacement_cong[of M, OF 1, THEN iffD1, OF _ _
    wfrec_pred_replacement[unfolded wfrec_replacement_def]] assms by simp
then show ?thesis
  using Pow_iff assms
  unfolding ordermap_rel_def
  apply(subst lam_cong[OF refl wfrec_on_pred_eq], simp_all)
  using wfrec_on_pred_closed lam_closed
  by simp
qed

lemma ordermap_rel_closed[intro, simp]:
  assumes wf[A](r) trans[A](r) r  $\in Pow(A \times A)$ 
  shows  $M(A) \implies M(r) \implies M(\text{ordermap\_rel}(M, A, r))$ 
  using ordermap_rel_closed' assms by simp

lemma is_ordermap_iff:
  assumes r  $\in Pow(A \times A)$  wf[A](r) trans[A](r)
  M(A) M(r) M(res)
  shows  $\text{is\_ordermap}(M, A, r, \text{res}) \longleftrightarrow \text{res} = \text{ordermap\_rel}(M, A, r)$ 
proof -
  from  $\langle r \in Pow(A \times A) \rangle$ 
  have  $r \cap A \times A = r$  by auto
  with assms have  $1: wf(r) \text{ trans}(r) \text{ relation}(r)$ 
    unfolding wf_on_def using trans_on_iff_trans relation_def by auto
  from assms
  have  $r \cap A \times A = r \subseteq A \times A \langle x, y \rangle \in r \implies x \in A \wedge y \in A$  for x y by auto
  then
  show ?thesis
using ordermap_rel_closed[of r A] assms wfrec_on_pred_closed wfrec_pred_replacement
1
  unfolding is_ordermap_def ordermap_rel_def
  apply (rule_tac lambda_abs2)
  apply (simp_all add: Relation1_def)
  apply clarify
  apply (rule trans_wfrec_on_abs)
  apply (auto dest: transM simp add: relation_Int relation2_def)
  by(rule_tac wfrec_on_pred_closed'[of A r], auto)
qed

end — M_pre_cardinal_arith

```


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

Discipline for *ordertype*

relativize functional *ordertype ordertype_rel* **external**

relationalize *ordertype_rel is_ordertype*

context *M_pre_cardinal_arith*

begin

lemma *is_ordertype_iff*:

assumes $r \in \text{Pow}(A \times A)$ *wf*[*A*](*r*) *trans*[*A*](*r*)

shows $M(A) \implies M(r) \implies M(\text{res}) \implies \text{is_ordertype}(M, A, r, \text{res}) \longleftrightarrow \text{res} = \text{ordertype_rel}(M, A, r)$

using *assms is_ordermap_iff*[of *r A*] *trans_on_iff_trans*
ordermap_rel_closed[of *A r*]

unfolding *is_ordertype_def ordertype_rel_def wf_on_def* **by** *simp*

lemma *is_ordertype_iff'*:

assumes $r \in \text{Pow_rel}(M, A \times A)$ *well_ord*(*A*, *r*)

shows $M(A) \implies M(r) \implies M(\text{res}) \implies \text{is_ordertype}(M, A, r, \text{res}) \longleftrightarrow \text{res} = \text{ordertype_rel}(M, A, r)$

using *assms is_ordertype_iff Pow_rel_char*

unfolding *well_ord_def part_ord_def tot_ord_def* **by** *simp*

lemma *is_ordertype_iff''*:

assumes *well_ord*(*A*, *r*) $r \subseteq A \times A$

shows $M(A) \implies M(r) \implies M(\text{res}) \implies \text{is_ordertype}(M, A, r, \text{res}) \longleftrightarrow \text{res} = \text{ordertype_rel}(M, A, r)$

using *assms is_ordertype_iff*

unfolding *well_ord_def part_ord_def tot_ord_def* **by** *simp*

end — *M_pre_cardinal_arith*

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

— NOTE: not quite the same as *jump_cardinal*, note $\text{Pow}(X \times X)$.

definition

jump_cardinal' :: $i \Rightarrow i$ **where**

jump_cardinal'(*K*) \equiv

$\bigcup X \in \text{Pow}(K). \{z. r \in \text{Pow}(X * X), \text{well_ord}(X, r) \ \& \ z = \text{ordertype}(X, r)\}$

relativize functional *jump_cardinal' jump_cardinal'_rel* **external**

relationalize *jump_cardinal'_rel is_jump_cardinal'*

synthesize *is_jump_cardinal'* **from_definition** *assuming nonempty*

arity_theorem **for** *is_jump_cardinal'_fm*

definition *jump_cardinal_body'* **where**

jump_cardinal_body'(*X*) $\equiv \{z . r \in \text{Pow}(X \times X), \text{well_ord}(X, r) \wedge z = \text{ordertype}(X, r)\}$

```

relativize functional jump_cardinal_body' jump_cardinal_body'_rel external
relationalize jump_cardinal_body'_rel is_jump_cardinal_body'
synthesize is_jump_cardinal_body' from_definition assuming nonempty
arity_theorem for is_jump_cardinal_body'_fm

context M_pre_cardinal_arith
begin

lemma ordertype_rel_closed':
  assumes wf[A](r) trans[A](r) r ∈ Pow(A×A) M(r) M(A)
  shows M(ordertype_rel(M,A,r))
    unfolding ordertype_rel_def
    using ordermap_rel_closed image_closed assms by simp

lemma ordertype_rel_closed[intro,simp]:
  assumes well_ord(A,r) r ∈ Pow_rel(M,A×A) M(A)
  shows M(ordertype_rel(M,A,r))
    using assms Pow_rel_char ordertype_rel_closed'
    unfolding well_ord_def tot_ord_def part_ord_def
    by simp

lemma ordertype_rel_abs:
  assumes wellordered(M,X,r) M(X) M(r)
  shows ordertype_rel(M,X,r) = ordertype(X,r)
  using assms ordetypes_are_absolute[of X r]
  unfolding ordertype_def ordertype_rel_def ordermap_rel_def ordermap_def
  by simp

lemma univalent_aux1: M(X) ⇒ univalent(M,Pow_rel(M,X×X),
  λr z. M(z) ∧ M(r) ∧ r ∈ Pow_rel(M,X×X) ∧ is_well_ord(M, X, r) ∧ is_ordertype(M,
  X, r, z))
  using is_well_ord_iff_wellordered
  is_ordertype_iff[of X]
  trans_on_subset[OF well_ord_is_trans_on]
  well_ord_is_wf[THEN wf_on_subset_A] mem_Pow_rel_abs
  unfolding univalent_def
  by (simp)

lemma jump_cardinal_body_eq :
  M(X) ⇒ jump_cardinal_body(M,X) = jump_cardinal_body'_rel(M,X)
  unfolding jump_cardinal_body_def jump_cardinal_body'_rel_def
  using ordertype_rel_abs
  by auto

end — M_pre_cardinal_arith

context M_cardinal_arith
begin
lemma jump_cardinal_closed_aux1:

```

```

assumes  $M(X)$ 
shows
   $M(\text{jump\_cardinal\_body}(M, X))$ 
unfolding  $\text{jump\_cardinal\_body\_def}$ 
using  $\langle M(X) \rangle \text{ordertype\_rel\_abs}$ 
   $\text{ordertype\_replacement}[OF \langle M(X) \rangle]$   $\text{univalent\_aux1}[OF \langle M(X) \rangle]$ 
   $\text{strong\_replacement\_closed}$  where  $A = \text{Pow}^M(X \times X)$  and
     $P = \lambda r z. M(z) \wedge M(r) \wedge r \in \text{Pow}^M(X \times X) \wedge \text{well\_ord}(X, r) \wedge z =$ 
 $\text{ordertype}(X, r)]$ 
by auto

lemma  $\text{univalent\_jc\_body}$ :  $M(X) \implies \text{univalent}(M, X, \lambda x z. M(z) \wedge M(x) \wedge z$ 
 $= \text{jump\_cardinal\_body}(M, x))$ 
using  $\text{transM}[of\_X]$   $\text{jump\_cardinal\_closed\_aux1}$  by auto

lemma  $\text{jump\_cardinal\_body\_closed}$ :
assumes  $M(K)$ 
shows  $M(\{a. X \in \text{Pow}^M(K), M(a) \wedge M(X) \wedge a = \text{jump\_cardinal\_body}(M, X)\})$ 
using  $\text{assms}$   $\text{univalent\_jc\_body}$   $\text{jump\_cardinal\_closed\_aux1}$   $\text{strong\_replacement\_jc\_body}$ 
by simp

rel\_closed for  $\text{jump\_cardinal}'$ 
using  $\text{jump\_cardinal\_body\_closed}$   $\text{ordertype\_rel\_abs}$ 
unfolding  $\text{jump\_cardinal\_body\_def}$   $\text{jump\_cardinal}'\_rel\_def$ 
by simp

is\_iff\_rel for  $\text{jump\_cardinal}'$ 
proof -
assume  $\text{types}$ :  $M(K)$   $M(\text{res})$ 
have  $\text{is\_Replace}(M, \text{Pow\_rel}(M, X \times X), \lambda r z. M(z) \wedge M(r) \wedge \text{is\_well\_ord}(M,$ 
 $X, r) \wedge \text{is\_ordertype}(M, X, r, z),$ 
   $a) \longleftrightarrow a = \{z. r \in \text{Pow\_rel}(M, X \times X), M(z) \wedge M(r) \wedge \text{is\_well\_ord}(M, X, r)$ 
 $\wedge \text{is\_ordertype}(M, X, r, z)\}$ 
if  $M(X)$   $M(a)$  for  $X$   $a$ 
using  $\text{that}$   $\text{univalent\_aux1}$ 
by  $(\text{rule\_tac } \text{Replace\_abs}) (\text{simp\_all})$ 
then
have  $\text{is\_Replace}(M, \text{Pow\_rel}(M, X \times X), \lambda r z. M(z) \wedge M(r) \wedge \text{is\_well\_ord}(M,$ 
 $X, r) \wedge \text{is\_ordertype}(M, X, r, z),$ 
   $a) \longleftrightarrow a = \{z. r \in \text{Pow\_rel}(M, X \times X), M(z) \wedge M(r) \wedge \text{well\_ord}(X, r) \wedge z =$ 
 $\text{ordertype\_rel}(M, X, r)\}$ 
if  $M(X)$   $M(a)$  for  $X$   $a$ 
using  $\text{that}$   $\text{univalent\_aux1}$   $\text{is\_ordertype\_iff'}$   $\text{is\_well\_ord\_iff\_wellordered}$   $\text{well\_ord\_abs}$ 
by auto
moreover
have  $\text{is\_Replace}(M, d, \lambda X a. M(a) \wedge M(X) \wedge$ 
 $a = \{z. r \in \text{Pow}^M(X \times X), M(z) \wedge M(r) \wedge \text{well\_ord}(X, r) \wedge z = \text{ordertype}(X,$ 
 $r)\}, e)$ 
 $\longleftrightarrow$ 

```

```

    e = {a . X ∈ d, M(a) ∧ M(X) ∧ a = jump_cardinal_body(M,X)}
    if M(d) M(e) for d e
    using jump_cardinal_closed_aux1 that
    unfolding jump_cardinal_body_def
    by (rule_tac Replace_abs) simp_all
  ultimately
  show ?thesis
    using Pow_rel_iff_jump_cardinal_body_closed[of K] ordertype_rel_abs
    unfolding is_jump_cardinal'_def jump_cardinal'_rel_def jump_cardinal_body_def
    by (simp add: types)
qed

end

```

```

context M_cardinal_arith
begin

```

```

lemma (in M_ordertype) ordermap_closed[intro,simp]:
  assumes wellordered(M,A,r) and types:M(A) M(r)
  shows M(ordermap(A,r))
proof -
  note assms
  moreover from this
  obtain i f where Ord(i) f ∈ ord_iso(A, r, i, Memrel(i))
    M(i) M(f) using ordertype_exists by blast
  moreover from calculation
  have i = ordertype(A,r) using ordertypes_are_absolute by force
  moreover from calculation
  have ordermap(A,r) ∈ ord_iso(A, r, i, Memrel(i))
    using ordertype_ord_iso by simp
  ultimately
  have f = ordermap(A,r) using well_ord_iso_unique by fastforce
  with ⟨M(f)⟩
  show ?thesis by simp
qed

```

```

lemma ordermap_eqpoll_pred:
  [| well_ord(A,r); x ∈ A ; M(A);M(r);M(x)|] ==> ordermap(A,r) `x ≈M
  Order.pred(A,x,r)
  apply (simp add: def_eqpoll_rel)
  apply (rule rexI)
  apply (simp add: ordermap_eq_image well_ord_is_wf)
  apply (erule ordermap_bij [THEN bij_is_inj, THEN restrict_bij,
    THEN bij_converse_bij])
  apply (rule pred_subset, simp)
done

```

Kunen: "each $\langle x, y \rangle \in K \times K$ has no more than $z \times z$ predecessors..." (page

29)

lemma *ordermap_csquare_le*:

assumes K : *Limit*(K) **and** x : $x < K$ **and** y : $y < K$

and types: $M(K)$ $M(x)$ $M(y)$

shows $|ordermap(K \times K, csquare_rel(K))|^{M(x,y)} \leq |succ(succ(x \cup y))|^M \otimes^M |succ(succ(x \cup y))|^M$

using *types*

proof (*simp add: cmult_rel_def, rule_tac well_ord_lepoll_rel_imp_cardinal_rel_le*)

let $?z = succ(x \cup y)$

show $well_ord(|succ(?z)|^M \times |succ(?z)|^M, rmult(|succ(?z)|^M, Memrel(|succ(?z)|^M), |succ(?z)|^M, Memrel(|succ(?z)|^M)))$

by (*blast intro: well_ord_Memrel well_ord_rmult types*)

next

let $?z = succ(x \cup y)$

have zK : $?z < K$ **using** x y K

by (*blast intro: Un_least_lt Limit_has_succ*)

hence oz : $Ord(?z)$ **by** (*elim ltE*)

from *assms*

have Mom : $M(ordermap(K \times K, csquare_rel(K)))$

using *well_ord_csquare Limit_is_Ord* **by** *fastforce*

then

have $ordermap(K \times K, csquare_rel(K))|^{M(x,y)} \lesssim^M ordermap(K \times K, csquare_rel(K))|^{M(x,y)}$

by (*blast intro: ordermap_z_lt leI le_imp_lepoll_rel K x y types*)

also have $\dots \approx^M Order.pred(K \times K, \langle ?z, ?z \rangle, csquare_rel(K))$

proof (*rule ordermap_eqpoll_pred*)

show $well_ord(K \times K, csquare_rel(K))$ **using** K

by (*rule Limit_is_Ord [THEN well_ord_csquare]*)

next

show $\langle ?z, ?z \rangle \in K \times K$ **using** zK

by (*blast intro: ltD*)

qed (*simp_all add:types*)

also have $\dots \lesssim^M succ(?z) \times succ(?z)$ **using** zK

by (*rule_tac pred_csquare_subset [THEN subset_imp_lepoll_rel]*) (*simp_all add:types*)

also have $\dots \approx^M |succ(?z)|^M \times |succ(?z)|^M$ **using** oz

by (*blast intro: prod_eqpoll_rel_cong Ord_cardinal_rel_eqpoll_rel_eqpoll_rel_sym types*)

finally show $ordermap(K \times K, csquare_rel(K))|^{M(x,y)} \lesssim^M |succ(?z)|^M \times |succ(?z)|^M$

by (*simp_all add:types Mom*)

from Mom

show $M(ordermap(K \times K, csquare_rel(K))|^{M(x,y)})$ **by** (*simp_all add:types*)

qed (*simp_all add:types*)

Kunen: "... so the order type is $\leq K$ "

lemma *ordertype_csquare_le M*:

assumes IK : $InfCard^M(K)$ **and** eq : $\bigwedge y. y \in K \implies InfCard^M(y) \implies M(y) \implies y \otimes^M y = y$

— Note the weakened hypothesis $\llbracket ?y \in K; \text{InfCard}^M(?y); M(?y) \rrbracket \implies ?y \otimes^M ?y = ?y$
 and *types*: $M(K)$
 shows $\text{ordertype}(K * K, \text{csquare_rel}(K)) \leq K$
proof -
 have $CK: \text{Card}^M(K)$ **using** IK **by** $(\text{rule_tac } \text{InfCard_rel_is_Card_rel}) (\text{simp_all } \text{add:types})$
 hence $OK: \text{Ord}(K)$ **by** $(\text{rule } \text{Card_rel_is_Ord}) (\text{simp_all } \text{add:types})$
 moreover have $\text{Ord}(\text{ordertype}(K \times K, \text{csquare_rel}(K)))$ **using** OK
by $(\text{rule } \text{well_ord_csquare } [\text{THEN } \text{Ord_ordertype}])$
 ultimately show $?thesis$
proof $(\text{rule } \text{all_lt_imp_le})$
 fix i
 assume $i: i < \text{ordertype}(K \times K, \text{csquare_rel}(K))$
 hence $Oi: \text{Ord}(i)$ **by** $(\text{elim } \text{ltE})$
 obtain $x \ y$ **where** $x: x \in K$ **and** $y: y \in K$
 and $ieq: i = \text{ordermap}(K \times K, \text{csquare_rel}(K)) \text{ ‘ } \langle x, y \rangle$
using i **by** $(\text{auto } \text{simp } \text{add: ordertype_unfold } \text{elim: ltE})$
 hence $xy: \text{Ord}(x) \ \text{Ord}(y) \ x < K \ y < K$ **using** OK
by $(\text{blast } \text{intro: Ord_in_Ord } \text{ltI})+$
 hence $ou: \text{Ord}(x \cup y)$
by (simp)
 from OK *types*
 have $M(\text{ordertype}(K \times K, \text{csquare_rel}(K)))$
using well_ord_csquare **by** fastforce
 with $i \ x \ y$ *types*
 have $\text{types}': M(K) \ M(i) \ M(x) \ M(y)$
using *types* **by** $(\text{auto } \text{dest:transM } \text{ltD})$
 show $i < K$
proof $(\text{rule } \text{Card_rel_lt_imp_lt } [\text{OF } Oi \ CK])$
 have $|i|^M \leq |\text{succ}(\text{succ}(x \cup y))|^M \otimes^M |\text{succ}(\text{succ}(x \cup y))|^M$ **using** $IK \ xy$
by $(\text{auto } \text{simp } \text{add: ieq } \text{types } \text{intro: InfCard_rel_is_Limit } [\text{THEN } \text{ordermap_csquare_le}] \text{types'})$
 moreover have $|\text{succ}(\text{succ}(x \cup y))|^M \otimes^M |\text{succ}(\text{succ}(x \cup y))|^M < K$
proof $(\text{cases } \text{rule: Ord_linear2 } [\text{OF } ou \ \text{Ord_nat}])$
 assume $x \cup y < \text{nat}$
 hence $|\text{succ}(\text{succ}(x \cup y))|^M \otimes^M |\text{succ}(\text{succ}(x \cup y))|^M \in \text{nat}$
by $(\text{simp } \text{add: lt_def } \text{nat_cmult_rel_eq_mult } \text{nat_succI } \text{nat_into_Card_rel } [\text{THEN } \text{Card_rel_cardinal_rel_eq}] \text{types'})$
 also have $\dots \subseteq K$ **using** IK
by $(\text{simp } \text{add: InfCard_rel_def } \text{le_imp_subset } \text{types})$
 finally show $|\text{succ}(\text{succ}(x \cup y))|^M \otimes^M |\text{succ}(\text{succ}(x \cup y))|^M < K$
by $(\text{simp } \text{add: ltI } OK)$
 next
 assume $\text{natxy: } \text{nat} \leq x \cup y$
 hence $\text{seq: } |\text{succ}(\text{succ}(x \cup y))|^M = |x \cup y|^M$ **using** xy
by $(\text{simp } \text{add: le_imp_subset } \text{nat_succ_eqpoll_rel } [\text{THEN } \text{cardinal_rel_cong}] \text{le_succ_iff } \text{types'})$
 also have $\dots < K$ **using** xy

```

      by (simp add: Un_least_lt Ord_cardinal_rel_le [THEN lt_trans1]
types')
    finally have |succ(succ(x ∪ y))|^M < K .
    moreover have InfCard^M(|succ(succ(x ∪ y))|^M) using xy natxy
      by (simp add: seq InfCard_rel_def nat_le_cardinal_rel types')
    ultimately show ?thesis by (simp add: eq ltD types')
  qed
  ultimately show |i|^M < K by (blast intro: lt_trans1)
qed (simp_all add: types')
qed
qed

lemma InfCard_rel_csquare_eq:
  assumes IK: InfCard^M(K) and
  types: M(K)
  shows K ⊗^M K = K
proof -
  have OK: Ord(K) using IK by (simp add: Card_rel_is_Ord InfCard_rel_is_Card_rel
types)
  from OK assms
  show K ⊗^M K = K
  proof (induct rule: trans_induct)
    case (step i)
    note types = ⟨M(K)⟩ ⟨M(i)⟩
    show i ⊗^M i = i
    proof (rule le_anti_sym)
      from step types
      have Mot: M(ordertype(i × i, csquare_rel(i))) M(ordertype(i × i, csquare_rel(i)))
        using well_ord_csquare Limit_is_Ord by simp_all
      then
      have |i × i|^M = |ordertype(i × i, csquare_rel(i))|^M
        by (rule_tac cardinal_rel_cong,
simp_all add: step.hyps well_ord_csquare [THEN ordermap_bij, THEN
bij_imp_eqpoll_rel] types)
      with Mot
      have i ⊗^M i ≤ ordertype(i × i, csquare_rel(i))
      by (simp add: step.hyps cmult_rel_def Ord_cardinal_rel_le well_ord_csquare
[THEN Ord_ordertype] types)
      moreover
      have ordertype(i × i, csquare_rel(i)) ≤ i using step
        by (rule_tac ordertype_csquare_le_M) (simp add: types)
      ultimately show i ⊗^M i ≤ i by (rule le_trans)
    next
    show i ≤ i ⊗^M i using step
      by (blast intro: cmult_rel_square_le InfCard_rel_is_Card_rel)
  qed
qed
qed

```

```

lemma well_ord_InfCard_rel_square_eq:
  assumes  $r$ : well_ord( $A, r$ ) and  $I$ : InfCard $^M(|A|^M)$  and
    types:  $M(A)$   $M(r)$ 
  shows  $A \times A \approx^M A$ 
proof -
  have  $A \times A \approx^M |A|^M \times |A|^M$ 
  by (blast intro: prod_eqpoll_rel_cong well_ord_cardinal_rel_eqpoll_rel_eqpoll_rel_sym
 $r$  types)
  also have  $\dots \approx^M A$ 
  proof (rule well_ord_cardinal_rel_eqE [OF  $r$ ])
  show well_ord( $|A|^M \times |A|^M$ ,  $rmult(|A|^M, Memrel(|A|^M), |A|^M, Memrel(|A|^M))$ )
    by (blast intro: well_ord_rmult well_ord_Memrel  $r$  types)
  next
  show  $|A|^M \times |A|^M = |A|^M$  using InfCard_rel_csquare_eq  $I$ 
  by (simp add: cmult_rel_def types)
  qed (simp_all add: types)
  finally show ?thesis by (simp_all add: types)
qed

lemma InfCard_rel_square_eqpoll:
  assumes InfCard $^M(K)$  and types: $M(K)$  shows  $K \times K \approx^M K$ 
  using assms
apply (rule_tac well_ord_InfCard_rel_square_eq)
apply (erule InfCard_rel_is_Card_rel [THEN Card_rel_is_Ord, THEN well_ord_Memrel])
apply (simp_all add: InfCard_rel_is_Card_rel [THEN Card_rel_cardinal_rel_eq]
types)
done

lemma Inf_Card_rel_is_InfCard_rel: [| Card $^M(i)$ ;  $\sim$  Finite_rel( $M, i$ ) ;  $M(i)$  |]
==> InfCard $^M(i)$ 
  by (simp add: InfCard_rel_def Card_rel_is_Ord [THEN nat_le_infinite_Ord])

```

16.5.1 Toward's Kunen's Corollary 10.13 (1)

```

lemma InfCard_rel_le_cmult_rel_eq: [| InfCard $^M(K)$ ;  $L \leq K$ ;  $0 < L$ ;  $M(K)$  ;
 $M(L)$  |] ==>  $K \otimes^M L = K$ 
apply (rule le_anti_sym)
prefer 2
apply (erule ltE, blast intro: cmult_rel_le_self InfCard_rel_is_Card_rel)
apply (frule InfCard_rel_is_Card_rel [THEN Card_rel_is_Ord, THEN le_refl])
prefer 3
apply (rule cmult_rel_le_mono [THEN le_trans], assumption+)
apply (simp_all add: InfCard_rel_csquare_eq)
done

```



```

lemma InfCard_rel_cmult_rel_eq: [| InfCardM(K); InfCardM(L); M(K) ; M(L)
|] ==>  $K \otimes^M L = K \cup L$ 
apply (rule_tac i = K and j = L in Ord_linear_le)
apply (typecheck add: InfCard_rel_is_Card_rel Card_rel_is_Ord)
apply (rule cmult_rel_commute [THEN ssubst]) prefer 3
apply (rule Un_commute [THEN ssubst])
apply (simp_all add: InfCard_rel_is_Limit [THEN Limit_has_0] InfCard_rel_le_cmult_rel_eq
subset_Un_iff2 [THEN iffD1] le_imp_subset)

done

```

```

lemma InfCard_rel_cdouble_eq: InfCardM(K)  $\implies$  M(K)  $\implies$   $K \oplus^M K = K$ 
apply (simp add: cmult_rel_2 [symmetric] InfCard_rel_is_Card_rel cmult_rel_commute)
apply (simp add: InfCard_rel_le_cmult_rel_eq InfCard_rel_is_Limit Limit_has_0
Limit_has_succ)

done

```

```

lemma InfCard_rel_le_cadd_rel_eq: [| InfCardM(K); L ≤ K ; M(K) ; M(L)|]
==>  $K \oplus^M L = K$ 
apply (rule le_anti_sym)
prefer 2
apply (erule ltE, blast intro: cadd_rel_le_self InfCard_rel_is_Card_rel)
apply (frule InfCard_rel_is_Card_rel [THEN Card_rel_is_Ord, THEN le_refl])
prefer 3
apply (rule cadd_rel_le_mono [THEN le_trans], assumption+)
apply (simp_all add: InfCard_rel_cdouble_eq)

done

```

```

lemma InfCard_rel_cadd_rel_eq: [| InfCardM(K); InfCardM(L); M(K) ; M(L)
|] ==>  $K \oplus^M L = K \cup L$ 
apply (rule_tac i = K and j = L in Ord_linear_le)
apply (typecheck add: InfCard_rel_is_Card_rel Card_rel_is_Ord)
apply (rule cadd_rel_commute [THEN ssubst]) prefer 3
apply (rule Un_commute [THEN ssubst])
apply (simp_all add: InfCard_rel_le_cadd_rel_eq subset_Un_iff2 [THEN iffD1]
le_imp_subset)

done

```

end — *M_cardinal_arith*

16.6 For Every Cardinal Number There Exists A Greater One

This result is Kunen's Theorem 10.16, which would be trivial using AC

```

locale M_cardinal_arith_jump = M_cardinal_arith + M_ordertype
begin

```

lemma *well_ord_restr*: $\text{well_ord}(X, r) \implies \text{well_ord}(X, r \cap X \times X)$

proof -

have $r \cap X \times X \cap X \times X = r \cap X \times X$ **by** *auto*

moreover

assume $\text{well_ord}(X, r)$

ultimately

show *?thesis*

unfolding *well_ord_def tot_ord_def part_ord_def linear_def*

irrefl_def wf_on_def

by *simp_all (simp only: trans_on_def, blast)*

qed

lemma *ordertype_restr_eq* :

assumes $\text{well_ord}(X, r)$

shows $\text{ordertype}(X, r) = \text{ordertype}(X, r \cap X \times X)$

using *ordemap_restr_eq assms unfolding ordertype_def*

by *simp*

lemma *def_jump_cardinal_rel_aux*:

$X \in \text{Pow}^M(K) \implies \text{well_ord}(X, w) \implies M(K) \implies$

$\{z . r \in \text{Pow}^M(X \times X), M(z) \wedge \text{well_ord}(X, r) \wedge z = \text{ordertype}(X, r)\} =$

$\{z . r \in \text{Pow}^M(K \times K), M(z) \wedge \text{well_ord}(X, r) \wedge z = \text{ordertype}(X, r)\}$

proof(*rule, auto simp: Pow_rel_char dest: transM*)

let $?L = \{z . r \in \text{Pow}^M(X \times X), M(z) \wedge \text{well_ord}(X, r) \wedge z = \text{ordertype}(X, r)\}$

let $?R = \{z . r \in \text{Pow}^M(K \times K), M(z) \wedge \text{well_ord}(X, r) \wedge z = \text{ordertype}(X, r)\}$

show $\text{ordertype}(X, r) \in \{y . x \in \{x \in \text{Pow}(X \times X) . M(x)\}, M(y) \wedge \text{well_ord}(X, x) \wedge y = \text{ordertype}(X, x)\}$

if $M(K) \ M(r) \ r \subseteq K \times K \ X \subseteq K \ M(X) \ \text{well_ord}(X, r)$ **for** r

proof -

from *that*

have $\text{ordertype}(X, r) = \text{ordertype}(X, r \cap X \times X) \ (r \cap X \times X) \subseteq X \times X \ M(r \cap X \times X)$

$\text{well_ord}(X, r \cap X \times X) \ \text{wellordered}(M, X, r \cap X \times X)$

using *well_ord_restr ordertype_restr_eq* **by** *auto*

moreover from *this*

have $\text{ordertype}(X, r \cap X \times X) \in ?L$

using *that Pow_rel_char*

ReplaceI[$\text{of } \lambda z r . M(z) \wedge \text{well_ord}(X, r) \wedge z = \text{ordertype}(X, r)$

ordertype(X, r ∩ X × X)]

by *auto*

ultimately

show *?thesis* **using** *Pow_rel_char* **by** *auto*

qed

qed

lemma *def_jump_cardinal_rel*:

assumes $M(K)$

shows $\text{jump_cardinal}'_rel(M, K) =$

$(\bigcup X \in \text{Pow_rel}(M, K). \{z . r \in \text{Pow_rel}(M, K * K), \text{well_ord}(X, r) \ \& \ z = \text{ordertype}(X, r)\})$

```

proof -
  have  $M(\{z . r \in \text{Pow}^M(X \times X), M(z) \wedge \text{well\_ord}(X, r) \wedge z = \text{ordertype}(X, r)\})$ 
    (is  $M(\text{Replace}(\_, ?P))$ )
  if  $M(X)$  for  $X$ 
    using that  $\text{jump\_cardinal\_closed\_aux1}$ [of  $X$ ]  $\text{ordertype\_rel\_abs}$ [of  $X$ ]
       $\text{jump\_cardinal\_body\_def}$ 
    by (subst  $\text{Replace\_cong}$ [where  $P = ?P$ 
      and  $Q = \lambda r z. M(z) \wedge M(r) \wedge \text{well\_ord}(X, r) \wedge z = \text{ordertype\_rel}(M, X, r),$ 
       $\text{OF refl, of Pow}^M(X \times X)]$ ) (auto dest:transM)
  then
    have  $M(\{z . r \in \text{Pow}^M(Y \times Y), M(z) \wedge \text{well\_ord}(X, r) \wedge z = \text{ordertype}(X, r)\})$ 
      (if  $M(Y)$   $M(X)$   $X \in \text{Pow}^M(Y)$   $\text{well\_ord}(X, r)$  for  $Y$   $X$   $r$ 
        using that  $\text{def\_jump\_cardinal\_rel\_aux}$ [of  $X$   $Y$   $r$ , symmetric] by simp
        moreover from  $\langle M(K) \rangle$ 
        have  $R \in \text{Pow}^M(X \times X) \implies X \in \text{Pow}^M(K) \implies R \in \text{Pow}^M(K \times K)$ 
        for  $X$   $R$  using  $\text{mem\_Pow\_rel\_abs transM}$ [ $\text{OF } \_ \text{Pow\_rel\_closed, of } R$   $X \times X$ ]
           $\text{transM}$ [ $\text{OF } \_ \text{Pow\_rel\_closed, of } X$   $K$ ] by auto
        ultimately
        show ?thesis
          using assms is_ordertype_iff is_well_ord_iff wellordered
             $\text{ordertype\_rel\_abs transM}$ [of  $\_ \text{Pow}^M(K)$ ]  $\text{transM}$ [of  $\_ \text{Pow}^M(K \times K)$ ]
             $\text{def\_jump\_cardinal\_rel\_aux}$ 
          unfolding  $\text{jump\_cardinal'\_rel\_def}$ 
          apply (intro equalityI)
          apply (auto dest:transM)
          apply (rename_tac X R)
          apply (rule_tac x=X in bexI)
          apply (rule_tac x=R in ReplaceI)
          apply auto
          apply (rule_tac x={y . xa \in Pow^M(K \times K), M(y) \wedge M(xa) \wedge well\_ord(X, xa) \wedge y = ordertype(X, xa)} in bexI)
          apply auto
          by (rule_tac x=X in ReplaceI) auto
      )
  qed

```

notation $\text{jump_cardinal'_rel}$ ($\langle \text{jump'_cardinal'_rel} \rangle$)

```

lemma  $\text{Ord\_jump\_cardinal\_rel}: M(K) \implies \text{Ord}(\text{jump\_cardinal\_rel}(M, K))$ 
apply (unfold def_jump_cardinal_rel)
apply (rule Ord_is_Transset [THEN [2] OrdI])
prefer 2 apply (blast intro!: Ord_ordertype)
apply (unfold Transset_def)
apply (safe del: subsetI)
apply (subst ordertype_pred_unfold, simp, safe)
apply (rule UN_I)
apply (rule_tac [2] ReplaceI)
prefer 4 apply (blast intro: well_ord_subset elim!: predE, simp_all)

```

```

    prefer 2 apply (blast intro: well_ord_subset elim!: predE)
proof -
  fix X r xb
  assume M(K) X ∈ PowM(K) r ∈ PowM(K × K) well_ord(X, r) xb ∈ X
  moreover from this
  have M(X) M(r)
    using cartprod_closed trans_Pow_rel_closed by auto
  moreover from this
  have M(xb) using transM[OF ‹xb∈X›] by simp
  ultimately
  show Order.pred(X, xb, r) ∈ PowM(K)
    using def_Pow_rel by (auto dest: predE)
qed

```

declare *conj_cong* [*cong del*]
 — incompatible with some of the proofs of the original theory

```

lemma jump_cardinal_rel_iff_old:
  M(i) ⇒ M(K) ⇒ i ∈ jump_cardinal_rel(M, K) ⇔
    (∃ r[M]. ∃ X[M]. r ⊆ K*K & X ⊆ K & well_ord(X, r) & i = ordertype(X, r))
apply (unfold def_jump_cardinal_rel)
apply (auto del: subsetI)
apply (rename_tac y r)
apply (rule_tac x=r in rexI, intro conjI) prefer 2
  apply (rule_tac x=y in rexI, intro conjI)
    apply (auto dest: mem_Pow_rel transM)
apply (rule_tac A=r in rev_subsetD, assumption)
defer
apply (rename_tac r y)
apply (rule_tac x=y in bexI)
  apply (rule_tac x=r in ReplaceI, auto)
using def_Pow_rel
apply (force+)[2]
apply (rule_tac A=r in rev_subsetD, assumption)
using mem_Pow_rel[THEN conjunct1]
apply auto
done

```

```

lemma K_lt_jump_cardinal_rel: Ord(K) ==> M(K) ⇒ K < jump_cardinal_rel(M, K)
apply (rule Ord_jump_cardinal_rel [THEN [2] ltI])
apply (rule jump_cardinal_rel_iff_old [THEN iffD2], assumption+)
apply (rule_tac x=Memrel(K) in rexI)
apply (rule_tac x=K in rexI)
  apply (simp add: ordertype_Memrel well_ord_Memrel)
    using Memrel_closed
apply (simp_all add: Memrel_def subset_iff)
done

```

```

lemma Card_rel_jump_cardinal_rel_lemma:
  [| well_ord(X,r); r ⊆ K * K; X ⊆ K;
    f ∈ bij(ordertype(X,r), jump_cardinal_rel(M,K));
    M(X); M(r); M(K); M(f) |]
  ==> jump_cardinal_rel(M,K) ∈ jump_cardinal_rel(M,K)
apply (subgoal_tac f O ordermap (X,r) ∈ bij (X, jump_cardinal_rel (M,K)))
prefer 2 apply (blast intro: comp_bij ordermap_bij)
apply (rule jump_cardinal_rel_iff_old [THEN iffD2], simp+)
apply (intro rexI conjI)
apply (rule subset_trans [OF rvimage_type Sigma_mono], assumption+)
apply (erule bij_is_inj [THEN well_ord_rvimage])
  apply (rule Ord_jump_cardinal_rel [THEN well_ord_Memrel])
apply (simp_all add: well_ord_Memrel [THEN [2] bij_ordertype_vimage]
  ordertype_Memrel Ord_jump_cardinal_rel)
done

```

```

lemma Card_rel_jump_cardinal_rel: M(K) ==> Card_rel(M,jump_cardinal_rel(M,K))
  apply (rule Ord_jump_cardinal_rel [THEN Card_relI])
  apply (simp_all add: def_eqpoll_rel)
  apply (drule_tac i1=j in jump_cardinal_rel_iff_old [THEN iffD1, OF __ ltD,
of _ K], safe)
  apply (blast intro: Card_rel_jump_cardinal_rel_lemma [THEN mem_irrefl])
done

```

16.7 Basic Properties of Successor Cardinals

```

lemma csucc_rel_basic: Ord(K) ==> M(K) ==> Card_rel(M,csucc_rel(M,K))
& K < csucc_rel(M,K)
apply (unfold csucc_rel_def)
apply (rule LeastI[of λi. M(i) ∧ Card_rel(M,i) ∧ K < i, THEN conjunct2])
apply (blast intro: Card_rel_jump_cardinal_rel K_lt_jump_cardinal_rel Ord_jump_cardinal_rel)+
done

```

```

lemmas Card_rel_csucc_rel = csucc_rel_basic [THEN conjunct1]

```

```

lemmas lt_csucc_rel = csucc_rel_basic [THEN conjunct2]

```

```

lemma Ord_0_lt_csucc_rel: Ord(K) ==> M(K) ==> 0 < csucc_rel(M,K)
by (blast intro: Ord_0_le lt_csucc_rel lt_trans1)

```

```

lemma csucc_rel_le: [| Card_rel(M,L); K < L; M(K); M(L) |] ==> csucc_rel(M,K)
≤ L
apply (unfold csucc_rel_def)
apply (rule Least_le)
apply (blast intro: Card_rel_is_Ord)+
done

```

```

lemma lt_csucc_rel_iff: [| Ord(i); Card_rel(M,K); M(K); M(i) |] ==> i <
csucc_rel(M,K) <=> |i|M ≤ K
apply (rule iffI)
apply (rule_tac [2] Card_rel_lt_imp_lt)
apply (erule_tac [2] lt_trans1)
apply (simp_all add: lt_csucc_rel Card_rel_csucc_rel Card_rel_is_Ord)
apply (rule notI [THEN not_lt_imp_le])
apply (rule Card_rel_cardinal_rel [THEN csucc_rel_le, THEN lt_trans1, THEN
lt_irrefl], simp_all+)
apply (rule Ord_cardinal_rel_le [THEN lt_trans1])
apply (simp_all add: Card_rel_is_Ord)
done

```

```

lemma Card_rel_lt_csucc_rel_iff:
  [| Card_rel(M,K'); Card_rel(M,K); M(K'); M(K) |] ==> K' < csucc_rel(M,K)
  <=> K' ≤ K
by (simp add: lt_csucc_rel_iff Card_rel_cardinal_rel_eq Card_rel_is_Ord)

```

```

lemma InfCard_rel_csucc_rel: InfCard_rel(M,K) ==> M(K) ==> InfCard_rel(M,csucc_rel(M,K))
by (simp add: InfCard_rel_def Card_rel_csucc_rel Card_rel_is_Ord
  lt_csucc_rel [THEN leI, THEN [2] le_trans])

```

16.7.1 Theorems by Krzysztof Grabczewski, proofs by lcp

```

lemma nat_sum_eqpoll_rel_sum:
  assumes m: m ∈ nat and n: n ∈ nat shows m + n ≈M m #+ n
proof -
  have m + n ≈M |m+n|M using m n
  by (blast intro: nat_implies_well_ord well_ord_radd well_ord_cardinal_rel_eqpoll_rel
eqpoll_rel_sym)
  also have ... = m #+ n using m n
  by (simp add: nat_cadd_rel_eq_add [symmetric] cadd_rel_def transM[OF _
M_nat])
  finally show ?thesis .
qed

```

```

lemma Ord_nat_subset_into_Card_rel: [| Ord(i); i ⊆ nat |] ==> CardM(i)
by (blast dest: Ord_subset_natD intro: Card_rel_nat nat_into_Card_rel)

```

end — *M_cardinal_arith_jump*

end

theory *Aleph_Relative*

imports

Univ_Relative

CardinalArith_Relative

Cardinal_Relative

begin

definition

$HAleph :: [i, i] \Rightarrow i$ **where**
 $HAleph(i, r) \equiv \text{if}(\neg(\text{Ord}(i)), i, \text{if}(i=0, \text{nat}, \text{if}(\neg\text{Limit}(i) \wedge i \neq 0,$
 $\text{csucc}(r'(\bigcup i)),$
 $\bigcup_{j \in i. r'j)))$

reldb__add functional *Limit Limit*
relationalize *Limit is__Limit external*
synthesize is__Limit from__definition
arity__theorem for is__Limit_fm

relativize functional *HAleph HAleph_rel*
relationalize *HAleph_rel is__HAleph*

synthesize is__HAleph from__definition assuming *nonempty*
arity__theorem for is__HAleph_fm

definition
 $Aleph' :: i \Rightarrow i$ **where**
 $Aleph'(a) == \text{transrec}(a, \lambda i r. HAleph(i, r))$

relativize functional *Aleph' Aleph_rel*
relationalize *Aleph_rel is__Aleph*

The extra assumptions $a < \text{length}(\text{env})$ and $c < \text{length}(\text{env})$ in this schematic goal (and the following results on synthesis that depend on it) are imposed by $\llbracket \bigwedge a0\ a1\ a2\ a3\ a4\ a5\ a6\ a7. \llbracket a0 \in ?A; a1 \in ?A; a2 \in ?A; a3 \in ?A; a4 \in ?A; a5 \in ?A; a6 \in ?A; a7 \in ?A \rrbracket \Longrightarrow ?MH(a2, a1, a0) \longleftrightarrow ?A, \text{Cons}(a0, \text{Cons}(a1, \text{Cons}(a2, \text{Cons}(a3, \text{Cons}(a4, \text{Cons}(a5, \text{Cons}(a6, \text{Cons}(a7, ?env)))))))) \models ?p; \text{nth}(?i, ?env) = ?x; \text{nth}(?k, ?env) = ?z; ?i < \text{length}(?env); ?k < \text{length}(?env); ?env \in \text{list}(?A) \rrbracket \Longrightarrow \text{is_transrec}(\#\# ?A, ?MH, ?x, ?z) \longleftrightarrow ?A, ?env \models \text{is_transrec_fm}(?p, ?i, ?k)$.

schematic__goal sats_is__Aleph_fm_auto:
 $a \in \text{nat} \Longrightarrow c \in \text{nat} \Longrightarrow \text{env} \in \text{list}(A) \Longrightarrow$
 $a < \text{length}(\text{env}) \Longrightarrow c < \text{length}(\text{env}) \Longrightarrow 0 \in A \Longrightarrow$
 $\text{is_Aleph}(\#\# A, \text{nth}(a, \text{env}), \text{nth}(c, \text{env})) \longleftrightarrow A, \text{env} \models ?fm(a, c)$
unfolding is__Aleph_def
proof (*rule is__transrec_iff_sats, rule_tac [1] is__HAleph_iff_sats*)
fix $a0\ a1\ a2\ a3\ a4\ a5\ a6\ a7$
let $?env' = \text{Cons}(a0, \text{Cons}(a1, \text{Cons}(a2, \text{Cons}(a3, \text{Cons}(a4, \text{Cons}(a5, \text{Cons}(a6, \text{Cons}(a7, \text{env}))))))))$
show $\text{nth}(2, ?env') = a2$
 $\text{nth}(1, ?env') = a1$
 $\text{nth}(0, ?env') = a0$
 $\text{nth}(c, \text{env}) = \text{nth}(c, \text{env})$
by simp_all
qed simp_all

synthesize__notc is__Aleph from__schematic

notation is_Aleph_fm ($\langle \cdot \rangle'$ ($_$) $is _$ \cdot)

lemma $is_Aleph_fm_type$ $[TC]$: $a \in nat \implies c \in nat \implies is_Aleph_fm(a, c) \in formula$
unfolding $is_Aleph_fm_def$ **by** $simp$

lemma $sats_is_Aleph_fm$:
assumes $f \in nat$ $r \in nat$ $env \in list(A)$ $0 \in A$ $f < length(env)$ $r < length(env)$
shows $is_Aleph(\#\#A, nth(f, env), nth(r, env)) \longleftrightarrow A, env \models is_Aleph_fm(f, r)$
using $assms$ $sats_is_Aleph_fm_auto$ **unfolding** is_Aleph_def $is_Aleph_fm_def$
by $simp$

lemma $is_Aleph_iff_sats$ $[iff_sats]$:
assumes
 $nth(f, env) = fa$ $nth(r, env) = ra$ $f < length(env)$ $r < length(env)$
 $f \in nat$ $r \in nat$ $env \in list(A)$ $0 \in A$
shows $is_Aleph(\#\#A, fa, ra) \longleftrightarrow A, env \models is_Aleph_fm(f, r)$
using $assms$ $sats_is_Aleph_fm$ $[of\ f\ r\ env\ A]$ **by** $simp$

arity_theorem **for** is_Aleph_fm

lemma $(in\ M_cardinal_arith_jump)$ is_Limit_iff :
assumes $M(a)$
shows $is_Limit(M, a) \longleftrightarrow Limit(a)$
unfolding is_Limit_def $Limit_def$ **using** lt_abs $transM$ $[OF\ ltD\ \langle M(a) \rangle]$ $assms$
by $auto$

lemma $HAleph_eq_Aleph_recursive$:
 $Ord(i) \implies HAleph(i, r) = (if\ i = 0\ then\ nat$
 $\quad else\ if\ \exists j. i = succ(j)\ then\ csucc(r\ ' (THE\ j. i = succ(j)))\ else\ \bigcup j < i.$
 $\quad r\ ' j)$

proof -
assume $Ord(i)$
moreover from $this$
have $i = succ(j) \implies (\bigcup succ(j)) = j$ **for** j
using $Ord_Union_succ_eq$ **by** $simp$
moreover from $\langle Ord(i) \rangle$
have $(\exists j. i = succ(j)) \longleftrightarrow \neg Limit(i) \wedge i \neq 0$
using Ord_cases_disj **by** $auto$
ultimately
show $?thesis$
unfolding $HAleph_def$ $OUnion_def$
by $auto$

qed

lemma $Aleph'_eq_Aleph$: $Ord(a) \implies Aleph'(a) = Aleph(a)$
unfolding $Aleph'_def$ $Aleph_def$ $transrec2_def$
using $HAleph_eq_Aleph_recursive$
by $(intro\ transrec_equal_on_Ord)\ auto$


```

reldb_rem functional Aleph'
reldb_rem relational is_Aleph
reldb_add functional Aleph Aleph_rel
reldb_add relational Aleph is_Aleph

abbreviation
  Aleph_r :: [i,i⇒o] ⇒ i (⟨ℵ _⟩) where
  Aleph_r(a,M) ≡ Aleph_rel(M,a)

abbreviation
  Aleph_r_set :: [i,i] ⇒ i (⟨ℵ _⟩) where
  Aleph_r_set(a,M) ≡ Aleph_rel(##M,a)

lemma Aleph_rel_def': Aleph_rel(M,a) ≡ transrec(a, λi r. HAleph_rel(M, i, r))
  unfolding Aleph_rel_def .

lemma succ_mem_Limit: Limit(j) ⇒ i ∈ j ⇒ succ(i) ∈ j
  using Limit_has_succ[THEN ltD] ltI Limit_is_Ord by auto

locale M_pre_aleph = M_eclose + M_cardinal_arith_jump +
  assumes
    haleph_transrec_replacement: M(a) ⇒ transrec_replacement(M,is_HAleph(M),a)

begin

lemma aux_ex_Replace_funapply:
  assumes M(a) M(f)
  shows ∃ x[M]. is_Replace(M, a, λj y. f ' j = y, x)
proof -
  have {f'j . j∈a} = {y . j∈a , f ' j=y}
    {y . j∈a , f ' j=y} = {y . j∈a , y =f ' j}
  by auto
  moreover
  note assms
  moreover from calculation
  have x ∈ a ⇒ y = f ' x ⇒ M(y) for x y
    using transM[OF _ ⟨M(a)⟩] by auto
  moreover from assms
  have M({f'j . j∈a})
    using transM[OF _ ⟨M(a)⟩] RepFun_closed[OF apply_replacement] by simp
  ultimately
  have 2:is_Replace(M, a, λj y. y = f ' j, {f'j . j∈a})
    using Replace_abs[of _ _ λj y. y = f ' j,OF _ ⟨M(a)⟩,THEN iffD2]
    by auto
  with ⟨M({f'j . j∈a})⟩
  show ?thesis
  using
    is_Replace_cong[of _ _ M λj y. y = f ' j λj y. f ' j = y, THEN iffD1,OF _

```

— 2]
 by *auto*
 qed

lemma *is_HAleph_zero*:
 assumes $M(f)$
 shows $\text{is_HAleph}(M, 0, f, \text{res}) \longleftrightarrow \text{res} = \text{nat}$
 unfolding *is_HAleph_def*
 using *Ord_0 If_abs is_Limit_iff is_csucc_iff assms aux_ex_Replace_funapply*
 by *auto*

lemma *is_HAleph_succ*:
 assumes $M(f) \ M(x) \ \text{Ord}(x) \ M(\text{res})$
 shows $\text{is_HAleph}(M, \text{succ}(x), f, \text{res}) \longleftrightarrow \text{res} = \text{csucc_rel}(M, f'x)$
 unfolding *is_HAleph_def*
 using *assms is_Limit_iff is_csucc_iff aux_ex_Replace_funapply If_abs Ord_Union_succ_eq*
 by *simp*

lemma *is_HAleph_limit*:
 assumes $M(f) \ M(x) \ \text{Limit}(x) \ M(\text{res})$
 shows $\text{is_HAleph}(M, x, f, \text{res}) \longleftrightarrow \text{res} = (\bigcup \{y . i \in x, M(i) \wedge M(y) \wedge y = f'i\})$
proof -
 from *assms*
 have $\text{univalent}(M, x, \lambda j y. y = f'j)$
 ($\bigwedge xa y. xa \in x \implies f'xa = y \implies M(y)$)
 $\{y . x \in x, f'x = y\} = \{y . i \in x, M(i) \wedge M(y) \wedge y = f'i\}$
 using *univalent_triv[of M x $\lambda j . f'j$] transM[OF $\langle M(x) \rangle$]*
 by *auto*
moreover
 from *this*
 have $\text{univalent}(M, x, \lambda j y. f'j = y)$
 by (*rule_tac univalent_cong[of x x M $\lambda j y. y = f'j \ \lambda j y. f'j=y$, THEN iffD1], auto*)
moreover
 from *this*
 have $\text{univalent}(M, x, \lambda j y. M(j) \wedge M(y) \wedge f'j = y)$
 by *auto*
 ultimately
 show *?thesis*
 unfolding *is_HAleph_def*
 using *assms is_Limit_iff Limit_is_Ord zero_not_Limit If_abs is_csucc_iff Replace_abs apply_replacement*
 by *auto*
 qed

lemma *is_HAleph_iff*:
 assumes $M(a) \ M(f) \ M(\text{res})$
 shows $\text{is_HAleph}(M, a, f, \text{res}) \longleftrightarrow \text{res} = \text{HAleph_rel}(M, a, f)$
proof(*cases Ord(a)*)

```

case True
note Ord_cases[OF  $\langle \text{Ord}(a) \rangle$ ]
then
show ?thesis
proof(cases )
  case 1
  with True assms
  show ?thesis
    using is_HAleph_zero unfolding HAleph_rel_def
    by simp
next
  case (2 j)
  with True assms
  show ?thesis
    using is_HAleph_succ Ord_Union_succ_eq
    unfolding HAleph_rel_def
    by simp
next
  case 3
  with assms
  show ?thesis
    using is_HAleph_limit zero_not_Limit Limit_is_Ord
    unfolding HAleph_rel_def
    by auto
qed
next
  case False
  then
  have  $\neg \text{Limit}(a) \ a \neq 0 \ \wedge \ x . \text{Ord}(x) \implies a \neq \text{succ}(x)$ 
    using Limit_is_Ord by auto
  with False
  show ?thesis
    unfolding is_HAleph_def HAleph_rel_def
    using assms is_Limit_iff If_abs is_csucc_iff aux_ex_Replace_funapply
    by auto
qed

lemma HAleph_rel_closed [intro, simp]:
  assumes function(f) M(a) M(f)
  shows  $M(\text{HAleph\_rel}(M, a, f))$ 
  unfolding HAleph_rel_def
  using assms apply_replacement
  by simp

lemma Aleph_rel_closed[intro, simp]:
  assumes  $\text{Ord}(a) \ M(a)$ 
  shows  $M(\text{Aleph\_rel}(M, a))$ 
proof -
  have relation2(M, is_HAleph(M), HAleph_rel(M))

```

```

    unfolding relation2_def using is_HAleph_iff assms by simp
  moreover
  have  $\forall x[M]. \forall g[M]. \text{function}(g) \longrightarrow M(\text{HAleph\_rel}(M, x, g))$ 
    using HAleph_rel_closed by simp
  moreover
  note assms
  ultimately
  show ?thesis
    unfolding Aleph_rel_def
    using transrec_closed[of is_HAleph(M) a HAleph_rel(M)]
      haleph_transrec_replacement by simp
qed

```

```

lemma Aleph_rel_zero:  $\aleph_0^M = \text{nat}$ 
  using def_transrec [OF Aleph_rel_def', of _ 0]
  unfolding HAleph_rel_def by simp

```

```

lemma Aleph_rel_succ:  $\text{Ord}(\alpha) \implies M(\alpha) \implies \aleph_{\text{succ}(\alpha)}^M = (\aleph_\alpha^{M+})^M$ 
  using Ord_Union_succ_eq
  by (subst def_transrec [OF Aleph_rel_def'])
    (simp add: HAleph_rel_def)

```

```

lemma Aleph_rel_limit:
  assumes Limit( $\alpha$ )  $M(\alpha)$ 
  shows  $\aleph_\alpha^M = \bigcup \{\aleph_j^M . j \in \alpha\}$ 
proof -
  note trans=transM[OF _  $\langle M(\alpha) \rangle$ ]
  from  $\langle M(\alpha) \rangle$ 
  have  $\aleph_\alpha^M = \text{HAleph\_rel}(M, \alpha, \lambda x \in \alpha. \aleph_x^M)$ 
    using def_transrec [OF Aleph_rel_def', of M  $\alpha$ ] by simp
  also
  have  $\dots = \bigcup \{a . j \in \alpha, M(a) \wedge a = \aleph_j^M\}$ 
    unfolding HAleph_rel_def
    using assms zero_not_Limit Limit_is_Ord trans by auto
  also
  have  $\dots = \bigcup \{\aleph_j^M . j \in \alpha\}$ 
    using Aleph_rel_closed[OF _ trans] Ord_in_Ord Limit_is_Ord[OF  $\langle \text{Limit}(\alpha) \rangle$ ]
  by auto
  finally
  show ?thesis .
qed

```

```

lemma is_Aleph_iff:
  assumes Ord(a)  $M(a)$   $M(\text{res})$ 
  shows  $\text{is\_Aleph}(M, a, \text{res}) \longleftrightarrow \text{res} = \aleph_a^M$ 
proof -
  have relation2(M, is_HAleph(M), HAleph_rel(M))
    unfolding relation2_def using is_HAleph_iff assms by simp
  moreover

```

```

have  $\forall x[M]. \forall g[M]. \text{function}(g) \longrightarrow M(\text{HAleph\_rel}(M, x, g))$ 
  using HAleph_rel_closed by simp
ultimately
show ?thesis
  using assms transrec_abs haleph_transrec_replacement
  unfolding is_Aleph_def Aleph_rel_def
  by simp
qed

end — M_pre_aleph

locale M_aleph = M_pre_aleph +
  assumes
    aleph_rel_replacement: strong_replacement(M,  $\lambda x y. \text{Ord}(x) \wedge y = \aleph_x^M$ )
begin

lemma Aleph_rel_cont:  $\text{Limit}(l) \implies M(l) \implies \aleph_l^M = (\bigcup_{i < l} \aleph_i^M)$ 
  using Limit_is_Ord Aleph_rel_limit
  by (simp add: OUnion_def)

lemma Ord_Aleph_rel:
  assumes Ord(a)
  shows  $M(a) \implies \text{Ord}(\aleph_a^M)$ 
  using  $\langle \text{Ord}(a) \rangle$ 
proof(induct a rule:trans_induct3)
  case 0
  show ?case using Aleph_rel_zero by simp
next
  case (succ x)
  with  $\langle \text{Ord}(x) \rangle$ 
  have  $M(x) \text{ Ord}(\aleph_x^M)$  by simp_all
  with  $\langle \text{Ord}(x) \rangle$ 
  have  $\text{Ord}(\text{csucc\_rel}(M, \aleph_x^M))$ 
    using Card_rel_is_Ord Card_rel_csucc_rel
    by simp
  with  $\langle \text{Ord}(x) \rangle \langle M(x) \rangle$ 
  show ?case using Aleph_rel_succ by simp
next
  case (limit x)
  note  $\text{trans} = \text{trans}M[\text{OF\_} \langle M(x) \rangle]$ 
  from limit
  have  $\aleph_x^M = (\bigcup_{i \in x} \aleph_i^M)$ 
    using Aleph_rel_cont OUnion_def Limit_is_Ord
    by auto
  with limit
  show ?case using Ord_UN trans by auto
qed

lemma Card_rel_Aleph_rel [simp, intro]:

```

```

    assumes  $Ord(a)$  and types:  $M(a)$  shows  $Card^M(\aleph_a^M)$ 
    using assms
  proof (induct rule: trans_induct3)
    case 0
    then
    show ?case
      using Aleph_rel_zero Card_rel_nat by simp
  next
    case (succ x)
    then
    show ?case
      using Card_rel_csucc_rel Ord_Aleph_rel Aleph_rel_succ
      by simp
  next
    case (limit x)
    moreover
    from this
    have  $M(\{y . z \in x, M(y) \wedge M(z) \wedge Ord(z) \wedge y = \aleph_z^M\})$ 
      using aleph_rel_replacement
      by auto
    moreover
    have  $\{y . z \in x, M(y) \wedge M(z) \wedge y = \aleph_z^M\} = \{y . z \in x, M(y) \wedge M(z) \wedge Ord(z) \wedge y = \aleph_z^M\}$ 
    using Ord_in_Ord Limit_is_Ord[OF limit(1)] by simp
    ultimately
    show ?case
      using Ord_Aleph_rel Card_nat Limit_is_Ord Card_relI
      by (subst def_transrec [OF Aleph_rel_def'])
      (auto simp add: HAleph_rel_def)
  qed

lemma Aleph_rel_increasing:
  assumes  $a < b$  and types:  $M(a) M(b)$ 
  shows  $\aleph_a^M < \aleph_b^M$ 
proof -
  { fix  $x$ 
    from assms
    have  $Ord(b)$ 
      by (blast intro: lt_Ord2)
    moreover
    assume  $M(x)$ 
    moreover
    note  $\langle M(b) \rangle$ 
    ultimately
    have  $x < b \implies \aleph_x^M < \aleph_b^M$ 
  }
  proof (induct  $b$  arbitrary:  $x$  rule: trans_induct3)
    case 0 thus ?case by simp
  next
    case (succ b)

```

```

then
show ?case
using Card_rel_csucc_rel Ord_Aleph_rel Ord_Union_succ_eq lt_csucc_rel
  lt_trans[of  $\aleph_b^M$  csucc $^M(\aleph_b^M)$ ]
  by (subst (2) def_transrec[OF Aleph_rel_def])
    (auto simp add: le_iff HAleph_rel_def)
next
case (limit l)
then
have sc: succ(x) < l
  by (blast intro: Limit_has_succ)
then
have  $\aleph_x^M < (\bigcup_{j < l} \aleph_j^M)$ 
  using limit Ord_Aleph_rel Ord_OUN
proof(rule_tac OUN_upper_lt, blast intro: Card_rel_is_Ord ltD lt_Ord)
  from  $\langle x < l \rangle \langle \text{Limit}(l) \rangle$ 
  have Ord(x)
    using Limit_is_Ord Ord_in_Ord
    by (auto dest!: ltD)
  with  $\langle M(x) \rangle$ 
  show  $\aleph_x^M < \aleph_{\text{succ}(x)}^M$ 
    using Card_rel_csucc_rel Ord_Aleph_rel lt_csucc_rel
      ltD[THEN [2] Ord_in_Ord] succ_in_MI[OF  $\langle M(x) \rangle$ ]
      Aleph_rel_succ[of x]
    by (simp)
next
  from  $\langle M(l) \rangle \langle \text{Limit}(l) \rangle$ 
  show Ord( $\bigcup_{j < l} \aleph_j^M$ )
    using Ord_Aleph_rel lt_Ord Limit_is_Ord Ord_in_Ord
    by (rule_tac Ord_OUN)
      (auto dest: transM ltD intro!: Ord_Aleph_rel)
qed
then
show ?case using limit Aleph_rel_cont by simp
qed
}
with types assms
show ?thesis by simp
qed

end — M_aleph

end

```

17 Relative, Cardinal Arithmetic Using AC

```

theory Cardinal_AC_Relative
imports

```

CardinalArith_Relative

begin

locale *M_AC* =

fixes *M*

assumes

choice_ax: *choice_ax*(*M*)

locale *M_cardinal_AC* = *M_cardinal_arith* + *M_AC*

begin

lemma *well_ord_surj_imp_lepoll_rel*:

assumes *well_ord*(*A*,*r*) *h* ∈ *surj*(*A*,*B*) **and**

types: *M*(*A*) *M*(*r*) *M*(*h*) *M*(*B*)

shows $B \lesssim^M A$

proof -

note *eq*=*vimage_fun_sing*[*OF surj_is_fun*[*OF* $\langle h \in _ \rangle$]]

from *assms*

have $(\lambda b \in B. \text{minimum}(r, \{a \in A. h \text{ ` } a = b\})) \in \text{inj}(B, A)$ (**is** $?f \in _$)

using *well_ord_surj_imp_inj_inverse* *assms*(1,2) **by** *simp*

with *assms*

have *M*($?f \text{ ` } b$) **if** $b \in B$ **for** *b*

using *apply_type*[*OF inj_is_fun*[*OF* $\langle ?f \in _ \rangle$]] *that* *transM*[*OF* $_ \langle M(A) \rangle$] **by**

simp

with *assms*

have *M*($?f$)

using *lam_closed_surj_imp_inj_replacement4* *eq* **by** *auto*

with $\langle ?f \in _ \rangle$ *assms*

have $?f \in \text{inj}^M(B, A)$

using *mem_inj_abs* **by** *simp*

with $\langle M(?f) \rangle$

show *thesis* **unfolding** *lepoll_rel_def* **by** *auto*

qed

lemma *surj_imp_well_ord_M*:

assumes *wos*: *well_ord*(*A*,*r*) *h* ∈ *surj*(*A*,*B*)

and

types: *M*(*A*) *M*(*r*) *M*(*h*) *M*(*B*)

shows $\exists s[M]. \text{well_ord}(B, s)$

using *assms* *lepoll_rel_well_ord*

well_ord_surj_imp_lepoll_rel **by** *fast*

lemma *choice_ax_well_ord*: $M(S) \implies \exists r[M]. \text{well_ord}(S, r)$

using *choice_ax* *well_ord_Memrel*[*THEN* *surj_imp_well_ord_M*]

unfolding *choice_ax_def* **by** *auto*

lemma *Finite_cardinal_rel_Finite*:


```

    assumes  $Finite(|i|^M)$   $M(i)$ 
    shows  $Finite(i)$ 
  proof -
    note assms
    moreover from this
    obtain  $r$  where  $M(r)$   $well\_ord(i, r)$ 
      using choice_ax_well_ord by auto
    moreover from calculation
    have  $|i|^M \approx^M i$ 
      using  $well\_ord\_cardinal\_rel\_eqpoll\_rel$ 
      by auto
    ultimately
    show ?thesis
      using  $eqpoll\_rel\_imp\_Finite$ 
      by auto
  qed

end —  $M\_cardinal\_AC$ 

locale  $M\_Pi\_assumptions\_choice = M\_Pi\_assumptions + M\_cardinal\_AC +$ 
  assumes
     $B\_replacement: strong\_replacement(M, \lambda x y. y = B(x))$ 
  and
    — The next one should be derivable from (some variant) of  $B\_replacement$ .
    Proving both instances each time seems inconvenient.
     $minimum\_replacement: M(r) \implies strong\_replacement(M, \lambda x y. y = \langle x, minimum(r, B(x)) \rangle)$ 
  begin

lemma  $AC\_M$ :
  assumes  $a \in A \wedge x. x \in A \implies \exists y. y \in B(x)$ 
  shows  $\exists z[M]. z \in Pi^M(A, B)$ 
  proof -
    have  $M(\bigcup_{x \in A} B(x))$  using assms family_union_closed  $Pi\_assumptions$   $B\_replacement$ 
    by simp
    then
    obtain  $r$  where  $well\_ord(\bigcup_{x \in A} B(x), r)$   $M(r)$ 
      using choice_ax_well_ord by blast
    let  $?f = \lambda x \in A. minimum(r, B(x))$ 
    have  $M(minimum(r, B(x)))$  if  $x \in A$  for  $x$ 
    proof -
      from  $\langle well\_ord(\_, r) \rangle \langle x \in A \rangle$ 
      have  $well\_ord(B(x), r)$  using  $well\_ord\_subset$   $UN\_upper$  by simp
      with assms  $\langle x \in A \rangle \langle M(r) \rangle$ 
      show ?thesis using  $Pi\_assumptions$  by blast
    qed
    with assms and  $\langle M(r) \rangle$ 
    have  $M(?f)$ 
      using  $Pi\_assumptions$   $minimum\_replacement$   $lam\_closed$ 

```

```

    by simp
  moreover from assms and calculation
  have ?f ∈  $Pi^M(A, B)$ 
    using lam_type[OF minimum_in, OF  $\langle well\_ord(\bigcup_{x \in A} B(x), r) \rangle$ , of A B]
    Pi_rel_char by auto
  ultimately
  show ?thesis by blast
qed

lemma AC_Pi_rel: assumes  $\bigwedge x. x \in A \implies \exists y. y \in B(x)$ 
  shows  $\exists z[M]. z \in Pi^M(A, B)$ 
proof (cases A=0)
  interpret Pi0:M_Pi_assumptions_0
    using Pi_assumptions by unfold_locales auto
  case True
  then
  show ?thesis using assms by simp
next
  case False
  then
  obtain a where a ∈ A by auto
  — It is noteworthy that without obtaining an element of A, the final step won't
  work
  with assms
  show ?thesis by (blast intro!: AC_M)
qed

end — M_Pi_assumptions_choice

context M_cardinal_AC
begin

```

17.1 Strengthened Forms of Existing Theorems on Cardinals

```

lemma cardinal_rel_eqpoll_rel:  $M(A) \implies |A|^M \approx^M A$ 
apply (rule choice_ax_well_ord [THEN rexE])
apply (auto intro:well_ord_cardinal_rel_eqpoll_rel)
done

lemmas cardinal_rel_idem = cardinal_rel_eqpoll_rel [THEN cardinal_rel_cong,
simp]

lemma cardinal_rel_eqE:  $|X|^M = |Y|^M \implies M(X) \implies M(Y) \implies X \approx^M Y$ 
apply (rule choice_ax_well_ord [THEN rexE], assumption)
  apply (rule choice_ax_well_ord [THEN rexE, of Y], assumption)
  apply (rule well_ord_cardinal_rel_eqE, assumption+)
done

```

lemma *cardinal_rel_eqpoll_rel_iff*: $M(X) \implies M(Y) \implies |X|^M = |Y|^M \longleftrightarrow X \approx^M Y$

by (*blast intro: cardinal_rel_cong cardinal_rel_eqE*)

lemma *cardinal_rel_disjoint_Un*:

$[| |A|^M = |B|^M; |C|^M = |D|^M; A \cap C = 0; B \cap D = 0; M(A); M(B); M(C); M(D)]$

$\implies |A \cup C|^M = |B \cup D|^M$

by (*simp add: cardinal_rel_eqpoll_rel_iff eqpoll_rel_disjoint_Un*)

lemma *lepoll_rel_imp_cardinal_rel_le*: $A \lesssim^M B \implies M(A) \implies M(B) \implies |A|^M \leq |B|^M$

apply (*rule choice_ax_well_ord [THEN rexE]*) **prefer** 2

apply (*erule well_ord_lepoll_rel_imp_cardinal_rel_le, assumption+*)

done

lemma *cadd_rel_assoc*: $\llbracket M(i); M(j); M(k) \rrbracket \implies (i \oplus^M j) \oplus^M k = i \oplus^M (j \oplus^M k)$

apply (*rule choice_ax_well_ord [THEN rexE]*) **prefer** 2

apply (*rule choice_ax_well_ord [THEN rexE]*) **prefer** 2

apply (*rule choice_ax_well_ord [THEN rexE]*) **prefer** 2

apply (*rule well_ord_cadd_rel_assoc, assumption+*)

done

lemma *cmult_rel_assoc*: $\llbracket M(i); M(j); M(k) \rrbracket \implies (i \otimes^M j) \otimes^M k = i \otimes^M (j \otimes^M k)$

apply (*rule choice_ax_well_ord [THEN rexE]*) **prefer** 2

apply (*rule choice_ax_well_ord [THEN rexE]*) **prefer** 2

apply (*rule choice_ax_well_ord [THEN rexE]*) **prefer** 2

apply (*rule well_ord_cmult_rel_assoc, assumption+*)

done

lemma *cadd_cmult_distrib*: $\llbracket M(i); M(j); M(k) \rrbracket \implies (i \oplus^M j) \otimes^M k = (i \otimes^M k) \oplus^M (j \otimes^M k)$

apply (*rule choice_ax_well_ord [THEN rexE]*) **prefer** 2

apply (*rule choice_ax_well_ord [THEN rexE]*) **prefer** 2

apply (*rule choice_ax_well_ord [THEN rexE]*) **prefer** 2

apply (*rule well_ord_cadd_cmult_distrib, assumption+*)

done

lemma *InfCard_rel_square_eq*: $\text{InfCard}^M(|A|^M) \implies M(A) \implies A \times A \approx^M A$

apply (*rule choice_ax_well_ord [THEN rexE]*) **prefer** 2

apply (*erule well_ord_InfCard_rel_square_eq, assumption, simp_all*)

done

17.2 The relationship between cardinality and le-pollence

lemma *Card_rel_le_imp_lepoll_rel*:

assumes $|A|^M \leq |B|^M$
and types: $M(A) \ M(B)$
shows $A \lesssim^M B$
proof -
have $A \approx^M |A|^M$
by (rule cardinal_rel_eqpoll_rel [THEN eqpoll_rel_sym], simp_all add:types)
also have $\dots \lesssim^M |B|^M$
by (rule le_imp_subset [THEN subset_imp_levpoll_rel]) (rule assms, simp_all add:types)
also have $\dots \approx^M B$
by (rule cardinal_rel_eqpoll_rel, simp_all add:types)
finally show ?thesis **by** (simp_all add:types)
qed

lemma le_Card_rel_iff: $\text{Card}^M(K) \implies M(K) \implies M(A) \implies |A|^M \leq K \longleftrightarrow A \lesssim^M K$
apply (erule Card_rel_cardinal_rel_eq [THEN subst], assumption, rule iffI,
 erule Card_rel_le_imp_levpoll_rel, assumption+)
apply (erule lepoll_rel_imp_cardinal_rel_le, assumption+)
done

lemma cardinal_rel_0_iff_0 [simp]: $M(A) \implies |A|^M = 0 \longleftrightarrow A = 0$
using cardinal_rel_0_eqpoll_rel_0_iff [THEN iffD1]
 cardinal_rel_eqpoll_rel_iff [THEN iffD1, OF _ nonempty]
by auto

lemma cardinal_rel_lt_iff_lesspoll_rel:
assumes $i: \text{Ord}(i)$ **and**
 types: $M(i) \ M(A)$
shows $i < |A|^M \longleftrightarrow i \prec^M A$
proof
assume $i < |A|^M$
hence $i \prec^M |A|^M$
by (blast intro: lt_Card_rel_imp_lesspoll_rel types)
also have $\dots \approx^M A$
by (rule cardinal_rel_eqpoll_rel) (simp_all add:types)
finally show $i \prec^M A$ **by** (simp_all add:types)
next
assume $i \prec^M A$
also have $\dots \approx^M |A|^M$
by (blast intro: cardinal_rel_eqpoll_rel eqpoll_rel_sym types)
finally have $i \prec^M |A|^M$ **by** (simp_all add:types)
thus $i < |A|^M$ **using** i types
by (force intro: cardinal_rel_lt_imp_lt_lesspoll_rel_cardinal_rel_lt)
qed

lemma cardinal_rel_le_imp_levpoll_rel: $i \leq |A|^M \implies M(i) \implies M(A) \implies i \lesssim^M A$
by (blast intro: lt_Ord Card_rel_le_imp_levpoll_rel Ord_cardinal_rel_le le_trans)

17.3 Other Applications of AC

We have an example of instantiating a locale involving higher order variables inside a proof, by using the assumptions of the first order, active locale.

```

lemma surj_rel_implies_inj_rel:
  assumes  $f: f \in \text{surj}^M(X, Y)$  and
    types:  $M(f) \ M(X) \ M(Y)$ 
  shows  $\exists g[M]. \ g \in \text{inj}^M(Y, X)$ 
proof -
  from types
  interpret  $M\_Pi\_assumptions\_choice \_ Y \ \lambda y. f - \{y\}$ 
    by unfold_locales (auto intro:surj_imp_inj_replacement dest:transM)
  from  $f \ AC\_Pi\_rel$ 
  obtain  $z$  where  $z: z \in Pi^M(Y, \lambda y. f - \{y\})$ 
    — In this and the following ported result, it is not clear how uniformly are
    ”_char” theorems to be used
    using surj_rel_char
    by (auto simp add: surj_def types) (fast dest: apply_Pair)
  show ?thesis
proof
  show  $z \in \text{inj}^M(Y, X) \ M(z)$ 
    using  $z \ \text{surj\_is\_fun}[of \ f \ X \ Y] \ f \ Pi\_rel\_char$ 
    by (auto dest: apply_type Pi_memberD
      intro: apply_equality Pi_type f_imp_injective
      simp add:types mem_surj_abs)
  qed
qed

```

Kunen’s Lemma 10.20

```

lemma surj_rel_implies_cardinal_rel_le:
  assumes  $f: f \in \text{surj}^M(X, Y)$  and
    types:  $M(f) \ M(X) \ M(Y)$ 
  shows  $|Y|^M \leq |X|^M$ 
proof (rule lepoll_rel_imp_cardinal_rel_le)
  from  $f \ [THEN \ \text{surj\_rel\_implies\_inj\_rel}]$ 
  obtain  $g$  where  $g \in \text{inj}^M(Y, X)$ 
    by (blast intro:types)
  then
  show  $Y \lesssim^M X$ 
    using inj_rel_char
    by (auto simp add: def_lepoll_rel types)
qed (simp_all add:types)

```

end — $M_cardinal_AC$

The set-theoretic universe.

abbreviation

```

Universe ::  $i \Rightarrow o \ (\langle \mathcal{V} \rangle)$  where
   $\mathcal{V}(x) \equiv \text{True}$ 

```

```

lemma separation_absolute: separation( $\mathcal{V}$ ,  $P$ )
  unfolding separation_def
  by (rule rallI, rule_tac  $x = \{x \in \_ . P(x)\}$  in rexI) auto

lemma univalent_absolute:
  assumes univalent( $\mathcal{V}$ ,  $A$ ,  $P$ )  $P(x, b)$   $x \in A$ 
  shows  $P(x, y) \implies y = b$ 
  using assms
  unfolding univalent_def by force

lemma replacement_absolute: strong_replacement( $\mathcal{V}$ ,  $P$ )
  unfolding strong_replacement_def
proof (intro rallI impI)
  fix  $A$ 
  assume univalent( $\mathcal{V}$ ,  $A$ ,  $P$ )
  then
  show  $\exists Y[\mathcal{V}]. \forall b[\mathcal{V}]. b \in Y \longleftrightarrow (\exists x[\mathcal{V}]. x \in A \wedge P(x, b))$ 
    by (rule_tac  $x = \{y. x \in A, P(x, y)\}$  in rexI)
      (auto dest:univalent_absolute[of _ P])
qed

lemma Union_ax_absolute: Union_ax( $\mathcal{V}$ )
  unfolding Union_ax_def big_union_def
  by (auto intro:rexI[of _  $\bigcup$ ])

lemma upair_ax_absolute: upair_ax( $\mathcal{V}$ )
  unfolding upair_ax_def upair_def rall_def rex_def
  by (auto)

lemma power_ax_absolute: power_ax( $\mathcal{V}$ )
proof -
  {
    fix  $x$ 
    have  $\forall y[\mathcal{V}]. y \in Pow(x) \longleftrightarrow (\forall z[\mathcal{V}]. z \in y \longrightarrow z \in x)$ 
      by auto
  }
  then
  show power_ax( $\mathcal{V}$ )
    unfolding power_ax_def powerset_def subset_def by blast
qed

locale  $M\_cardinal\_UN = M\_Pi\_assumptions\_choice\_K\ X$  for  $K\ X +$ 
  assumes
  — The next assumption is required by  $(\bigwedge x. \llbracket ?Q(x); Ord(x) \rrbracket \implies \exists y[M]. ?Q(y)$ 
 $\wedge Ord(y) \implies M(\mu x. ?Q(x))$ 
   $X\_witness\_in\ M: w \in X(x) \implies M(x)$ 
  and
   $lam\_m\_replacement: M(f) \implies strong\_replacement(M,$ 

```

```

     $\lambda x y. y = \langle x, \mu i. x \in X(i), f \text{ ' } (\mu i. x \in X(i)) \text{ ' } x \rangle$ 
and
    inj_replacement:
     $M(x) \implies \text{strong\_replacement}(M, \lambda y z. y \in \text{inj}^M(X(x), K) \wedge z = \{\langle x, y \rangle\})$ 
     $\text{strong\_replacement}(M, \lambda x y. y = \text{inj}^M(X(x), K))$ 
     $\text{strong\_replacement}(M,$ 
       $\lambda x z. z = \text{Sigfun}(x, \lambda i. \text{inj}^M(X(i), K)))$ 
     $M(r) \implies \text{strong\_replacement}(M,$ 
       $\lambda x y. y = \langle x, \text{minimum}(r, \text{inj}^M(X(x), K)) \rangle)$ 

begin

lemma UN_closed:  $M(\bigcup_{i \in K} X(i))$ 
  using family_union_closed B_replacement Pi_assumptions by simp

Kunen's Lemma 10.21

lemma cardinal_rel_UN_le:
  assumes  $K: \text{InfCard}^M(K)$ 
  shows  $(\bigwedge i. i \in K \implies |X(i)|^M \leq K) \implies |\bigcup_{i \in K} X(i)|^M \leq K$ 
proof (simp add: K InfCard_rel_is_Card_rel le_Card_rel_iff Pi_assumptions)
  have  $M(f) \implies M(\lambda x \in (\bigcup_{x \in K} X(x)). \langle \mu i. x \in X(i), f \text{ ' } (\mu i. x \in X(i)) \text{ ' } x \rangle)$ 
for  $f$ 
    using lam_m_replacement X_witness_in_M Least_closed' Pi_assumptions
    UN_closed
    by (rule_tac lam_closed) (auto dest:transM)
  note types = this Pi_assumptions UN_closed
  have [intro]:  $\text{Ord}(K)$  by (blast intro: InfCard_rel_is_Card_rel
    Card_rel_is_Ord K types)
  interpret pri:  $M\_Pi\_assumptions\_choice \_ K \lambda i. \text{inj}^M(X(i), K)$ 
    using inj_replacement Pi_assumptions transM[of \_ K]
    by unfold_locales (simp_all del:mem_inj_abs)
  assume asm:  $\bigwedge i. i \in K \implies X(i) \lesssim^M K$ 
  then
  have  $\bigwedge i. i \in K \implies M(\text{inj}^M(X(i), K))$ 
    by (auto simp add: types)
  interpret V:  $M\_N\_Perm\ M\ \mathcal{V}$ 
    using separation_absolute replacement_absolute Union_ax_absolute
    power_ax_absolute upair_ax_absolute
    by unfold_locales auto
  note bad_simps[simp del] = V.N.Forall_in_M_iff V.N.Equal_in_M_iff
    V.N.nonempty
  have  $\text{abs:inj\_rel}(\mathcal{V}, x, y) = \text{inj}(x, y)$  for  $x\ y$ 
    using V.N.inj_rel_char by simp
  from asm
  have  $\bigwedge i. i \in K \implies \exists f[M]. f \in \text{inj}^M(X(i), K)$ 
    by (simp add: types def_lepoll_rel)
  then
  obtain  $f$  where  $f \in (\prod_{i \in K} \text{inj}^M(X(i), K))\ M(f)$ 
    using pri.AC_Pi_rel pri.Pi_rel_char by auto

```

```

with abs
have  $f: f \in (\prod i \in K. \text{inj}(X(i), K))$ 
  using Pi_weaken_type[OF  $V.\text{inj\_rel\_transfer}, \text{of } f \text{ } K \text{ } X \text{ } \lambda \_ . K$ ]
  Pi_assumptions by simp
{ fix z
  assume  $z: z \in (\bigcup i \in K. X(i))$ 
  then obtain i where  $i: i \in K \text{ } \text{Ord}(i) \text{ } z \in X(i)$ 
    by (blast intro: Ord_in_Ord [of K])
  hence  $(\mu i. z \in X(i)) \leq i$  by (fast intro: Least_le)
  hence  $(\mu i. z \in X(i)) < K$  by (best intro: lt_trans1 ltI i)
  hence  $(\mu i. z \in X(i)) \in K$  and  $z \in X(\mu i. z \in X(i))$ 
    by (auto intro: LeastI ltD i)
} note mems = this
have  $(\bigcup i \in K. X(i)) \lesssim^M K \times K$ 
proof (simp add:types def lepoll_rel)
  show  $\exists f[M]. f \in \text{inj}(\bigcup x \in K. X(x), K \times K)$ 
    apply (rule rexI)
    apply (rule_tac  $c = \lambda z. \langle \mu i. z \in X(i), f \text{ } \langle \mu i. z \in X(i) \rangle \text{ } z \rangle$ 
      and  $d = \lambda \langle i, j \rangle. \text{converse } (f \text{ } i) \text{ } j$  in lam_injective)
    apply (force intro: f inj_is_fun mems apply_type Perm.left_inverse) +
    apply (simp add:types  $\langle M(f) \rangle$ )
    done
  qed
also have  $\dots \approx^M K$ 
by (simp add: K InfCard_rel_square_eq InfCard_rel_is_Card_rel
  Card_rel_cardinal_rel_eq types)
finally have  $(\bigcup i \in K. X(i)) \lesssim^M K$  by (simp_all add:types)
then
show ?thesis
  by (simp add: K InfCard_rel_is_Card_rel le_Card_rel_iff types)
qed

end — M_cardinal_UN

end

```

18 Relativization of Finite Functions

```

theory FiniteFun_Relative
imports
  Delta_System_Lemma.ZF_Library
  Lambda_Replacement
begin

lemma function_subset:
   $\text{function}(f) \implies g \subseteq f \implies \text{function}(g)$ 
  unfolding function_def subset_def by auto

```



```

lemma FiniteFunI :
  assumes  $f \in \text{Fin}(A \times B)$  function(f)
  shows  $f \in A \multimap B$ 
  using assms
proof(induct)
  case 0
  then show ?case using emptyI by simp
next
  case (cons p f)
  moreover
  from assms this
  have  $\text{fst}(p) \in A$   $\text{snd}(p) \in B$  function(f)
    using snd_type[OF <p∈_>] function_subset
    by auto
  moreover
  from  $\langle \text{function}(\text{cons}(p, f)) \rangle \langle p \notin f \rangle \langle p \in \_ \rangle$ 
  have  $\text{fst}(p) \notin \text{domain}(f)$ 
    unfolding function_def
    by force
  ultimately
  show ?case
    using consI[of fst(p) _ snd(p)]
    by auto
qed

```

18.1 The set of finite binary sequences

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

definition

seqspace :: $[i, i] \Rightarrow i \multimap [100, 1]100$ **where**
 $B^{<\alpha} \equiv \bigcup_{n \in \alpha}. (n \rightarrow B)$

lemma *seqspaceI[intro]*: $n \in \alpha \Longrightarrow f: n \rightarrow B \Longrightarrow f \in B^{<\alpha}$
unfolding *seqspace_def* **by** *blast*

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

locale *M_seqspace* = *M_trancl* + *M_replacement* +
assumes
seqspace_replacement: $M(B) \Longrightarrow \text{strong_replacement}(M, \lambda n z. n \in \text{nat} \wedge \text{is_funspace}(M, n, B, z))$
begin

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

```

unfolding seqspace_def using seqspace_replacement[of B] RepFun_closed2
by simp

end

schematic_goal seqspace_fm_auto:
  assumes
     $i \in \text{nat } j \in \text{nat } h \in \text{nat } \text{env} \in \text{list}(A)$ 
  shows
     $(\exists om \in A. \text{omega}(\#\#A, om) \wedge \text{nth}(i, \text{env}) \in om \wedge \text{is\_funspace}(\#\#A, \text{nth}(i, \text{env}),$ 
 $\text{nth}(h, \text{env}), \text{nth}(j, \text{env}))) \longleftrightarrow (A, \text{env} \models (?sqsprp(i, j, h)))$ 
  unfolding is_funspace_def
  by (insert assms ; (rule iff_sats | simp)+)
synthesize seqspace_rel from schematic seqspace_fm_auto
arity_theorem for seqspace_rel_fm

```

18.2 Representation of finite functions

A function $f \in A \rightarrow_{\text{fin}} B$ can be represented by a function $g \in |f| \rightarrow A \times B$. It is clear that f can be represented by any $g' = g \cdot \pi$, where π is a permutation $\pi \in \text{dom}(g) \rightarrow \text{dom}(g)$. We use this representation of $A \rightarrow_{\text{fin}} B$ to prove that our model is closed under $_ \rightarrow_{\text{fin}} _$.

A function $g \in n \rightarrow A \times B$ that is functional in the first components.

definition $\text{cons_like} :: i \Rightarrow o$ **where**
 $\text{cons_like}(f) \equiv \forall i \in \text{domain}(f) . \forall j \in i . \text{fst}(f'i) \neq \text{fst}(f'j)$

relativize cons_like cons_like_rel

lemma (in $M_seqspace$) cons_like_abs :
 $M(f) \implies \text{cons_like}(f) \longleftrightarrow \text{cons_like_rel}(M, f)$
unfolding cons_like_def cons_like_rel_def
using fst_abs
by simp

definition $\text{FiniteFun_iso} :: [i, i, i, i, i] \Rightarrow o$ **where**
 $\text{FiniteFun_iso}(A, B, n, g, f) \equiv (\forall i \in n . g'i \in f) \wedge (\forall ab \in f. (\exists i \in n. g'i = ab))$

From a function $g \in n \rightarrow A \times B$ we obtain a finite function in $A \dashv\!\mid\!> B$.

definition $\text{to_FiniteFun} :: i \Rightarrow i$ **where**
 $\text{to_FiniteFun}(f) \equiv \{f'i. i \in \text{domain}(f)\}$

definition $\text{FiniteFun_Repr} :: [i, i] \Rightarrow i$ **where**
 $\text{FiniteFun_Repr}(A, B) \equiv \{f \in (A \times B)^{<\omega} . \text{cons_like}(f) \}$

locale $M_FiniteFun = M_seqspace +$
assumes

```

    cons_like_separation : separation(M, λf. cons_like_rel(M, f))
  and
    separation_is_function : separation(M, is_function(M))
begin

lemma supset_separation: separation(M, λ x. ∃ a. ∃ b. x = ⟨a, b⟩ ∧ b ⊆ a)
  using separation_pair separation_subset lam_replacement_fst lam_replacement_snd
  by simp

lemma to_finiteFun_replacement: strong_replacement(M, λx y. y = range(x))
  using lam_replacement_range lam_replacement_imp_strong_replacement
  by simp

lemma fun_range_eq: f ∈ A → B ⇒ {f' i . i ∈ domain(f)} = range(f)
  using ZF_Library.range_eq_image[of f] domain_of_fun image_fun func.apply_rangeI
  by simp

lemma FiniteFun_fst_type:
  assumes h ∈ A -||> B p ∈ h
  shows fst(p) ∈ domain(h)
  using assms
  by(induct h, auto)

lemma FinFun_closed:
  M(A) ⇒ M(B) ⇒ M(⋃ {n → A × B . n ∈ ω})
  using cartprod_closed seqspace_closed
  unfolding seqspace_def by simp

lemma cons_like_lt :
  assumes n ∈ ω f ∈ succ(n) → A × B cons_like(f)
  shows restrict(f, n) ∈ n → A × B cons_like(restrict(f, n))
  using assms
proof (auto simp add: le_imp_subset restrict_type2)
  from ⟨f ∈ _⟩
  have D: domain(restrict(f, n)) = n domain(f) = succ(n)
    using domain_of_fun domain_restrict by auto
  {
    fix i j
    assume i ∈ domain(restrict(f, n)) (is i ∈ ?D) j ∈ i
    with ⟨n ∈ _⟩ D
    have j ∈ ?D i ∈ n j ∈ n using Ord_trans[of j] by simp_all
    with D ⟨cons_like(f)⟩ ⟨j ∈ n⟩ ⟨i ∈ n⟩ ⟨j ∈ i⟩
    have fst(restrict(f, n) 'i) ≠ fst(restrict(f, n) 'j)
      using restrict_if unfolding cons_like_def by auto
  }
  then show cons_like(restrict(f, n))
    unfolding cons_like_def by auto
qed

```

A finite function $f \in A -||> B$ can be represented by a function $g \in n \rightarrow$

$A \times B$, with $n = |f|$.

lemma *FiniteFun_iso_intro1*:

assumes $f \in (A \rightarrow B)$

shows $\exists n \in \omega . \exists g \in n \rightarrow A \times B. \text{FiniteFun_iso}(A, B, n, g, f) \wedge \text{cons_like}(g)$

using *assms*

proof(*induct f, force simp add: emptyI FiniteFun_iso_def cons_like_def*)

case (*consI a b h*)

then obtain $n g$ **where**

HI: $n \in \omega \ g \in n \rightarrow A \times B \ \text{FiniteFun_iso}(A, B, n, g, h) \ \text{cons_like}(g)$ **by** *auto*

let $?G = \lambda i \in \text{succ}(n) . \text{if } i = n \text{ then } \langle a, b \rangle \text{ else } g'i$

from *HI* $\langle a \in _ \rangle \langle b \in _ \rangle$

have $G: ?G \in \text{succ}(n) \rightarrow A \times B$

by (*auto intro: lam_type*)

have $\text{FiniteFun_iso}(A, B, \text{succ}(n), ?G, \text{cons}(\langle a, b \rangle, h))$

unfolding *FiniteFun_iso_def*

proof(*intro conjI*)

{

fix i

assume $i \in \text{succ}(n)$

then consider $i = n \mid i \in n \wedge i \neq n$ **by** *auto*

then have $?G \ 'i \in \text{cons}(\langle a, b \rangle, h)$

using *HI*

by(*cases, simp; auto simp add: HI FiniteFun_iso_def*)

}

then show $\forall i \in \text{succ}(n). ?G \ 'i \in \text{cons}(\langle a, b \rangle, h) ..$

next

{ **fix** ab'

assume $ab' \in \text{cons}(\langle a, b \rangle, h)$

then

consider $ab' = \langle a, b \rangle \mid ab' \in h$ **using** *cons_iff* **by** *auto*

then

have $\exists i \in \text{succ}(n) . ?G \ 'i = ab'$ **unfolding** *FiniteFun_iso_def*

proof(*cases, simp*)

case 2

with *HI* **obtain** i

where $i \in n \ g'i = ab'$ **unfolding** *FiniteFun_iso_def* **by** *auto*

with *HI* **show** *?thesis* **using** *ltI[OF i ∈ _]* **by** *auto*

qed

}

then

show $\forall ab \in \text{cons}(\langle a, b \rangle, h). \exists i \in \text{succ}(n). ?G \ 'i = ab ..$

qed

with *HI G*

have 1: $?G \in \text{succ}(n) \rightarrow A \times B \ \text{FiniteFun_iso}(A, B, \text{succ}(n), ?G, \text{cons}(\langle a, b \rangle, h)) \ \text{succ}(n) \in \omega$

by *simp_all*

have *cons_like*($?G$)

proof -

from $\langle ?G \in _ \rangle \langle g \in _ \rangle$

have *domain*(g) = n **using** *domain_of_fun* **by** *simp*

```

{
  fix i j
  assume  $i \in \text{domain}(?G)$   $j \in i$ 
  with  $\langle n \in \_ \rangle$ 
  have  $j \in n$  using Ord_trans[of  $j \_ n$ ] by auto
  from  $\langle i \in \_ \rangle$  consider (a)  $i = n \wedge i \notin n$  | (b)  $i \in n$  by auto
  then
  have  $\text{fst}(?G'i) \neq \text{fst}(?G'j)$ 
  proof(cases)
    case a
    with  $\langle j \in n \rangle$  HI
    have  $?G'i = \langle a, b \rangle$   $?G'j = g'j$   $g'j \in h$ 
      unfolding FiniteFun_iso_def by auto
    with  $\langle a \notin \_ \rangle$   $\langle h \in \_ \rangle$ 
    show ?thesis using FiniteFun_fst_type by auto
  next
  case b
  with  $\langle i \in n \rangle$   $\langle j \in i \rangle$   $\langle j \in n \rangle$  HI  $\langle \text{domain}(g) = n \rangle$ 
  show ?thesis unfolding cons_like_def
    using mem_not_refl by auto
  qed
}
then show ?thesis unfolding cons_like_def by auto
qed
with 1 show ?case by auto
qed

```

All the representations of $f \in A \multimap B$ are equal.

```

lemma FiniteFun_isoD :
  assumes  $n \in \omega$   $g \in n \rightarrow A \times B$   $f \in A \multimap B$  FiniteFun_iso( $A, B, n, g, f$ )
  shows  $\text{to\_FiniteFun}(g) = f$ 
proof
  show  $\text{to\_FiniteFun}(g) \subseteq f$ 
  proof
    fix ab
    assume  $ab \in \text{to\_FiniteFun}(g)$ 
    moreover
    note assms
    moreover from calculation
    obtain i where  $i \in n$   $g'i = ab$   $ab \in A \times B$ 
      unfolding to_FiniteFun_def using domain_of_fun by auto
    ultimately
    show  $ab \in f$  unfolding FiniteFun_iso_def by auto
  qed
next
show  $f \subseteq \text{to\_FiniteFun}(g)$ 
proof
  fix ab
  assume  $ab \in f$ 

```

```

with assms
obtain i where i ∈ n g' i = ab ab ∈ A × B
  unfolding FiniteFun_iso_def by auto
with assms
show ab ∈ to_FiniteFun(g)
  unfolding to_FiniteFun_def
  using domain_of_fun by auto
qed
qed

```

```

lemma to_FiniteFun_succ_eq :
  assumes n ∈ ω f ∈ succ(n) → A
  shows to_FiniteFun(f) = cons(f'n, to_FiniteFun(restrict(f, n)))
  using assms domain_restrict domain_of_fun
  unfolding to_FiniteFun_def by auto

```

If $g \in n \rightarrow A \times B$ is *cons_like*, then it is a representation of $to_FiniteFun(g)$.

```

lemma FiniteFun_iso_intro_to:
  assumes n ∈ ω g ∈ n → A × B cons_like(g)
  shows to_FiniteFun(g) ∈ (A -||> B) ∧ FiniteFun_iso(A, B, n, g, to_FiniteFun(g))
  using assms
proof(induct n arbitrary: g rule: nat_induct)
  case 0
  fix g
  assume g ∈ 0 → A × B
  then
  have g = 0 by simp
  then have to_FiniteFun(g) = 0 unfolding to_FiniteFun_def by simp
  then show to_FiniteFun(g) ∈ (A -||> B) ∧ FiniteFun_iso(A, B, 0, g, to_FiniteFun(g))
    using emptyI unfolding FiniteFun_iso_def by simp
next
  case (succ x)
  fix g
  let ?g' = restrict(g, x)
  assume g ∈ succ(x) → A × B cons_like(g)
  with succ.hyps ⟨g ∈ _⟩
  have cons_like(?g') ?g' ∈ x → A × B g' x ∈ A × B domain(g) = succ(x)
    using cons_like_lt succI1 apply_funtype domain_of_fun by simp_all
  with succ.hyps ⟨?g' ∈ _⟩ ⟨x ∈ ω⟩
  have HI:
    to_FiniteFun(?g') ∈ A -||> B (is (?h) ∈ _)
    FiniteFun_iso(A, B, x, ?g', to_FiniteFun(?g'))
    by simp_all
  then
  have fst(g' x) ∉ domain(?h)
  proof -
  {
    assume fst(g' x) ∈ domain(?h)
    with HI ⟨x ∈ _⟩

```

```

    obtain i b
      where  $i \in x \text{ } \langle \text{fst}(\text{?}g' i), b \rangle \in \text{?}h \text{ } i \in x \text{ } \text{fst}(g' x) = \text{fst}(\text{?}g' i)$ 
      unfolding FiniteFun_iso_def using ltI by auto
    with  $\langle \text{cons\_like}(g), \text{domain}(g) = \_ \rangle$ 
    have False
      unfolding cons_like_def by auto
  }
  then show ?thesis ..
qed
with HI assms  $\langle g' x \in \_ \rangle$ 
have  $\text{cons}(g' x, \text{?}h) \in A \text{ } \text{--} \text{--} B$  (is  $\text{?}h' \in \_$ ) using consI by auto
have FiniteFun_iso(A,B,succ(x),g, $\text{?}h'$ )
  unfolding FiniteFun_iso_def
proof
  { fix i
    assume  $i \in \text{succ}(x)$ 
    with  $\langle x \in \_ \rangle$  consider (a)  $i = x$  | (b)  $i \in x \wedge i \neq x$  by auto
    then have  $g' i \in \text{?}h'$ 
    proof(cases,simp)
      case b
      with  $\langle \text{FiniteFun\_iso}(\_, \_, \_, \text{?}g', \text{?}h) \rangle$ 
      show ?thesis unfolding FiniteFun_iso_def by simp
    qed
  }
  then show  $\forall i \in \text{succ}(x). g' i \in \text{cons}(g' x, \text{?}h)$  ..
next
{
  fix ab
  assume  $ab \in \text{?}h'$ 
  then consider  $ab = g' x \mid ab \in \text{?}h$  using cons_iff by auto
  then
    have  $\exists i \in \text{succ}(x). g' i = ab$  unfolding FiniteFun_iso_def
    proof(cases,simp)
      case 2
      with HI obtain i
        where  $2: i \in x \text{ } \text{?}g' i = ab$  unfolding FiniteFun_iso_def by auto
      with  $\langle x \in \_ \rangle$ 
      have  $i \neq x \text{ } i \in \text{succ}(x)$  using ltI[OF  $\langle i \in \_ \rangle$ ] by auto
      with 2 HI show ?thesis by auto
    qed
  } then show  $\forall ab \in \text{cons}(g' x, \text{?}h). \exists i \in \text{succ}(x). g' i = ab$  ..
qed
with  $\langle \text{?}h' \in \_ \rangle$ 
show  $\text{to\_FiniteFun}(g) \in A \text{ } \text{--} \text{--} B \wedge \text{FiniteFun\_iso}(A, B, \text{succ}(x), g, \text{to\_FiniteFun}(g))$ 
  using to\_FiniteFun_succ_eq[OF  $\langle x \in \_ \rangle \text{ } \langle g \in \_ \rangle, \text{symmetric}$ ] by auto
qed

lemma FiniteFun_iso_intro2:
  assumes  $n \in \omega \text{ } f \in n \rightarrow A \times B \text{ } \text{cons\_like}(f)$ 

```

```

shows  $\exists g \in (A \multimap B) . \text{FiniteFun\_iso}(A, B, n, f, g)$ 
using assms FiniteFun_iso_intro_to by blast

lemma FiniteFun_eq_range_Repr :
  shows  $\{\text{range}(h) . h \in \text{FiniteFun\_Repr}(A, B)\} = \{\text{to\_FiniteFun}(h) . h \in \text{FiniteFun\_Repr}(A, B)\}$ 
  unfolding FiniteFun_Repr_def to_FiniteFun_def seqspace_def
  using fun_range_eq
  by(intro equalityI subsetI, auto)

lemma FiniteFun_eq_to_FiniteFun_Repr :
  shows  $A \multimap B = \{\text{to\_FiniteFun}(h) . h \in \text{FiniteFun\_Repr}(A, B)\}$ 
  (is  $?Y = ?X$ )
proof
  {
    fix f
    assume  $f \in A \multimap B$ 
    then obtain n g where
      1:  $n \in \omega$   $g \in n \rightarrow A \times B$  FiniteFun_iso(A, B, n, g, f) cons_like(g)
      using FiniteFun_iso_intro1 by blast
    with  $\langle f \in \_ \rangle$ 
    have cons_like(g)  $f = \text{to\_FiniteFun}(g)$   $\text{domain}(g) = n$   $g \in \text{FiniteFun\_Repr}(A, B)$ 
      using FiniteFun_isoD domain_of_fun
      unfolding FiniteFun_Repr_def
      by auto
    with 1 have  $f \in ?X$ 
      by auto
  } then show  $?Y \subseteq ?X$  ..
next
  {
    fix f
    assume  $f \in ?X$ 
    then obtain g where
      A:  $g \in \text{FiniteFun\_Repr}(A, B)$   $f = \text{to\_FiniteFun}(g)$  cons_like(g)
      using RepFun_iff unfolding FiniteFun_Repr_def by auto
    then obtain n where  $n \in \omega$   $g \in n \rightarrow A \times B$   $\text{domain}(g) = n$ 
      unfolding FiniteFun_Repr_def using domain_of_fun by force
    with A
    have  $f \in ?Y$ 
      using FiniteFun_iso_intro_to by simp
  } then show  $?X \subseteq ?Y$  ..
qed

lemma FiniteFun_Repr_closed :
  assumes  $M(A) \ M(B)$ 
  shows  $M(\text{FiniteFun\_Repr}(A, B))$ 
  unfolding FiniteFun_Repr_def
  using assms cartprod_closed

```



```

    seqspace_closed separation_closed cons_like_abs cons_like_separation
  by simp

lemma to_FiniteFun_closed:
  assumes  $M(A) \ f \in A$ 
  shows  $M(\text{range}(f))$ 
  using assms transM[of _ A] by simp

lemma To_FiniteFun_Repr_closed :
  assumes  $M(A) \ M(B)$ 
  shows  $M(\{ \text{range}(h) \ . \ h \in \text{FiniteFun\_Repr}(A,B) \})$ 
  using assms FiniteFun_Repr_closed
    RepFun_closed to_finiteFun_replacement
    to_FiniteFun_closed[OF FiniteFun_Repr_closed]
  by simp

lemma FiniteFun_closed[intro,simp] :
  assumes  $M(A) \ M(B)$ 
  shows  $M(A \rightarrow B)$ 
  using assms To_FiniteFun_Repr_closed FiniteFun_eq_to_FiniteFun_Repr
    FiniteFun_eq_range_Repr
  by simp

end —  $M\_FiniteFun$ 

end

```

19 Library of basic ZF results

```

theory ZF_Library_Relative
  imports
    Aleph_Relative — must be before Cardinal_AC_Relative!
    Cardinal_AC_Relative
    FiniteFun_Relative
begin

lemma (in  $M\_cardinal\_arith\_jump$ ) csucc_rel_cardinal_rel:
  assumes  $\text{Ord}(\kappa) \ M(\kappa)$ 
  shows  $(|\kappa|^{M+})^M = (\kappa^+)^M$ 
proof (intro le_anti_sym) — show both inequalities
  from assms
  have hips:  $M((\kappa^+)^M) \ \text{Ord}((\kappa^+)^M) \ \kappa < (\kappa^+)^M$ 
    using Card_rel_csucc_rel[THEN Card_rel_is_Ord]
    csucc_rel_basic by simp_all
  then
  show  $(|\kappa|^{M+})^M \leq (\kappa^+)^M$ 
    using Ord_cardinal_rel_le[THEN lt_trans1]
    Card_rel_csucc_rel
  unfolding csucc_rel_def

```

```

    by (rule_tac Least_antitone) (assumption, simp_all add:assms)
  from assms
  have  $\kappa < L$  if  $\text{Card}^M(L) \mid \kappa \mid^M < L$   $M(L)$  for  $L$ 
    using that
      Card_rel_is_Ord leI Card_rel_le_iff[of  $\kappa$   $L$ ]
    by (rule_tac ccontr, auto dest: not_lt_imp_le) (fast dest: le_imp_not_lt)
  with hips
  show  $(\kappa^+)^M \leq (\mid \kappa \mid^{M+})^M$ 
    using Ord_cardinal_rel_le[THEN lt_trans1] Card_rel_csucc_rel
    unfolding csucc_rel_def
    by (rule_tac Least_antitone) (assumption, auto simp add:assms)
qed

lemma (in  $M$  cardinal_arith_jump) csucc_rel_le_mono:
  assumes  $\kappa \leq \nu$   $M(\kappa)$   $M(\nu)$ 
  shows  $(\kappa^+)^M \leq (\nu^+)^M$ 
proof (cases  $\kappa = \nu$ )
  case True
  with assms
  show ?thesis using Card_rel_csucc_rel [THEN Card_rel_is_Ord] by simp
next
  case False
  with assms
  have  $\kappa < \nu$  using le_neq_imp_lt by simp
  show ?thesis— by way of contradiction
proof (rule ccontr)
  assume  $\neg (\kappa^+)^M \leq (\nu^+)^M$ 
  with assms
  have  $(\nu^+)^M < (\kappa^+)^M$ 
    using Card_rel_csucc_rel[THEN Card_rel_is_Ord] le_Ord2 lt_Ord
    by (intro not_le_iff_lt[THEN iffD1]) auto
  with assms
  have  $(\nu^+)^M \leq \mid \kappa \mid^M$ 
    using le_Ord2[THEN Card_rel_csucc_rel, of  $\kappa$   $\nu$ ]
    Card_rel_lt_csucc_rel_iff[of  $(\nu^+)^M \mid \kappa \mid^M$ , THEN iffD1]
    csucc_rel_cardinal_rel[OF lt_Ord] leI[of  $(\nu^+)^M (\kappa^+)^M$ ]
    by simp
  with assms
  have  $(\nu^+)^M \leq \kappa$ 
    using Ord_cardinal_rel_le[OF lt_Ord] le_trans by auto
  with assms
  have  $\nu < \kappa$ 
    using csucc_rel_basic le_Ord2[THEN Card_rel_csucc_rel, of  $\kappa$   $\nu$ ] Card_rel_is_Ord
    le_Ord2
    by (rule_tac j= $(\nu^+)^M$  in lt_trans2) simp_all
  with  $\langle \kappa < \nu \rangle$ 
  show False using le_imp_not_lt leI by blast
qed
qed

```

lemma (in *M_cardinal_AC*) *cardinal_rel_succ_not_0*: $|A|^M = \text{succ}(n) \implies M(A) \implies M(n) \implies A \neq 0$
by *auto*

reldb_add functional *Finite Finite* — wrongly done? Finite is absolute
relativize functional *Finite_to_one Finite_to_one_rel external*

notation *Finite_to_one_rel* ($\langle \text{Finite}'_to_one -'(_,_)' \rangle$)

abbreviation

Finite_to_one_r_set :: $[i, i, i] \Rightarrow i \langle \text{Finite}'_to_one -'(_,_)' \rangle$ **where**
Finite_to_one^M(*X*, *Y*) \equiv *Finite_to_one_rel*($\#\#M, X, Y$)

locale *M_ZF_library* = *M_cardinal_arith* + *M_aleph* + *M_FiniteFun* + *M_replacement_extra*
begin

lemma *Finite_Collect_imp*: *Finite*($\{x \in X \mid Q(x)\}$) \implies *Finite*($\{x \in X \mid M(x) \wedge Q(x)\}$)
(is *Finite*(*?A*) \implies *Finite*(*?B*))
using *subset_Finite[of ?B ?A]* **by** *auto*

lemma *Finite_to_one_relI[intro]*:
assumes $f: X \rightarrow^M Y \bigwedge y. y \in Y \implies \text{Finite}(\{x \in X \mid f'x = y\})$
and *types*: *M*(*f*) *M*(*X*) *M*(*Y*)
shows $f \in \text{Finite_to_one}^M(X, Y)$
using *assms Finite_Collect_imp unfolding Finite_to_one_rel_def*
by (*simp*)

lemma *Finite_to_one_relI'[intro]*:
assumes $f: X \rightarrow^M Y \bigwedge y. y \in Y \implies \text{Finite}(\{x \in X \mid M(x) \wedge f'x = y\})$
and *types*: *M*(*f*) *M*(*X*) *M*(*Y*)
shows $f \in \text{Finite_to_one}^M(X, Y)$
using *assms unfolding Finite_to_one_rel_def*
by (*simp*)

lemma *Finite_to_one_relD[dest]*:
 $f \in \text{Finite_to_one}^M(X, Y) \implies f: X \rightarrow^M Y$
 $f \in \text{Finite_to_one}^M(X, Y) \implies y \in Y \implies M(Y) \implies \text{Finite}(\{x \in X \mid M(x) \wedge f'x = y\})$
unfolding *Finite_to_one_rel_def* **by** *simp_all*

lemma *Diff_bij_rel*:
assumes $\forall A \in F. X \subseteq A$
and *types*: *M*(*F*) *M*(*X*) **shows** $(\lambda A \in F. A - X) \in \text{bij}^M(F, \{A - X \mid A \in F\})$
using *assms def_inj_rel def_surj_rel unfolding bij_rel_def*
apply (*auto*)

```

apply (subgoal_tac M( $\lambda A \in F. A - X$ ) M( $\{A - X . A \in F\}$ ))
apply (auto simp add:mem_function_space_rel_abs)
apply (rule_tac lam_type, auto)
prefer 4
apply (subgoal_tac M( $\lambda A \in F. A - X$ ) M( $\{A - X . A \in F\}$ ))
apply (tactic <distinct_subgoals_tac>)
apply (auto simp add:mem_function_space_rel_abs)
apply (rule_tac lam_type, auto) prefer 3
apply (subst subset_Diff_Un[of X])
apply auto
proof -
  from types
  show M( $\{A - X . A \in F\}$ )
    using diff_replacement
    by (rule_tac RepFun_closed) (auto dest:transM[of __ F])
  from types
  show M( $\lambda A \in F. A - X$ )
    using Pair_diff_replacement
    by (rule_tac lam_closed, auto dest:transM)
qed

lemma function_space_rel_nonempty:
  assumes  $b \in B$  and types: M(B) M(A)
  shows ( $\lambda x \in A. b$ ) :  $A \rightarrow^M B$ 
proof -
  note assms
  moreover from this
  have M( $\lambda x \in A. b$ )
    using tag_replacement by (rule_tac lam_closed, auto dest:transM)
  ultimately
  show ?thesis
    by (simp add:mem_function_space_rel_abs)
qed

lemma mem_function_space_rel:
  assumes  $f \in A \rightarrow^M y$  M(A) M(y)
  shows  $f \in A \rightarrow y$ 
  using assms function_space_rel_char by simp

lemmas range_fun_rel_subset_codomain = range_fun_subset_codomain[OF mem_function_space_rel]

end — M_ZF_library

context M_Pi_assumptions
begin

lemma mem_Pi_rel:  $f \in Pi^M(A, B) \implies f \in Pi(A, B)$ 
  using trans_closed mem_Pi_rel_abs
  by force

```

```

lemmas  $Pi\_rel\_rangeD = Pi\_rangeD[OF\ mem\_Pi\_rel]$ 

lemmas  $rel\_apply\_Pair = apply\_Pair[OF\ mem\_Pi\_rel]$ 

lemmas  $rel\_apply\_rangeI = apply\_rangeI[OF\ mem\_Pi\_rel]$ 

lemmas  $Pi\_rel\_range\_eq = Pi\_range\_eq[OF\ mem\_Pi\_rel]$ 

lemmas  $Pi\_rel\_vimage\_subset = Pi\_vimage\_subset[OF\ mem\_Pi\_rel]$ 

end —  $M\_Pi\_assumptions$ 

context  $M\_ZF\_library$ 
begin

lemma  $mem\_bij\_rel$ :  $\llbracket f \in bij^M(A,B); M(A); M(B) \rrbracket \implies f \in bij(A,B)$ 
  using  $bij\_rel\_char$  by  $simp$ 

lemma  $mem\_inj\_rel$ :  $\llbracket f \in inj^M(A,B); M(A); M(B) \rrbracket \implies f \in inj(A,B)$ 
  using  $inj\_rel\_char$  by  $simp$ 

lemma  $mem\_surj\_rel$ :  $\llbracket f \in surj^M(A,B); M(A); M(B) \rrbracket \implies f \in surj(A,B)$ 
  using  $surj\_rel\_char$  by  $simp$ 

lemmas  $rel\_apply\_in\_range = apply\_in\_range[OF\_ \_ mem\_function\_space\_rel]$ 

lemmas  $rel\_range\_eq\_image = ZF\_Library.range\_eq\_image[OF\ mem\_function\_space\_rel]$ 

lemmas  $rel\_Image\_sub\_codomain = Image\_sub\_codomain[OF\ mem\_function\_space\_rel]$ 

lemma  $rel\_inj\_to\_Image$ :  $\llbracket f:A \rightarrow^M B; f \in inj^M(A,B); M(A); M(B) \rrbracket \implies f \in$ 
 $inj^M(A, f''A)$ 
  using  $inj\_to\_Image[OF\ mem\_function\_space\_rel\ mem\_inj\_rel]$ 
   $transM[OF\_ function\_space\_rel\_closed]$  by  $simp$ 

lemma  $inj\_rel\_imp\_surj\_rel$ :
  fixes  $f\ b$ 
  defines  $[simp]$ :  $ifx(x) \equiv$  if  $x \in range(f)$  then  $converse(f) 'x$  else  $b$ 
  assumes  $f \in inj^M(B,A)$   $b \in B$  and types:  $M(f)\ M(B)\ M(A)$ 
  shows  $(\lambda x \in A. ifx(x)) \in surj^M(A,B)$ 
proof -
  from types and  $\langle b \in B \rangle$ 
  have  $M(\lambda x \in A. ifx(x))$ 
    using  $ifx\_replacement$  by  $(rule\_tac\ lam\_closed)\ (auto\ dest:transM)$ 
  with  $assms(2-)$ 
  show  $?thesis$ 
  using  $inj\_imp\_surj\ mem\_surj\_abs$  by  $simp$ 
qed

```

lemma *function_space_rel_disjoint_Un*:
assumes $f \in A \rightarrow^M B$ $g \in C \rightarrow^M D$ $A \cap C = 0$
and types: $M(A)$ $M(B)$ $M(C)$ $M(D)$
shows $f \cup g \in (A \cup C) \rightarrow^M (B \cup D)$
using *assms fun_Pi_disjoint_Un* [*OF mem_function_space_rel*
mem_function_space_rel, *OF assms(1)* — *assms(2)*]
function_space_rel_char **by** *auto*

lemma *restrict_eq_imp_Un_into_function_space_rel*:
assumes $f \in A \rightarrow^M B$ $g \in C \rightarrow^M D$ $\text{restrict}(f, A \cap C) = \text{restrict}(g, A \cap C)$
and types: $M(A)$ $M(B)$ $M(C)$ $M(D)$
shows $f \cup g \in (A \cup C) \rightarrow^M (B \cup D)$
using *assms restrict_eq_imp_Un_into_Pi* [*OF mem_function_space_rel*
mem_function_space_rel, *OF assms(1)* — *assms(2)*]
function_space_rel_char **by** *auto*

lemma *lepoll_relD[dest]*: $A \lesssim^M B \implies \exists f[M]. f \in \text{inj}^M(A, B)$
unfolding *lepoll_rel_def* .

— Should the assumptions be on f or on A and B ? Should BOTH be intro rules?

lemma *lepoll_relI[intro]*: $f \in \text{inj}^M(A, B) \implies M(f) \implies A \lesssim^M B$
unfolding *lepoll_rel_def* **by** *blast*

lemma *eqpollD[dest]*: $A \approx^M B \implies \exists f[M]. f \in \text{bij}^M(A, B)$
unfolding *eqpoll_rel_def* .

— Same as $\llbracket ?f \in \text{inj}^M(?A, ?B); M(?f) \rrbracket \implies ?A \lesssim^M ?B$

lemma *bij_rel_imp_eqpoll_rel[intro]*: $f \in \text{bij}^M(A, B) \implies M(f) \implies A \approx^M B$
unfolding *eqpoll_rel_def* **by** *blast*

lemma *restrict_bij_rel*: — Unused
assumes $f \in \text{inj}^M(A, B)$ $C \subseteq A$
and types: $M(A)$ $M(B)$ $M(C)$
shows $\text{restrict}(f, C) \in \text{bij}^M(C, f \restriction C)$
using *assms restrict_bij inj_rel_char bij_rel_char* **by** *auto*

lemma *range_of_subset_eqpoll_rel*:
assumes $f \in \text{inj}^M(X, Y)$ $S \subseteq X$
and types: $M(X)$ $M(Y)$ $M(S)$
shows $S \approx^M f \restriction S$
using *assms restrict_bij bij_rel_char*
trans_inj_rel_closed [*OF* $\langle f \in \text{inj}^M(X, Y) \rangle$]
unfolding *eqpoll_rel_def*
by (*rule_tac* $x = \text{restrict}(f, S)$ **in** *rexI*) *auto*

lemmas *inj_rel_is_fun* = *inj_is_fun* [*OF mem_inj_rel*]

lemma *inj_rel_bij_rel_range*: $f \in \text{inj}^M(A, B) \implies M(A) \implies M(B) \implies f \in$

```

bijM(A, range(f))
  using bij_rel_char inj_rel_char inj_bij_range by force

lemma bij_rel_is_inj_rel: f ∈ bijM(A, B) ⇒ M(A) ⇒ M(B) ⇒ f ∈ injM(A, B)
  unfolding bij_rel_def by simp

lemma inj_rel_weaken_type: [| f ∈ injM(A, B); B ⊆ D; M(A); M(B); M(D) |]
  ==> f ∈ injM(A, D)
  using inj_rel_char inj_rel_is_fun inj_weaken_type by auto

lemma bij_rel_converse_bij_rel [TC]: f ∈ bijM(A, B) ⇒ M(A) ⇒ M(B) ==>
  converse(f) ∈ bijM(B, A)
  using bij_rel_char by force

lemma bij_rel_is_fun_rel: f ∈ bijM(A, B) ⇒ M(A) ⇒ M(B) ⇒ f ∈ A →M B
  using bij_rel_char mem_function_space_rel_abs bij_is_fun by simp

lemmas bij_rel_is_fun = bij_rel_is_fun_rel [THEN mem_function_space_rel]

lemma comp_bij_rel:
  g ∈ bijM(A, B) ⇒ f ∈ bijM(B, C) ⇒ M(A) ⇒ M(B) ⇒ M(C) ⇒ (f O
  g) ∈ bijM(A, C)
  using bij_rel_char comp_bij by force

lemma inj_rel_converse_fun: f ∈ injM(A, B) ⇒ M(A) ⇒ M(B) ⇒ converse(f)
  ∈ range(f) →M A
proof -
  assume f ∈ injM(A, B) M(A) M(B)
  then
  have M(f) M(converse(f)) M(range(f)) f ∈ inj(A, B)
    using inj_rel_char converse_closed range_closed
    by auto
  then
  show ?thesis
    using inj_converse_inj function_space_rel_char inj_is_fun ⟨M(A)⟩ by auto
qed

lemma fg_imp_bijective_rel:
  assumes f ∈ A →M B g ∈ B →M A f O g = id(B) g O f = id(A) M(A) M(B)
  shows f ∈ bijM(A, B)
  using assms mem_bij_abs fg_imp_bijective mem_function_space_rel_abs [THEN
  iffD2] function_space_rel_char
  by auto

end — M_ZF_library

relativize functional cexp cexp_rel external
relationalize cexp_rel is_cexp

```

```

context M_ZF_library
begin

— MOVE THIS to an appropriate place
is_iff_rel for function_space
proof -
  assume  $M(A) \ M(B) \ M(res)$ 
  then
    show ?thesis
    using function_space_rel_char mem_function_space_rel_abs
    unfolding is_funspace_def is_function_space_def
    by (safe, intro equalityI; clarsimp) (auto dest:transM[of _ res])
qed

is_iff_rel for cexp
  using is_cardinal_iff is_function_space_iff unfolding cexp_rel_def is_cexp_def
  by (simp)

rel_closed for cexp unfolding cexp_rel_def by simp

end — M_ZF_library

synthesize is_cexp from definition assuming nonempty
notation is_cexp_fm ( $\langle \cdot \uparrow \cdot \rangle$  is  $\cdot \rangle$ )
arity_theorem for is_cexp_fm

abbreviation
   $cexp\_r :: [i, i, i \Rightarrow o] \Rightarrow i \ (\langle \cdot \uparrow \cdot \rangle \text{ is } \cdot)$  where
   $cexp\_r(x, y, M) \equiv cexp\_rel(M, x, y)$ 

abbreviation
   $cexp\_r\_set :: [i, i, i] \Rightarrow i \ (\langle \cdot \uparrow \cdot \rangle \text{ is } \cdot)$  where
   $cexp\_r\_set(x, y, M) \equiv cexp\_rel(\#\#M, x, y)$ 

context M_ZF_library
begin

lemma Card_rel_cexp_rel:  $M(\kappa) \Longrightarrow M(\nu) \Longrightarrow Card^M(\kappa \uparrow \nu, M)$ 
  unfolding cexp_rel_def by simp

— Restoring congruence rule, but NOTE: beware
declare conj_cong[cong]

lemma eq_csucc_rel_ord:
   $Ord(i) \Longrightarrow M(i) \Longrightarrow (i^+)^M = (|i|^{M+})^M$ 
  using Card_rel_lt_iff Least_cong unfolding csucc_rel_def by auto

lemma lesspoll_succ_rel:

```



```

assumes  $Ord(\kappa)$   $M(\kappa)$ 
shows  $\kappa \lesssim^M (\kappa^+)^M$ 
using  $csucc\_rel\_basic$   $assms$   $lt\_Card\_rel\_imp\_lesspoll\_rel$ 
 $Card\_rel\_csucc\_rel$   $lepoll\_rel\_iff\_lepoll\_rel$ 
by auto

lemma  $lesspoll\_rel\_csucc\_rel$ :
  assumes  $Ord(\kappa)$ 
  and  $types: M(\kappa)$   $M(d)$ 
  shows  $d \prec^M (\kappa^+)^M \longleftrightarrow d \lesssim^M \kappa$ 
proof
  assume  $d \prec^M (\kappa^+)^M$ 
  moreover
  note  $Card\_rel\_csucc\_rel$   $assms$   $Card\_rel\_is\_Ord$ 
  moreover from calculation
  have  $Card^M((\kappa^+)^M)$   $M((\kappa^+)^M)$   $Ord((\kappa^+)^M)$ 
    using  $Card\_rel\_is\_Ord$  by simp_all
  moreover from calculation
  have  $d \prec^M (|\kappa|^{M+})^M$   $d \approx^M |d|^M$ 
    using  $eq\_csucc\_rel\_ord[OF\_ \langle M(\kappa) \rangle]$ 
 $lesspoll\_rel\_imp\_eqpoll\_rel$   $eqpoll\_rel\_sym$  by simp_all
  moreover from calculation
  have  $|d|^M < (|\kappa|^{M+})^M$ 
    using  $lesspoll\_cardinal\_lt\_rel$  by simp
  moreover from calculation
  have  $|d|^M \lesssim^M |\kappa|^M$ 
    using  $Card\_rel\_lt\_csucc\_rel\_iff\_le\_imp\_lepoll\_rel$  by simp
  moreover from calculation
  have  $|d|^M \lesssim^M \kappa$ 
    using  $Ord\_cardinal\_rel\_eqpoll\_rel$   $lepoll\_rel\_eq\_trans$ 
by simp
  ultimately
  show  $d \lesssim^M \kappa$ 
    using  $eq\_lepoll\_rel\_trans$  by simp
next
  from  $\langle Ord(\kappa) \rangle$ 
  have  $\kappa < (\kappa^+)^M$   $Card^M((\kappa^+)^M)$   $M((\kappa^+)^M)$ 
    using  $Card\_rel\_csucc\_rel$   $lt\_csucc\_rel\_iff\_types$   $eq\_csucc\_rel\_ord[OF\_ \langle M(\kappa) \rangle]$ 
by simp_all
  then
  have  $\kappa \prec^M (\kappa^+)^M$ 
    using  $lt\_Card\_rel\_imp\_lesspoll\_rel[OF\_ \langle \kappa < \_ \rangle]$   $types$  by simp
  moreover
  assume  $d \lesssim^M \kappa$ 
  ultimately
  have  $d \lesssim^M (\kappa^+)^M$ 
    using  $Card\_rel\_csucc\_rel$   $types$   $lesspoll\_succ\_rel$   $lepoll\_rel\_trans$   $\langle Ord(\kappa) \rangle$ 
by simp

```

```

moreover
from  $\langle d \lesssim^M \kappa \rangle \langle \text{Ord}(\kappa) \rangle$ 
have  $(\kappa^+)^M \lesssim^M \kappa$  if  $d \approx^M (\kappa^+)^M$ 
  using eqpoll_rel_sym[OF that] types eq_lepoll_rel_trans[OF _  $\langle d \lesssim^M \kappa \rangle$ ]
  by simp
moreover from calculation  $\langle \kappa \prec^M (\kappa^+)^M \rangle$ 
have False if  $d \approx^M (\kappa^+)^M$ 
  using lesspoll_rel_irrefl[OF _  $\langle M((\kappa^+)^M) \rangle$ ] lesspoll_rel_trans1 types that
  by auto
ultimately
show  $d \prec^M (\kappa^+)^M$ 
  unfolding lesspoll_rel_def by auto
qed

```

```

lemma Infinite_imp_nats_lepoll:
  assumes Infinite(X)  $n \in \omega$ 
  shows  $n \lesssim X$ 
  using  $\langle n \in \omega \rangle$ 
proof (induct)
  case 0
  then
    show ?case using empty_lepollI by simp
next
  case (succ x)
  show ?case
  proof -
    from  $\langle \text{Infinite}(X) \rangle$  and  $\langle x \in \omega \rangle$ 
    have  $\neg (x \approx X)$ 
      using eqpoll_sym unfolding Finite_def by auto
    with  $\langle x \lesssim X \rangle$ 
    obtain f where  $f \in \text{inj}(x, X)$   $f \notin \text{surj}(x, X)$ 
      unfolding bij_def eqpoll_def by auto
    moreover from this
    obtain b where  $b \in X \ \forall a \in x. f'a \neq b$ 
      using inj_is_fun unfolding surj_def by auto
    ultimately
    have  $f \in \text{inj}(x, X - \{b\})$ 
      unfolding inj_def by (auto intro: Pi_type)
    then
    have  $\text{cons}(\langle x, b \rangle, f) \in \text{inj}(\text{succ}(x), \text{cons}(b, X - \{b\}))$ 
      using inj_extend[of  $f \ x \ X - \{b\} \ x \ b$ ] unfolding succ_def
      by (auto dest: mem_irrefl)
    moreover from  $\langle b \in X \rangle$ 
    have  $\text{cons}(b, X - \{b\}) = X$  by auto
    ultimately
    show  $\text{succ}(x) \lesssim X$  by auto
  qed
qed

```

```

lemma nepoll_imp_nepoll_rel :
  assumes  $\neg x \approx X \ M(x) \ M(X)$ 
  shows  $\neg (x \approx^M X)$ 
  using assms unfolding eqpoll_def eqpoll_rel_def by simp

lemma Infinite_imp_nats_lepoll_rel:
  assumes Infinite( $X$ )  $n \in \omega$ 
  and types:  $M(X)$ 
  shows  $n \lesssim^M X$ 
  using  $\langle n \in \omega \rangle$ 
proof (induct)
  case 0
  then
  show ?case using empty_lepoll_rell types by simp
next
  case (succ  $x$ )
  show ?case
  proof -
    from  $\langle \text{Infinite}(X) \rangle$  and  $\langle x \in \omega \rangle$ 
    have  $\neg (x \approx X) \ M(x) \ M(\text{succ}(x))$ 
      using eqpoll_sym unfolding Finite_def by auto
    then
    have  $\neg (x \approx^M X)$ 
      using nepoll_imp_nepoll_rel types by simp
    with  $\langle x \lesssim^M X \rangle$ 
    obtain  $f$  where  $f \in \text{inj}^M(x, X) \ f \notin \text{surj}^M(x, X) \ M(f)$ 
      unfolding bij_rel_def eqpoll_rel_def by auto
    with  $\langle M(X) \rangle \ \langle M(x) \rangle$ 
    have  $f \notin \text{surj}(x, X) \ f \in \text{inj}(x, X)$ 
      using surj_rel_char by simp_all
    moreover
    from this
    obtain  $b$  where  $b \in X \ \forall a \in x. f'a \neq b$ 
      using inj_is_fun unfolding surj_def by auto
    moreover
    from this calculation  $\langle M(x) \rangle$ 
    have  $f \in \text{inj}(x, X - \{b\}) \ M(\langle x, b \rangle)$ 
      unfolding inj_def using transM[OF _ M(X)]
      by (auto intro: Pi_type)
    moreover
    from this
    have  $\text{cons}(\langle x, b \rangle, f) \in \text{inj}(\text{succ}(x), \text{cons}(b, X - \{b\})) \ (\text{is } ?g \in \_)$ 
      using inj_extend[of f x X - {b} x b] unfolding succ_def
      by (auto dest: mem_irrefl)
    moreover
    note  $\langle M(\langle x, b \rangle) \rangle \ \langle M(f) \rangle \ \langle b \in X \rangle \ \langle M(X) \rangle \ \langle M(\text{succ}(x)) \rangle$ 
    moreover from this
    have  $M(?g) \ \text{cons}(b, X - \{b\}) = X$  by auto
    moreover from calculation

```

```

have ?g∈inj_rel(M,succ(x),X)
  using mem_inj_abs by simp
with ⟨M(?g)⟩
show succ(x) ≲M X using lepoll_relI by simp
qed
qed

lemma lepoll_rel_imp_lepoll: A ≲M B ⇒ M(A) ⇒ M(B) ⇒ A ≲ B
  unfolding lepoll_rel_def by auto

lemma zero_lesspoll_rel: assumes 0<κ M(κ) shows 0 ≲M κ
  using assms eqpoll_rel_0_iff[THEN iffD1, of κ] eqpoll_rel_sym
  unfolding lesspoll_rel_def lepoll_rel_def
  by (auto simp add:inj_def)

lemma lepoll_rel_nat_imp_Infinite: ω ≲M X ⇒ M(X) ⇒ Infinite(X)
  using lepoll_nat_imp_Infinite lepoll_rel_imp_lepoll by simp

lemma InfCard_rel_imp_Infinite: InfCardM(κ) ⇒ M(κ) ⇒ Infinite(κ)
  using le_imp_lepoll_rel[THEN lepoll_rel_nat_imp_Infinite, of κ]
  unfolding InfCard_rel_def by simp

lemma lt_surj_rel_empty_imp_Card_rel:
  assumes Ord(κ) ∧ α. α < κ ⇒ surjM(α,κ) = 0
  and types:M(κ)
  shows CardM(κ)
proof -
{
  define min where min≡μ x. ∃ f[M]. f ∈ bijM(x,κ)
  moreover
  note ⟨Ord(κ)⟩ ⟨M(κ)⟩
  moreover
  assume |κ|M < κ
  moreover from calculation
  have ∃ f. f ∈ bijM(min,κ)
    using LeastI[of λ i. i ≈M κ κ, OF eqpoll_rel_refl]
    unfolding Card_rel_def cardinal_rel_def eqpoll_rel_def
    by (auto)
  moreover from calculation
  have min < κ
    using lt_transI[of min μ i. M(i) ∧ (∃ f[M]. f ∈ bijM(i, κ)) κ]
    Least_le[of λ i. i ≈M κ |κ|M, OF Ord_cardinal_rel_eqpoll_rel]
    unfolding Card_rel_def cardinal_rel_def eqpoll_rel_def
    by (simp)
  moreover
  note ⟨min < κ ⇒ surjM(min,κ) = 0⟩
  ultimately
  have False
    unfolding bij_rel_def by simp
}

```

```

}
with assms
show ?thesis
  using Ord_cardinal_rel_le[of  $\kappa$ ] not_lt_imp_le[of  $|\kappa|^M \kappa$ ] le_anti_sym
  unfolding Card_rel_def by auto
qed

end — M_ZF_library

relativize functional mono_map mono_map_rel external
relationalize mono_map_rel is mono_map
synthesize is_mono_map from_definition assuming nonempty

notation mono_map_rel ( $\langle \text{mono}'\_map^{-'}(\_,\_,\_,\_) \rangle$ )

abbreviation
  mono_map_r_set ::  $[i,i,i,i,i] \Rightarrow i$  ( $\langle \text{mono}'\_map^{-'}(\_,\_,\_,\_) \rangle$ ) where
  mono_mapM(a,r,b,s)  $\equiv$  mono_map_rel( $\#\#M,a,r,b,s$ )

context M_ZF_library
begin

lemma mono_map_rel_char:
  assumes  $M(a) \ M(b)$ 
  shows  $\text{mono\_map}^M(a,r,b,s) = \{f \in \text{mono\_map}(a,r,b,s) . M(f)\}$ 
  using assms function_space_rel_char unfolding mono_map_rel_def mono_map_def
  by auto

Just a sample of porting results on mono_map

lemma mono_map_rel_mono:
  assumes
     $f \in \text{mono\_map}^M(A,r,B,s) \ B \subseteq C$ 
    and types: $M(A) \ M(B) \ M(C)$ 
  shows
     $f \in \text{mono\_map}^M(A,r,C,s)$ 
  using assms mono_map_mono mono_map_rel_char by auto

lemma nats_le_InfCard_rel:
  assumes  $n \in \omega \ \text{InfCard}^M(\kappa)$ 
  shows  $n \leq \kappa$ 
  using assms Ord_is_Transset
    le_trans[of  $n \ \omega \ \kappa$ , OF le_subset_iff[THEN iffD2]]
  unfolding InfCard_rel_def Transset_def by simp

lemma nat_into_InfCard_rel:
  assumes  $n \in \omega \ \text{InfCard}^M(\kappa)$ 
  shows  $n \in \kappa$ 
  using assms le_imp_subset[of  $\omega \ \kappa$ ]
  unfolding InfCard_rel_def by auto

```

lemma *Finite_cardinal_rel_in_nat* [*simp*]:
 assumes *Finite*(*A*) *M*(*A*) **shows** $|A|^M \in \omega$
proof -
 note *assms*
 moreover from *this*
 obtain *n* where $n \in \omega$ *M*(*n*) $A \approx n$
 unfolding *Finite_def* by *auto*
 moreover from *calculation*
 obtain *f* where $f \in \text{bij}(A, n)$ $f: A \rightarrow n$
 using *Finite_Fin*[*THEN fun_FiniteFunI, OF subset_refl*] *bij_is_fun*
 unfolding *eqpoll_def* by *auto*
 ultimately
 have $A \approx^M n$ unfolding *eqpoll_rel_def* by (*auto dest:transM*)
 with *assms* and $\langle M(n) \rangle$
 have $n \approx^M A$ using *eqpoll_rel_sym* by *simp*
 moreover
 note $\langle n \in \omega \rangle$ $\langle M(n) \rangle$
 ultimately
 show ?*thesis*
 using *assms* *Least_le*[*of* $\lambda i. M(i) \wedge i \approx^M A$ *n*]
lt_trans1[*of* $n \in \omega$, *THEN ltD*]
 unfolding *cardinal_rel_def* *Finite_def*
 by (*auto dest!:naturals_lt_nat*)
qed

lemma *Finite_cardinal_rel_eq_cardinal*:
 assumes *Finite*(*A*) *M*(*A*) **shows** $|A|^M = |A|$
proof -
 — Copy-paste from $\llbracket \text{Finite}(\text{?}A); M(\text{?}A) \rrbracket \implies |\text{?}A|^M \in \omega$
 note *assms*
 moreover from *this*
 obtain *n* where $n \in \omega$ *M*(*n*) $A \approx n$
 unfolding *Finite_def* by *auto*
 moreover from *this*
 have $|A| = n$
 using *cardinal_cong*[*of* *A n*]
nat_into_Card[*THEN Card_cardinal_eq, of n*] by *simp*
 moreover from *calculation*
 obtain *f* where $f \in \text{bij}(A, n)$ $f: A \rightarrow n$
 using *Finite_Fin*[*THEN fun_FiniteFunI, OF subset_refl*] *bij_is_fun*
 unfolding *eqpoll_def* by *auto*
 ultimately
 have $A \approx^M n$ unfolding *eqpoll_rel_def* by (*auto dest:transM*)
 with *assms* and $\langle M(n) \rangle$ $\langle n \in \omega \rangle$
 have $|A|^M = n$
 using *cardinal_rel_cong*[*of* *A n*]
nat_into_Card_rel[*THEN Card_rel_cardinal_rel_eq, of n*]
 by *simp*

with $\langle |A| = n \rangle$
show *?thesis* **by** *simp*
qed

lemma *Finite_imp_cardinal_rel_cons*:
assumes *FA*: *Finite*(*A*) **and** *a*: $a \notin A$ **and** *types*: $M(A) \ M(a)$
shows $|cons(a, A)|^M = succ(|A|^M)$
using *assms Finite_imp_cardinal_cons Finite_cardinal_rel_eq_cardinal* **by** *simp*

lemma *Finite_imp_succ_cardinal_rel_Diff*:
assumes *Finite*(*A*) $a \in A$ $M(A)$
shows $succ(|A - \{a\}|^M) = |A|^M$
proof -
from *assms*
have *inM*: $M(A - \{a\}) \ M(a) \ M(A)$ **by** (*auto dest:transM*)
with $\langle Finite(A) \rangle$
have $succ(|A - \{a\}|^M) = succ(|A - \{a\}|)$
using *Diff_subset THEN subset_Finite,*
THEN Finite_cardinal_rel_eq_cardinal, of A {a} **by** *simp*
also from *assms*
have $\dots = |A|$
using *Finite_imp_succ_cardinal_Diff* **by** *simp*
also from *assms*
have $\dots = |A|^M$ **using** *Finite_cardinal_rel_eq_cardinal* **by** *simp*
finally
show *?thesis* .
qed

lemma *InfCard_rel_Aleph_rel*:
notes *Aleph_rel_zero*[*simp*]
assumes *Ord*(α)
and *types*: $M(\alpha)$
shows $InfCard^M(\aleph_\alpha^M)$
proof -
have $\neg (\aleph_\alpha^M \in \omega)$
proof (*cases* $\alpha = 0$)
case *True*
then show *?thesis* **using** *mem_irrefl* **by** *auto*
next
case *False*
with *assms*
have $\omega \in \aleph_\alpha^M$ **using** *Ord_0_lt[of* α] *ltD* **by** (*auto dest:Aleph_rel_increasing*)
then show *?thesis* **using** *foundation* **by** *blast*
qed
with *assms*
have $\neg (|\aleph_\alpha^M|^M \in \omega)$
using *Card_rel_cardinal_rel_eq* **by** *auto*
with *assms*

```

have Infinite( $\aleph_\alpha^M$ ) using Ord_Aleph_rel by clarsimp
with assms
show ?thesis
  using Inf_Card_rel_is_InfCard_rel by simp
qed

lemmas Limit_Aleph_rel = InfCard_rel_Aleph_rel[THEN InfCard_rel_is_Limit]

bundle Ord_dests = Limit_is_Ord[dest] Card_rel_is_Ord[dest]
bundle Aleph_rel_dests = Aleph_rel_cont[dest]
bundle Aleph_rel_intros = Aleph_rel_increasing[intro!]
bundle Aleph_rel_mem_dests = Aleph_rel_increasing[OF ltI, THEN ltD, dest]

lemma f_imp_injective_rel:
  assumes  $f \in A \rightarrow^M B \ \forall x \in A. d(f \cdot x) = x \ M(A) \ M(B)$ 
  shows  $f \in \text{inj}^M(A, B)$ 
  using assms
  apply (simp (no_asm_simp) add: def_inj_rel)
  apply (auto intro: subst_context [THEN box_equals])
  done

lemma lam_injective_rel:
  assumes  $\bigwedge x. x \in A \implies c(x) \in B$ 
     $\bigwedge x. x \in A \implies d(c(x)) = x$ 
     $\forall x[M]. M(c(x)) \ \text{lam\_replacement}(M, c)$ 
     $M(A) \ M(B)$ 
  shows  $(\lambda x \in A. c(x)) \in \text{inj}^M(A, B)$ 
  using assms function_space_rel_char lam_replacement_iff_lam_closed
  by (rule_tac d = d in f_imp_injective_rel)
    (auto simp add: lam_type)

lemma f_imp_surjective_rel:
  assumes  $f \in A \rightarrow^M B \ \bigwedge y. y \in B \implies d(y) \in A \ \bigwedge y. y \in B \implies f \cdot d(y) = y$ 
     $M(A) \ M(B)$ 
  shows  $f \in \text{surj}^M(A, B)$ 
  using assms
  by (simp add: def_surj_rel, blast)

lemma lam_surjective_rel:
  assumes  $\bigwedge x. x \in A \implies c(x) \in B$ 
     $\bigwedge y. y \in B \implies d(y) \in A$ 
     $\bigwedge y. y \in B \implies c(d(y)) = y$ 
     $\forall x[M]. M(c(x)) \ \text{lam\_replacement}(M, c)$ 
     $M(A) \ M(B)$ 
  shows  $(\lambda x \in A. c(x)) \in \text{surj}^M(A, B)$ 
  using assms function_space_rel_char lam_replacement_iff_lam_closed
  by (rule_tac d = d in f_imp_surjective_rel)
    (auto simp add: lam_type)

```



```

lemma lam_bijective_rel:
  assumes  $\bigwedge x. x \in A \implies c(x) \in B$ 
     $\bigwedge y. y \in B \implies d(y) \in A$ 
     $\bigwedge x. x \in A \implies d(c(x)) = x$ 
     $\bigwedge y. y \in B \implies c(d(y)) = y$ 
     $\forall x[M]. M(c(x)) \text{ lam\_replacement}(M, c)$ 
     $M(A) \ M(B)$ 
  shows  $(\lambda x \in A. c(x)) \in \text{bij}^M(A, B)$ 
  using assms
  apply (unfold bij_rel_def)
  apply (blast intro!: lam_injective_rel lam_surjective_rel)
  done

lemma function_space_rel_eqpoll_rel_cong:
  assumes
     $A \approx^M A' \ B \approx^M B' \ M(A) \ M(A') \ M(B) \ M(B')$ 
  shows
     $A \rightarrow^M B \approx^M A' \rightarrow^M B'$ 
proof -
  from assms(1)[THEN eqpoll_rel_sym] assms(2) assms lam_type
  obtain f g where  $f \in \text{bij}^M(A', A) \ g \in \text{bij}^M(B, B')$ 
  by blast
  with assms
  have  $\text{converse}(g) : \text{bij}^M(B', B) \ \text{converse}(f) : \text{bij}^M(A, A')$ 
  using bij_converse_bij by auto
  let ?H =  $\lambda h \in A \rightarrow^M B. g \ O \ h \ O \ f$ 
  let ?I =  $\lambda h \in A' \rightarrow^M B'. \text{converse}(g) \ O \ h \ O \ \text{converse}(f)$ 
  have  $go : g \ O \ F \ O \ f : A' \rightarrow^M B' \ \text{if } F : A \rightarrow^M B \ \text{for } F$ 
  proof -
  note assms  $\langle f \in \_ \rangle \ \langle g \in \_ \rangle$  that
  moreover from this
  have  $g \ O \ F \ O \ f : A' \rightarrow B'$ 
  using bij_rel_is_fun[OF  $\langle g \in \_ \rangle$ ] bij_rel_is_fun[OF  $\langle f \in \_ \rangle$ ] comp_fun
    mem_function_space_rel[OF  $\langle F \in \_ \rangle$ ]
  by blast
  ultimately
  show  $g \ O \ F \ O \ f : A' \rightarrow^M B'$ 
  using comp_closed_function_space_rel_char bij_rel_char
  by auto
qed
have  $og : \text{converse}(g) \ O \ F \ O \ \text{converse}(f) : A \rightarrow^M B \ \text{if } F : A' \rightarrow^M B' \ \text{for } F$ 
proof -
  note assms that  $\langle \text{converse}(f) \in \_ \rangle \ \langle \text{converse}(g) \in \_ \rangle$ 
  moreover from this
  have  $\text{converse}(g) \ O \ F \ O \ \text{converse}(f) : A \rightarrow B$ 
  using bij_rel_is_fun[OF  $\langle \text{converse}(g) \in \_ \rangle$ ] bij_rel_is_fun[OF  $\langle \text{converse}(f) \in \_ \rangle$ ]
  comp_fun
    mem_function_space_rel[OF  $\langle F \in \_ \rangle$ ]
  by blast

```

```

ultimately
show converse(g) O F O converse(f) : A →M B (is ?G∈_)
  using comp_closed function_space_rel_char bij_rel_char
  by auto
qed
with go
have tc: ?H ∈ (A →M B) → (A' →M B') ?I ∈ (A' →M B') → (A →M B)
  using lam_type by auto
with assms ⟨f∈_⟩ ⟨g∈_⟩
have M(g O x O f) and M(converse(g) O x O converse(f)) if M(x) for x
  using bij_rel_char comp_closed that by auto
with assms ⟨f∈_⟩ ⟨g∈_⟩
have M(?H) M(?I)
  using lam_replacement_iff_lam_closed[THEN iffD1, OF _ lam_replacement_comp]
  bij_rel_char by auto
show ?thesis
  unfolding eqpoll_rel_def
proof (intro rexI[of _ ?H] fg_imp_bijective_rel)
  from og go
  have (λx. x ∈ A' →M B' ⇒ converse(g) O x O converse(f) ∈ A →M B)
    by simp
next
  show M(A →M B) using assms by simp
next
  show M(A' →M B') using assms by simp
next
  from og assms
  have ?H O ?I = (λx∈A' →M B'. (g O converse(g)) O x O (converse(f) O f))
    using lam_cong[OF refl[of A' →M B']] comp_assoc comp_lam
    by auto
  also
  have ... = (λx∈A' →M B'. id(B') O x O (id(A')))
    using left_comp_inverse[OF mem_inj_rel[OF bij_rel_is_inj_rel]] ⟨f∈_⟩
    right_comp_inverse[OF bij_is_surj[OF mem_bij_rel]] ⟨g∈_⟩ assms
    by auto
  also
  have ... = (λx∈A' →M B'. x)
    using left_comp_id[OF fun_is_rel[OF mem_function_space_rel]]
    right_comp_id[OF fun_is_rel[OF mem_function_space_rel]] assms
    by auto
  also
  have ... = id(A' →M B') unfolding id_def by simp
  finally
  show ?H O ?I = id(A' →M B') .
next
  from go assms
  have ?I O ?H = (λx∈A →M B. (converse(g) O g) O x O (f O converse(f)))
    using lam_cong[OF refl[of A →M B]] comp_assoc comp_lam by auto
  also

```

```

have ... = ( $\lambda x \in A \rightarrow^M B . id(B) \circ x \circ id(A)$ )
  using
    left_comp_inverse[OF mem_inj_rel[OF bij_rel_is_inj_rel[OF  $\langle g \in \_ \rangle$ ]]]
    right_comp_inverse[OF bij_is_surj[OF mem_bij_rel[OF  $\langle f \in \_ \rangle$ ]]] assms
  by auto
also
have ... = ( $\lambda x \in A \rightarrow^M B . x$ )
  using left_comp_id[OF fun_is_rel[OF mem_function_space_rel]]
    right_comp_id[OF fun_is_rel[OF mem_function_space_rel]]
    assms
  by auto
also
have ... =  $id(A \rightarrow^M B)$  unfolding id_def by simp
finally
show ?I  $\circ$  ?H =  $id(A \rightarrow^M B)$  .
next
from assms tc  $\langle M(?H) \rangle \langle M(?I) \rangle$ 
show ?H  $\in (A \rightarrow^M B) \rightarrow^M (A' \rightarrow^M B')$   $M(?H)$ 
  ?I  $\in (A' \rightarrow^M B') \rightarrow^M (A \rightarrow^M B)$ 
  using mem_function_space_rel_abs by auto
qed
qed

lemma curry_eqpoll_rel:
  fixes  $\nu 1 \ \nu 2 \ \kappa$ 
  assumes  $M(\nu 1) \ M(\nu 2) \ M(\kappa)$ 
  shows  $\nu 1 \rightarrow^M (\nu 2 \rightarrow^M \kappa) \approx^M \nu 1 \times \nu 2 \rightarrow^M \kappa$ 
  unfolding eqpoll_rel_def
proof (intro rexI, rule lam_bijective_rel,
  rule_tac [1-2] mem_function_space_rel_abs[THEN iffD2],
  rule_tac [4] lam_type, rule_tac [8] lam_type,
  rule_tac [8] mem_function_space_rel_abs[THEN iffD2],
  rule_tac [11] lam_type, simp_all add:assms)
let ?cur =  $\lambda x. \lambda w \in \nu 1 \times \nu 2. x \text{ 'fst}(w) \text{ 'snd}(w)$ 
fix f z
assume  $f : \nu 1 \rightarrow^M (\nu 2 \rightarrow^M \kappa)$ 
moreover
note assms
moreover from calculation
have  $M(\nu 2 \rightarrow^M \kappa)$ 
  using function_space_rel_closed by simp
moreover from calculation
have  $M(f) \ f : \nu 1 \rightarrow (\nu 2 \rightarrow^M \kappa)$ 
  using function_space_rel_char by (auto dest:transM)
moreover from calculation
have  $x \in \nu 1 \implies f'x : \nu 2 \rightarrow \kappa$  for x
  by (auto dest:transM intro!:mem_function_space_rel_abs[THEN iffD1])
moreover from this
show ( $\lambda a \in \nu 1. \lambda b \in \nu 2. ?cur(f) \text{ '}\langle a, b \rangle$ ) = f

```

```

    using Pi_type[OF ⟨f ∈ ν1 → ν2 →M κ⟩, of λ_.ν2 → κ] by simp
  moreover
  assume z ∈ ν1 × ν2
  moreover from calculation
  have f'fst(z): ν2 →M κ by simp
  ultimately
  show f'fst(z) 'snd(z) ∈ κ
    using mem_function_space_rel_abs by (auto dest:transM)
next — one composition is the identity:
  let ?cur=λx. λw∈ν1 × ν2. x ' fst(w) ' snd(w)
  fix f
  assume f : ν1 × ν2 →M κ
  with assms
  show ?cur(λx∈ν1. λxa∈ν2. f ' ⟨x, xa⟩) = f
    using function_space_rel_char mem_function_space_rel_abs
    by (auto dest:transM intro:fun_extension)
  fix x y
  assume x∈ν1 y∈ν2
  with assms ⟨f : ν1 × ν2 →M κ⟩
  show f'⟨x,y⟩ ∈ κ
    using function_space_rel_char mem_function_space_rel_abs
    by (auto dest:transM[of _ ν1 × ν2 →M κ])
next
  let ?cur=λx. λw∈ν1 × ν2. x ' fst(w) ' snd(w)
  note assms
  moreover from this
  show ∀ x[M]. M(?cur(x))
    using lam_replacement_fst lam_replacement_snd
    lam_replacement_apply2[THEN [5] lam_replacement_hcomp2,
      THEN [1] lam_replacement_hcomp2, where h=( ' ), OF
      lam_replacement_constant] lam_replacement_apply2
    by (auto intro: lam_replacement_iff_lam_closed[THEN iffD1, rule_format])
  moreover from calculation
  show x ∈ ν1 →M (ν2 →M κ) ⇒ M(?cur(x)) for x
    by (auto dest:transM)
  moreover from assms
  show lam_replacement(M, ?cur)
    using lam_replacement_Lambda_apply_fst_snd by simp
  ultimately
  show M(λx∈ν1 →M (ν2 →M κ). ?cur(x))
    using lam_replacement_iff_lam_closed
    by (auto dest:transM)
  from assms
  show y ∈ ν1 × ν2 →M κ ⇒ x ∈ ν1 ⇒ M(λxa∈ν2. y ' ⟨x, xa⟩) for x y
    using lam_replacement_apply_const_id
    by (rule_tac lam_replacement_iff_lam_closed[THEN iffD1, rule_format])
    (auto dest:transM)
  from assms
  show y ∈ ν1 × ν2 →M κ ⇒ M(λx∈ν1. λxa∈ν2. y ' ⟨x, xa⟩) for y

```

```

using lam_replacement_apply2[THEN [5] lam_replacement_hcomp2,
  OF lam_replacement_constant lam_replacement_const_id]
  lam_replacement_Lambda_apply_Pair[of  $\nu 2$ ]
by (auto dest:transM
  intro!: lam_replacement_iff_lam_closed[THEN iffD1, rule_format])
qed

lemma Pow_rel_eqpoll_rel_function_space_rel:
  fixes d X
  notes bool_of_o_def [simp]
  defines [simp]:  $d(A) \equiv (\lambda x \in X. \text{bool\_of\_o}(x \in A))$ 
  — the witnessing map for the thesis:
  assumes  $M(X)$ 
  shows  $\text{Pow}^M(X) \approx^M X \rightarrow^M 2$ 
proof -
  from assms
  interpret M_Pi_assumptions M X  $\lambda \_. 2$ 
  using Pi_replacement Pi_separation lam_replacement_identity
    lam_replacement_Sigfun[THEN lam_replacement_imp_strong_replacement]
    Pi_replacement1[of  $\_ 2$ ] transM[of  $\_ X$ ] lam_replacement_constant
  by unfold_locales auto
  have lam_replacement(M,  $\lambda x. \text{bool\_of\_o}(x \in A)$ ) if  $M(A)$  for A
  using that lam_replacement_if lam_replacement_constant
    separation_in_constant by simp
  with assms
  have lam_replacement(M,  $\lambda x. d(x)$ )
  using separation_in_constant[THEN [3] lam_replacement_if, of  $\lambda \_. 1 \lambda \_. 0$ ]
    lam_replacement_identity lam_replacement_constant lam_replacement_Lambda_if_mem
  by simp
  show ?thesis
  unfolding eqpoll_rel_def
  proof (intro rexI, rule lam_bijective_rel)
  — We give explicit mutual inverses
  fix A
  assume  $A \in \text{Pow}^M(X)$ 
  moreover
  note  $\langle M(X) \rangle$ 
  moreover from calculation
  have  $M(A)$  by (auto dest:transM)
  moreover
  note  $\langle \_ \implies \text{lam\_replacement}(M, \lambda x. \text{bool\_of\_o}(x \in A)) \rangle$ 
  ultimately
  show  $d(A) : X \rightarrow^M 2$ 
  using function_space_rel_char lam_replacement_iff_lam_closed[THEN
iffD1]
  by (simp, rule_tac lam_type[of X  $\lambda x. \text{bool\_of\_o}(x \in A)$   $\lambda \_. 2$ , simplified])
  auto
  from  $\langle A \in \text{Pow}^M(X) \rangle \langle M(X) \rangle$ 
  show  $\{y \in X. d(A) 'y = 1\} = A$ 

```

```

    using Pow_rel_char by auto
next
fix f
assume f:  $X \rightarrow^M 2$ 
with assms
have f:  $X \rightarrow 2$   $M(f)$  using function_space_rel_char by simp_all
then
show  $d(\{y \in X . f \text{ ' } y = 1\}) = f$ 
  using apply_type[OF  $\langle f: X \rightarrow 2 \rangle$ ] by (force intro:fun_extension)
from  $\langle M(X) \rangle \langle M(f) \rangle$ 
show  $\{ya \in X . f \text{ ' } ya = 1\} \in Pow^M(X)$ 
  using Pow_rel_char separation_equal_apply by auto
next
from assms  $\langle lam\_replacement(M, \lambda x. d(x)) \rangle$ 
   $\langle \bigwedge A. \_ \implies lam\_replacement(M, \lambda x. bool\_of\_o(x \in A)) \rangle$ 
show  $M(\lambda x \in Pow^M(X). d(x)) \ lam\_replacement(M, \lambda x. d(x))$ 
   $\forall x[M]. M(d(x))$ 
  using lam_replacement_iff_lam_closed[THEN iffD1] by auto
qed (auto simp:  $\langle M(X) \rangle$ )
qed

lemma Pow_rel_bottom:  $M(B) \implies 0 \in Pow^M(B)$ 
  using Pow_rel_char by simp

lemma cantor_surj_rel:
  assumes  $M(f) \ M(A)$ 
  shows  $f \notin surj^M(A, Pow^M(A))$ 
proof
  assume  $f \in surj^M(A, Pow^M(A))$ 
  with assms
  have  $f \in surj(A, Pow^M(A))$  using surj_rel_char by simp
  moreover
  note assms
  moreover from this
  have  $M(\{x \in A . x \in f \text{ ' } x\}) \ \{x \in A . x \notin f \text{ ' } x\} = A - \{x \in A . x \in f \text{ ' } x\}$ 
    using lam_replacement_apply[THEN [4] separation_in, of  $\lambda x. x$ ]
      lam_replacement_identity lam_replacement_constant by auto
  with  $\langle M(A) \rangle$ 
  have  $\{x \in A . x \notin f \text{ ' } x\} \in Pow^M(A)$ 
    by (intro mem_Pow_rel_abs[THEN iffD2]) auto
  ultimately
  obtain d where  $d \in A \ f \text{ ' } d = \{x \in A . x \notin f \text{ ' } x\}$ 
    unfolding surj_def by blast
  show False
proof (cases  $d \in f \text{ ' } d$ )
  case True
  note  $\langle d \in f \text{ ' } d \rangle$ 
  also
  note  $\langle f \text{ ' } d = \{x \in A . x \notin f \text{ ' } x\} \rangle$ 

```

```

    finally
    have  $d \notin f'd$  using  $\langle d \in A \rangle$  by simp
    then
    show False using  $\langle d \in f' d \rangle$  by simp
next
case False
with  $\langle d \in A \rangle$ 
have  $d \in \{x \in A . x \notin f'x\}$  by simp
also from  $\langle f'd = \dots \rangle$ 
have  $\dots = f'd$  by simp
finally
show False using  $\langle d \notin f'd \rangle$  by simp
qed
qed

lemma cantor_inj_rel:  $M(f) \implies M(A) \implies f \notin \text{inj}^M(\text{Pow}^M(A), A)$ 
  using inj_rel_imp_surj_rel[OF Pow_rel_bottom, of f A A]
  cantor_surj_rel[of  $\lambda x \in A. \text{if } x \in \text{range}(f) \text{ then } \text{converse}(f) 'x \text{ else } 0 A$ ]
  lam_replacement_if_separation_in_constant[of range(f)]
  lam_replacement_converse_app[THEN [5] lam_replacement_hcomp2]
  lam_replacement_identity lam_replacement_constant
  lam_replacement_iff_lam_closed by auto

end — M_ZF_library

end

```

20 Lambda-replacements required for cardinal inequalities

```

theory Replacement_Lepoll
  imports
    ZF_Library_Relative
begin

definition
  lepoll_assumptions1 ::  $[i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$  where
    lepoll_assumptions1( $M, A, F, S, fa, K, x, f, r$ )  $\equiv \forall x \in S. \text{strong\_replacement}(M, \lambda y$ 
 $z. y \in F(A, x) \wedge z = \{x, y\})$ )

definition
  lepoll_assumptions2 ::  $[i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$  where
    lepoll_assumptions2( $M, A, F, S, fa, K, x, f, r$ )  $\equiv \text{strong\_replacement}(M, \lambda x z. z =$ 
 $\text{Sigfun}(x, F(A)))$ 

definition
  lepoll_assumptions3 ::  $[i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$  where
    lepoll_assumptions3( $M, A, F, S, fa, K, x, f, r$ )  $\equiv \text{strong\_replacement}(M, \lambda x y. y =$ 
 $F(A, x))$ 

```

definition

$lepoll_assumptions4 :: [i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$ **where**
 $lepoll_assumptions4(M, A, F, S, fa, K, x, f, r) \equiv strong_replacement(M, \lambda x y. y = \langle x, minimum(r, F(A, x)) \rangle)$

definition

$lepoll_assumptions5 :: [i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$ **where**
 $lepoll_assumptions5(M, A, F, S, fa, K, x, f, r) \equiv strong_replacement(M, \lambda x y. y = \langle x, \mu i. x \in F(A, i), f \text{ ‘ } (\mu i. x \in F(A, i)) \text{ ‘ } x \rangle)$

definition

$lepoll_assumptions6 :: [i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$ **where**
 $lepoll_assumptions6(M, A, F, S, fa, K, x, f, r) \equiv strong_replacement(M, \lambda y z. y \in inj^M(F(A, x), S) \wedge z = \{\langle x, y \rangle\})$

definition

$lepoll_assumptions7 :: [i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$ **where**
 $lepoll_assumptions7(M, A, F, S, fa, K, x, f, r) \equiv strong_replacement(M, \lambda x y. y = inj^M(F(A, x), S))$

definition

$lepoll_assumptions8 :: [i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$ **where**
 $lepoll_assumptions8(M, A, F, S, fa, K, x, f, r) \equiv strong_replacement(M, \lambda x z. z = Sigfun(x, \lambda i. inj^M(F(A, i), S)))$

definition

$lepoll_assumptions9 :: [i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$ **where**
 $lepoll_assumptions9(M, A, F, S, fa, K, x, f, r) \equiv strong_replacement(M, \lambda x y. y = \langle x, minimum(r, inj^M(F(A, x), S)) \rangle)$

definition

$lepoll_assumptions10 :: [i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$ **where**
 $lepoll_assumptions10(M, A, F, S, fa, K, x, f, r) \equiv strong_replacement(M, \lambda x z. z = Sigfun(x, \lambda k. if k \in range(f) then F(A, converse(f) \text{ ‘ } k) else 0))$

definition

$lepoll_assumptions11 :: [i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$ **where**
 $lepoll_assumptions11(M, A, F, S, fa, K, x, f, r) \equiv strong_replacement(M, \lambda x y. y = (if x \in range(f) then F(A, converse(f) \text{ ‘ } x) else 0))$

definition

$lepoll_assumptions12 :: [i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$ **where**
 $lepoll_assumptions12(M, A, F, S, fa, K, x, f, r) \equiv strong_replacement(M, \lambda y z. y \in F(A, converse(f) \text{ ‘ } x) \wedge z = \{\langle x, y \rangle\})$

definition

$lepoll_assumptions13 :: [i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$ **where**
 $lepoll_assumptions13(M, A, F, S, fa, K, x, f, r) \equiv strong_replacement$

$(M, \lambda x y. y = \langle x, \text{minimum}(r, \text{if } x \in \text{range}(f) \text{ then } F(A, \text{converse}(f) \text{ ' } x) \text{ else } 0) \rangle))$

definition

$\text{lepoll_assumptions14} :: [i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$ **where**
 $\text{lepoll_assumptions14}(M, A, F, S, fa, K, x, f, r) \equiv \text{strong_replacement}$
 $(M, \lambda x y. y = \langle x, \mu i. x \in (\text{if } i \in \text{range}(f) \text{ then } F(A, \text{converse}(f) \text{ ' } i) \text{ else } 0),$
 $fa \text{ ' } (\mu i. x \in (\text{if } i \in \text{range}(f) \text{ then } F(A, \text{converse}(f) \text{ ' } i) \text{ else } 0)) \text{ ' } x \rangle)$

definition

$\text{lepoll_assumptions15} :: [i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$ **where**
 $\text{lepoll_assumptions15}(M, A, F, S, fa, K, x, f, r) \equiv \text{strong_replacement}$
 $(M, \lambda y z. y \in \text{inj}^M(\text{if } x \in \text{range}(f) \text{ then } F(A, \text{converse}(f) \text{ ' } x) \text{ else } 0, K) \wedge$
 $z = \{\langle x, y \rangle\})$

definition

$\text{lepoll_assumptions16} :: [i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$ **where**
 $\text{lepoll_assumptions16}(M, A, F, S, fa, K, x, f, r) \equiv \text{strong_replacement}(M, \lambda x y. y =$
 $\text{inj}^M(\text{if } x \in \text{range}(f) \text{ then } F(A, \text{converse}(f) \text{ ' } x) \text{ else } 0, K))$

definition

$\text{lepoll_assumptions17} :: [i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$ **where**
 $\text{lepoll_assumptions17}(M, A, F, S, fa, K, x, f, r) \equiv \text{strong_replacement}$
 $(M, \lambda x z. z = \text{Sigfun}(x, \lambda i. \text{inj}^M(\text{if } i \in \text{range}(f) \text{ then } F(A, \text{converse}(f)$
 $\text{ ' } i) \text{ else } 0, K)))$

definition

$\text{lepoll_assumptions18} :: [i \Rightarrow o, i, [i, i] \Rightarrow i, i, i, i, i, i] \Rightarrow o$ **where**
 $\text{lepoll_assumptions18}(M, A, F, S, fa, K, x, f, r) \equiv \text{strong_replacement}$
 $(M, \lambda x y. y = \langle x, \text{minimum}(r, \text{inj}^M(\text{if } x \in \text{range}(f) \text{ then } F(A, \text{converse}(f)$
 $\text{ ' } x) \text{ else } 0, K) \rangle))$

lemmas $\text{lepoll_assumptions_defs[simp]} = \text{lepoll_assumptions1_def}$

$\text{lepoll_assumptions2_def}$ $\text{lepoll_assumptions3_def}$ $\text{lepoll_assumptions4_def}$
 $\text{lepoll_assumptions5_def}$ $\text{lepoll_assumptions6_def}$ $\text{lepoll_assumptions7_def}$
 $\text{lepoll_assumptions8_def}$ $\text{lepoll_assumptions9_def}$ $\text{lepoll_assumptions10_def}$
 $\text{lepoll_assumptions11_def}$ $\text{lepoll_assumptions12_def}$ $\text{lepoll_assumptions13_def}$
 $\text{lepoll_assumptions14_def}$ $\text{lepoll_assumptions15_def}$ $\text{lepoll_assumptions16_def}$
 $\text{lepoll_assumptions17_def}$ $\text{lepoll_assumptions18_def}$

definition if_range_F **where**

$[\text{simp}]: \text{if_range_F}(H, f, i) \equiv \text{if } i \in \text{range}(f) \text{ then } H(\text{converse}(f) \text{ ' } i) \text{ else } 0$

definition if_range_F_else_F **where**

$\text{if_range_F_else_F}(H, b, f, i) \equiv \text{if } b=0 \text{ then } \text{if_range_F}(H, f, i) \text{ else } H(i)$

lemma (in M_basic) $\text{lam_Least_assumption_general}$:

```

assumes
  separations:
 $\forall A'[M]. \text{separation}(M, \lambda y. \exists x \in A'. y = \langle x, \mu i. x \in \text{if\_range\_F\_else\_F}(F(A), b, f, i) \rangle)$ 
and
   $\text{mem\_F\_bound}: \bigwedge x c. x \in F(A, c) \implies c \in \text{range}(f) \cup U(A)$ 
and
   $\text{types}: M(A) \ M(b) \ M(f) \ M(U(A))$ 
shows  $\text{lam\_replacement}(M, \lambda x. \mu i. x \in \text{if\_range\_F\_else\_F}(F(A), b, f, i))$ 
proof -
have  $\forall x \in X. (\mu i. x \in \text{if\_range\_F\_else\_F}(F(A), b, f, i)) \in$ 
   $\text{Pow}^M(\bigcup (X \cup \text{range}(f) \cup U(A)))$  if  $M(X)$  for  $X$ 
proof
  fix  $x$ 
  assume  $x \in X$ 
  moreover
  note  $\langle M(X) \rangle$ 
  moreover from calculation
  have  $M(x)$  by (auto dest:transM)
  moreover
  note assms
  ultimately
  show  $(\mu i. x \in \text{if\_range\_F\_else\_F}(F(A), b, f, i)) \in$ 
     $\text{Pow}^M(\bigcup (X \cup \text{range}(f) \cup U(A)))$ 
  proof (rule_tac Least_in_Pow_rel_Union, cases b=0, simp_all)
    case True
      fix  $c$ 
      assume  $\text{asm}: x \in \text{if\_range\_F\_else\_F}(F(A), 0, f, c)$ 
      with  $\text{mem\_F\_bound}$ 
      show  $c \in X \vee c \in \text{range}(f) \vee c \in U(A)$ 
      unfolding  $\text{if\_range\_F\_else\_F\_def if\_range\_F\_def}$  by (cases c ∈ range(f))
    auto
    next
      case False
      fix  $c$ 
      assume  $x \in \text{if\_range\_F\_else\_F}(F(A), b, f, c)$ 
      with  $\text{False mem\_F\_bound[of } x \ c]$ 
      show  $c \in X \vee c \in \text{range}(f) \vee c \in U(A)$ 
      unfolding  $\text{if\_range\_F\_else\_F\_def if\_range\_F\_def}$  by auto
    qed
  qed
  with assms
  show ?thesis
  using  $\text{bounded\_lam\_replacement[of } \lambda x. (\mu i. x \in \text{if\_range\_F\_else\_F}(F(A), b, f, i))$ 
     $\lambda X. \text{Pow}^M(\bigcup (X \cup \text{range}(f) \cup U(A)))]$  by simp
qed

lemma (in M_basic)  $\text{lam\_Least\_assumption\_ifM\_b0}$ :
  fixes  $F$ 
  defines  $F \equiv \lambda x. \text{if } M(x) \text{ then } x \text{ else } 0$ 

```

```

assumes
  separations:
 $\forall A'[M]. \text{separation}(M, \lambda y. \exists x \in A'. y = \langle x, \mu i. x \in \text{if\_range\_F\_else\_F}(F(A), 0, f, i) \rangle)$ 
and
  types:  $M(A) \ M(f)$ 
shows  $\text{lam\_replacement}(M, \lambda x. \mu i. x \in \text{if\_range\_F\_else\_F}(F(A), 0, f, i))$ 
  (is  $\text{lam\_replacement}(M, \lambda x. \text{Least}(\lambda y. P(y)))$ )
proof -
  {
    fix  $x \ X$ 
    assume  $M(X) \ x \in X \ (\mu i. P(x, i)) \neq 0$ 
    moreover from this
    obtain  $m$  where  $\text{Ord}(m) \ P(x, m)$ 
    using  $\text{Least\_0}[of \ P(\_)]$  by auto
    moreover
    note assms
    moreover
    have  $P(x, i) \longleftrightarrow (M(\text{converse}(f) \restriction i) \wedge i \in \text{range}(f) \wedge x \in \text{converse}(f) \restriction i)$ 
for  $i$ 
    unfolding  $F\_def \ \text{if\_range\_F\_else\_F\_def} \ \text{if\_range\_F\_def}$  by auto
    ultimately
    have  $(\mu i. P(x, i)) \in \text{range}(f)$ 
    unfolding  $F\_def \ \text{if\_range\_F\_else\_F\_def} \ \text{if\_range\_F\_def}$ 
    by (rule_tac LeastI2) auto
  }
  with assms
  show ?thesis
  by (rule_tac bounded_lam_replacement[of _  $\lambda X. \text{range}(f) \cup \{0\}$ ]) auto
qed

lemma (in M_replacement_extra) lam_Least_assumption_ifM_bnot0:
fixes  $F$ 
defines  $F \equiv \lambda x. \text{if } M(x) \text{ then } x \text{ else } 0$ 
assumes
  separations:
 $\forall A'[M]. \text{separation}(M, \lambda y. \exists x \in A'. y = \langle x, \mu i. x \in \text{if\_range\_F\_else\_F}(F(A), b, f, i) \rangle)$ 
   $\text{separation}(M, \text{Ord})$ 
and
  types:  $M(A) \ M(f)$ 
and
   $b \neq 0$ 
shows  $\text{lam\_replacement}(M, \lambda x. \mu i. x \in \text{if\_range\_F\_else\_F}(F(A), b, f, i))$ 
  (is  $\text{lam\_replacement}(M, \lambda x. \text{Least}(\lambda y. P(y)))$ )
proof -
  have  $M(x) \implies (\mu i. (M(i) \longrightarrow x \in i) \wedge M(i)) = (\text{if } \text{Ord}(x) \text{ then } \text{succ}(x) \text{ else } 0)$ 
for  $x$ 
  using Ord_in_Ord
  apply (auto intro:Least_0, rule_tac Least_equality, simp_all)
  by (frule lt_Ord) (auto dest:le_imp_not_lt[of _ x] intro:ltI[of x])

```

```

moreover
have lam_replacement( $M, \lambda x. \text{if } \text{Ord}(x) \text{ then } \text{succ}(x) \text{ else } 0$ )
  using lam_replacement_if[ $OF \_ \_ \text{separations}(2)$ ] lam_replacement_identity
  lam_replacement_constant lam_replacement_hcomp lam_replacement_succ
  by simp
moreover
note types  $\langle b \neq 0 \rangle$ 
ultimately
show ?thesis
  using lam_replacement_cong
  unfolding F_def if_range_F_else_F_def if_range_F_def
  by auto
qed

lemma (in  $M\_replacement\_extra$ ) lam_Least_assumption_drSR_Y:
  fixes  $F \ r' \ D$ 
  defines  $F \equiv \text{drSR\_Y}(r', D)$ 
  assumes  $\forall A' [M]. \text{separation}(M, \lambda y. \exists x \in A'. y = \langle x, \mu i. x \in \text{if\_range\_F\_else\_F}(F(A), b, f, i) \rangle)$ 
     $M(A) \ M(b) \ M(f) \ M(r')$ 
  shows  $\text{lam\_replacement}(M, \lambda x. \mu i. x \in \text{if\_range\_F\_else\_F}(F(A), b, f, i))$ 
proof -
  from  $\text{assms}(2-)$ 
  have  $[\text{simp}]: M(X) \implies M(X \cup \text{range}(f) \cup \{ \text{domain}(x) . x \in A \})$ 
     $M(r') \implies M(X) \implies M(\{ \text{restrict}(x, r') . x \in A \})$ 
  for  $X \ r'$ 
  using lam_replacement_domain[ $THEN \text{lam\_replacement\_imp\_strong\_replacement},$ 
     $THEN \text{RepFun\_closed}, \text{ of } A$ ]
    lam_replacement_restrict'[ $THEN \text{lam\_replacement\_imp\_strong\_replacement},$ 
     $THEN \text{RepFun\_closed}, \text{ of } r' \ A$ ] by (auto dest:transM)
  have  $\forall x \in X. (\mu i. x \in \text{if\_range\_F\_else\_F}(F(A), b, f, i)) \in$ 
     $\text{Pow}^M(\bigcup (X \cup \text{range}(f) \cup \{ \text{domain}(x). x \in A \} \cup \{ \text{restrict}(x, r'). x \in A \} \cup \text{domain}(A)$ 
 $\cup \text{range}(A) \cup \bigcup A))$  if  $M(X)$  for  $X$ 
proof
  fix  $x$ 
  assume  $x \in X$ 
  moreover
  note  $\langle M(X) \rangle$ 
  moreover from calculation
  have  $M(x)$  by (auto dest:transM)
  moreover
  note  $\text{assms}(2-)$ 
  ultimately
  show  $(\mu i. x \in \text{if\_range\_F\_else\_F}(F(A), b, f, i)) \in$ 
     $\text{Pow}^M(\bigcup (X \cup \text{range}(f) \cup \{ \text{domain}(x). x \in A \} \cup \{ \text{restrict}(x, r'). x \in A \} \cup$ 
 $\text{domain}(A) \cup \text{range}(A) \cup \bigcup A))$ 
  unfolding if_range_F_else_F_def if_range_F_def
  proof (rule_tac Least_in_Pow_rel_Union, simp_all, cases  $b=0$ , simp_all)
  case True
  fix  $c$ 

```

```

    assume asm:  $x \in (\text{if } c \in \text{range}(f) \text{ then } F(A, \text{converse}(f) \text{ ‘ } c) \text{ else } 0)$ 
  then
    show  $c \in X \vee c \in \text{range}(f) \vee (\exists x \in A. c = \text{domain}(x)) \vee (\exists x \in A. c = \text{restrict}(x, r'))$ 
 $\vee c \in \text{domain}(A) \vee c \in \text{range}(A) \vee (\exists x \in A. c \in x)$  by auto
  next
    case False
    fix c
    assume  $x \in F(A, c)$ 
    then
      show  $c \in X \vee c \in \text{range}(f) \vee (\exists x \in A. c = \text{domain}(x)) \vee (\exists x \in A. c = \text{restrict}(x, r'))$ 
 $\vee c \in \text{domain}(A) \vee c \in \text{range}(A) \vee (\exists x \in A. c \in x)$ 
      using apply_0
      by (cases M(c), auto simp: F_def drSR_Y_def dC_F_def)
    qed
  qed
  with assms(2-)
  show ?thesis
    using bounded_lam_replacement[of  $\lambda x. (\mu i. x \in \text{if\_range\_F\_else\_F}(F(A), b, f, i))$ 
 $\lambda X. \text{Pow}^M(\bigcup (X \cup \text{range}(f) \cup \{\text{domain}(x). x \in A\} \cup \{\text{restrict}(x, r') . x \in A\} \cup$ 
 $\text{domain}(A) \cup \text{range}(A) \cup \bigcup A))]$  by simp
  qed

```

```

locale M_replacement_lepoll = M_replacement_extra + M_inj +
  fixes F
  assumes
    F_type[simp]:  $M(A) \implies \forall x[M]. M(F(A, x))$ 
  and
    lam_lepoll_assumption_F:  $M(A) \implies \text{lam\_replacement}(M, F(A))$ 
  and
    — Here b is a Boolean.
    lam_Least_assumption:  $M(A) \implies M(b) \implies M(f) \implies$ 
 $\text{lam\_replacement}(M, \lambda x . \mu i. x \in \text{if\_range\_F\_else\_F}(F(A), b, f, i))$ 
  and
    F_args_closed:  $M(A) \implies M(x) \implies x \in F(A, i) \implies M(i)$ 
  and
    lam_replacement_inj_rel:  $\text{lam\_replacement}(M, \lambda p. \text{inj}^M(\text{fst}(p), \text{snd}(p)))$ 
begin

```

```

declare if_range_F_else_F_def[simp]

```

```

lemma lepoll_assumptions1:

```

```

  assumes types[simp]:  $M(A) \ M(S)$ 
  shows lepoll_assumptions1(M, A, F, S, fa, K, x, f, r)
  using strong_replacement_separation[OF lam_replacement_sing_const_id separation_in_constant]
    transM[of _ S]
  by simp

```

```

lemma lepoll_assumptions2:

```

```

  assumes types[simp]:  $M(A) \ M(S)$ 

```

```

shows lepoll_assumptions2( $M, A, F, S, fa, K, x, f, r$ )
using lam_replacement_Sigfun lam_replacement_imp_strong_replacement
      assms lam_lepoll_assumption_F
by simp

lemma lepoll_assumptions3:
  assumes types[simp]: $M(A)$ 
  shows lepoll_assumptions3( $M, A, F, S, fa, K, x, f, r$ )
  using lam_lepoll_assumption_F[THEN lam_replacement_imp_strong_replacement]
  by simp

lemma lepoll_assumptions4:
  assumes types[simp]: $M(A)$   $M(r)$ 
  shows lepoll_assumptions4( $M, A, F, S, fa, K, x, f, r$ )
  using lam_replacement_minimum lam_replacement_constant lam_lepoll_assumption_F
  unfolding lepoll_assumptions_defs
      lam_replacement_def[symmetric]
  by (rule_tac lam_replacement_hcomp2[of _ _ minimum])
      (force intro: lam_replacement_identity)+

lemma lam_Least_closed :
  assumes  $M(A)$   $M(b)$   $M(f)$ 
  shows  $\forall x[M]. M(\mu i. x \in \text{if\_range\_F\_else\_F}(F(A), b, f, i))$ 
proof -
  have  $x \in (\text{if } i \in \text{range}(f) \text{ then } F(A, \text{converse}(f) \text{ ` } i) \text{ else } 0) \implies M(i)$  for  $x \ i$ 
  proof (cases  $i \in \text{range}(f)$ )
    case True
    with  $\langle M(f) \rangle$ 
    show ?thesis by (auto dest:transM)
  next
    case False
    moreover
    assume  $x \in (\text{if } i \in \text{range}(f) \text{ then } F(A, \text{converse}(f) \text{ ` } i) \text{ else } 0)$ 
    ultimately
    show ?thesis
    by auto
  qed
with assms
show ?thesis
  using F_args_closed[of A] unfolding if_range_F_else_F_def if_range_F_def
  by (clarify, rule_tac Least_closed', cases  $b=0$ ) simp_all
qed

lemma lepoll_assumptions5:
  assumes
    types[simp]: $M(A)$   $M(f)$ 
  shows lepoll_assumptions5( $M, A, F, S, fa, K, x, f, r$ )
  using
    lam_replacement_apply2[THEN [5] lam_replacement_hcomp2]

```

```

    lam_replacement_hcomp[OF lam_replacement_apply[of f]]
    lam_replacement_identity
    lam_replacement_product lam_Least_closed[where b=1]
    asms lam_Least_assumption[where b=1, OF  $\langle M(A) \rangle \_ \langle M(f) \rangle$ ]
unfolding lepoll_assumptions_defs
    lam_replacement_def[symmetric]
by simp

lemma lepoll_assumptions6:
  assumes types[simp]:  $M(A) \ M(S) \ M(x)$ 
  shows lepoll_assumptions6( $M, A, F, S, fa, K, x, f, r$ )
  using strong_replacement_separation[OF lam_replacement_sing_const_id separation_in_constant]
    lam_replacement_inj_rel
  by simp

lemma lepoll_assumptions7:
  assumes types[simp]:  $M(A) \ M(S) \ M(x)$ 
  shows lepoll_assumptions7( $M, A, F, S, fa, K, x, f, r$ )
  using lam_replacement_constant lam_lepoll_assumption_F lam_replacement_inj_rel
  unfolding lepoll_assumptions_defs
  by (rule_tac lam_replacement_imp_strong_replacement)
    (rule_tac lam_replacement_hcomp2[of  $\_ \_ inj\_rel(M)$ ], simp_all)

lemma lepoll_assumptions8:
  assumes types[simp]:  $M(A) \ M(S)$ 
  shows lepoll_assumptions8( $M, A, F, S, fa, K, x, f, r$ )
  using lam_replacement_Sigfun lam_replacement_imp_strong_replacement
    lam_replacement_inj_rel lam_replacement_constant
    lam_replacement_hcomp2[of  $\_ \_ inj\_rel(M)$ , OF lam_lepoll_assumption_F[of
A]]
  by simp

lemma lepoll_assumptions9:
  assumes types[simp]:  $M(A) \ M(S) \ M(r)$ 
  shows lepoll_assumptions9( $M, A, F, S, fa, K, x, f, r$ )
  using lam_replacement_minimum lam_replacement_constant lam_lepoll_assumption_F
    lam_replacement_hcomp2[of  $\_ \_ inj\_rel(M)$ ] lam_replacement_inj_rel lepoll_assumptions4
  unfolding lepoll_assumptions_defs lam_replacement_def[symmetric]
  by (rule_tac lam_replacement_hcomp2[of  $\_ \_ minimum$ ])
    (force intro: lam_replacement_identity)+

lemma lepoll_assumptions10:
  assumes types[simp]:  $M(A) \ M(f)$ 
  shows lepoll_assumptions10( $M, A, F, S, fa, K, x, f, r$ )
  using lam_replacement_Sigfun lam_replacement_imp_strong_replacement
    lam_replacement_constant[OF nonempty]
    lam_replacement_if[OF  $\_ \_ separation\_in\_constant$ ]
    lam_replacement_hcomp
    lam_replacement_apply[OF converse_closed[OF  $\langle M(f) \rangle$ ]]

```

```

    lam_lepoll_assumption_F[of A]
  by simp

lemma lepoll_assumptions11:
  assumes types[simp]:M(A) M(f)
  shows lepoll_assumptions11(M, A, F, S, fa, K, x, f, r)
  using lam_replacement_imp_strong_replacement
    lam_replacement_if[OF _ separation_in_constant[of range(f)]]
    lam_replacement_constant
    lam_replacement_hcomp lam_replacement_apply
    lam_lepoll_assumption_F
  by simp

lemma lepoll_assumptions12:
  assumes types[simp]:M(A) M(x) M(f)
  shows lepoll_assumptions12(M,A,F,S,fa,K,x,f,r)
  using strong_replacement_separation[OF lam_replacement_sing_const_id separation_in_constant]
  by simp

lemma lepoll_assumptions13:
  assumes types[simp]:M(A) M(r) M(f)
  shows lepoll_assumptions13(M,A,F,S,fa,K,x,f,r)
  using lam_replacement_constant[OF nonempty] lam_lepoll_assumption_F
    lam_replacement_hcomp lam_replacement_apply
    lam_replacement_hcomp2[OF lam_replacement_constant[OF ‹M(r)›]
      lam_replacement_if[OF _ separation_in_constant[of range(f)]] _
      lam_replacement_minimum] assms
  unfolding lepoll_assumptions_defs
    lam_replacement_def[symmetric]
  by simp

lemma lepoll_assumptions14:
  assumes types[simp]:M(A) M(f) M(fa)
  shows lepoll_assumptions14(M,A,F,S,fa,K,x,f,r)
  using
    lam_replacement_apply2[THEN [5] lam_replacement_hcomp2]
    lam_replacement_hcomp[OF _ lam_replacement_apply[of fa]]
    lam_replacement_identity
    lam_replacement_product lam_Least_closed[where b=0]
    assms lam_Least_assumption[where b=0, OF ‹M(A)› _ ‹M(f)›]
  unfolding lepoll_assumptions_defs
    lam_replacement_def[symmetric]
  by simp

lemma lepoll_assumptions15:
  assumes types[simp]:M(A) M(x) M(f) M(K)
  shows lepoll_assumptions15(M,A,F,S,fa,K,x,f,r)
  using strong_replacement_separation[OF lam_replacement_sing_const_id separation_in_constant]
  by simp

```



```

lemma lepoll_assumptions16:
  assumes types[simp]:M(A) M(f) M(K)
  shows lepoll_assumptions16(M,A,F,S,fa,K,x,f,r)
  using lam_replacement_imp_strong_replacement
    lam_replacement_inj_rel lam_replacement_constant
    lam_replacement_hcomp2[of _ _ inj_rel(M)]
    lam_replacement_constant[OF nonempty]
    lam_replacement_if[OF _ _ separation_in_constant]
    lam_replacement_hcomp
    lam_replacement_apply[OF converse_closed[OF  $\langle M(f) \rangle$ ]]
    lam_lepoll_assumption_F[of A]
  by simp

lemma lepoll_assumptions17:
  assumes types[simp]:M(A) M(f) M(K)
  shows lepoll_assumptions17(M,A,F,S,fa,K,x,f,r)
  using lam_replacement_Sigfun lam_replacement_imp_strong_replacement
    lam_replacement_inj_rel lam_replacement_constant
    lam_replacement_hcomp2[of _ _ inj_rel(M)]
    lam_replacement_constant[OF nonempty]
    lam_replacement_if[OF _ _ separation_in_constant]
    lam_replacement_hcomp
    lam_replacement_apply[OF converse_closed[OF  $\langle M(f) \rangle$ ]]
    lam_lepoll_assumption_F[of A]
  by simp

lemma lepoll_assumptions18:
  assumes types[simp]:M(A) M(K) M(f) M(r)
  shows lepoll_assumptions18(M,A,F,S,fa,K,x,f,r)
  using lam_replacement_constant lam_replacement_inj_rel lam_lepoll_assumption_F
    lam_replacement_minimum lam_replacement_identity lam_replacement_apply2
    separation_in_constant
  unfolding lepoll_assumptions18_def lam_replacement_def[symmetric]
  by (rule_tac lam_replacement_hcomp2[of _ _ minimum], simp_all,
    rule_tac lam_replacement_hcomp2[of _ _ inj_rel(M)], simp_all)
    (rule_tac lam_replacement_if, rule_tac lam_replacement_hcomp[of _ F(A)],
    rule_tac lam_replacement_hcomp2[of _ _ ( $\cdot$ )], simp_all)

lemmas lepoll_assumptions = lepoll_assumptions1 lepoll_assumptions2
  lepoll_assumptions3 lepoll_assumptions4 lepoll_assumptions5
  lepoll_assumptions6 lepoll_assumptions7 lepoll_assumptions8
  lepoll_assumptions9 lepoll_assumptions10 lepoll_assumptions11
  lepoll_assumptions12 lepoll_assumptions13 lepoll_assumptions14
  lepoll_assumptions15 lepoll_assumptions16
  lepoll_assumptions17 lepoll_assumptions18

end — M_replacement_lepoll

```

end

21 Cardinal Arithmetic under Choice

theory *Cardinal_Library_Relative*

imports

Replacement_Lepoll

begin

locale *M_library* = *M_ZF_library* + *M_cardinal_AC* +

assumes

separation_cardinal_rel_lesspoll_rel: $M(\kappa) \implies \text{separation}(M, \lambda x . |x|^M \prec^M \kappa)$

begin

declare *eqpoll_rel_refl* [*simp*]

21.1 Miscellaneous

lemma *cardinal_rel_RepFun_apply_le*:

assumes $S \in A \rightarrow B$ $M(S)$ $M(A)$ $M(B)$

shows $|\{S'a . a \in A\}|^M \leq |A|^M$

proof -

note *assms*

moreover from *this*

have $\{S' a . a \in A\} = S''A$

using *image_eq_UN RepFun_def UN_iff* **by** *force*

moreover from *calculation*

have $M(\lambda x \in A. S' x)$ $M(\{S' a . a \in A\})$

using *lam_closed[of $\lambda x. S'x$]* *apply_type[OF $\langle S \in _ \rangle$]*

transM[OF $_ \langle M(B) \rangle$] *image_closed*

by *auto*

moreover from *assms this*

have $(\lambda x \in A. S'x) \in \text{surj_rel}(M, A, \{S'a . a \in A\})$

using *mem_surj_abs lam_funtype[of $A \lambda x . S'x$]*

unfolding *surj_def* **by** *auto*

ultimately

show *?thesis*

using *surj_rel_char surj_rel_implies_cardinal_rel_le* **by** *simp*

qed

lemma *cardinal_rel_RepFun_le*:

assumes *lrf*:*lam_replacement*(*M*,*f*) **and** *f_closed*: $\forall x[M]. M(f(x))$ **and** $M(X)$

shows $|\{f(x) . x \in X\}|^M \leq |X|^M$

using $\langle M(X) \rangle$ *f_closed cardinal_rel_RepFun_apply_le*[*OF lam_funtype, of X*], *OF*

lrf[*THEN* [2] *lam_replacement_iff_lam_closed* [*THEN iffD1, THEN rspec*]]

lrf[*THEN lam_replacement_imp_strong_replacement*]

by *simp* (*auto simp flip:setclass_iff intro!:RepFun_closed dest:transM*)

```

lemma subset_imp_le_cardinal_rel:  $A \subseteq B \implies M(A) \implies M(B) \implies |A|^M \leq |B|^M$ 
using subset_imp_lepoll_rel[THEN lepoll_rel_imp_cardinal_rel_le] .

lemma lt_cardinal_rel_imp_not_subset:  $|A|^M < |B|^M \implies M(A) \implies M(B) \implies \neg B \subseteq A$ 
using subset_imp_le_cardinal_rel le_imp_not_lt by blast

lemma cardinal_rel_lt_csucc_rel_iff:
Card_rel(M,K)  $\implies M(K) \implies M(K') \implies |K'|^M < (K^+)^M \iff |K'|^M \leq K$ 
by (simp add: Card_rel_lt_csucc_rel_iff)

end — M_library

locale M_cardinal_UN_nat = M_cardinal_UN _  $\omega$  X for X
begin
lemma cardinal_rel_UN_le_nat:
assumes  $\bigwedge i. i \in \omega \implies |X(i)|^M \leq \omega$ 
shows  $|\bigcup_{i \in \omega}. X(i)|^M \leq \omega$ 
proof -
from assms
show ?thesis
by (simp add: cardinal_rel_UN_le InfCard_rel_nat)
qed

end — M_cardinal_UN_nat

locale M_cardinal_UN_inj = M_library +
j:M_cardinal_UN _ J +
y:M_cardinal_UN _ K  $\lambda k. \text{if } k \in \text{range}(f) \text{ then } X(\text{converse}(f) 'k) \text{ else } 0$  for J K
f +
assumes
f_inj:  $f \in \text{inj\_rel}(M, J, K)$ 
begin

lemma inj_rel_imp_cardinal_rel_UN_le:
notes [dest] = InfCard_is_Card Card_is_Ord
fixes Y
defines  $Y(k) \equiv \text{if } k \in \text{range}(f) \text{ then } X(\text{converse}(f) 'k) \text{ else } 0$ 
assumes  $\text{InfCard}^M(K) \wedge i. i \in J \implies |X(i)|^M \leq K$ 
shows  $|\bigcup_{i \in J}. X(i)|^M \leq K$ 
proof -
have  $M(K) \ M(J) \wedge w x. w \in X(x) \implies M(x)$ 
using y.Pi_assumptions j.Pi_assumptions j.X_witness_in_M by simp_all
then
have M(f)
using inj_rel_char f_inj by simp
note inM =  $\langle M(f) \rangle \ \langle M(K) \rangle \ \langle M(J) \rangle \ \langle \wedge w x. w \in X(x) \implies M(x) \rangle$ 

```

```

have  $i \in J \implies f^i \in K$  for  $i$ 
  using inj_rel_is_fun[OF f_inj] apply_type
  function_space_rel_char by (auto simp add:inM)
have  $(\bigcup_{i \in J}. X(i)) \subseteq (\bigcup_{i \in K}. Y(i))$ 
proof (standard, elim UN_E)
  fix  $x$   $i$ 
  assume  $i \in J$   $x \in X(i)$ 
  with  $\langle i \in J \implies f^i \in K \rangle$ 
  have  $x \in Y(f^i)$   $f^i \in K$ 
  unfolding Y_def
  using inj_is_fun right_inverse f_inj
  by (auto simp add:inM Y_def intro: apply_rangeI)
then
  show  $x \in (\bigcup_{i \in K}. Y(i))$  by auto
qed
then
have  $|\bigcup_{i \in J}. X(i)|^M \leq |\bigcup_{i \in K}. Y(i)|^M$ 
  using subset_imp_le_cardinal_rel j.UN_closed y.UN_closed
  unfolding Y_def by (simp add:inM)
moreover
note assms  $\langle \bigwedge i. i \in J \implies f^i \in K \rangle$  inM
moreover from this
have  $k \in \text{range}(f) \implies \text{converse}(f)^k \in J$  for  $k$ 
  using inj_rel_converse_fun[OF f_inj]
  range_fun_subset_codomain function_space_rel_char by simp
ultimately
show  $|\bigcup_{i \in J}. X(i)|^M \leq K$ 
  using InfCard_rel_is_Card_rel[THEN Card_rel_is_Ord, THEN Ord_0_le,
of K]
  by (rule_tac le_trans[OF _ y.cardinal_rel_UN_le])
  (auto intro: Ord_0_le simp: Y_def)+
qed

end — M_cardinal_UN_inj

locale M_cardinal_UN_lepoll = M_library + M_replacement_lepoll _  $\lambda$ _. X +
j:M_cardinal_UN _ J for J
begin

— FIXME: this "LEQpoll" should be "LEPOLL"; same correction in Delta System
lemma leqpoll_rel_imp_cardinal_rel_UN_le:
  notes [dest] = InfCard_is_Card Card_is_Ord
  assumes  $\text{InfCard}^M(K) \ J \lesssim^M K \ \bigwedge i. i \in J \implies |X(i)|^M \leq K$ 
  M(K)
  shows  $|\bigcup_{i \in J}. X(i)|^M \leq K$ 
proof -
  from  $\langle J \lesssim^M K \rangle$ 
  obtain  $f$  where  $f \in \text{inj\_rel}(M, J, K)$   $M(f)$  by blast
  moreover

```

```

let ?Y= $\lambda k$ . if  $k \in \text{range}(f)$  then  $X(\text{converse}(f) \cdot k)$  else 0
note  $\langle M(K) \rangle$ 
moreover from calculation
have  $k \in \text{range}(f) \implies \text{converse}(f) \cdot k \in J$  for  $k$ 
  using mem_inj_rel[THEN inj_converse_fun, THEN apply_type]
  j.Pi_assumptions by blast
moreover from  $\langle M(f) \rangle$ 
have  $w \in ?Y(x) \implies M(x)$  for  $w \ x$ 
  by (cases  $x \in \text{range}(f)$ ) (auto dest:transM)
moreover from calculation
interpret M_Pi_assumptions_choice  $\_ K ?Y$ 
  using j.Pi_assumptions lepoll_assumptions
proof (unfold locales, auto dest:transM)
  show strong_replacement( $M, \lambda y \ z. \text{False}$ )
    unfolding strong_replacement_def by auto
qed
from calculation
interpret M_cardinal_UN_inj  $\_ \_ \_ f$ 
  using lepoll_assumptions
  by unfold_locales auto
from assms
show ?thesis using inj_rel_imp_cardinal_rel_UN_le by simp
qed

end  $\_ M\_cardinal\_UN\_lepoll$ 

context M_library
begin

lemma cardinal_rel_lt_csucc_rel_iff':
  includes Ord_dests
  assumes Card_rel( $M, \kappa$ )
    and types: $M(\kappa) \ M(X)$ 
  shows  $\kappa < |X|^M \longleftrightarrow (\kappa^+)^M \leq |X|^M$ 
  using assms cardinal_rel_lt_csucc_rel_iff[of  $\kappa \ X$ ] Card_rel_csucc_rel[of  $\kappa$ ]
    not_le_iff_lt[of  $(\kappa^+)^M \ |X|^M$ ] not_le_iff_lt[of  $|X|^M \ \kappa$ ]
  by blast

lemma lepoll_rel_imp_subset_bij_rel:
  assumes  $M(X) \ M(Y)$ 
  shows  $X \lesssim^M Y \longleftrightarrow (\exists Z[M]. \ Z \subseteq Y \wedge Z \approx^M X)$ 
proof
  assume  $X \lesssim^M Y$ 
  then
  obtain  $j$  where  $j \in \text{inj\_rel}(M, X, Y)$ 
    by blast
  with assms
  have  $\text{range}(j) \subseteq Y \ j \in \text{bij\_rel}(M, X, \text{range}(j)) \ M(\text{range}(j)) \ M(j)$ 
    using inj_rel_bij_rel_range inj_rel_char

```

```

    inj_rel_is_fun[THEN range_fun_subset_codomain, of j X Y]
  by auto
with assms
have range(j)  $\subseteq$  Y X  $\approx^M$  range(j)
  unfolding eqpoll_rel_def by auto
with assms  $\langle M(j) \rangle$ 
show  $\exists Z[M]. Z \subseteq Y \wedge Z \approx^M X$ 
  using eqpoll_rel_sym[OF  $\langle X \approx^M \text{range}(j) \rangle$ ]
  by auto
next
assume  $\exists Z[M]. Z \subseteq Y \wedge Z \approx^M X$ 
then
obtain Z f where  $f \in \text{bij\_rel}(M, Z, X)$   $Z \subseteq Y$   $M(Z) \ M(f)$ 
  unfolding eqpoll_rel_def by blast
with assms
have converse(f)  $\in \text{inj\_rel}(M, X, Y)$   $M(\text{converse}(f))$ 
  using inj_rel_weaken_type[OF bij_rel_converse_bij_rel[THEN bij_rel_is_inj_rel], of
f Z X Y]
  by auto
then
show  $X \lesssim^M Y$ 
  unfolding lepoll_rel_def by auto
qed

```

The following result proves to be very useful when combining *cardinal_rel* and *eqpoll_rel* in a calculation.

lemma *cardinal_rel_Card_rel_eqpoll_rel_iff*:
 $\text{Card_rel}(M, \kappa) \implies M(\kappa) \implies M(X) \implies |X|^M = \kappa \iff X \approx^M \kappa$
 using *Card_rel_cardinal_rel_eq*[of κ] *cardinal_rel_eqpoll_rel_iff*[of $X \ \kappa$] **by**
auto

lemma *lepoll_rel_imp_lepoll_rel_cardinal_rel*:
 assumes $X \lesssim^M Y$ $M(X) \ M(Y)$
 shows $X \lesssim^M |Y|^M$
 using *assms* *cardinal_rel_Card_rel_eqpoll_rel_iff*[of $|Y|^M \ Y$]
Card_rel_cardinal_rel
lepoll_rel_eq_trans[of $_ _ |Y|^M$] **by** *simp*

lemma *lepoll_rel_Un*:
 assumes *InfCard_rel*(M, κ) $A \lesssim^M \kappa$ $B \lesssim^M \kappa$ $M(A) \ M(B) \ M(\kappa)$
 shows $A \cup B \lesssim^M \kappa$
proof -
 from *assms*
 have $A \cup B \lesssim^M \text{sum}(A, B)$
 using *Un_lepoll_rel_sum* **by** *simp*
 moreover
 note *assms*
 moreover from *this*
 have $|\text{sum}(A, B)|^M \leq \kappa \oplus^M \kappa$

```

    using sum_lepoll_rel_mono[of A  $\kappa$  B  $\kappa$ ] lepoll_rel_imp_cardinal_rel_le
    unfolding cadd_rel_def by auto
  ultimately
  show ?thesis
    using InfCard_rel_cdoube_eq Card_rel_cardinal_rel_eq
    InfCard_rel_is_Card_rel Card_rel_le_imp_lepoll_rel[of sum(A,B)  $\kappa$ ]
    lepoll_rel_trans[of A  $\cup$  B]
    by auto
qed

lemma cardinal_rel_Un_le:
  assumes InfCard_rel(M, $\kappa$ )  $|A|^M \leq \kappa$   $|B|^M \leq \kappa$  M( $\kappa$ ) M(A) M(B)
  shows  $|A \cup B|^M \leq \kappa$ 
  using assms lepoll_rel_Un_le_Card_rel_iff InfCard_rel_is_Card_rel by auto

lemma Finite_cardinal_rel_iff': M(i)  $\implies$  Finite( $|i|^M$ )  $\longleftrightarrow$  Finite(i)
  using eqpoll_rel_imp_Finite_iff[OF cardinal_rel_eqpoll_rel]
  by auto

lemma cardinal_rel_subset_of_Card_rel:
  assumes Card_rel(M, $\gamma$ )  $a \subseteq \gamma$  M(a) M( $\gamma$ )
  shows  $|a|^M < \gamma \vee |a|^M = \gamma$ 
proof -
  from assms
  have  $|a|^M < |\gamma|^M \vee |a|^M = |\gamma|^M$ 
    using subset_imp_le_cardinal_rel[THEN le_iff[THEN iffD1]] by simp
  with assms
  show ?thesis
    using Card_rel_cardinal_rel_eq by auto
qed

lemma cardinal_rel_cases:
  includes Ord_dests
  assumes M( $\gamma$ ) M(X)
  shows Card_rel(M, $\gamma$ )  $\implies$   $|X|^M < \gamma \longleftrightarrow \neg |X|^M \geq \gamma$ 
  using assms not_le_iff_lt Card_rel_is_Ord Ord_cardinal_rel
  by auto

end — M_library

```

21.2 Countable and uncountable sets

definition

countable :: $i \Rightarrow o$ where
 countable(X) $\equiv X \lesssim \omega$

relativize functional countable countable_rel external
 relationalize countable_rel is_countable

notation *countable_rel* ($\langle \text{countable_}'(_) \rangle$)

abbreviation

countable_r_set :: $[i, i] \Rightarrow o$ ($\langle \text{countable_}'(_) \rangle$) **where**
countable^M(i) \equiv *countable_rel*($\#\#M, i$)

context *M_library*

begin

lemma *countableI[intro]*: $X \lesssim^M \omega \implies \text{countable_rel}(M, X)$
unfolding *countable_rel_def* **by** *simp*

lemma *countableD[dest]*: $\text{countable_rel}(M, X) \implies X \lesssim^M \omega$
unfolding *countable_rel_def* **by** *simp*

lemma *countable_rel_iff_cardinal_rel_le_nat*: $M(X) \implies \text{countable_rel}(M, X)$
 $\iff |X|^M \leq \omega$
using *le_Card_rel_iff[of ω X]* *Card_rel_nat*
unfolding *countable_rel_def* **by** *simp*

lemma *lepoll_rel_countable_rel*: $X \lesssim^M Y \implies \text{countable_rel}(M, Y) \implies M(X)$
 $\implies M(Y) \implies \text{countable_rel}(M, X)$
using *lepoll_rel_trans[of X Y]* **by** *blast*

— Next lemma can be proved without using AC

lemma *surj_rel_countable_rel*:

$\text{countable_rel}(M, X) \implies f \in \text{surj_rel}(M, X, Y) \implies M(X) \implies M(Y) \implies M(f)$
 $\implies \text{countable_rel}(M, Y)$
using *surj_rel_implies_cardinal_rel_le[of f X Y, THEN le_trans]*
countable_rel_iff_cardinal_rel_le_nat **by** *simp*

lemma *Finite_imp_countable_rel*: $\text{Finite_rel}(M, X) \implies M(X) \implies \text{countable_rel}(M, X)$
unfolding *Finite_rel_def*
by (*auto intro: InfCard_rel_nat nats_le_InfCard_rel[of ω ,*
THEN le_imp_lepoll_rel] dest!: eq_lepoll_rel_trans[of X ω])

end — *M_library*

lemma (**in** *M_cardinal_UN_lepoll*) *countable_rel_imp_countable_rel_UN*:
assumes $\text{countable_rel}(M, J) \bigwedge i. i \in J \implies \text{countable_rel}(M, X(i))$
shows $\text{countable_rel}(M, \bigcup_{i \in J} X(i))$
using *assms lepoll_rel_imp_cardinal_rel_UN_le[of ω]* *InfCard_rel_nat*
InfCard_rel_is_Card_rel j.UN_closed
countable_rel_iff_cardinal_rel_le_nat j.Pi_assumptions
Card_rel_le_imp_lepoll_rel[of J ω] *Card_rel_cardinal_rel_eq[of ω]*
by *auto*

locale *M_cardinal_library* = *M_library* + *M_replacement* +
assumes


```

    lam_replacement_inj_rel: lam_replacement(M,  $\lambda x. inj^M(fst(x), snd(x))$ )
  and
    cdl_t_assms:  $M(G) \implies M(Q) \implies separation(M, \lambda p. \forall x \in G. x \in snd(p) \longleftrightarrow$ 
    ( $\forall s \in fst(p). \langle s, x \rangle \in Q$ )
  and
    cardinal_lib_assms1:
     $M(A) \implies M(b) \implies M(f) \implies$ 
     $separation(M, \lambda y. \exists x \in A. y = \langle x, \mu i. x \in if\_range\_F\_else\_F(\lambda x. if\ M(x)$ 
     $then\ x\ else\ 0, b, f, i) \rangle)$ 
     $separation(M, Ord)$ 
  and
    cardinal_lib_assms2:
     $M(A') \implies M(G) \implies M(b) \implies M(f) \implies$ 
     $separation(M, \lambda y. \exists x \in A'. y = \langle x, \mu i. x \in if\_range\_F\_else\_F(\lambda a. if\ M(a)$ 
     $then\ G'a\ else\ 0, b, f, i) \rangle)$ 
  and
    cardinal_lib_assms3:
     $M(A') \implies M(b) \implies M(f) \implies M(F) \implies$ 
     $separation(M, \lambda y. \exists x \in A'. y = \langle x, \mu i. x \in if\_range\_F\_else\_F(\lambda a. if\ M(a)$ 
     $then\ F'\{a\}\ else\ 0, b, f, i) \rangle)$ 
  and
    lam_replacement_cardinal_rel : lam_replacement(M, cardinal_rel(M))
  and
    cardinal_lib_assms6:
     $M(f) \implies M(\beta) \implies Ord(\beta) \implies$ 
     $strong\_replacement(M, \lambda x\ y. x \in \beta \wedge y = \langle x, transrec(x, \lambda a\ g. f\ ' (g\ ' a)) \rangle)$ 

```

begin

lemma cardinal_lib_assms5 :

$M(\gamma) \implies Ord(\gamma) \implies separation(M, \lambda Z. cardinal_rel(M, Z) < \gamma)$

unfolding lt_def

using separation_in lam_replacement_constant[of γ] separation_univ lam_replacement_cardinal_rel

unfolding lt_def

by simp_all

lemma separation_dist: $separation(M, \lambda x. \exists a. \exists b. x = \langle a, b \rangle \wedge a \neq b)$

using separation_pair separation_neg separation_eq lam_replacement_fst lam_replacement_snd

by simp

lemma cdl_t_assms': $M(x) \implies M(Q) \implies separation(M, \lambda a. \forall s \in x. \langle s, a \rangle \in Q)$

using separation_in[OF _

lam_replacement_hcomp2[OF _ _ _ lam_replacement_Pair] _

lam_replacement_constant]

separation_ball lam_replacement_hcomp lam_replacement_fst lam_replacement_snd

by simp_all

lemma countable_rel_union_countable_rel:

assumes $\bigwedge x. x \in C \implies countable_rel(M, x)$ countable_rel(M, C) $M(C)$

```

shows countable_rel( $M, \bigcup C$ )
proof -
  have  $x \in (\text{if } M(i) \text{ then } i \text{ else } 0) \implies M(i)$  for  $x \ i$ 
    by (cases  $M(i)$ ) auto
  then
  interpret  $M\_replacement\_lepoll \ M \ \lambda\_x. \text{if } M(x) \text{ then } x \text{ else } 0$ 
    using lam_replacement_if[OF lam_replacement_identity
      lam_replacement_constant[OF nonempty], where  $b=M$ ] lam_replacement_inj_rel
  proof(unfold_locales, auto simp add: separation_def)
    fix  $b \ f$ 
    assume  $M(b) \ M(f)$ 
    show lam_replacement( $M, \lambda x. \mu i. x \in \text{if\_range\_F\_else\_F}(\lambda x. \text{if } M(x) \text{ then } x \text{ else } 0, b, f, i)$ )
    proof (cases  $b=0$ )
      case True
      with  $\langle M(f) \rangle$ 
      show ?thesis
        using cardinal_lib_assms1
        by (simp_all; rule_tac lam_Least_assumption_ifM_b0)+
    next
      case False
      with  $\langle M(f) \rangle \ \langle M(b) \rangle$ 
      show ?thesis
        using cardinal_lib_assms1
        by (rule_tac lam_Least_assumption_ifM_bnot0) auto
    qed
  qed
  note  $\langle M(C) \rangle$ 
  moreover
  have  $w \in (\text{if } M(x) \text{ then } x \text{ else } 0) \implies M(x)$  for  $w \ x$ 
    by (cases  $M(x)$ ) auto
  ultimately
  interpret  $M\_cardinal\_UN\_lepoll \ \lambda c. \text{if } M(c) \text{ then } c \text{ else } 0 \ C$ 
    using lepoll_assumptions
    by unfold_locales simp_all
  have  $(\text{if } M(i) \text{ then } i \text{ else } 0) = i$  if  $i \in C$  for  $i$ 
    using transM[OF  $\langle M(C) \rangle$ ] that by simp
  then
  show ?thesis
    using assms countable_rel_imp_countable_rel_UN by simp
  qed

end —  $M\_cardinal\_library$ 

abbreviation
  uncountable_rel ::  $[i \Rightarrow o, i] \Rightarrow o$  where
  uncountable_rel( $M, X$ )  $\equiv \neg \text{countable\_rel}(M, X)$ 

context  $M\_cardinal\_library$ 

```

begin

lemma *uncountable_rel_iff_nat_lt_cardinal_rel*:

$M(X) \implies \text{uncountable_rel}(M, X) \longleftrightarrow \omega < |X|^M$

using *countable_rel_iff_cardinal_rel_le_nat not_le_iff_lt* **by** *simp*

lemma *uncountable_rel_not_empty*: $\text{uncountable_rel}(M, X) \implies X \neq 0$

using *empty_lepoll_relI* **by** *auto*

lemma *uncountable_rel_imp_Infinite*: $\text{uncountable_rel}(M, X) \implies M(X) \implies \text{Infinite}(X)$

using *uncountable_rel_iff_nat_lt_cardinal_rel[of X] lepoll_rel_nat_imp_Infinite[of X]*

cardinal_rel_le_imp_lepoll_rel[of ω X] leI

by *simp*

lemma *uncountable_rel_not_subset_countable_rel*:

assumes *countable_rel(M, X) uncountable_rel(M, Y) M(X) M(Y)*

shows $\neg (Y \subseteq X)$

using *assms lepoll_rel_trans subset_imp_lepoll_rel[of Y X]*

by *blast*

21.3 Results on Aleph_rels

lemma *nat_lt_Aleph_rel1*: $\omega < \aleph_1^M$

by (*simp add: Aleph_rel_succ Aleph_rel_zero lt_csucc_rel*)

lemma *zero_lt_Aleph_rel1*: $0 < \aleph_1^M$

by (*rule lt_trans[of ω], auto simp add: ltI nat_lt_Aleph_rel1*)

lemma *le_Aleph_rel1_nat*: $M(k) \implies \text{Card_rel}(M, k) \implies k < \aleph_1^M \implies k \leq \omega$

by (*simp add: Aleph_rel_succ Aleph_rel_zero Card_rel_lt_csucc_rel_iff Card_rel_nat*)

lemma *lesspoll_rel_Aleph_rel_succ*:

assumes *Ord(α)*

and *types: M(α) M(d)*

shows $d \prec^M \aleph_{\text{succ}(\alpha)}^M \longleftrightarrow d \lesssim^M \aleph_\alpha^M$

using *assms Aleph_rel_succ Card_rel_is_Ord Ord_Aleph_rel lesspoll_rel_csucc_rel*

by *simp*

lemma *cardinal_rel_Aleph_rel [simp]*: $\text{Ord}(\alpha) \implies M(\alpha) \implies |\aleph_\alpha^M|^M = \aleph_\alpha^M$

using *Card_rel_cardinal_rel_eq* **by** *simp*

— Could be proved without using AC

lemma *Aleph_rel_lesspoll_rel_increasing*:

includes *Aleph_rel_intros*

assumes *M(b) M(a)*

shows $a < b \implies \aleph_a^M \prec^M \aleph_b^M$

using *assms*

cardinal_rel_lt_iff_lesspoll_rel[of $\aleph_a^M \aleph_b^M$]

```

    Aleph_rel_increasing[of a b] Card_rel_cardinal_rel_eq[of  $\aleph_b$ ]
    lt_Ord lt_Ord2 Card_rel_Aleph_rel[THEN Card_rel_is_Ord]
  by auto

lemma uncountable_rel_iff_subset_eqpoll_rel_Aleph_rel1:
  includes Ord_dests
  assumes  $M(X)$ 
  notes Aleph_rel_zero[simp] Card_rel_nat[simp] Aleph_rel_succ[simp]
  shows  $\text{uncountable\_rel}(M, X) \longleftrightarrow (\exists S[M]. S \subseteq X \wedge S \approx^M \aleph_1^M)$ 
proof
  assume  $\text{uncountable\_rel}(M, X)$ 
  with  $\langle M(X) \rangle$ 
  have  $\aleph_1^M \lesssim^M X$ 
    using uncountable_rel_iff_nat_lt_cardinal_rel cardinal_rel_lt_csucc_rel_iff'
      cardinal_rel_le_imp_lepoll_rel by auto
  with  $\langle M(X) \rangle$ 
  obtain  $S$  where  $M(S) S \subseteq X S \approx^M \aleph_1^M$ 
    using lepoll_rel_imp_subset_bij_rel by auto
  then
  show  $\exists S[M]. S \subseteq X \wedge S \approx^M \aleph_1^M$ 
    using cardinal_rel_cong Card_rel_csucc_rel[of  $\omega$ ] Card_rel_cardinal_rel_eq
  by auto
next
  note Aleph_rel_lespoll_rel_increasing[of 1 0, simplified]
  assume  $\exists S[M]. S \subseteq X \wedge S \approx^M \aleph_1^M$ 
  moreover
  have  $\text{eq}:\aleph_1^M = (\omega^+)^M$  by auto
  moreover from calculation  $\langle M(X) \rangle$ 
  have  $A:(\omega^+)^M \lesssim^M X$ 
    using subset_imp_lepoll_rel[THEN [2] eq_lepoll_rel_trans, of  $\aleph_1^M \_ X$ ,
      OF eqpoll_rel_sym] by auto
  with  $\langle M(X) \rangle$ 
  show  $\text{uncountable\_rel}(M, X)$ 
    using
      lespoll_rel_trans1[OF lepoll_rel_trans[OF A  $\_$ ]  $\langle \omega \prec^M (\omega^+)^M \rangle$ 
      lespoll_rel_not_refl
    by auto
qed

lemma UN_if_zero:  $M(K) \implies (\bigcup_{x \in K}. \text{if } M(x) \text{ then } G \text{ ' } x \text{ else } 0) = (\bigcup_{x \in K}. G \text{ ' } x)$ 
  using transM[of  $\_ K$ ] by auto

lemma mem_F_bound1:
  fixes  $F G$ 
  defines  $F \equiv \lambda \_ x. \text{if } M(x) \text{ then } G \text{ ' } x \text{ else } 0$ 
  shows  $x \in F(A, c) \implies c \in (\text{range}(f) \cup \text{domain}(G))$ 
  using apply_0 unfolding F_def
  by (cases  $M(c)$ , auto simp:F_def drSR_Y_def dC_F_def)

```

```

lemma lt_Aleph_rel_imp_cardinal_rel_UN_le_nat: function(G)  $\implies$  domain(G)
 $\lesssim^M \omega \implies$ 
 $\forall n \in \text{domain}(G). |G'n|^M <_{\aleph_I} M \implies M(G) \implies |\bigcup n \in \text{domain}(G). G'n|^M \leq \omega$ 
proof -
  assume M(G)
  moreover from this
  have  $x \in (\text{if } M(i) \text{ then } G' i \text{ else } 0) \implies M(i)$  for  $x i$ 
    by (cases M(i)) auto
  moreover
  have separation(M, M) unfolding separation_def by auto
  ultimately
  interpret M_replacement_lepoll M  $\lambda x. \text{if } M(x) \text{ then } G'x \text{ else } 0$ 
  using lam_replacement_inj_rel cardinal_lib_assms2 mem_F_bound1[of _ _
G]
    lam_if_then_replacement_apply
  by (unfold_locales, simp_all)
    (rule lam_Least_assumption_general[where  $U = \lambda \_. \text{domain}(G)$ ], auto)
  note  $\langle M(G) \rangle$ 
  moreover
  have  $w \in (\text{if } M(x) \text{ then } G' x \text{ else } 0) \implies M(x)$  for  $w x$ 
    by (cases M(x)) auto
  ultimately
  interpret M_cardinal_UN_lepoll  $\lambda n. \text{if } M(n) \text{ then } G'n \text{ else } 0$  domain(G)
  using lepoll_assumptions1[where  $S = \text{domain}(G)$ , unfolded lepoll_assumptions1_def]
    cardinal_lib_assms2 lepoll_assumptions
  by (unfold_locales, auto)
  assume function(G)
  let  $?N = \text{domain}(G)$  and  $?R = \bigcup n \in \text{domain}(G). G'n$ 
  assume  $?N \lesssim^M \omega$ 
  assume Eq1:  $\forall n \in ?N. |G'n|^M <_{\aleph_I} M$ 
  {
    fix n
    assume  $n \in ?N$ 
    with Eq1  $\langle M(G) \rangle$ 
    have  $|G'n|^M \leq \omega$ 
      using le_Aleph_rel1_nat[of  $|G' n|^M$ ] leqpoll_rel_imp_cardinal_rel_UN_le
        UN_if_zero[of domain(G) G]
      by (auto dest:transM)
  }
  then
  have  $n \in ?N \implies |G'n|^M \leq \omega$  for  $n$  .
  moreover
  note  $\langle ?N \lesssim^M \omega \rangle \langle M(G) \rangle$ 
  moreover
  have  $(\text{if } M(i) \text{ then } G' i \text{ else } 0) \subseteq G' i$  for  $i$ 
    by auto
  with  $\langle M(G) \rangle$ 
  have  $|\text{if } M(i) \text{ then } G' i \text{ else } 0|^M \leq |G' i|^M$  for  $i$ 

```

```

proof(cases M(i))
  case True
    with  $\langle M(G) \rangle$  show ?thesis using Ord_cardinal_rel[OF apply_closed]
    by simp
  next
    case False
    then
      have  $i \notin \text{domain}(G)$ 
      using transM[OF _ domain_closed[OF  $\langle M(G) \rangle$ ]] by auto
    then
      show ?thesis
      using Ord_cardinal_rel[OF apply_closed] apply_0 by simp
  qed
ultimately
show ?thesis
  using InfCard_rel_nat lepoll_rel_imp_cardinal_rel_UN_le[of  $\omega$ ]
  UN_if_zero[of domain(G) G]
  le_trans[of  $\text{if } M(\_) \text{ then } G \text{ ' } \_ \text{ else } 0|^M \mid G \text{ ' } \_|^M \omega$ ]
  by auto blast
qed

lemma Aleph_rel1_eq_cardinal_rel_vimage:  $f: \aleph_1^M \rightarrow^M \omega \implies \exists n \in \omega. |f^{-1}\{n\}|^M = \aleph_1^M$ 
proof -
  assume  $f: \aleph_1^M \rightarrow^M \omega$ 
  then
    have function(f) domain(f) =  $\aleph_1^M$  range(f)  $\subseteq \omega$   $f \in \aleph_1^M \rightarrow^M \omega$  M(f)
    using mem_function_space_rel[OF  $\langle f \in \_ \rangle$ ] domain_of_fun fun_is_function
    range_fun_subset_codomain
    function_space_rel_char
    by auto
  let ?G =  $\lambda n \in \text{range}(f). f^{-1}\{n\}$ 
  from  $\langle f: \aleph_1^M \rightarrow^M \omega \rangle$ 
  have range(f)  $\subseteq \omega$  domain(?G) = range(f)
    using range_fun_subset_codomain
    by simp_all
  from  $\langle f: \aleph_1^M \rightarrow^M \omega \rangle$   $\langle M(f) \rangle$   $\langle \text{range}(f) \subseteq \omega \rangle$ 
  have  $M(f^{-1}\{n\})$  if  $n \in \text{range}(f)$  for  $n$ 
    using that transM[of _  $\omega$ ] by auto
  with  $\langle M(f) \rangle$   $\langle \text{range}(f) \subseteq \omega \rangle$ 
  have domain(?G)  $\lesssim^M \omega$  M(?G)
    using subset_imp_lepoll_rel lam_closed[of  $\lambda x. f^{-1}\{x\}$ ] cardinal_lib_assms4
    by simp_all
  have function(?G) by (simp add: function_lam)
  from  $\langle f: \aleph_1^M \rightarrow^M \omega \rangle$ 
  have  $n \in \omega \implies f^{-1}\{n\} \subseteq \aleph_1^M$  for  $n$ 
    using Pi_vimage_subset by simp
  with  $\langle \text{range}(f) \subseteq \omega \rangle$ 
  have  $\aleph_1^M = (\bigcup n \in \text{range}(f). f^{-1}\{n\})$ 

```

```

proof (intro equalityI, intro subsetI)
  fix x
  assume  $x \in \aleph_I^M$ 
  with  $\langle f: \aleph_I^M \rightarrow \omega \rangle$   $\langle \text{function}(f) \rangle$   $\langle \text{domain}(f) = \aleph_I^M \rangle$ 
  have  $x \in f^{-1}\{f(x)\}$  by  $\text{range\_mem}$ 
  using function_apply_Pair vimage_iff apply_rangeI by simp_all
  then
  show  $x \in \bigcup_{n \in \text{range}(f)} f^{-1}\{n\}$  by auto
qed auto
{
  assume  $\forall n \in \text{range}(f). |f^{-1}\{n\}|^M < \aleph_I^M$ 
  then
  have  $\forall n \in \text{domain}(?G). |?G^{-1}\{n\}|^M < \aleph_I^M$ 
  using zero_lt_Aleph_rel1 by (auto)
  with  $\langle \text{function}(?G) \rangle$   $\langle \text{domain}(?G) \lesssim^M \omega \rangle$   $\langle M(?G) \rangle$ 
  have  $|\bigcup_{n \in \text{domain}(?G)} ?G^{-1}\{n\}|^M \leq \omega$ 
  using lt_Aleph_rel_imp_cardinal_rel_UN_le_nat[of ?G]
  by auto
  then
  have  $|\bigcup_{n \in \text{range}(f)} f^{-1}\{n\}|^M \leq \omega$  by simp
  with  $\langle \aleph_I^M = \_ \rangle$ 
  have  $|\aleph_I^M|^M \leq \omega$  by auto
  then
  have  $\aleph_I^M \leq \omega$ 
  using Card_rel_Aleph_rel Card_rel_cardinal_rel_eq
  by auto
  then
  have False
  using nat_lt_Aleph_rel1 by (blast dest:lt_trans2)
}
with  $\langle \text{range}(f) \subseteq \omega \rangle$   $\langle M(f) \rangle$ 
obtain  $n$  where  $n \in \omega \wedge (|f^{-1}\{n\}|^M < \aleph_I^M) \wedge M(f^{-1}\{n\})$ 
using nat_into_M by auto
moreover from this
have  $\aleph_I^M \leq |f^{-1}\{n\}|^M$ 
using not_lt_iff_le Card_rel_is_Ord by simp
moreover
note  $\langle n \in \omega \implies f^{-1}\{n\} \subseteq \aleph_I^M \rangle$ 
ultimately
show ?thesis
using subset_imp_le_cardinal_rel[THEN le_anti_sym, of _ \aleph_I^M]
Card_rel_Aleph_rel Card_rel_cardinal_rel_eq
by auto
qed

```

— There is some asymmetry between assumptions and conclusion (*eqpoll_rel* versus *cardinal_rel*)

lemma *eqpoll_rel_Aleph_rel1_cardinal_rel_vimage*:

```

assumes  $Z \approx^M (\aleph_I^M) f \in Z \rightarrow^M \omega \ M(Z)$ 
shows  $\exists n \in \omega. |f \cdot \{\{n\}\}|^M = \aleph_I^M$ 
proof -
  have  $M(1) \ M(\omega)$  by simp_all
  then
  have  $M(\aleph_I^M)$  by simp
  with assms  $\langle M(1) \rangle$ 
  obtain  $g$  where  $A: g \in \text{bij\_rel}(M, \aleph_I^M, Z) \ M(g)$ 
    using eqpoll_rel_sym unfolding eqpoll_rel_def by blast
  with  $\langle f : Z \rightarrow^M \omega \rangle$  assms
  have  $M(f) \ \text{converse}(g) \in \text{bij\_rel}(M, Z, \aleph_I^M) \ f \in Z \rightarrow \omega \ g \in \aleph_I^M \rightarrow Z$ 
  using bij_rel_is_fun_rel bij_rel_converse_bij_rel bij_rel_char function_space_rel_char
    by simp_all
  with  $\langle g \in \text{bij\_rel}(M, \aleph_I^M, Z) \rangle \ \langle M(g) \rangle$ 
  have  $f \circ g : \aleph_I^M \rightarrow^M \omega \ M(\text{converse}(g))$ 
    using comp_fun[OF  $\langle f \in Z \rightarrow \omega \rangle$ , of  $g$ ] function_space_rel_char
    by simp_all
  then
  obtain  $n$  where  $n \in \omega \ |(f \circ g) \cdot \{\{n\}\}|^M = \aleph_I^M$ 
    using Aleph_rel1_eq_cardinal_rel_vimage
    by auto
  with  $\langle M(f) \rangle \ \langle M(\text{converse}(g)) \rangle$ 
  have  $M(\text{converse}(g) \cdot \{(f \cdot \{\{n\}\})\}) \ f \cdot \{\{n\}\} \subseteq Z$ 
  using image_comp converse_comp Pi_iff[THEN iffD1, OF  $\langle f \in Z \rightarrow \omega \rangle$ ] vimage_subset
    unfolding vimage_def
    using transM[OF  $\langle n \in \omega \rangle$ ]
    by auto
  from  $\langle n \in \omega \rangle \ \langle |(f \circ g) \cdot \{\{n\}\}|^M = \aleph_I^M \rangle$ 
  have  $\aleph_I^M = |\text{converse}(g) \cdot \{(f \cdot \{\{n\}\})\}|^M$ 
    using image_comp converse_comp unfolding vimage_def
    by auto
  also from  $\langle \text{converse}(g) \in \text{bij\_rel}(M, Z, \aleph_I^M) \rangle \ \langle f : Z \rightarrow^M \omega \rangle \ \langle M(Z) \rangle \ \langle M(f) \rangle$ 
   $\langle M(\aleph_I^M) \rangle$ 
   $\langle M(\text{converse}(g) \cdot \{(f \cdot \{\{n\}\})\}) \rangle$ 
  have  $\dots = |f \cdot \{\{n\}\}|^M$ 
    using range_of_subset_eqpoll_rel[of  $\text{converse}(g) \ Z \ \_ \ f \cdot \{\{n\}\}$ ,
      OF  $\text{bij\_rel\_is\_inj\_rel}[OF \ \langle \text{converse}(g) \in \_ \rangle] \ \langle f \cdot \{\{n\}\} \subseteq Z \rangle$ ]
      cardinal_rel_cong vimage_closed[OF singleton_closed[OF  $\text{transM}[OF \ \langle n \in \omega \rangle]$ ], of
f]
    by auto
  finally
  show ?thesis using  $\langle n \in \_ \rangle$  by auto
qed

```

21.4 Applications of transfinite recursive constructions

definition

$\text{rec_constr} :: [i, i] \Rightarrow i$ **where**
 $\text{rec_constr}(f, \alpha) \equiv \text{transrec}(\alpha, \lambda a \ g. f(g \cdot a))$

The function *rec_constr* allows to perform *recursive constructions*: given a choice function on the powerset of some set, a transfinite sequence is created by successively choosing some new element.

The next result explains its use.

lemma *rec_constr_unfold*: $\text{rec_constr}(f, \alpha) = f(\{\text{rec_constr}(f, \beta) \mid \beta \in \alpha\})$
using *def_transrec*[*OF* *rec_constr_def*, *of* *f* α] *image_lam* **by** *simp*

lemma *rec_constr_type*:

assumes $f: \text{Pow_rel}(M, G) \rightarrow^M G$ *Ord*(α) $M(G)$

shows $M(\alpha) \implies \text{rec_constr}(f, \alpha) \in G$

using *assms*(2)

proof(*induct rule:trans_induct*)

case (*step* β)

with *assms*

have $\{\text{rec_constr}(f, x) \mid x \in \beta\} = \{y \mid x \in \beta, y = \text{rec_constr}(f, x)\}$ (**is** $_ = ?Y$)
 $M(f)$

using *transM*[*OF* $\langle M(\beta) \rangle$] *function_space_rel_char* *Ord_in_Ord*

by *auto*

moreover from *assms this step* $\langle M(\beta) \rangle \langle \text{Ord}(\beta) \rangle$

have $M(\{y \mid x \in \beta, y = \langle x, \text{rec_constr}(f, x) \rangle\})$ (**is** $M(?Z)$)

using *strong_replacement_closed*[*OF* *cardinal_lib_assms6*(1), *of* *f* β , *OF* $_$

univalent_conjI2[**where** $P = \lambda x _ . x \in \beta, \text{OF } \text{univalent_triv}$]]

transM[*OF* $\langle M(\beta) \rangle$] *transM*[*OF* *step*(2) $\langle M(G) \rangle$] *Ord_in_Ord*

unfolding *rec_constr_def*

by *auto*

moreover from *assms this step* $\langle M(\beta) \rangle \langle \text{Ord}(\beta) \rangle$

have $?Y = \{\text{snd}(y) \mid y \in ?Z\}$

proof(*intro equalityI*, *auto*)

fix *u*

assume $u \in \beta$

with *assms this step* $\langle M(\beta) \rangle \langle \text{Ord}(\beta) \rangle$

have $\langle u, \text{rec_constr}(f, u) \rangle \in ?Z$ $\text{rec_constr}(f, u) = \text{snd}(\langle u, \text{rec_constr}(f, u) \rangle)$

by *auto*

then

show $\exists x \in \{y \mid x \in \beta, y = \langle x, \text{rec_constr}(f, x) \rangle\}. \text{rec_constr}(f, u) = \text{snd}(x)$

using *beXI*[*of* $_ u$] **by** *force*

qed

moreover from $\langle M(?Z) \rangle \langle ?Y = _ \rangle$

have $M(?Y)$

using

RepFun_closed[*OF* *lam_replacement_imp_strong_replacement*[*OF* *lam_replacement_snd*]

$\langle M(?Z) \rangle$]

fst_snd_closed[*THEN* *conjunct2*] *transM*[*OF* $\langle M(?Z) \rangle$]

by *simp*

moreover from *assms step*

have $\{\text{rec_constr}(f, x) \mid x \in \beta\} \in \text{Pow}(G)$ (**is** $?X \in _$)

using *transM*[*OF* $\langle M(\beta) \rangle$] *function_space_rel_char*

```

    by auto
  moreover from assms calculation step
  have  $M(?X)$ 
    by simp
  moreover from calculation  $\langle M(G) \rangle$ 
  have  $?X \in \text{Pow\_rel}(M, G)$ 
    using Pow_rel_char by simp
  ultimately
  have  $f' ?X \in G$ 
    using assms apply_type [OF mem_function_space_rel [of f], of Pow_rel (M, G)
G ?X]
    by auto
  then
  show ?case
    by (subst rec_constr_unfold, simp)
qed

```

```

lemma rec_constr_closed :
  assumes  $f : \text{Pow\_rel}(M, G) \rightarrow^M G \text{ Ord}(\alpha) \ M(G) \ M(\alpha)$ 
  shows  $M(\text{rec\_constr}(f, \alpha))$ 
  using transM [OF rec_constr_type  $\langle M(G) \rangle$ ] assms by auto

```

```

lemma lambda_rec_constr_closed :
  assumes  $\text{Ord}(\gamma) \ M(\gamma) \ M(f) \ f : \text{Pow\_rel}(M, G) \rightarrow^M G \ M(G)$ 
  shows  $M(\lambda \alpha \in \gamma . \text{rec\_constr}(f, \alpha))$ 
  using lam_closed2 [OF cardinal_lib_assms6 (1), unfolded rec_constr_def [symmetric], of
f  $\gamma$ ]
    rec_constr_type [OF  $\langle f \in \_ \rangle \ \text{Ord\_in\_Ord}$  [of  $\gamma$ ]] transM [OF  $\_ \ \langle M(G) \rangle$ ] assms
  by simp

```

The next lemma is an application of recursive constructions. It works under the assumption that whenever the already constructed subsequence is small enough, another element can be added.

```

lemma bounded_cardinal_rel_selection:
  includes Ord_dests
  assumes
     $\bigwedge Z. |Z|^M < \gamma \implies Z \subseteq G \implies M(Z) \implies \exists a \in G. \forall s \in Z. \langle s, a \rangle \in Q \ b \in G$ 
Card_rel (M, γ)
     $M(G) \ M(Q) \ M(\gamma)$ 
  shows
     $\exists S[M]. S : \gamma \rightarrow^M G \wedge (\forall \alpha \in \gamma. \forall \beta \in \gamma. \alpha < \beta \longrightarrow \langle S'\alpha, S'\beta \rangle \in Q)$ 
  proof -
    from assms
    have  $M(x) \implies M(\{a \in G . \forall s \in x. \langle s, a \rangle \in Q\})$  for  $x$ 
      using cdlt_assms' by simp
    let  $?cdlt\gamma = \{Z \in \text{Pow\_rel}(M, G) . |Z|^M < \gamma\}$  — “cardinal_rel less than  $\gamma$ ”
      and  $?inQ = \lambda Y. \{a \in G. \forall s \in Y. \langle s, a \rangle \in Q\}$ 
    from  $\langle M(G) \rangle \ \langle \text{Card\_rel}(M, \gamma) \rangle \ \langle M(\gamma) \rangle$ 
    have  $M(?cdlt\gamma) \ \text{Ord}(\gamma)$ 

```

```

    using cardinal_lib_assms5[OF  $\langle M(\gamma) \rangle$ ] Card_rel_is_Ord
    by simp_all
  from assms
  have  $H: \exists a. a \in ?inQ(Y)$  if  $Y \in ?cdlt\gamma$  for  $Y$ 
  proof -
    {
      fix Y
      assume  $Y \in ?cdlt\gamma$ 
      then
      have  $A: Y \in Pow\_rel(M, G) \mid Y \mid^M < \gamma$  by simp_all
      then
      have  $Y \subseteq G \ M(Y)$  using Pow_rel_char[OF  $\langle M(G) \rangle$ ] by simp_all
      with A
      obtain a where  $a \in G \ \forall s \in Y. \langle s, a \rangle \in Q$ 
        using assms(1) by force
      with  $\langle M(G) \rangle$ 
      have  $\exists a. a \in ?inQ(Y)$  by auto
    }
    then show ?thesis using that by simp
  qed
  then
  have  $\exists f[M]. f \in Pi\_rel(M, ?cdlt\gamma, ?inQ) \wedge f \in Pi(?cdlt\gamma, ?inQ)$ 
  proof -
    from  $\langle \bigwedge x. M(x) \implies M(\{a \in G . \forall s \in x. \langle s, a \rangle \in Q\}) \rangle \langle M(G) \rangle$ 
    have  $x \in \{Z \in Pow^M(G) . |Z|^M < \gamma\} \implies M(\{a \in G . \forall s \in x. \langle s, a \rangle \in Q\})$ 
  for x
    by (auto dest: transM)
    with  $\langle M(G) \rangle \langle \bigwedge x. M(x) \implies M(\{a \in G . \forall s \in x. \langle s, a \rangle \in Q\}) \rangle \langle M(Q) \rangle$ 
   $\langle M(?cdlt\gamma) \rangle$ 
  interpret M_Pi_assumptions_choice M ?cdlt\gamma ?inQ
  using cdl_t_assms[where Q=Q] lam_replacement_Collect_ball_Pair[THEN
    lam_replacement_imp_strong_replacement] surj_imp_inj_replacement3
    lam_replacement_hcomp2[OF lam_replacement_constant
      lam_replacement_Collect_ball_Pair _ lam_replacement_minimum,
      unfolded lam_replacement_def]
    lam_replacement_hcomp lam_replacement_Sigfun[OF
      lam_replacement_Collect_ball_Pair, of G Q, THEN
      lam_replacement_imp_strong_replacement] cdl_t_assms'
  by unfold_locales (blast dest: transM, auto dest: transM)
  show ?thesis using AC_Pi_rel Pi_rel_char H by auto
  qed
  then
  obtain f where  $f\_type: f \in Pi\_rel(M, ?cdlt\gamma, ?inQ) \wedge f \in Pi(?cdlt\gamma, ?inQ)$  and
   $M(f)$ 
    by auto
  moreover
  define Cb where  $Cb \equiv \lambda\_ \in Pow\_rel(M, G) - ?cdlt\gamma. b$ 
  moreover from  $\langle b \in G \rangle \langle M(?cdlt\gamma) \rangle \langle M(G) \rangle$ 
  have  $Cb \in Pow\_rel(M, G) - ?cdlt\gamma \rightarrow G \ M(Cb)$ 

```

```

using lam_closed[of  $\lambda_.b$  Pow_rel( $M, G$ )-?cdlt $\gamma$ ]
tag_replacement transM[OF  $\langle b \in G \rangle$ ]
unfolding Cb_def by auto
moreover
note  $\langle \text{Card\_rel}(M, \gamma) \rangle$ 
ultimately
have  $f \cup Cb : (\prod x \in \text{Pow\_rel}(M, G). \text{?in}Q(x) \cup G)$  using
  fun_Pi_disjoint_Un[of  $f$  ?cdlt $\gamma$  ?in $Q$  Cb Pow_rel( $M, G$ )-?cdlt $\gamma$   $\lambda_.G$ ]
  Diff_partition[of  $\{Z \in \text{Pow\_rel}(M, G). |Z|^M < \gamma\}$  Pow_rel( $M, G$ ), OF Collect_subset]
by auto
moreover
have  $\text{?in}Q(x) \cup G = G$  for  $x$  by auto
moreover from calculation
have  $f \cup Cb : \text{Pow\_rel}(M, G) \rightarrow G$ 
using function_space_rel_char by simp
ultimately
have  $f \cup Cb : \text{Pow\_rel}(M, G) \xrightarrow{M} G$ 
using function_space_rel_char  $\langle M(f) \rangle \langle M(Cb) \rangle \text{Pow\_rel\_closed} \langle M(G) \rangle$ 
by auto
define  $S$  where  $S \equiv \lambda \alpha \in \gamma. \text{rec\_constr}(f \cup Cb, \alpha)$ 
from  $\langle f \cup Cb : \text{Pow\_rel}(M, G) \xrightarrow{M} G \rangle \langle \text{Card\_rel}(M, \gamma) \rangle \langle M(\gamma) \rangle \langle M(G) \rangle$ 
have  $S : \gamma \rightarrow G$   $M(f \cup Cb)$ 
unfolding S_def
using Ord_in_Ord[OF Card_rel_is_Ord] rec_constr_type lam_type transM[OF
 $\langle M(\gamma) \rangle$ ]
function_space_rel_char
by auto
moreover from  $\langle f \cup Cb \in \_ \xrightarrow{M} G \rangle \langle \text{Card\_rel}(M, \gamma) \rangle \langle M(\gamma) \rangle \langle M(G) \rangle \langle M(f \cup$ 
 $Cb) \rangle \langle \text{Ord}(\gamma) \rangle$ 
have  $M(S)$ 
unfolding S_def
using lambda_rec_constr_closed
by simp
moreover
have  $\forall \alpha \in \gamma. \forall \beta \in \gamma. \alpha < \beta \longrightarrow \langle S \text{ ` } \alpha, S \text{ ` } \beta \rangle \in Q$ 
proof (intro ballI impI)
  fix  $\alpha \beta$ 
  assume  $\beta \in \gamma$ 
  with  $\langle \text{Card\_rel}(M, \gamma) \rangle \langle M(S) \rangle \langle M(\gamma) \rangle$ 
  have  $\beta \subseteq \gamma \ M(S \text{ ` } \beta) \ M(\beta) \ \{S \text{ ` } x . x \in \beta\} = \{\text{restrict}(S, \beta) \text{ ` } x . x \in \beta\}$ 
using transM[OF  $\langle \beta \in \gamma \rangle \langle M(\gamma) \rangle$ ] image_closed Card_rel_is_Ord OrdmemD
by auto
  with  $\langle \beta \in \_ \rangle \langle \text{Card\_rel}(M, \gamma) \rangle \langle M(\gamma) \rangle$ 
  have  $\{\text{rec\_constr}(f \cup Cb, x) . x \in \beta\} = \{S \text{ ` } x . x \in \beta\}$ 
using Ord_trans[OF  $\_ \_ \text{Card\_rel\_is\_Ord}$ , of  $\_ \beta \gamma$ ]
unfolding S_def
by auto
  moreover from  $\langle \beta \in \gamma \rangle \langle S : \gamma \rightarrow G \rangle \langle \text{Card\_rel}(M, \gamma) \rangle \langle M(\gamma) \rangle \langle M(S \text{ ` } \beta) \rangle$ 
  have  $\{S \text{ ` } x . x \in \beta\} \subseteq G \ M(\{S \text{ ` } x . x \in \beta\})$ 

```

```

using Ord_trans[OF __ Card_rel_is_Ord, of __  $\beta$   $\gamma$ ]
      apply_type[of  $S$   $\gamma$   $\lambda\_.$   $G$ ]
by(auto,simp add:image_fun_subset[OF  $\langle S \in \_ \rangle \langle \beta \subseteq \_ \rangle$ ])
moreover from  $\langle \text{Card\_rel}(M, \gamma) \rangle \langle \beta \in \gamma \rangle \langle S \in \_ \rangle \langle \beta \subseteq \gamma \rangle \langle M(S) \rangle \langle M(\beta) \rangle \langle M(G) \rangle$ 
 $\langle M(\gamma) \rangle$ 
have  $|\{S'x . x \in \beta\}|^M < \gamma$ 
using
   $\langle \{S'x . x \in \beta\} = \{\text{restrict}(S, \beta)'x . x \in \beta\} \rangle [\text{symmetric}]$ 
  cardinal_rel_RepFun_apply_le[of  $\text{restrict}(S, \beta)$   $\beta$   $G$ ,
    OF restrict_type2[of  $S$   $\gamma$   $\lambda\_.$   $G$   $\beta$ ] restrict_closed]
  Ord_in_Ord Ord_cardinal_rel
  lt_trans1[of  $|\{S'x . x \in \beta\}|^M |\beta|^M \gamma$ ]
  Card_rel_lt_iff[THEN iffD2, of  $\beta$   $\gamma$ , OF __ __ __ ltI]
  Card_rel_is_Ord
by auto
moreover
have  $\forall x \in \beta. \langle S'x, f' \{S'x . x \in \beta\} \rangle \in Q$ 
proof -
  from calculation and f_type
have  $f' \{S'x . x \in \beta\} \in \{a \in G. \forall x \in \beta. \langle S'x, a \rangle \in Q\}$ 
    using apply_type[of  $f$  ?cdlt $\gamma$  ?inQ  $\{S'x . x \in \beta\}$ ]
    Pow_rel_char[OF  $\langle M(G) \rangle$ ]
    by simp
  then
    show ?thesis by simp
qed
moreover
assume  $\alpha \in \gamma$   $\alpha < \beta$ 
moreover from this
have  $\alpha \in \beta$  using ltD by simp
moreover
note  $\langle \beta \in \gamma \rangle \langle Cb \in \text{Pow\_rel}(M, G) \text{--} ?cdlt\gamma \rightarrow G \rangle$ 
ultimately
show  $\langle S' \alpha, S' \beta \rangle \in Q$ 
  using fun_disjoint_apply1[of  $\{S'x . x \in \beta\}$   $Cb$   $f$ ]
  domain_of_fun[of  $Cb$ ] ltD[of  $\alpha$   $\beta$ ]
  by (subst (2) S_def, auto) (subst rec_constr_unfold, auto)
qed
moreover
note  $\langle M(G) \rangle \langle M(\gamma) \rangle$ 
ultimately
show ?thesis using function_space_rel_char by auto
qed

```

The following basic result can, in turn, be proved by a bounded-cardinal_rel selection.

lemma Infinite_iff_lepoll_rel_nat: $M(Z) \implies \text{Infinite}(Z) \longleftrightarrow \omega \lesssim^M Z$

proof

define Distinct **where** Distinct = $\{\langle x, y \rangle \in Z \times Z . x \neq y\}$

```

have Distinct = { $xy \in Z \times Z . \exists a \ b. xy = \langle a, b \rangle \wedge a \neq b$ } (is  $\_=?A$ )
  unfolding Distinct_def by auto
moreover
assume Infinite(Z) M(Z)
moreover from calculation
have M(Distinct)
  using cardinal_lib_assms6_separation_dist by simp
from  $\langle \text{Infinite}(Z) \rangle \langle M(Z) \rangle$ 
obtain b where  $b \in Z$ 
  using Infinite_not_empty by auto
{
  fix Y
  assume  $|Y|^M < \omega$  M(Y)
  then
  have Finite(Y)
    using Finite_cardinal_rel_iff' ltD nat_into_Finite by auto
  with  $\langle \text{Infinite}(Z) \rangle$ 
  have  $Z \neq Y$  by auto
}
moreover
have  $(\bigwedge W. M(W) \implies |W|^M < \omega \implies W \subseteq Z \implies \exists a \in Z. \forall s \in W. \langle s, a \rangle \in \text{Distinct})$ 
proof -
  fix W
  assume M(W)  $|W|^M < \omega$   $W \subseteq Z$ 
  moreover from this
  have Finite_rel(M, W)
    using
      cardinal_rel_closed[OF  $\langle M(W) \rangle$  Card_rel_nat
      lt_Card_rel_imp_lespoll_rel[of  $\omega$ ,simplified,OF  $\_ \langle |W|^M < \omega \rangle$ ]
      lespoll_rel_nat_is_Finite_rel[of W]
      eqpoll_rel_imp_lepoll_rel eqpoll_rel_sym[OF cardinal_rel_eqpoll_rel,of W]
      lespoll_rel_trans1[of W  $|W|^M \omega$ ] by auto
  moreover from calculation
  have  $\neg Z \subseteq W$ 
    using equalityI  $\langle \text{Infinite}(Z) \rangle$  by auto
  moreover from calculation
  show  $\exists a \in Z. \forall s \in W. \langle s, a \rangle \in \text{Distinct}$ 
    unfolding Distinct_def by auto
qed
moreover from  $\langle b \in Z \rangle \langle M(Z) \rangle \langle M(\text{Distinct}) \rangle$  this
obtain S where  $S : \omega \rightarrow^M Z$   $M(S) \forall \alpha \in \omega. \forall \beta \in \omega. \alpha < \beta \longrightarrow \langle S'\alpha, S'\beta \rangle \in$ 
Distinct
  using bounded_cardinal_rel_selection[OF  $\_ \langle b \in Z \rangle$  Card_rel_nat,of Distinct]
  by blast
moreover from this
have  $\alpha \in \omega \implies \beta \in \omega \implies \alpha \neq \beta \implies S'\alpha \neq S'\beta$  for  $\alpha \beta$ 
  unfolding Distinct_def
  by (rule_tac lt_neq_symmetry[of  $\omega \lambda \alpha \beta. S'\alpha \neq S'\beta$ ])
    auto

```

```

moreover from this  $\langle S \in \_ \rangle \langle M(Z) \rangle$ 
have  $S \in \text{inj}(\omega, Z)$  using function_space_rel_char unfolding inj_def by auto
ultimately
show  $\omega \lesssim^M Z$ 
  unfolding lepoll_rel_def using inj_rel_char  $\langle M(Z) \rangle$  by auto
next
  assume  $\omega \lesssim^M Z \ M(Z)$ 
  then
    show Infinite( $Z$ ) using lepoll_rel_nat_imp_Infinite by simp
qed

lemma Infinite_InfCard_rel_cardinal_rel:  $\text{Infinite}(Z) \implies M(Z) \implies \text{InfCard\_rel}(M, |Z|^M)$ 
  using lepoll_rel_eq_trans eqpoll_rel_sym lepoll_rel_nat_imp_Infinite
  Infinite_iff_lepoll_rel_nat_Inf_Card_rel_is_InfCard_rel cardinal_rel_eqpoll_rel
  by simp

lemma (in M_trans) mem_F_bound2:
  fixes  $F \ A$ 
  defines  $F \equiv \lambda \_ x. \text{if } M(x) \text{ then } A \cdot \{\!x\! \} \text{ else } 0$ 
  shows  $x \in F(A, c) \implies c \in (\text{range}(f) \cup \text{range}(A))$ 
  using apply_0 unfolding F_def
  by (cases  $M(c)$ , auto simp: F_def drSR_Y_def dC_F_def)

lemma Finite_to_one_rel_surj_rel_imp_cardinal_rel_eq:
  assumes  $F \in \text{Finite\_to\_one\_rel}(M, Z, Y) \cap \text{surj\_rel}(M, Z, Y)$  Infinite( $Z$ )  $M(Z)$ 
   $M(Y)$ 
  shows  $|Y|^M = |Z|^M$ 
proof -
  have sep_true: separation( $M, M$ ) unfolding separation_def by auto
  note  $\langle M(Z) \rangle \langle M(Y) \rangle$ 
  moreover from this assms
  have  $M(F) \ F \in Z \rightarrow Y$ 
    unfolding Finite_to_one_rel_def
    using function_space_rel_char by simp_all
  moreover from this
  have  $x \in (\text{if } M(i) \text{ then } F \cdot \{\!i\! \} \text{ else } 0) \implies M(i)$  for  $x \ i$ 
    by (cases  $M(i)$ ) auto
  moreover from calculation
  interpret M_replacement_lepoll  $M \ \lambda \_ x. \text{if } M(x) \text{ then } F \cdot \{\!x\! \} \text{ else } 0$ 
    using lam_replacement_inj_rel mem_F_bound2 cardinal_lib_assms3
    lam_replacement_vimage_sing_fun
    lam_replacement_if[OF _
      lam_replacement_constant[OF nonempty], where b=M] sep_true
    by (unfold locales, simp_all)
    (rule lam_Least_assumption_general[where U= $\lambda \_. \text{range}(F)$ ], auto)
  have  $w \in (\text{if } M(y) \text{ then } F \cdot \{\!y\! \} \text{ else } 0) \implies M(y)$  for  $w \ y$ 
    by (cases  $M(y)$ ) auto
  moreover from  $\langle F \in \_ \cap \_ \rangle$ 
  have  $0 : \text{Finite}(F \cdot \{\!y\! \})$  if  $y \in Y$  for  $y$ 

```

```

    unfolding Finite_to_one_rel_def
    using vimage_fun_sing ⟨F ∈ Z → Y⟩ transM[OF that ⟨M(Y)⟩] transM[OF _
⟨M(Z)⟩] that by simp
    ultimately
    interpret M_cardinal_UN_lepoll _ λy. if M(y) then F-“{y} else 0 Y
    using cardinal_lib_assms3 lepoll_assumptions
    by unfold_locales (auto dest:transM simp del:mem_inj_abs)
    from ⟨F ∈ Z → Y⟩
    have Z = (⋃ y ∈ Y. {x ∈ Z . F'x = y})
    using apply_type by auto
    then
    show ?thesis
    proof (cases Finite(Y))
    case True
    with ⟨Z = (⋃ y ∈ Y. {x ∈ Z . F'x = y})⟩ and assms and ⟨F ∈ Z → Y⟩
    show ?thesis
    using Finite_RepFun[THEN [2] Finite_Union, of Y λy. F-“{y}]] 0 vimage_fun_sing[OF
⟨F ∈ Z → Y⟩]
    by simp
    next
    case False
    moreover from this ⟨M(Y)⟩
    have Y ≲M |Y|M
    using cardinal_rel_eqpoll_rel eqpoll_rel_sym eqpoll_rel_imp_lepoll_rel by
auto
    moreover
    note assms
    moreover from ⟨F ∈ _ ∩ _⟩
    have Finite({x ∈ Z . F'x = y}) M(F-“{y}) if y ∈ Y for y
    unfolding Finite_to_one_rel_def
    using transM[OF that ⟨M(Y)⟩] transM[OF _ ⟨M(Z)⟩] vimage_fun_sing[OF
⟨F ∈ Z → Y⟩] that
    by simp_all
    moreover from calculation
    have |{x ∈ Z . F'x = y}|M ∈ ω if y ∈ Y for y
    using Finite_cardinal_rel_in_nat that transM[OF that ⟨M(Y)⟩] vimage_fun_sing[OF
⟨F ∈ Z → Y⟩] that
    by simp
    moreover from calculation
    have |{x ∈ Z . F'x = y}|M ≤ |Y|M if y ∈ Y for y
    using Infinite_imp_nats_lepoll_rel[THEN lepoll_rel_imp_cardinal_rel_le,
of _ |{x ∈ Z . F'x = y}|M]
    that cardinal_rel_idem transM[OF that ⟨M(Y)⟩] vimage_fun_sing[OF
⟨F ∈ Z → Y⟩]
    by auto
    ultimately
    have |⋃ y ∈ Y. {x ∈ Z . F'x = y}|M ≤ |Y|M
    using lepoll_rel_imp_cardinal_rel_UN_le
    Infinite_InfCard_rel_cardinal_rel[of Y] vimage_fun_sing[OF ⟨F ∈ Z → Y⟩]

```



```

    by(auto simp add:transM[OF _ ⟨M(Y)⟩])
  moreover from ⟨F ∈ Finite_to_one_rel(M,Z,Y) ∩ surj_rel(M,Z,Y)⟩ ⟨M(Z)⟩
  ⟨M(F)⟩ ⟨M(Y)⟩
  have |Y|^M ≤ |Z|^M
    using surj_rel_implies_cardinal_rel_le by auto
  moreover
  note ⟨Z = (⋃ y∈Y. {x∈Z . F'x = y})⟩
  ultimately
  show ?thesis
    using le_anti_sym by auto
qed
qed

lemma cardinal_rel_map_Un:
  assumes Infinite(X) Finite(b) M(X) M(b)
  shows |{a ∪ b . a ∈ X}|^M = |X|^M
proof -
  have (λa∈X. a ∪ b) ∈ Finite_to_one_rel(M,X,{a ∪ b . a ∈ X})
    (λa∈X. a ∪ b) ∈ surj_rel(M,X,{a ∪ b . a ∈ X})
    M({a ∪ b . a ∈ X})
  unfolding def_surj_rel
proof
  fix d
  have Finite({a ∈ X . a ∪ b = d}) (is Finite(?Y(b,d)))
    using ⟨Finite(b)⟩
  proof (induct arbitrary:d)
    case 0
    have {a ∈ X . a ∪ 0 = d} = (if d∈X then {d} else 0)
      by auto
    then
    show ?case by simp
  next
    case (cons c b)
    from ⟨c ∉ b⟩
    have ?Y(cons(c,b),d) ⊆ (if c∈d then ?Y(b,d) ∪ ?Y(b,d-{c}) else 0)
      by auto
    with cons
    show ?case
      using subset_Finite
      by simp
  qed
  moreover
  assume d ∈ {x ∪ b . x ∈ X}
  ultimately
  show Finite({a ∈ X . M(a) ∧ (λx∈X. x ∪ b) ' a = d})
    using subset_Finite[of {a ∈ X . M(a) ∧ (λx∈X. x ∪ b) ' a = d}
      {a ∈ X . (λx∈X. x ∪ b) ' a = d}] by auto
next
  note ⟨M(X)⟩ ⟨M(b)⟩

```

```

moreover
show  $M(\lambda a \in X. a \cup b)$ 
  using lam_closed[of  $\lambda x. x \cup b, OF\_ \langle M(X) \rangle$ ] Un_closed[OF transM[OF\_
 $\langle M(X) \rangle$ ]  $\langle M(b) \rangle$ ]
    tag_union_replacement[OF  $\langle M(b) \rangle$ ]
  by simp
moreover from this
have  $\{a \cup b . a \in X\} = (\lambda x \in X. x \cup b) \text{ `` } X$ 
  using image_lam by simp
with calculation
show  $M(\{a \cup b . a \in X\})$  by auto
moreover from calculation
show  $(\lambda a \in X. a \cup b) \in X \rightarrow^M \{a \cup b . a \in X\}$ 
  using function_space_rel_char by (auto intro: lam_funtype)
ultimately
show  $(\lambda a \in X. a \cup b) \in \text{surj}^M(X, \{a \cup b . a \in X\})$   $M(\{a \cup b . a \in X\})$ 
  using surj_rel_char function_space_rel_char
  unfolding surj_def by auto
next
qed (simp add: \langle M(X) \rangle)
moreover from assms this
show ?thesis
  using Finite_to_one_rel_surj_rel_imp_cardinal_rel_eq by simp
qed

```

21.5 Results on relative cardinal exponentiation

lemma *cecp_rel_eqpoll_rel_cong*:

assumes

$$A \approx^M A' \quad B \approx^M B' \quad M(A) \quad M(A') \quad M(B) \quad M(B')$$

shows

$$A^{\uparrow B, M} = A'^{\uparrow B', M}$$

unfolding *cecp_rel_def* **using** *cardinal_rel_eqpoll_rel_iff*

function_space_rel_eqpoll_rel_cong *assms*

by *simp*

lemma *cecp_rel_cecp_rel_cmult*:

assumes $M(\kappa) \quad M(\nu 1) \quad M(\nu 2)$

shows $(\kappa^{\uparrow \nu 1, M})^{\uparrow \nu 2, M} = \kappa^{\uparrow \nu 2} \otimes^M \nu 1, M$

proof -

have $(\kappa^{\uparrow \nu 1, M})^{\uparrow \nu 2, M} = (\nu 1 \rightarrow^M \kappa)^{\uparrow \nu 2, M}$

using *cardinal_rel_eqpoll_rel*

by (*intro cecp_rel_eqpoll_rel_cong*) (*simp_all add: assms cecp_rel_def*)

also from *assms*

have $\dots = \kappa^{\uparrow \nu 2} \times \nu 1, M$

unfolding *cecp_rel_def* **using** *curry_eqpoll_rel*[*THEN cardinal_rel_cong*] **by**

blast

also

have $\dots = \kappa^{\uparrow \nu 2} \otimes^M \nu 1, M$

```

    using cardinal_rel_eqpoll_rel[THEN eqpoll_rel_sym]
    unfolding cmult_rel_def by (intro cexp_rel_eqpoll_rel_cong) (auto simp
add:assms)
    finally
    show ?thesis .
qed

```

lemma *cardinal_rel_Pow_rel*: $M(X) \implies |Pow_rel(M, X)|^M = 2^{\uparrow X, M}$ — Perhaps
it's better with $|X|$

```

    using cardinal_rel_eqpoll_rel_iff[THEN iffD2,
      OF _ _ Pow_rel_eqpoll_rel_function_space_rel]
    unfolding cexp_rel_def by simp

```

lemma *cantor_cexp_rel*:

```

    assumes Card_rel(M, ν) M(ν)
    shows ν < 2↑ν, M
    using assms Card_rel_is_Ord Card_rel_cexp_rel
proof (intro not_le_iff_lt[THEN iffD1] notI)
    assume 2↑ν, M ≤ ν
    with assms
    have |Pow_rel(M, ν)|M ≤ ν
      using cardinal_rel_Pow_rel[of ν] by simp
    with assms
    have Pow_rel(M, ν) ≲M ν
    using cardinal_rel_eqpoll_rel_iff Card_rel_le_imp_lepoll_rel Card_rel_cardinal_rel_eq
    by auto
    then
    obtain g where g ∈ inj_rel(M, Pow_rel(M, ν), ν)
      by blast
    moreover
    note ⟨M(ν)⟩
    moreover from calculation
    have M(g) by (auto dest:transM)
    ultimately
    show False
      using cantor_inj_rel by simp
qed simp

```

lemma *countable_iff_le_rel_Aleph_rel_one*:

```

    notes iff_trans[trans]
    assumes M(C)
    shows countableM(C) ⟷ |C|M ≲M ℵ1M
proof -
    have countableM(C) ⟷ C ≲M ℵ1M
      using assms lesspoll_rel_csucc_rel[of ω C] Aleph_rel_succ Aleph_rel_zero
      unfolding countable_rel_def by simp
    also from assms
    have ... ⟷ |C|M ≲M ℵ1M
    using cardinal_rel_eqpoll_rel[THEN eqpoll_rel_sym, THEN eq_lespoll_rel_trans]

```

```

    by (auto intro:cardinal_rel_eqpoll_rel[THEN eq_lespoll_rel_trans])
  finally
    show ?thesis .
qed

end — M_cardinal_library

lemma (in M_cardinal_library) countable_fun_imp_countable_image:
  assumes  $f:C \rightarrow^M B$  countableM(C)  $\wedge c. c \in C \implies$  countableM(f'c)
    M(C) M(B)
  shows countableM( $\bigcup (f''C)$ )
  using assms function_space_rel_char_image_fun[of f]
    cardinal_rel_RepFun_apply_le[of f C B]
    countable_rel_iff_cardinal_rel_le_nat[THEN iffD1, THEN [2] le_trans, of _
]
    countable_rel_iff_cardinal_rel_le_nat
  by (rule_tac countable_rel_union_countable_rel)
    (auto dest:transM del:imageE)

end

```

22 The Delta System Lemma, Relativized

```

theory Delta_System_Relative
  imports
    Cardinal_Library_Relative
begin

definition
  delta_system ::  $i \Rightarrow o$  where
    delta_system(D)  $\equiv \exists r. \forall A \in D. \forall B \in D. A \neq B \longrightarrow A \cap B = r$ 

lemma delta_systemI[intro]:
  assumes  $\forall A \in D. \forall B \in D. A \neq B \longrightarrow A \cap B = r$ 
  shows delta_system(D)
  using assms unfolding delta_system_def by simp

lemma delta_systemD[dest]:
  delta_system(D)  $\implies \exists r. \forall A \in D. \forall B \in D. A \neq B \longrightarrow A \cap B = r$ 
  unfolding delta_system_def by simp

lemma delta_system_root_eq_Inter:
  assumes delta_system(D)
  shows  $\forall A \in D. \forall B \in D. A \neq B \longrightarrow A \cap B = \bigcap D$ 
proof (clarify, intro equalityI, auto)
  fix A' B' x C
  assume hyp:  $A' \in D \ B' \in D \ A' \neq B' \ x \in A' \ x \in B' \ C \in D$ 

```

```

with assms
obtain r where delta: $\forall A \in D. \forall B \in D. A \neq B \longrightarrow A \cap B = r$ 
  by auto
show  $x \in C$ 
proof (cases  $C=A'$ )
  case True
    with hyp and assms
    show ?thesis by simp
  next
    case False
    moreover
      note hyp
      moreover from calculation and delta
      have  $r = C \cap A' \ A' \cap B' = r \ x \in r$  by auto
      ultimately
        show ?thesis by simp
  qed
qed

```

relativize functional *delta_system delta_system_rel external*

```

locale M_delta = M_cardinal_library +
  assumes
    countable_lepoll_assms:
       $M(G) \implies M(A) \implies M(b) \implies M(f) \implies \text{separation}(M, \lambda y. \exists x \in A. \\
        y = \langle x, \mu \ i. x \in \text{if\_range\_F\_else\_F}(\lambda x. \{xa \in G . x \in xa\}, \\
        b, f, i)))$ 
  begin

```

lemmas *cardinal_replacement* = *lam_replacement_cardinal_rel*[*unfolded lam_replacement_def*]

```

lemma disjoint_separation:  $M(c) \implies \text{separation}(M, \lambda x. \exists a. \exists b. x = \langle a, b \rangle \wedge a \cap \\
b = c)$ 
  using separation_pair separation_eq lam_replacement_constant lam_replacement_Int
  by simp

```

```

lemma insnd_ball:  $M(G) \implies \text{separation}(M, \lambda p. \forall x \in G. x \in \text{snd}(p) \longleftrightarrow \text{fst}(p) \in \\
x)$ 
  using separation_ball separation_iff' lam_replacement_fst lam_replacement_snd \\
separation_in lam_replacement_hcomp
  by simp

```

```

lemma (in M_trans) mem_F_bound6:
  fixes F G
  defines  $F \equiv \lambda x. \text{Collect}(G, (\in)(x))$ 
  shows  $x \in F(G, c) \implies c \in (\text{range}(f) \cup \bigcup G)$ 
  using apply_0 unfolding F_def
  by (cases  $M(c)$ , auto simp:F_def)

```

```

lemma delta_system_Aleph_rel1:
  assumes  $\forall A \in F. \text{Finite}(A) \ F \approx^M \aleph_1^M M(F)$ 
  shows  $\exists D[M]. \ D \subseteq F \wedge \text{delta\_system}(D) \wedge D \approx^M \aleph_1^M$ 
proof -
  have  $M(G) \implies M(p) \implies M(\{A \in G . p \in A\})$  for  $G \ p$ 
  proof -
    assume  $M(G) \ M(p)$ 
    have  $\{A \in G . p \in A\} = G \cap (\text{Memrel}(\{p\} \cup G) \text{ `` } \{p\})$ 
    unfolding Memrel_def by auto
    with  $\langle M(G) \rangle \ \langle M(p) \rangle$ 
    show ?thesis by simp
  qed
from  $\langle M(F) \rangle$ 
have  $M(\lambda A \in F. |A|^M)$ 
  using cardinal_replacement
  by (rule_tac lam_closed) (auto dest:transM)

```

Since all members are finite,

```

with  $\langle \forall A \in F. \text{Finite}(A) \rangle \ \langle M(F) \rangle$ 
have  $(\lambda A \in F. |A|^M) : F \rightarrow^M \omega$  (is ?cards : _)
  by (simp add:mem_function_space_rel_abs, rule_tac lam_type)
  (force dest:transM)
moreover from this
have  $a: ?cards \text{ `` } \{n\} = \{ A \in F . |A|^M = n \}$  for  $n$ 
  using vimage_lam by auto
moreover
note  $\langle F \approx^M \aleph_1^M \rangle \ \langle M(F) \rangle$ 
moreover from calculation

```

there are uncountably many have the same cardinal:

```

obtain  $n$  where  $n \in \omega \ \mid \ ?cards \text{ `` } \{n\} \mid^M = \aleph_1^M$ 
  using eqpoll_rel_Aleph_rel1_cardinal_rel_vimage[of F ?cards] by auto
moreover
define  $G$  where  $G \equiv ?cards \text{ `` } \{n\}$ 
moreover from calculation and  $\langle M(?cards) \rangle$ 
have  $M(G)$  by blast
moreover from calculation
have  $G \subseteq F$  by auto
ultimately

```

Therefore, without loss of generality, we can assume that all elements of the family have cardinality $n \in \omega$.

```

have  $A \in G \implies |A|^M = n$  and  $G \approx^M \aleph_1^M$  and  $M(G)$  for  $A$ 
  using cardinal_rel_Card_rel_eqpoll_rel_iff by auto
with  $\langle n \in \omega \rangle$ 

```

So we prove the result by induction on this n and generalizing G , since the argument requires changing the family in order to apply the inductive hypothesis.

```

have  $\exists D[M]. D \subseteq G \wedge \text{delta\_system}(D) \wedge D \approx^M \aleph_1^M$ 
proof (induct arbitrary:G)
  case 0 — This case is impossible
  then
  have  $G \subseteq \{0\}$ 
  using cardinal_rel_0_iff_0 by (blast dest:transM)
  with  $\langle G \approx^M \aleph_1^M \rangle, \langle M(G) \rangle$ 
  show ?case
  using nat_lt_Aleph_rel1_subset_imp_le_cardinal_rel[of G {0}]
  lt_trans2 cardinal_rel_Card_rel_eqpoll_rel_iff by auto
next
case (succ n)
have  $\forall a \in G. \text{Finite}(a)$ 
proof
  fix a
  assume  $a \in G$ 
  moreover
  note  $\langle M(G) \rangle, \langle n \in \omega \rangle$ 
  moreover from calculation
  have  $M(a)$  by (auto dest: transM)
  moreover from succ and  $\langle a \in G \rangle$ 
  have  $|a|^M = \text{succ}(n)$  by simp
  ultimately
  show  $\text{Finite}(a)$ 
  using Finite_cardinal_rel_iff' nat_into_Finite[of succ(n)]
  by fastforce
qed
show  $\exists D[M]. D \subseteq G \wedge \text{delta\_system}(D) \wedge D \approx^M \aleph_1^M$ 
proof (cases  $\exists p[M]. \{A \in G . p \in A\} \approx^M \aleph_1^M$ )
  case True — the positive case, uncountably many sets with a common element
  then
  obtain p where  $\{A \in G . p \in A\} \approx^M \aleph_1^M$  by blast
  moreover
  note 1= $\langle M(G) \rangle, \langle M(G) \implies M(p) \implies M(\{A \in G . p \in A\}) \rangle$  singleton_closed[OF
 $\langle M(p) \rangle$ ]
  moreover from this
  have  $M(\{x - \{p\} . x \in \{x \in G . p \in x\}\})$ 
  using RepFun_closed[OF lam_replacement_Diff'[THEN
  lam_replacement_imp_strong_replacement]]
  Diff_closed[OF transM[OF 1(2)]] by auto
  moreover from 1
  have  $M(\text{converse}(\lambda x \in \{x \in G . p \in x\}. x - \{p\}))$  (is  $M(\text{converse}(?h))$ )
  using converse_closed[of ?h] lam_closed[OF diff_Pair_replacement]
  Diff_closed[OF transM[OF 1(2)]]
  by auto
  moreover from calculation
  have  $\{A - \{p\} . A \in \{X \in G. p \in X\}\} \approx^M \aleph_1^M$  (is  $?F \approx^M \_$ )
  using Diff_bij_rel[of  $\{A \in G . p \in A\} \{p\}$ , THEN
  comp_bij_rel[OF bij_rel_converse_bij_rel, where  $C = \aleph_1^M$ ,

```

THEN $\text{bij_rel_imp_eqpoll_rel, of } __ \text{ ?F}]$
 unfolding eqpoll_rel_def
 by (auto simp del: mem_bij_abs)

Now using the hypothesis of the successor case,

moreover from $\langle \bigwedge A. A \in G \implies |A|^M = \text{succ}(n) \rangle \langle \forall a \in G. \text{Finite}(a) \rangle$
 and this $\langle M(G) \rangle$
 have $p \in A \implies A \in G \implies |A - \{p\}|^M = n$ for A
 using $\text{Finite_imp_succ_cardinal_rel_Diff}[of _ p]$ by (force dest: transM)
 moreover
 have $\forall a \in ?F. \text{Finite}(a)$
 proof (clarsimp)
 fix A
 assume $p \in A \ A \in G$
 with $\langle \bigwedge A. p \in A \implies A \in G \implies |A - \{p\}|^M = n \rangle$ and $\langle n \in \omega \rangle \langle M(G) \rangle$
 have $\text{Finite}(|A - \{p\}|^M)$
 using nat_into_Finite by simp
 moreover from $\langle p \in A \rangle \langle A \in G \rangle \langle M(G) \rangle$
 have $M(A - \{p\})$ by (auto dest: transM)
 ultimately
 show $\text{Finite}(A - \{p\})$
 using $\text{Finite_cardinal_rel_iff'}$ by simp
 qed
 moreover

we may apply the inductive hypothesis to the new family $\{A - \{p\} . A \in \{X \in G . p \in X\}\}$:

note $\langle (\bigwedge A. A \in ?F \implies |A|^M = n) \implies ?F \approx^M \aleph_1^M \implies M(?F) \implies \exists D[M]. D \subseteq ?F \wedge \text{delta_system}(D) \wedge D \approx^M \aleph_1^M \rangle$
 moreover
 note $1 = \langle M(G) \rangle \langle M(G) \implies M(p) \implies M(\{A \in G . p \in A\}) \rangle \text{singleton_closed}[OF \langle M(p) \rangle]$
 moreover from this
 have $M(\{x - \{p\} . x \in \{x \in G . p \in x\}\})$
 using $\text{RepFun_closed}[OF \text{lam_replacement_Diff'}[THEN \text{lam_replacement_imp_strong_replacement}]]$
 $\text{Diff_closed}[OF \text{transM}[OF _ 1(2)]]$ by auto
 ultimately
 obtain D where $D \subseteq \{A - \{p\} . A \in \{X \in G . p \in X\}\}$ $\text{delta_system}(D)$ $D \approx^M \aleph_1^M$
 $M(D)$
 by auto
 moreover from this
 obtain r where $\forall A \in D. \forall B \in D. A \neq B \longrightarrow A \cap B = r$
 by fastforce
 then
 have $\forall A \in D. \forall B \in D. A \cup \{p\} \neq B \cup \{p\} \longrightarrow (A \cup \{p\}) \cap (B \cup \{p\}) = r \cup \{p\}$
 by blast
 ultimately
 have $\text{delta_system}(\{B \cup \{p\} . B \in D\})$ (is $\text{delta_system}(?D)$)


```

    by fastforce
  moreover from  $\langle M(D) \rangle \langle M(p) \rangle$ 
  have  $M(?D)$ 
    using RepFun_closed_un_Pair_replacement transM[of _ D] by auto
  moreover from  $\langle D \approx^M \aleph_1^M \rangle \langle M(D) \rangle$ 
  have  $\text{Infinite}(D) \mid D \mid^M = \aleph_1^M$ 
    using uncountable_rel_iff_subset_eqpoll_rel_Aleph_rel1[THEN iffD2,
      THEN uncountable_rel_imp_Infinite, of D]
      cardinal_rel_eqpoll_rel_iff[of D  $\aleph_1^M$ ]  $\langle M(D) \rangle \langle D \approx^M \aleph_1^M \rangle$ 
    by auto
  moreover from this  $\langle M(?D) \rangle \langle M(D) \rangle \langle M(p) \rangle$ 
  have  $?D \approx^M \aleph_1^M$ 
    using cardinal_rel_map_Un[of D {p}] naturals_lt_nat
      cardinal_rel_eqpoll_rel_iff[THEN iffD1] by simp
  moreover
  note  $\langle D \subseteq \{A - \{p\} \mid A \in \{X \in G. p \in X\}\} \rangle$ 
  have  $?D \subseteq G$ 
  proof -
    {
      fix A
      assume  $A \in G \mid p \in A$ 
      moreover from this
      have  $A = A - \{p\} \cup \{p\}$ 
        by blast
      ultimately
      have  $A - \{p\} \cup \{p\} \in G$ 
        by auto
    }
    with  $\langle D \subseteq \{A - \{p\} \mid A \in \{X \in G. p \in X\}\} \rangle$ 
    show ?thesis
      by blast
  qed
  moreover
  note  $\langle M(?D) \rangle$ 
  ultimately
  show  $\exists D[M]. D \subseteq G \wedge \text{delta\_system}(D) \wedge D \approx^M \aleph_1^M$  by auto
next
case False
note  $\langle \neg (\exists p[M]. \{A \in G \mid p \in A\} \approx^M \aleph_1^M) \rangle$  — the other case
 $\langle M(G) \rangle \langle \bigwedge p. M(G) \implies M(p) \implies M(\{A \in G \mid p \in A\}) \rangle$ 
moreover from  $\langle G \approx^M \aleph_1^M \rangle$  and this
have  $M(p) \implies \{A \in G \mid p \in A\} \lesssim^M \aleph_1^M$  (is  $\_ \implies ?G(p) \lesssim^M \_$ ) for p
  by (auto intro!: lepoll_rel_eq_trans[OF subset_imp_lepoll_rel] dest: transM)
moreover from calculation
have  $M(p) \implies ?G(p) \prec^M \aleph_1^M$  for p
  using  $\langle M(G) \implies M(p) \implies M(\{A \in G \mid p \in A\}) \rangle$ 
  unfolding lesspoll_rel_def by simp
moreover from calculation
have  $M(p) \implies ?G(p) \lesssim^M \omega$  for p

```

```

    using lesspoll_rel_Aleph_rel_succ[of 0] Aleph_rel_zero by auto
  moreover
  have  $\{A \in G . S \cap A \neq 0\} = (\bigcup_{p \in S} ?G(p))$  for  $S$ 
    by auto
  moreover from calculation
  have  $M(S) \implies i \in S \implies M(\{x \in G . i \in x\})$  for  $i \in S$ 
    by (auto dest: transM)
  moreover
  have  $M(S) \implies \text{countable\_rel}(M, S) \implies \text{countable\_rel}(M, \{A \in G . S \cap A \neq 0\})$  for  $S$ 
  proof -
    from  $\langle M(G) \rangle$ 
    interpret  $M\_replacement\_lepoll\ M\ \lambda\_x.\ Collect(G, (\in)(x))$ 
    using countable_lepoll_assms lam_replacement_inj_rel separation_in_rev
      lam_replacement_Collect[OF  $\_ \_ \text{insnd\_ball}$ ] mem_F_bound6[of  $\_ G$ ]
    by unfold_locales
      (auto dest: transM intro: lam_Least_assumption_general[of  $\_ \_ \_$ 
Union])
    fix  $S$ 
    assume  $M(S)$ 
    with  $\langle M(G) \rangle \langle \bigwedge i. M(S) \implies i \in S \implies M(\{x \in G . i \in x\}) \rangle$ 
    interpret  $M\_cardinal\_UN\_lepoll\_ ?G\ S$ 
      using lepoll_assumptions
    by unfold_locales (auto dest: transM)
    assume countable_rel(M, S)
    with  $\langle M(S) \rangle$  calculation(6) calculation(7,8)[of  $S$ ]
    show countable_rel(M,  $\{A \in G . S \cap A \neq 0\})$ 
      using InfCard_rel_nat Card_rel_nat
      le_Card_rel_iff[THEN iffD2, THEN [3] leqpoll_rel_imp_cardinal_rel_UN_le,
        THEN [4] le_Card_rel_iff[THEN iffD1], of  $\omega$ ] j.UN_closed
    unfolding countable_rel_def by (auto dest: transM)
  qed
  define Disjoint where  $Disjoint = \{ \langle A, B \rangle \in G \times G . B \cap A = 0 \}$ 
  have  $Disjoint = \{x \in G \times G . \exists a\ b. x = \langle a, b \rangle \wedge a \cap b = 0\}$ 
    unfolding Disjoint_def by force
  with  $\langle M(G) \rangle$ 
  have  $M(Disjoint)$ 
    using disjoint_separation by simp

```

For every countable_rel subfamily of G there is another some element disjoint from all of them:

```

  have  $\exists A \in G. \forall S \in X. \langle S, A \rangle \in Disjoint$  if  $|X|^M < \aleph_I^M$   $X \subseteq G$   $M(X)$  for  $X$ 
  proof -
    note  $\langle n \in \omega \rangle \langle M(G) \rangle$ 
    moreover from this and  $\langle \bigwedge A. A \in G \implies |A|^M = \text{succ}(n) \rangle$ 
    have  $|A|^M = \text{succ}(n) \implies M(A)$  if  $A \in G$  for  $A$ 
    using that Finite_cardinal_rel_eq_cardinal[of  $A$ ] Finite_cardinal_rel_iff'[of
A]
      nat_into_Finite transM[of  $A\ G$ ] by (auto dest: transM)

```

```

ultimately
have  $A \in G \implies \text{Finite}(A)$  for  $A$ 
  using cardinal_rel_Card_rel_eqpoll_rel_iff[of succ(n) A]
    Finite_cardinal_rel_eq_cardinal[of A] nat_into_Card_rel[of succ(n)]
    nat_into_M[of n] unfolding Finite_def eqpoll_rel_def by (auto)
with  $\langle X \subseteq G \rangle \langle M(X) \rangle$ 
have  $A \in X \implies \text{countable\_rel}(M, A)$  for  $A$ 
  using Finite_imp_countable_rel by (auto dest: transM)
moreover from  $\langle M(X) \rangle$ 
have  $M(\bigcup X)$  by simp
moreover
note  $\langle |X|^M < \aleph_1^M \rangle \langle M(X) \rangle$ 
ultimately
have countable_rel(M,  $\bigcup X$ )
  using Card_rel_nat[THEN cardinal_rel_lt_csucc_rel_iff, of X]
    countable_rel_union_countable_rel[of X]
    countable_rel_iff_cardinal_rel_le_nat[of X] Aleph_rel_succ
    Aleph_rel_zero by simp
with  $\langle M(\bigcup X) \rangle \langle M(\_) \implies \text{countable\_rel}(M, \_) \implies \text{countable\_rel}(M, \{A$ 
 $\in G . \_ \cap A \neq 0\}) \rangle$ 
have countable_rel(M,  $\{A \in G . (\bigcup X) \cap A \neq 0\}$ ) by simp
with  $\langle G \approx^M \aleph_1^M \rangle \langle M(G) \rangle$ 
obtain  $B$  where  $B \in G$   $B \notin \{A \in G . (\bigcup X) \cap A \neq 0\}$ 
  using nat_lt_Aleph_rel1 cardinal_rel_Card_rel_eqpoll_rel_iff[of  $\aleph_1^M$ 
 $G$ ]
    uncountable_rel_not_subset_countable_rel
    [of  $\{A \in G . (\bigcup X) \cap A \neq 0\}$   $G$ ]
    uncountable_rel_iff_nat_lt_cardinal_rel[of  $G$ ]
  by force
then
have  $\exists A \in G. \forall S \in X. A \cap S = 0$  by auto
with  $\langle X \subseteq G \rangle$ 
show  $\exists A \in G. \forall S \in X. \langle S, A \rangle \in \text{Disjoint}$  unfolding Disjoint_def
  using subsetD by simp
qed
moreover from  $\langle G \approx^M \aleph_1^M \rangle \langle M(G) \rangle$ 
obtain  $b$  where  $b \in G$ 
  using uncountable_rel_iff_subset_eqpoll_rel_Aleph_rel1
    uncountable_rel_not_empty by blast
ultimately

```

Hence, the hypotheses to perform a bounded-cardinal selection are satisfied,

```

obtain  $S$  where  $S: \aleph_1^M \rightarrow^M G$   $\alpha \in \aleph_1^M \implies \beta \in \aleph_1^M \implies \alpha < \beta \implies \langle S' \alpha, S' \beta \rangle$ 
 $\in \text{Disjoint}$ 
  for  $\alpha \beta$ 
  using bounded_cardinal_rel_selection[of  $\aleph_1^M$   $G$  Disjoint]  $\langle M(\text{Disjoint}) \rangle$ 
  by force
moreover from this  $\langle n \in \omega \rangle \langle M(G) \rangle$ 
have  $\text{in } M: M(S) \ M(n) \wedge x. x \in \aleph_1^M \implies S' x \in G \wedge x. x \in \aleph_1^M \implies M(x)$ 

```

```

    using function_space_rel_char by (auto dest: transM)
  ultimately
  have  $\alpha \in \aleph_1^M \implies \beta \in \aleph_1^M \implies \alpha \neq \beta \implies S'\alpha \cap S'\beta = 0$  for  $\alpha \beta$ 
    unfolding Disjoint_def
    using lt_neq_symmetry[of  $\aleph_1^M$   $\lambda \alpha \beta. S'\alpha \cap S'\beta = 0$ ] Card_rel_is_Ord
  by auto (blast)

```

and a symmetry argument shows that obtained S is an injective \aleph_1^M -sequence of disjoint elements of G .

```

  moreover from this and  $\langle \bigwedge A. A \in G \implies |A|^M = \text{succ}(n) \rangle$  inM
     $\langle S : \aleph_1^M \rightarrow^M G \rangle \langle M(G) \rangle$ 
  have  $S \in \text{inj\_rel}(M, \aleph_1^M, G)$ 
    using def_inj_rel[OF Aleph_rel_closed  $\langle M(G) \rangle$ , of 1]
  proof (clarsimp)
    fix w x
    from inM
    have  $a \in \aleph_1^M \implies b \in \aleph_1^M \implies a \neq b \implies S'a \neq S'b$  for  $a b$ 
    using  $\langle \bigwedge A. A \in G \implies |A|^M = \text{succ}(n) \rangle$  [THEN [4] cardinal_rel_succ_not_0] [THEN
[4]
      Int_eq_zero_imp_not_eq[OF calculation, of  $\aleph_1^M \lambda x. x$ ,
      of  $\lambda \_ . n$ ], OF  $\_ \_ \_ \_ \text{apply\_closed}$ ] by auto
    moreover
    assume  $w \in \aleph_1^M \ x \in \aleph_1^M \ S'w = S'x$ 
    ultimately
    show  $w = x$  by blast
  qed
  moreover from this  $\langle M(G) \rangle$ 
  have  $\text{range}(S) \approx^M \aleph_1^M$ 
    using inj_rel_bij_rel_range eqpoll_rel_sym unfolding eqpoll_rel_def
    by (blast dest: transM)
  moreover
  note  $\langle M(G) \rangle$ 
  moreover from calculation
  have  $\text{range}(S) \subseteq G$ 
    using inj_rel_is_fun range_fun_subset_codomain
    by (fastforce dest: transM)
  moreover
  note  $\langle M(S) \rangle$ 
  ultimately
  show  $\exists D[M]. D \subseteq G \wedge \text{delta\_system}(D) \wedge D \approx^M \aleph_1^M$ 
    using inj_rel_is_fun ZF_Library.range_eq_image[of  $S \aleph_1^M G$ ]
      image_function[OF fun_is_function, OF inj_rel_is_fun, of  $S \aleph_1^M G$ ]
      domain_of_fun[OF inj_rel_is_fun, of  $S \aleph_1^M G$ ] apply_replacement[of  $S$ ]
    by (rule_tac  $x = S'\aleph_1^M$  in rexI) (auto dest: transM intro!: RepFun_closed)

```

This finishes the successor case and hence the proof.

```

  qed
  qed
  with  $\langle G \subseteq F \rangle$ 

```

```

  show ?thesis by blast
qed

lemma delta_system_uncountable_rel:
  assumes  $\forall A \in F. \text{Finite}(A) \text{ uncountable\_rel}(M, F) \text{ } M(F)$ 
  shows  $\exists D[M]. D \subseteq F \wedge \text{delta\_system}(D) \wedge D \approx^M \aleph_1^M$ 
proof -
  from assms
  obtain S where  $S \subseteq F \text{ } S \approx^M \aleph_1^M \text{ } M(S)$ 
    using uncountable_rel_iff_subset_eqpoll_rel_Aleph_rel1[of F] by auto
  moreover from  $\langle \forall A \in F. \text{Finite}(A) \rangle$  and this
  have  $\forall A \in S. \text{Finite}(A)$  by auto
  ultimately
  show ?thesis using delta_system_Aleph_rel1[of S]
    by (auto dest:transM)
qed

end — M_delta

end

```

23 Relative DC

```

theory Pointed_DC_Relative
  imports
    Cardinal_Library_Relative

begin

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

lemmas dc_witness_def = dc_witness_nat_def

relativize_functional dc_witness dc_witness_rel
relationalize dc_witness_rel is_dc_witness

schematic_goal sats_is_dc_witness_fm_auto:
  assumes  $na < length(env) \text{ } e < length(env)$ 
  shows
     $na \in \omega \implies$ 
     $A \in \omega \implies$ 
     $a \in \omega \implies$ 
     $s \in \omega \implies$ 
     $R \in \omega \implies$ 
     $e \in \omega \implies$ 
     $env \in list(Aa) \implies$ 

```

```

0 ∈ Aa ⇒
  is_dc_witness(##Aa, nth(na, env), nth(A, env), nth(a, env), nth(s, env),
nth(R, env), nth(e, env)) ⇔
  Aa, env ⊨ ?fm(nat, A, a, s, R, e)
unfolding is_dc_witness_def is_recurser_def
by (rule is_transrec_iff_sats | simp_all)
(rule iff_sats is_nat_case_iff_sats is_eclose_iff_sats sep_rules | simp add:assms)+

```

synthesize *is_dc_witness* **from_schematic**

```

manual_arity for is_dc_witness_fm
unfolding is_dc_witness_fm_def apply (subst arity_transrec_fm)
apply (simp add:arity) defer 3
apply (simp add:arity) defer
apply (subst arity_is_nat_case_fm)
apply (simp add:arity del:arity_transrec_fm) prefer 5
by (simp add:arity del:arity_transrec_fm)+

```

```

definition dcwit_body :: [i,i,i,i,i] ⇒ o where
  dcwit_body(A,a,g,R) ≡ λp. snd(p) = dc_witness(fst(p), A, a, g, R)

```

```

relativize_functional dcwit_body dcwit_body_rel
relationalize dcwit_body_rel is_dcwit_body

```

```

synthesize is_dcwit_body from_definition assuming nonempty
arity_theorem for is_dcwit_body_fm

```

```

context M_replacement
begin

```

```

lemma dc_witness_closed[intro,simp]:
  assumes M(n) M(A) M(a) M(s) M(R) n ∈ nat
  shows M(dc_witness(n,A,a,s,R))
  using ⟨n ∈ nat⟩
proof(induct)
  case 0
  with ⟨M(a)⟩
  show ?case
    unfolding dc_witness_def by simp
next
  case (succ x)
  with assms
  have M(dc_witness(x,A,a,s,R)) (is M(?b))
    by simp
  moreover from this assms
  have M(({ ?b } × A) ∩ R) (is M(?X)) by auto
  moreover
  have {x ∈ A. ⟨?b,x⟩ ∈ R} = {snd(y) . y ∈ ?X} (is _ = ?Y)
    by(intro equalityI subsetI,force,auto)

```

```

moreover from calculation
have  $M(?Y)$ 
using lam_replacement_snd lam_replacement_imp_strong_replacement RepFun_closed
      snd_closed[OF transM]
by auto
ultimately
show ?case
using  $\langle M(s) \rangle$  apply_closed
unfolding dc_witness_def by simp
qed

```

```

lemma dc_witness_rel_char:
  assumes  $M(A)$ 
  shows  $dc\_witness\_rel(M, n, A, a, s, R) = dc\_witness(n, A, a, s, R)$ 
proof -
  from assms
  have  $\{x \in A . \langle rec, x \rangle \in R\} = \{x \in A . M(x) \wedge \langle rec, x \rangle \in R\}$  for rec
    by (auto dest:transM)
  then
  show ?thesis
    unfolding dc_witness_def dc_witness_rel_def by simp
qed

```

```

lemma (in M_basic) first_section_closed:
  assumes
     $M(A)$   $M(a)$   $M(R)$ 
  shows  $M(\{x \in A . \langle a, x \rangle \in R\})$ 
proof -
  have  $\{x \in A . \langle a, x \rangle \in R\} = range(\{a\} \times range(R) \cap R) \cap A$ 
    by (intro equalityI) auto
  with assms
  show ?thesis
    by simp
qed

```

```

lemma witness_into_A [TC]:
  assumes  $a \in A$ 
     $\forall X[M]. X \neq 0 \wedge X \subseteq A \longrightarrow s'X \in A$ 
     $\forall y \in A. \{x \in A. \langle y, x \rangle \in R\} \neq 0$   $n \in nat$ 
     $M(A)$   $M(a)$   $M(s)$   $M(R)$ 
  shows  $dc\_witness(n, A, a, s, R) \in A$ 
  using  $\langle n \in nat \rangle$ 
proof(induct n)
  case 0
  then show ?case using  $\langle a \in A \rangle$  by simp
next
  case (succ x)
  with succ assms(1,3-)
  show ?case

```

```

using nat_into_M_first_section_closed
by (simp, rule_tac rev_subsetD, rule_tac assms(2)[rule_format])
    auto
qed

end — M_replacement

locale M_DC = M_trancl + M_replacement + M_eclose +
assumes
  separation_is_dcwit_body:
     $M(A) \implies M(a) \implies M(g) \implies M(R) \implies \text{separation}(M, \text{is\_dcwit\_body}(M, A,$ 
a, g, R))
and
  dcwit_replacement:  $\text{Ord}(na) \implies$ 
     $M(na) \implies$ 
     $M(A) \implies$ 
     $M(a) \implies$ 
     $M(s) \implies$ 
     $M(R) \implies$ 
    transrec_replacement
    (M,  $\lambda n f \text{ ntc.}$ 
      is_nat_case
      (M, a,
         $\lambda m \text{ bmfm.}$ 
           $\exists fm[M]. \exists cp[M].$ 
            is_apply(M, f, m, fm)  $\wedge$ 
            is_Collect(M, A,  $\lambda x. \exists fm x[M]. (M(x) \wedge fm x \in R) \wedge \text{pair}(M, fm,$ 
x, fm x), cp) \wedge
            is_apply(M, s, cp, bmfm),
            n, ntc), na)
    )
begin

lemma is_dc_witness_iff:
assumes  $\text{Ord}(na) \ M(na) \ M(A) \ M(a) \ M(s) \ M(R) \ M(res)$ 
shows  $\text{is\_dc\_witness}(M, na, A, a, s, R, res) \longleftrightarrow res = \text{dc\_witness\_rel}(M, na,$ 
A, a, s, R)
proof -
note assms
moreover from this
have  $\{x \in A . M(x) \wedge \langle f, x \rangle \in R\} = \{x \in A . \langle f, x \rangle \in R\}$  for f
by (auto dest:transM)
moreover from calculation
have  $M(fm) \implies M(\{x \in A . M(x) \wedge \langle fm, x \rangle \in R\})$  for fm
using first_section_closed by (auto dest:transM)
moreover from calculation
have  $M(x) \implies M(z) \implies M(mesa) \implies$ 
  ( $\exists ya[M]. \text{pair}(M, x, ya, z) \wedge$ 
     $\text{is\_wfrec}(M, \lambda n f. \text{is\_nat\_case}(M, a, \lambda m \text{ bmfm. } \exists fm[M]. \text{is\_apply}(M, f, m,$ 
fm) \wedge

```



```

    is_apply(M, s, {x ∈ A . ⟨fm, x⟩ ∈ R}, bmfM), n), mesa, x, ya))
  <=>
  (∃ y[M]. pair(M, x, y, z) ∧
   is_wfrec(M, λn f. is_nat_case(M, a,
    λm bmfM.
      ∃ fm[M]. ∃ cp[M]. is_apply(M, f, m, fm) ∧
      is_Collect(M, A, λx. M(x) ∧ ⟨fm, x⟩ ∈ R, cp) ∧ is_apply(M, s, cp,
bmfM), n),
    mesa, x, y)) for x z mesa by simp
moreover from calculation
show ?thesis
using assms dcwit_replacement[of na A a s R]
unfolding is_dc_witness_def dc_witness_rel_def
by (rule_tac recursor_abs) (auto dest:transM)
qed

```

```

lemma dcwit_body_abs:
  fst(x) ∈ ω ⇒ M(A) ⇒ M(a) ⇒ M(g) ⇒ M(R) ⇒ M(x) ⇒
  is_dcwit_body(M, A, a, g, R, x) <=> dcwit_body(A, a, g, R, x)
using pair_in_M_iff_apply_closed transM[of _ A]
  is_dc_witness_iff[of fst(x) A a g R snd(x)]
  fst_snd_closed dc_witness_closed
unfolding dcwit_body_def is_dcwit_body_def
by (auto dest:transM simp:absolut dc_witness_rel_char del:beXI intro!:beXI)

```

```

lemma separation_eq_dc_witness:
  M(A) ⇒
  M(a) ⇒
  M(g) ⇒
  M(R) ⇒ separation(M, λp. fst(p) ∈ ω → snd(p) = dc_witness(fst(p), A, a,
g, R))
using separation_is_dcwit_body dcwit_body_abs unfolding is_dcwit_body_def
oops

```

```

lemma Lambda_dc_witness_closed:
  assumes g ∈ PowM(A) - {0} → A a ∈ A ∀ y ∈ A. {x ∈ A . ⟨y, x⟩ ∈ R} ≠ 0
  M(g) M(A) M(a) M(R)
  shows M(λn ∈ nat. dc_witness(n, A, a, g, R))
proof -
from assms
have (λn ∈ nat. dc_witness(n, A, a, g, R)) = {p ∈ ω × A . is_dcwit_body(M, A, a, g, R, p)}
using witness_into_A[of a A g R]
  Pow_rel_char_apply_type[of g {x ∈ Pow(A) . M(x)} - {0} λ_. A]
unfolding lam_def split_def
apply (intro equalityI subsetI)
apply (auto)
by (subst dcwit_body_abs, simp_all add:transM[of _ ω] dcwit_body_def,
  subst (asm) dcwit_body_abs, auto dest:transM simp:dcwit_body_def)

```

```

with assms
show ?thesis
  using separation_is_dcwit_body dc_witness_rel_char unfolding split_def by
simp
qed

```

```

lemma witness_related:
  assumes  $a \in A$ 
     $\forall X[M]. X \neq 0 \wedge X \subseteq A \longrightarrow s'X \in X$ 
     $\forall y \in A. \{x \in A. \langle y, x \rangle \in R\} \neq 0 \quad n \in \text{nat}$ 
     $M(a) \ M(A) \ M(s) \ M(R) \ M(n)$ 
  shows  $\langle \text{dc\_witness}(n, A, a, s, R), \text{dc\_witness}(\text{succ}(n), A, a, s, R) \rangle \in R$ 
proof -
  note assms
  moreover from this
  have  $(\forall X[M]. X \neq 0 \wedge X \subseteq A \longrightarrow s'X \in A)$  by auto
  moreover from calculation
  have  $\text{dc\_witness}(n, A, a, s, R) \in A$  (is  $?x \in A$ )
    using witness_into_A[of _ _ s R n] by simp
  moreover from assms
  have  $M(\{x \in A. \langle \text{dc\_witness}(n, A, a, s, R), x \rangle \in R\})$ 
    using first_section_closed[of A dc_witness(n, A, a, s, R)]
    by simp
  ultimately
  show ?thesis by auto
qed

```

```

lemma witness_funtype:
  assumes  $a \in A$ 
     $\forall X[M]. X \neq 0 \wedge X \subseteq A \longrightarrow s'X \in A$ 
     $\forall y \in A. \{x \in A. \langle y, x \rangle \in R\} \neq 0$ 
     $M(A) \ M(a) \ M(s) \ M(R)$ 
  shows  $(\lambda n \in \text{nat}. \text{dc\_witness}(n, A, a, s, R)) \in \text{nat} \rightarrow A$  (is  $?f \in \_ \rightarrow \_$ )
proof -
  have  $?f \in \text{nat} \rightarrow \{\text{dc\_witness}(n, A, a, s, R). n \in \text{nat}\}$  (is  $\_ \in \_ \rightarrow ?B$ )
    using lam_funtype assms by simp
  then
  have  $?B \subseteq A$ 
    using witness_into_A assms by auto
  with  $\langle ?f \in \_ \rangle$ 
  show ?thesis
    using fun_weaken_type
    by simp
qed

```

```

lemma witness_to_fun:
  assumes  $a \in A$ 
     $\forall X[M]. X \neq 0 \wedge X \subseteq A \longrightarrow s'X \in A$ 
     $\forall y \in A. \{x \in A. \langle y, x \rangle \in R\} \neq 0$ 

```

```

    M(A) M(a) M(s) M(R)
  shows  $\exists f \in \text{nat} \rightarrow A. \forall n \in \text{nat}. f' n = \text{dc\_witness}(n, A, a, s, R)$ 
  using assms beXI[of _  $\lambda n \in \text{nat}. \text{dc\_witness}(n, A, a, s, R)$ ] witness_funtype
  by simp

end — M_DC

locale M_library_DC = M_library + M_DC
begin

lemma AC_M_func:
  assumes  $\bigwedge x. x \in A \implies (\exists y. y \in x) M(A)$ 
  shows  $\exists f \in A \rightarrow^M \bigcup(A). \forall x \in A. f' x \in x$ 
proof -
  from  $\langle M(A) \rangle$ 
  interpret mpiA: M_Pi_assumptions _ A  $\lambda x. x$ 
    using Pi_replacement Pi_separation lam_replacement_identity
    lam_replacement_Sigfun[THEN lam_replacement_imp_strong_replacement]
    by unfold_locales (simp_all add:transM[of _ A])
  from  $\langle M(A) \rangle$ 
  interpret mpic_A: M_Pi_assumptions_choice _ A  $\lambda x. x$ 
    apply unfold_locales
    using lam_replacement_constant lam_replacement_identity
    lam_replacement_identity[THEN lam_replacement_imp_strong_replacement]
    lam_replacement_minimum[THEN [5] lam_replacement_hcomp2]
    unfolding lam_replacement_def[symmetric]
    by auto
  from  $\langle M(A) \rangle$ 
  interpret mpi2: M_Pi_assumptions2 _ A  $\lambda_. \bigcup A \lambda x. x$ 
    using Pi_replacement Pi_separation lam_replacement_constant
    lam_replacement_Sigfun[THEN lam_replacement_imp_strong_replacement,
    of  $\lambda_. \bigcup A$ ] Pi_replacement1[of _  $\bigcup A$ ] transM[of _ A]
    by unfold_locales auto
  from assms
  show ?thesis
    using mpi2.Pi_rel_type apply_type mpiA.mem_Pi_rel_abs mpi2.mem_Pi_rel_abs
    function_space_rel_char
    by (rule_tac mpic_A.AC_Pi_rel[THEN rexE], simp, rule_tac x=x in beXI)
    (auto, rule_tac C= $\lambda x. x$  in Pi_type, auto)
qed

lemma non_empty_family: [ $0 \notin A; x \in A$ ] ==>  $\exists y. y \in x$ 
by (subgoal_tac  $x \neq 0$ , blast+)

lemma AC_M_func0:  $0 \notin A \implies M(A) \implies \exists f \in A \rightarrow^M \bigcup(A). \forall x \in A. f' x \in x$ 
by (rule AC_M_func) (simp_all add: non_empty_family)

lemma AC_M_func_Pow_rel:

```

```

assumes  $M(C)$ 
shows  $\exists f \in (Pow^M(C) - \{0\}) \rightarrow^M C. \forall x \in Pow^M(C) - \{0\}. f'x \in x$ 
proof -
  have  $0 \notin Pow^M(C) - \{0\}$  by simp
  with assms
  obtain  $f$  where  $f \in (Pow^M(C) - \{0\}) \rightarrow^M \bigcup (Pow^M(C) - \{0\}) \forall x \in Pow^M(C) - \{0\}. f'x \in x$ 
    using  $AC\_M\_func0[OF \langle 0 \notin \_ \rangle]$  by auto
  moreover
  have  $x \subseteq C$  if  $x \in Pow^M(C) - \{0\}$  for  $x$ 
    using that Pow_rel_char assms
    by auto
  moreover
  have  $\bigcup (Pow^M(C) - \{0\}) \subseteq C$ 
    using assms Pow_rel_char by auto
  ultimately
  show ?thesis
    using assms function_space_rel_char
    by (rule_tac bexI, auto, rule_tac Pi_weaken_type, simp_all)
qed

theorem pointed_DC:
  assumes  $\forall x \in A. \exists y \in A. \langle x, y \rangle \in R \ M(A) \ M(R)$ 
  shows  $\forall a \in A. \exists f \in nat \rightarrow^M A. f'0 = a \wedge (\forall n \in nat. \langle f'n, f'succ(n) \rangle \in R)$ 
proof -
  have  $0: \forall y \in A. \{x \in A. \langle y, x \rangle \in R\} \neq 0$ 
    using assms by auto
  from  $\langle M(A) \rangle$ 
  obtain  $g$  where  $1: g \in Pow^M(A) - \{0\} \rightarrow A \forall X[M]. X \neq 0 \wedge X \subseteq A \longrightarrow g'X \in X$ 
    using  $M(g)$ 
    using  $AC\_M\_func\_Pow\_rel[of A] \ mem\_Pow\_rel\_abs$ 
     $function\_space\_rel\_char[of Pow^M(A) - \{0\} A]$  by auto
  then
  have  $2: \forall X[M]. X \neq 0 \wedge X \subseteq A \longrightarrow g'X \in A$  by auto
  {
    fix  $a$ 
    assume  $a \in A$ 
    let  $?f = \lambda n \in nat. dc\_witness(n, A, a, g, R)$ 
    note  $\langle a \in A \rangle$ 
    moreover from this
    have  $f'0: ?f'0 = a$  by simp
    moreover
    note  $\langle a \in A \rangle \ \langle M(g) \rangle \ \langle M(A) \rangle \ \langle M(R) \rangle$ 
    moreover from calculation
    have  $\langle ?f' n, ?f' succ(n) \rangle \in R$  if  $n \in nat$  for  $n$ 
      using witness_related[OF \langle a \in A \rangle _ 0, of g n] 1 that
      by (auto dest:transM)
    ultimately

```

```

have  $\exists f[M]. f \in \text{nat} \rightarrow A \wedge f \text{ ' } 0 = a \wedge (\forall n \in \text{nat}. \langle f \text{ ' } n, f \text{ ' } \text{succ}(n) \rangle \in R)$ 
  using 0 1  $\langle a \in \_ \rangle$ 
  by (rule_tac  $x = (\lambda n \in \omega. \text{dc\_witness}(n, A, a, g, R))$  in rexI)
    (simp_all, rule_tac witness_funtype,
      auto intro!: Lambda_dc_witness_closed dest:transM)
}
with  $\langle M(A) \rangle$ 
show ?thesis using function_space_rel_char by auto
qed

lemma aux_DC_on_AxNat2 :  $\forall x \in A \times \text{nat}. \exists y \in A. \langle x, \langle y, \text{succ}(\text{snd}(x)) \rangle \rangle \in R \implies$ 
 $\forall x \in A \times \text{nat}. \exists y \in A \times \text{nat}. \langle x, y \rangle \in \{ \langle a, b \rangle \in R. \text{snd}(b) = \text{succ}(\text{snd}(a)) \}$ 
  by (rule ballI, erule_tac  $x = x$  in ballE, simp_all)

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

corollary DC_on_A_x_nat :
  assumes  $(\forall x \in A \times \text{nat}. \exists y \in A. \langle x, \langle y, \text{succ}(\text{snd}(x)) \rangle \rangle \in R) \ a \in A \ M(A) \ M(R)$ 
  shows  $\exists f \in \text{nat} \rightarrow^M A. f \text{ ' } 0 = a \wedge (\forall n \in \text{nat}. \langle \langle f \text{ ' } n, n \rangle, \langle f \text{ ' } \text{succ}(n), \text{succ}(n) \rangle \rangle \in R)$  (is
 $\exists x \in \_. ?P(x)$ )
proof -
  let ?R' =  $\{ \langle a, b \rangle \in R. \text{snd}(b) = \text{succ}(\text{snd}(a)) \}$ 
  from assms(1)
  have  $\forall x \in A \times \text{nat}. \exists y \in A \times \text{nat}. \langle x, y \rangle \in ?R'$ 
    using aux_DC_on_AxNat2 by simp
  moreover
  note  $\langle a \in \_ \rangle \ \langle M(A) \rangle$ 
  moreover from this
  have  $M(A \times \omega)$  by simp
  have lam_replacement(M,  $\lambda x. \text{succ}(\text{snd}(\text{fst}(x)))$ )
    using lam_replacement_fst lam_replacement_snd lam_replacement_hcomp
      lam_replacement_hcomp[of  $\_ \lambda x. \text{succ}(\text{snd}(x))$ ]
      lam_replacement_succ by simp
  with  $\langle M(R) \rangle$ 
  have  $M(?R')$ 
    using separation_eq lam_replacement_fst lam_replacement_snd
      lam_replacement_succ lam_replacement_hcomp lam_replacement_identity
      unfolding split_def by simp
  ultimately
  obtain f where
     $F : f \in \text{nat} \rightarrow^M A \times \omega \ f \text{ ' } 0 = \langle a, 0 \rangle \ \forall n \in \text{nat}. \langle f \text{ ' } n, f \text{ ' } \text{succ}(n) \rangle \in ?R'$ 
    using pointed_DC[of  $A \times \text{nat} \ ?R'$ ] by blast
  with  $\langle M(A) \rangle$ 
  have  $M(f)$  using function_space_rel_char by simp
  then
  have  $M(\lambda x \in \text{nat}. \text{fst}(f \text{ ' } x))$  (is  $M(?f)$ )
    using lam_replacement_fst lam_replacement_hcomp
      lam_replacement_constant lam_replacement_identity

```

```

    lam_replacement_apply
  by (rule_tac lam_replacement_iff_lam_closed[THEN iffD1, rule_format])
    auto
with F ⟨M(A)⟩
have ?f∈nat→M A ?f ‘ 0 = a using function_space_rel_char by auto
have 1:n∈ nat ⇒ f’n = ⟨?f’n, n⟩ for n
proof(induct n set:nat)
  case 0
  then show ?case using F by simp
next
  case (succ x)
  with ⟨M(A)⟩
  have ⟨f’x, f’succ(x)⟩ ∈ ?R' f’x ∈ A×nat f’succ(x)∈A×nat
    using F function_space_rel_char[of ω A×ω] by auto
  then
  have snd(f’succ(x)) = succ(snd(f’x)) by simp
  with succ ⟨f’x∈_⟩
  show ?case using infer_snd[OF ⟨f’succ(⟦_⟧)∈_⟩] by auto
qed
have ⟨⟨?f’n,n⟩,⟨?f’succ(n),succ(n)⟩⟩ ∈ R if n∈nat for n
  using that 1[of succ(n)] 1[OF ⟨n∈_⟩] F(3) by simp
with ⟨f’0=⟨a,0⟩⟩
show ?thesis
  using rev_bexI[OF ⟨?f∈_⟩] by simp
qed

```

lemma *aux_sequence_DC* :

```

  assumes ∀ x∈A. ∀ n∈nat. ∃ y∈A. ⟨x,y⟩ ∈ S’n
  R={⟨⟨x,n⟩,⟨y,m⟩⟩ ∈ (A×nat)×(A×nat). ⟨x,y⟩∈S’m }
  shows ∀ x∈A×nat . ∃ y∈A. ⟨x,⟨y,succ(snd(x))⟩⟩ ∈ R
  using assms Pairfst_snd_eq by auto

```

lemma *aux_sequence_DC2* : ∀ x∈A. ∀ n∈nat. ∃ y∈A. ⟨x,y⟩ ∈ S’n ⇒

```

  ∀ x∈A×nat. ∃ y∈A. ⟨x,⟨y,succ(snd(x))⟩⟩ ∈ {⟨⟨x,n⟩,⟨y,m⟩⟩∈(A×nat)×(A×nat).
  ⟨x,y⟩∈S’m }
  by auto

```

lemma *sequence_DC*:

```

  assumes ∀ x∈A. ∀ n∈nat. ∃ y∈A. ⟨x,y⟩ ∈ S’n M(A) M(S)
  shows ∀ a∈A. (∃ f ∈ nat→M A. f’0 = a ∧ (∀ n ∈ nat. ⟨f’n,f’succ(n)⟩∈S’succ(n)))
proof -
  from ⟨M(S)⟩
  have lam_replacement(M, λx. S ‘ snd(snd(x)))
    using lam_replacement_snd lam_replacement_hcomp
    lam_replacement_hcomp[of _ λx. S’snd(x)] lam_replacement_apply by simp
  with assms
  have M({x ∈ (A × ω) × A × ω . (λ⟨⟨x,n⟩,y,m⟩. ⟨x, y⟩ ∈ S ‘ m)(x)})
    using lam_replacementfst lam_replacement_snd
    lam_replacement_Pair[THEN [5] lam_replacement_hcomp2,

```

```

      of  $\lambda x. fst(fst(x)) \lambda x. fst(snd(x))$ , THEN [2] separation_in,
      of  $\lambda x. S \text{ ' } snd(snd(x))$ ] lam_replacement_apply[of S]
      lam_replacement_hcomp unfolding split_def by simp
with assms
show ?thesis
      by (rule_tac ballI) (drule aux_sequence_DC2, drule DC_on_A_x_nat, auto)
qed

end — M_library_DC

end

```

24 Cohen forcing notions

```

theory Partial_Functions_Relative
imports
  FiniteFun_Relative
  Cardinal_Library_Relative
begin

```

definition

```

  Fn :: [i,i,i]  $\Rightarrow$  i where
  Fn( $\kappa$ ,I,J)  $\equiv \bigcup \{y . d \in Pow(I), y=(d \rightarrow J) \wedge d \prec \kappa\}$ 

```

lemma *domain_function_lepoll* :

```

  assumes function(r)
  shows domain(r)  $\lesssim$  r

```

proof -

```

  let ?f =  $\lambda x \in domain(r) . \langle x, THE\ y . \langle x, y \rangle \in r \rangle$ 
  have  $1: \bigwedge x. x \in domain(r) \implies \exists !y. \langle x, y \rangle \in r$ 
    using assms unfolding domain_def function_def by auto
  then
  have ?f  $\in inj(domain(r), r)$ 
    using theI[OF 1]
    by(rule_tac lam_injective,auto)
  then
  show ?thesis unfolding lepoll_def
    by force

```

qed

lemma *function_lepoll*:

```

  assumes  $r: d \rightarrow J$ 
  shows  $r \lesssim d$ 

```

proof -

```

  let ?f =  $\lambda x \in r . fst(x)$ 
  note assms Pi_iff[THEN iffD1,OF assms]
  moreover from this
  have  $1: \bigwedge x. x \in domain(r) \implies \exists !y. \langle x, y \rangle \in r$ 
    unfolding function_def by auto

```

```

moreover from calculation
have (THE  $u . \langle \text{fst}(x), u \rangle \in r = \text{snd}(x)$  if  $x \in r$  for  $x$ 
  using that subsetD[of  $r \times J$   $x$ ] theI[OF 1]
  by(auto, rule_tac the_equality2[OF 1], auto)
moreover from calculation
have  $\bigwedge x. x \in r \implies \langle \text{fst}(x), \text{THE } y . \langle \text{fst}(x), y \rangle \in r \rangle = x$ 
  by auto
ultimately
have  $?f \in \text{inj}(r, d)$ 
  by(rule_tac  $d = \lambda u . \langle u, \text{THE } y . \langle u, y \rangle \in r \rangle$  in lam_injective, force, simp)
then
show ?thesis
  unfolding lepoll_def
  by auto
qed

lemma function_eqpoll :
  assumes  $r : d \rightarrow J$ 
  shows  $r \approx d$ 
  using assms domain_of_fun domain_function_lepoll Pi_iff[THEN iffD1, OF assms]
  eqpollI[OF function_lepoll[OF assms]] subset_imp_lepoll
  by force

lemma Fn_char :  $\text{Fn}(\kappa, I, J) = \{f \in \text{Pow}(I \times J) . \text{function}(f) \wedge f \prec \kappa\}$  (is  $?L = ?R$ )
proof (intro equalityI subsetI)
  fix  $x$ 
  assume  $x \in ?R$ 
  moreover from this
  have  $\text{domain}(x) \in \text{Pow}(I)$   $\text{domain}(x) \lesssim x \prec \kappa$ 
    using domain_function_lepoll
    by auto
  ultimately
  show  $x \in ?L$ 
    unfolding Fn_def
    using lesspoll_trans1 Pi_iff
    by (auto, rule_tac rev_bexI[of  $\text{domain}(x) \rightarrow J$ ], auto)
next
  fix  $x$ 
  assume  $x \in ?L$ 
  then
  obtain  $d$  where  $x : d \rightarrow J$   $d \in \text{Pow}(I)$   $d \prec \kappa$ 
    unfolding Fn_def
    by auto
  moreover from this
  have  $x \prec \kappa$ 
    using function_lepoll[THEN lesspoll_trans1] by auto
  moreover from calculation

```



```

have  $x \in \text{Pow}(I \times J)$  function( $x$ )
  using  $Pi\_iff$  by auto
ultimately
show  $x \in ?R$  by simp
qed

```

```

lemma zero_in_Fn:
  assumes  $0 < \kappa$ 
  shows  $0 \in \text{Fn}(\kappa, I, J)$ 
  using lt_Card_imp_lespoll assms zero_lespoll
  unfolding Fndef
  by (simp, rule_tac  $x=0 \rightarrow J$  in bexI, simp)
  (rule ReplaceI[of  $\_ 0$ ], simp_all)

```

```

lemma Fndef_nat_eq_FiniteFun:  $\text{Fn}(\text{nat}, I, J) = I -||> J$ 
proof (intro equalityI subsetI)
  fix  $x$ 
  assume  $x \in I -||> J$ 
  then
  show  $x \in \text{Fn}(\text{nat}, I, J)$ 
  proof (induct)
    case emptyI
    then
    show ?case
      using zero_in_Fn ltI
      by simp
  next
    case (consI  $a\ b\ h$ )
    then
    obtain  $d$  where  $h: d \rightarrow J\ d \prec_{\text{nat}} d \subseteq I$ 
      unfolding Fndef by auto
    moreover from this
    have Finite( $d$ )
      using lespoll_nat_is_Finite by simp
    ultimately
    have  $h : d -||> J$ 
      using fun_FiniteFunI Finite_into_Fin by blast
    note  $\langle h: d \rightarrow J \rangle$ 
    moreover from this
    have  $\text{domain}(\text{cons}(\langle a, b \rangle, h)) = \text{cons}(a, d)$  (is  $\text{domain}(?h) = ?d$ )
      and  $\text{domain}(h) = d$ 
      using domain_of_fun by simp_all
    moreover
    note consI
    moreover from calculation
    have  $\text{cons}(\langle a, b \rangle, h) \in \text{cons}(a, d) \rightarrow J$ 
      using fun_extend3 by simp
    moreover from  $\langle \text{Finite}(d) \rangle$ 
    have Finite( $\text{cons}(a, d)$ ) by simp

```

```

    moreover from this
    have  $\text{cons}(a,d) \prec \text{nat}$  using Finite_imp_lespoll_nat by simp
    ultimately
    show ?case
      unfolding Fn_def
      by (simp, rule_tac  $x = ?d \rightarrow J$  in bexI)
        (force dest: app_fun) +
  qed
next
fix  $x$ 
assume  $x \in \text{Fn}(\text{nat}, I, J)$ 
then
obtain  $d$  where  $x: d \rightarrow J$   $d \in \text{Pow}(I)$   $d \prec \text{nat}$ 
  unfolding Fn_def
  by auto
moreover from this
have Finite( $d$ )
  using lespoll_nat_is_Finite by simp
moreover from calculation
have  $d \in \text{Fin}(I)$ 
  using Finite_into_Fin[of d] Fin_mono by auto
ultimately
show  $x \in I \dashv|| > J$  using fun_FiniteFunI FiniteFun_mono by blast
qed

lemma Fn_nat_subset_Pow:  $\text{Fn}(\kappa, I, J) \subseteq \text{Pow}(I \times J)$ 
  using Fn_char by auto

lemma FnI:
  assumes  $p : d \rightarrow J$   $d \subseteq I$   $d \prec \kappa$ 
  shows  $p \in \text{Fn}(\kappa, I, J)$ 
  using assms
  unfolding Fn_def by auto

lemma FnD[dest]:
  assumes  $p \in \text{Fn}(\kappa, I, J)$ 
  shows  $\exists d. p : d \rightarrow J \wedge d \subseteq I \wedge d \prec \kappa$ 
  using assms
  unfolding Fn_def by auto

lemma Fn_is_function:  $p \in \text{Fn}(\kappa, I, J) \implies \text{function}(p)$ 
  unfolding Fn_def using fun_is_function by auto

lemma Fn_succ:
  assumes Ord( $\kappa$ )
  shows  $\text{Fn}(\text{csucc}(\kappa), I, J) = \bigcup \{y . d \in \text{Pow}(I), y = (d \rightarrow J) \wedge d \lesssim \kappa\}$ 
  using assms
  unfolding Fn_def using lespoll_succ by (simp)

```

definition

$FnleR :: i \Rightarrow i \Rightarrow o$ (**infixl** $\langle \supseteq \rangle$ 50) **where**
 $f \supseteq g \equiv g \subseteq f$

lemma $FnleR_iff_subset$ [*iff*]: $f \supseteq g \longleftrightarrow g \subseteq f$
unfolding $FnleR_def$..

definition

$Fnlrel :: i \Rightarrow i$ **where**
 $Fnlrel(A) \equiv Rrel(\lambda x y. x \supseteq y, A)$

definition

$Fnle :: [i, i, i] \Rightarrow i$ **where**
 $Fnle(\kappa, I, J) \equiv Fnlrel(Fn(\kappa, I, J))$

lemma $FnleI$ [*intro*]:

assumes $p \in Fn(\kappa, I, J)$ $q \in Fn(\kappa, I, J)$ $p \supseteq q$
shows $\langle p, q \rangle \in Fnle(\kappa, I, J)$
using *assms* **unfolding** $Fnlrel_def$ $Fnle_def$ $FnleR_def$ $Rrel_def$
by *auto*

lemma $FnleD$ [*dest*]:

assumes $\langle p, q \rangle \in Fnle(\kappa, I, J)$
shows $p \in Fn(\kappa, I, J)$ $q \in Fn(\kappa, I, J)$ $p \supseteq q$
using *assms* **unfolding** $Fnlrel_def$ $Fnle_def$ $FnleR_def$ $Rrel_def$
by *auto*

definition $PFun_Space_Rel :: [i, i \Rightarrow o, i] \Rightarrow i$ ($_ \multimap _$)
where $A \multimap^M B \equiv \{f \in Pow(A \times B) . M(f) \wedge function(f)\}$

lemma (**in** $M_library$) $PFun_Space_subset_Powrel$:

assumes $M(A)$ $M(B)$
shows $A \multimap^M B = \{f \in Pow^M(A \times B) . function(f)\}$
using Pow_rel_char *assms*
unfolding $PFun_Space_Rel_def$
by *auto*

lemma (**in** $M_library$) $PFun_Space_closed$:

assumes $M(A)$ $M(B)$
shows $M(A \multimap^M B)$
using *assms* $PFun_Space_subset_Powrel$ *separation_is_function*
by *auto*

lemma $Un_filter_fun_space_closed$:

assumes $G \subseteq I \rightarrow J \wedge f g . f \in G \implies g \in G \implies \exists d \in I \rightarrow J . d \supseteq f \wedge d \supseteq g$
shows $\bigcup G \in Pow(I \times J)$ $function(\bigcup G)$

proof -

from *assms*
show $\bigcup G \in Pow(I \times J)$

```

    using Union_Pow_iff
    unfolding Pi_def
    by auto
next
show function( $\bigcup G$ )
  unfolding function_def
proof(auto)
  fix B B' x y y'
  assume B  $\in G$   $\langle x, y \rangle \in B$  B'  $\in G$   $\langle x, y \rangle \in B'$ 
  moreover from assms this
  have B  $\in I \rightarrow J$  B'  $\in I \rightarrow J$ 
  by auto
  moreover from calculation assms(2)[of B B']
  obtain d where d  $\supseteq B$  d  $\supseteq B'$  d  $\in I \rightarrow J$   $\langle x, y \rangle \in d$   $\langle x, y \rangle \in d$ 
    using subsetD[OF  $\langle G \subseteq \_ \rangle$ ]
    by auto
  then
  show y=y'
    using fun_is_function[OF d  $\in \_$ ]
    unfolding function_def
    by force
qed
qed

lemma Un_filter_is_fun :
  assumes G  $\subseteq I \rightarrow J$   $\bigwedge f g . f \in G \implies g \in G \implies \exists d \in I \rightarrow J . d \supseteq f \wedge d \supseteq g$  G  $\neq 0$ 
  shows  $\bigcup G \in I \rightarrow J$ 
  using assms Un_filter_fun_space_closed Pi_iff
proof(simp_all)
  show I  $\subseteq$  domain( $\bigcup G$ )
  proof -
    from  $\langle G \neq 0 \rangle$ 
    obtain f where f  $\subseteq \bigcup G$  f  $\in G$ 
    by auto
    with  $\langle G \subseteq \_ \rangle$ 
    have f  $\in I \rightarrow J$  by auto
    then
    show ?thesis
      using subset_trans[OF domain_mono[OF f  $\subseteq \bigcup G$ ], of I]
      unfolding Pi_def by auto
  qed
qed

context M_cardinals
begin

lemma mem_function_space_relD:
  assumes f  $\in$  function_space_rel(M,A,y) M(A) M(y)
  shows f  $\in A \rightarrow y$  and M(f)

```

```

using assms function_space_rel_char by simp_all

lemma pfunI :
  assumes  $C \subseteq A \rightarrow^M B$   $M(C) \subseteq M(B)$ 
  shows  $f \in A \rightarrow^M B$ 
proof -
  from assms
  have  $f \in C \rightarrow B$   $M(f)$ 
    using mem_function_space_relD
    by simp_all
  with assms
  show ?thesis
    using Pi_iff
    unfolding PFun_Space_Rel_def
    by auto
qed

lemma zero_in_PFun_rel:
  assumes  $M(I) \subseteq M(J)$ 
  shows  $0 \in I \rightarrow^M J$ 
  using pfunI[of 0] nonempty mem_function_space_rel_abs assms
  by simp

lemma pfun_subsetI :
  assumes  $f \in A \rightarrow^M B$   $g \subseteq f$   $M(g)$ 
  shows  $g \in A \rightarrow^M B$ 
using assms function_subset
  unfolding PFun_Space_Rel_def
  by auto

lemma pfun_is_function :
   $f \in A \rightarrow^M B \implies \text{function}(f)$ 
  unfolding PFun_Space_Rel_def by simp

lemma pfun_Un_filter_closed:
  assumes  $G \subseteq I \rightarrow^M J$   $f \in G \implies g \in G \implies \exists d \in I \rightarrow^M J. d \supseteq f \wedge d \supseteq g$ 
  shows  $\bigcup G \in \text{Pow}(I \times J)$   $\text{function}(\bigcup G)$ 
proof -
  from assms
  show  $\bigcup G \in \text{Pow}(I \times J)$ 
    using Union_Pow_iff
    unfolding PFun_Space_Rel_def
    by auto
next
  show  $\text{function}(\bigcup G)$ 
    unfolding function_def
  proof(auto)
    fix  $B \subseteq B'$   $x \in B$   $y' \in B'$ 
    assume  $B \in G$   $\langle x, y \rangle \in B$   $B' \in G$   $\langle x, y' \rangle \in B'$ 

```

```

moreover from calculation assms
obtain  $d$  where  $d \in I \multimap^M J$  function( $d$ )  $\langle x, y \rangle \in d \ \langle x, y' \rangle \in d$ 
  using pfun_is_function
  by force
ultimately
show  $y=y'$ 
  unfolding function_def
  by auto
qed
qed

lemma pfun_Un_filter_closed'':
  assumes  $G \subseteq I \multimap^M J \wedge f g . f \in G \implies g \in G \implies \exists d \in G . d \supseteq f \wedge d \supseteq g$ 
  shows  $\bigcup G \in Pow(I \times J)$  function( $\bigcup G$ )
proof -
  from assms
  have  $\bigwedge f g . f \in G \implies g \in G \implies \exists d \in I \multimap^M J . d \supseteq f \wedge d \supseteq g$ 
    using subsetD[OF assms(1), THEN [2] bexI]
    by force
  then
  show  $\bigcup G \in Pow(I \times J)$  function( $\bigcup G$ )
    using assms pfun_Un_filter_closed
    unfolding PFun_Space_Rel_def
    by auto
qed

lemma pfun_Un_filter_closed':
  assumes  $G \subseteq I \multimap^M J \wedge f g . f \in G \implies g \in G \implies \exists d \in G . d \supseteq f \wedge d \supseteq g$   $M(G)$ 
  shows  $\bigcup G \in I \multimap^M J$ 
  using assms pfun_Un_filter_closed''
  unfolding PFun_Space_Rel_def
  by auto

lemma pfunD :
  assumes  $f \in A \multimap^M B$ 
  shows  $\exists C[M]. C \subseteq A \wedge f \in C \rightarrow B$ 
proof -
  note assms
  moreover from this
  have  $f \in Pow(A \times B)$  function( $f$ )  $M(f)$ 
  unfolding PFun_Space_Rel_def
  by simp_all
  moreover from this
  have  $domain(f) \subseteq A$   $f \in domain(f) \rightarrow B$   $M(domain(f))$ 
    using assms Pow_iff[of f A × B] domain_subset Pi_iff
    by auto
  ultimately
  show ?thesis by auto
qed

```

```

lemma pfunD_closed :
  assumes  $f \in A \multimap^M B$ 
  shows  $M(f)$ 
  using assms
  unfolding PFun_Space_Rel_def by simp

lemma pfun_singletonI :
  assumes  $x \in A \ b \in B \ M(A) \ M(B)$ 
  shows  $\{\langle x, b \rangle\} \in A \multimap^M B$ 
  using assms transM[of x A] transM[of b B]
  unfolding PFun_Space_Rel_def function_def
  by auto

lemma pfun_unionI :
  assumes  $f \in A \multimap^M B \ g \in A \multimap^M B \ \text{domain}(f) \cap \text{domain}(g) = 0$ 
  shows  $f \cup g \in A \multimap^M B$ 
  using assms
  unfolding PFun_Space_Rel_def function_def
  by blast

lemma (in M_library) pfun_restrict_eq_imp_compat:
  assumes  $f \in I \multimap^M J \ g \in I \multimap^M J \ M(J)$ 
  shows  $\text{restrict}(f, \text{domain}(f) \cap \text{domain}(g)) = \text{restrict}(g, \text{domain}(f) \cap \text{domain}(g))$ 
  using  $f \cup g \in I \multimap^M J$ 
proof -
  note assms
  moreover from this
  obtain  $C \ D$  where  $f : C \rightarrow J \ C \subseteq I \ D \subseteq I \ M(C) \ M(D) \ g : D \rightarrow J$ 
  using pfunD[of f] pfunD[of g] by force
  moreover from calculation
  have  $f \cup g \in C \cup D \rightarrow J$ 
  using restrict_eq_imp_Un_into_Pi'[OF <f ∈ C → > <g ∈ D → >]
  by auto
  ultimately
  show ?thesis
  using pfunI[of C ∪ D _ f ∪ g] Un_subset_iff pfunD_closed function_space_rel_char
  by auto
qed

lemma FiniteFun_pfunI :
  assumes  $f \in A \multimap^M B \ M(A) \ M(B)$ 
  shows  $f \in A \multimap^M B$ 
  using assms(1)
proof(induct)
case emptyI
  then
  show ?case
  using assms nonempty mem_function_space_rel_abs pfunI[OF empty_subsetI,

```

```

of 0]
  by simp
next
  case (consI a b h)
  note consI
  moreover from this
  have  $M(a) \ M(b) \ M(h) \ \text{domain}(h) \subseteq A$ 
    using  $\text{transM}[OF \ \_ \ \langle M(A) \rangle] \ \text{transM}[OF \ \_ \ \langle M(B) \rangle]$ 
       $\text{FinD}$ 
       $\text{FiniteFun\_domain\_Fin}$ 
       $\text{pfunD\_closed}$ 
    by simp_all
  moreover from calculation
  have  $\{a\} \cup \text{domain}(h) \subseteq A \ M(\{a\} \cup \text{domain}(h)) \ M(\text{cons}(\langle a, b \rangle, h)) \ \text{domain}(\text{cons}(\langle a, b \rangle, h))$ 
    =  $\{a\} \cup \text{domain}(h)$ 
    by auto
  moreover from calculation
  have  $\text{cons}(\langle a, b \rangle, h) \in \{a\} \cup \text{domain}(h) \rightarrow B$ 
    using  $\text{FiniteFun\_is\_fun}[OF \ \text{FiniteFun.consI, of } a \ A \ b \ B \ h]$ 
    by auto
  ultimately
  show  $\text{cons}(\langle a, b \rangle, h) \in A \multimap^M B$ 
    using  $\text{assms} \ \text{mem\_function\_space\_rel\_abs} \ \text{pfunI}$ 
    by simp_all
qed

lemma  $\text{PFun\_FiniteFunI}$  :
  assumes  $f \in A \multimap^M B \ \text{Finite}(f)$ 
  shows  $f \in A \multimap B$ 
proof -
  from  $\text{assms}$ 
  have  $f \in \text{Fin}(A \times B) \ \text{function}(f)$ 
    using  $\text{Finite\_Fin} \ \text{Pow\_iff}$ 
    unfolding  $\text{PFun\_Space\_Rel\_def}$ 
    by auto
  then
  show ?thesis
    using  $\text{FiniteFunI}$  by simp
qed

end

```

definition

$\text{Fn_rel} :: [i \Rightarrow o, i, i, i] \Rightarrow i \ (\langle \text{Fn}'(_, _, _) \rangle)$ **where**

$\text{Fn_rel}(M, \kappa, I, J) \equiv \{f \in I \multimap^M J \ . \ |f|^M \prec^M \kappa\}$

context $M_library$

begin

lemma $Fn_rel_subset_PFun_rel : Fn^M(\kappa, I, J) \subseteq I \multimap^M J$
unfolding Fn_rel_def **by** *auto*

lemma $Fn_relI[intro]$:
assumes $f : d \rightarrow J \ d \subseteq I \ |f|^M \prec^M \kappa \ M(d) \ M(J) \ M(f)$
shows $f \in Fn_rel(M, \kappa, I, J)$
using *assms pfunI mem_function_space_rel_abs*
unfolding Fn_rel_def
by *auto*

lemma $Fn_relD[dest]$:
assumes $p \in Fn_rel(M, \kappa, I, J)$
shows $\exists C[M]. \ C \subseteq I \wedge p : C \rightarrow J \wedge |p|^M \prec^M \kappa$
using *assms pfunD*
unfolding Fn_rel_def
by *simp*

lemma $Fn_rel_is_function$:
assumes $p \in Fn_rel(M, \kappa, I, J)$
shows $function(p) \ M(p) \ |p|^M \prec^M \kappa \ p \in I \multimap^M J$
using *assms*
unfolding $Fn_rel_def \ PFun_Space_Rel_def$ **by** *simp_all*

lemma Fn_rel_mono :
assumes $p \in Fn_rel(M, \kappa, I, J) \ \kappa \prec^M \kappa' \ M(\kappa) \ M(\kappa')$
shows $p \in Fn_rel(M, \kappa', I, J)$
using *assms lesspoll_rel_trans[OF assms(2)] cardinal_rel_closed*
 $Fn_rel_is_function(2)[OF assms(1)]$
unfolding Fn_rel_def
by *simp*

lemma Fn_rel_mono' :
assumes $p \in Fn_rel(M, \kappa, I, J) \ \kappa \lesssim^M \kappa' \ M(\kappa) \ M(\kappa')$
shows $p \in Fn_rel(M, \kappa', I, J)$
proof -
note *assms*
then
consider $\kappa \prec^M \kappa' \mid \kappa \approx^M \kappa'$
using *lepoll_rel_iff_leqpoll_rel*
by *auto*
then
show *?thesis*
proof(*cases*)
case 1
with *assms* **show** *?thesis* **using** Fn_rel_mono **by** *simp*
next
case 2
then **show** *?thesis*

```

    using assms cardinal_rel_closed Fn_rel_is_function[OF assms(1)]
      lesspoll_rel_eq_trans
  unfolding Fn_rel_def
  by simp
qed
qed

lemma Fn_csucc:
  assumes Ord( $\kappa$ ) M( $\kappa$ )
  shows  $Fn\_rel(M, (\kappa^+)^M, I, J) = \{p \in I \multimap^M J \mid |p|^M \lesssim^M \kappa\}$  (is ?L=?R)
  using assms
proof(intro equalityI)
  show ?L  $\subseteq$  ?R
  proof(intro subsetI)
    fix p
    assume p $\in$ ?L
    then
      have  $|p|^M \prec^M csucc\_rel(M, \kappa) M(p) \ p \in I \multimap^M J$ 
        using Fn_rel_is_function by simp_all
      then
        show p $\in$ ?R
        using assms lesspoll_rel_csucc_rel by simp
    qed
  next
    show ?R  $\subseteq$  ?L
    proof(intro subsetI)
      fix p
      assume p $\in$ ?R
      then
        have  $p \in I \multimap^M J \mid |p|^M \lesssim^M \kappa$ 
          using assms lesspoll_rel_csucc_rel by simp_all
        then
          show p $\in$ ?L
          using assms lesspoll_rel_csucc_rel pfunD_closed
            unfolding Fn_rel_def
            by simp
        qed
    qed
  qed

lemma Finite_imp_lespoll_nat:
  assumes Finite(A)
  shows  $A \prec nat$ 
  using assms subset_imp_lepoll[OF naturals_subset_nat] eq_lepoll_trans
    n_lespoll_nat eq_lespoll_trans
  unfolding Finite_def lesspoll_def by auto

lemma FinD_Finite :
  assumes a $\in$ Fin(A)

```

```

shows Finite(a)
using assms
by(induct,simp_all)

lemma Fn_rel_nat_eq_FiniteFun:
  assumes M(I) M(J)
  shows I -||> J = Fn_rel(M,ω,I,J)
proof(intro equalityI subsetI)
  fix p
  assume p∈I -||> J
  with assms
  have p∈ I  $\rightarrow^M$  J Finite(p)
  using FiniteFun_pfunI FinD_Finite[OF subsetD[OF FiniteFun.dom_subset, OF
    ⟨p∈_⟩]]
  by auto
  moreover from this
  have |p|^M  $\prec^M$  ω
    using Finite_cardinal_rel_in_nat pfunD_closed[of p] n_lespoll_rel_nat
    by simp
  ultimately
  show p∈ Fn_rel(M,ω,I,J)
    unfolding Fn_rel_def by simp
next
  fix p
  assume p∈ Fn_rel(M,ω,I,J)
  then
  have p∈ I  $\rightarrow^M$  J |p|^M  $\prec^M$  ω
    unfolding Fn_rel_def by simp_all
  moreover from this
  have Finite(p)
    using Finite_cardinal_rel_Finite lesspoll_rel_nat_is_Finite_rel pfunD_closed
    cardinal_rel_closed[of p] Finite_cardinal_rel_iff'[THEN iffD1]
    by simp
  ultimately
  show p∈I -||> J
    using PFun_FiniteFunI
    by simp
qed

lemma Fn_nat_abs:
  assumes M(I) M(J)
  shows Fn(nat,I,J) = Fn_rel(M,ω,I,J)
  using assms Fn_rel_nat_eq_FiniteFun Fn_nat_eq_FiniteFun
  by simp
end

lemma (in M_library) Fn_rel_singletonI:
  assumes x ∈ I j ∈ J InfCardM(κ) M(κ) M(I) M(J)
  shows {⟨x,j⟩} ∈ FnM(κ,I,J)

```

```

using assms pfun_singletonI transM[of x ] transM[of j]
      cardinal_rel_singleton
      lt_Card_rel_imp_lesspoll_rel ltI[OF nat_into_InfCard_rel]
      Card_rel_cardinal_rel Card_rel_is_Ord InfCard_rel_is_Card_rel
unfolding Fn_rel_def
by auto

```

definition

```

Fnle_rel :: [i ⇒ o, i, i, i] ⇒ i (⟨Fnle—'(_,_,'_)'⟩) where
Fnle_rel(M, κ, I, J) ≡ Fnlerel(FnM(κ, I, J))

```

abbreviation

```

Fn_r_set :: [i, i, i, i] ⇒ i (⟨Fn—'(_,_,'_)'⟩) where
Fn_r_set(M) ≡ Fn_rel(##M)

```

abbreviation

```

Fnle_r_set :: [i, i, i, i] ⇒ i (⟨Fnle—'(_,_,'_)'⟩) where
Fnle_r_set(M) ≡ Fnle_rel(##M)

```

context *M_library*

begin

lemma *Fnle_relI*[*intro*]:

```

assumes p ∈ Fn_rel(M, κ, I, J) q ∈ Fn_rel(M, κ, I, J) p ⊇ q
shows <p, q> ∈ Fnle_rel(M, κ, I, J)
using assms unfolding Fnlerel_def Fnle_rel_def FnleR_def Rrel_def
by auto

```

lemma *Fnle_relD*[*dest*]:

```

assumes <p, q> ∈ Fnle_rel(M, κ, I, J)
shows p ∈ Fn_rel(M, κ, I, J) q ∈ Fn_rel(M, κ, I, J) p ⊇ q
using assms unfolding Fnlerel_def Fnle_rel_def FnleR_def Rrel_def
by auto

```

end

context *M_library*

begin

lemma *Fn_rel_closed*[*intro*, *simp*]:

```

assumes M(κ) M(I) M(J)
shows M(FnM(κ, I, J))
using assms separation_cardinal_rel_lesspoll_rel PFun_Space_closed
unfolding Fn_rel_def
by auto

```

lemma *Fn_rel_subset_Pow*:

```

assumes M(κ) M(I) M(J)

```

```

shows  $F_n^M(\kappa, I, J) \subseteq \text{Pow}(I \times J)$ 
unfolding  $F_n\_rel\_def$   $PFun\_Space\_Rel\_def$ 
by auto

lemma  $F_n\_rel\_closed[intro, simp]$ :
  assumes  $M(\kappa) \ M(I) \ M(J)$ 
  shows  $M(F_n^M(\kappa, I, J))$ 
  unfolding  $F_n\_rel\_def$   $F_nlerel\_def$   $Rrel\_def$   $F_nleR\_def$ 
  using  $assms$   $supset\_separation$   $F_n\_rel\_closed$ 
  by auto

lemma  $zero\_in\_F_n\_rel$ :
  assumes  $0 < \kappa \ M(\kappa) \ M(I) \ M(J)$ 
  shows  $0 \in F_n^M(\kappa, I, J)$ 
  unfolding  $F_n\_rel\_def$ 
  using  $zero\_in\_PFun\_rel$   $zero\_lesspoll\_rel$   $assms$ 
  by simp

lemma  $zero\_top\_F_n\_rel$ :
  assumes  $p \in F_n^M(\kappa, I, J) \ 0 < \kappa \ M(\kappa) \ M(I) \ M(J)$ 
  shows  $\langle p, 0 \rangle \in F_n^M(\kappa, I, J)$ 
  using  $assms$   $zero\_in\_F_n\_rel$  unfolding  $preorder\_on\_def$   $refl\_def$   $trans\_on\_def$ 
  by auto

lemma  $preorder\_on\_F_nle\_rel$ :
  assumes  $M(\kappa) \ M(I) \ M(J)$ 
  shows  $preorder\_on(F_n^M(\kappa, I, J), F_nle^M(\kappa, I, J))$ 
  unfolding  $preorder\_on\_def$   $refl\_def$   $trans\_on\_def$ 
  by blast

end —  $M\_library$ 

end
theory  $M\_Basic\_No\_Repl$ 
  imports  $ZF\text{-Constructible.Relative}$ 
begin

```

This locale is exactly M_basic without its only replacement instance.

```

locale  $M\_basic\_no\_repl = M\_trivial +$ 
  assumes  $Inter\_separation$ :
     $M(A) \implies separation(M, \lambda x. \forall y[M]. y \in A \longrightarrow x \in y)$ 
  and  $Diff\_separation$ :
     $M(B) \implies separation(M, \lambda x. x \notin B)$ 
  and  $cartprod\_separation$ :
     $[| M(A); M(B) |]$ 
     $\implies separation(M, \lambda z. \exists x[M]. x \in A \ \& \ (\exists y[M]. y \in B \ \& \ pair(M, x, y, z)))$ 
  and  $image\_separation$ :
     $[| M(A); M(r) |]$ 
     $\implies separation(M, \lambda y. \exists p[M]. p \in r \ \& \ (\exists x[M]. x \in A \ \& \ pair(M, x, y, p)))$ 

```

```

and converse_separation:
  M(r) ==> separation(M,
    λz. ∃ p[M]. p ∈ r & (∃ x[M]. ∃ y[M]. pair(M,x,y,p) & pair(M,y,x,z)))
and restrict_separation:
  M(A) ==> separation(M, λz. ∃ x[M]. x ∈ A & (∃ y[M]. pair(M,x,y,z)))
and comp_separation:
  [| M(r); M(s) |]
    ==> separation(M, λxz. ∃ x[M]. ∃ y[M]. ∃ z[M]. ∃ xy[M]. ∃ yz[M].
      pair(M,x,z,xz) & pair(M,x,y,xy) & pair(M,y,z,yz) &
      xy ∈ s & yz ∈ r)
and pred_separation:
  [| M(r); M(x) |] ==> separation(M, λy. ∃ p[M]. p ∈ r & pair(M,y,x,p))
and Memrel_separation:
  separation(M, λz. ∃ x[M]. ∃ y[M]. pair(M,x,y,z) & x ∈ y)
and is_recfun_separation:
  — for well-founded recursion: used to prove is_recfun_equal
  [| M(r); M(f); M(g); M(a); M(b) |]
    ==> separation(M,
      λx. ∃ xa[M]. ∃ xb[M].
        pair(M,x,a,xa) & xa ∈ r & pair(M,x,b,xb) & xb ∈ r &
        (∃ fx[M]. ∃ gx[M]. fun_apply(M,f,x,fx) & fun_apply(M,g,x,gx) &
          fx ≠ gx))
and power_ax:      power_ax(M)

lemma (in M_basic_no_repl) cartprod_iff:
  [| M(A); M(B); M(C) |]
    ==> cartprod(M,A,B,C) <=>
      (∃ p1[M]. ∃ p2[M]. powerset(M,A ∪ B,p1) & powerset(M,p1,p2) &
        C = {z ∈ p2. ∃ x ∈ A. ∃ y ∈ B. z = <x,y>})
apply (simp add: Pair_def cartprod_def, safe)
defer 1
  apply (simp add: powerset_def)
apply blast

Final, difficult case: the left-to-right direction of the theorem.

apply (insert power_ax, simp add: power_ax_def)
apply (frule_tac x=A ∪ B and P=λx. rex(M,Q(x)) for Q in rspec)
apply (blast, clarify)
apply (drule_tac x=z and P=λx. rex(M,Q(x)) for Q in rspec)
apply assumption
apply (blast intro: cartprod_iff_lemma)
done

lemma (in M_basic_no_repl) cartprod_closed_lemma:
  [| M(A); M(B) |] ==> ∃ C[M]. cartprod(M,A,B,C)
apply (simp del: cartprod_abs add: cartprod_iff)
apply (insert power_ax, simp add: power_ax_def)
apply (frule_tac x=A ∪ B and P=λx. rex(M,Q(x)) for Q in rspec)
apply (blast, clarify)

```

```

apply (drule_tac  $x=z$  and  $P=\lambda x. rex(M, Q(x))$  for  $Q$  in rspec, auto)
apply (intro rexI conjI, simp+)
apply (insert cartprod_separation [of A B], simp)
done

```

All the lemmas above are necessary because Powerset is not absolute. I should have used Replacement instead!

```

lemma (in M_basic_no_repl) cartprod_closed [intro, simp]:
  [|  $M(A); M(B)$  |]  $\impl M(A*B)$ 
by (frule cartprod_closed_lemma, assumption, force)

```

```

lemma (in M_basic_no_repl) sum_closed [intro, simp]:
  [|  $M(A); M(B)$  |]  $\impl M(A+B)$ 
by (simp add: sum_def)

```

```

lemma (in M_basic_no_repl) sum_abs [simp]:
  [|  $M(A); M(B); M(Z)$  |]  $\impl is\_sum(M, A, B, Z) \longleftrightarrow (Z = A+B)$ 
by (simp add: is_sum_def sum_def singleton_0 nat_into_M)

```

```

lemma (in M_basic_no_repl) M_converse_iff:
   $M(r) \impl$ 
   $converse(r) =$ 
   $\{z \in \bigcup (\bigcup (r)) * \bigcup (\bigcup (r)).$ 
   $\exists p \in r. \exists x[M]. \exists y[M]. p = \langle x, y \rangle \ \& \ z = \langle y, x \rangle\}$ 
apply (rule equalityI)
prefer 2 apply (blast dest: transM, clarify, simp)
apply (simp add: Pair_def)
apply (blast dest: transM)
done

```

```

lemma (in M_basic_no_repl) converse_closed [intro, simp]:
   $M(r) \impl M(converse(r))$ 
apply (simp add: M_converse_iff)
apply (insert converse_separation [of r], simp)
done

```

```

lemma (in M_basic_no_repl) converse_abs [simp]:
  [|  $M(r); M(z)$  |]  $\impl is\_converse(M, r, z) \longleftrightarrow z = converse(r)$ 
apply (simp add: is_converse_def)
apply (rule iffI)
prefer 2 apply blast
apply (rule M_equalityI)
apply simp
apply (blast dest: transM) +
done

```

24.0.1 image, preimage, domain, range

```

lemma (in M_basic_no_repl) image_closed [intro, simp]:

```

```

    [| M(A); M(r) |] ==> M(r-''A)
  apply (simp add: image_iff_Collect)
  apply (insert image_separation [of A r], simp)
done

```

```

lemma (in M_basic_no_repl) vimage_abs [simp]:
  [| M(r); M(A); M(z) |] ==> pre_image(M,r,A,z) <=> z = r-''A
  apply (simp add: pre_image_def)
  apply (rule iffI)
  apply (blast intro!: equalityI dest: transM, blast)
done

```

```

lemma (in M_basic_no_repl) vimage_closed [intro,simp]:
  [| M(A); M(r) |] ==> M(r-''A)
  by (simp add: vimage_def)

```

24.0.2 Domain, range and field

```

lemma (in M_basic_no_repl) domain_closed [intro,simp]:
  M(r) ==> M(domain(r))
  apply (simp add: domain_eq_vimage)
done

```

```

lemma (in M_basic_no_repl) range_closed [intro,simp]:
  M(r) ==> M(range(r))
  apply (simp add: range_eq_image)
done

```

```

lemma (in M_basic_no_repl) field_abs [simp]:
  [| M(r); M(z) |] ==> is_field(M,r,z) <=> z = field(r)
  by (simp add: is_field_def field_def)

```

```

lemma (in M_basic_no_repl) field_closed [intro,simp]:
  M(r) ==> M(field(r))
  by (simp add: field_def)

```

24.0.3 Relations, functions and application

```

lemma (in M_basic_no_repl) apply_closed [intro,simp]:
  [| M(f); M(a) |] ==> M(f'a)
  by (simp add: apply_def)

```

```

lemma (in M_basic_no_repl) apply_abs [simp]:
  [| M(f); M(x); M(y) |] ==> fun_apply(M,f,x,y) <=> f'x = y
  apply (simp add: fun_apply_def apply_def, blast)
done

```

```

lemma (in M_basic_no_repl) injection_abs [simp]:
  [| M(A); M(f) |] ==> injection(M,A,B,f) <=> f ∈ inj(A,B)
  apply (simp add: injection_def apply_iff inj_def)

```


apply (*blast dest: transM [of _ A]*)
done

lemma (**in** *M_basic_no_repl*) *surjection_abs [simp]*:

$$[| M(A); M(B); M(f) |] ==> \text{surjection}(M,A,B,f) \longleftrightarrow f \in \text{surj}(A,B)$$

by (*simp add: surjection_def surj_def*)

lemma (**in** *M_basic_no_repl*) *bijection_abs [simp]*:

$$[| M(A); M(B); M(f) |] ==> \text{bijection}(M,A,B,f) \longleftrightarrow f \in \text{bij}(A,B)$$

by (*simp add: bijection_def bij_def*)

24.0.4 Composition of relations

lemma (**in** *M_basic_no_repl*) *M_comp_iff*:

$$[| M(r); M(s) |]$$

$$==> r \ O \ s =$$

$$\{xz \in \text{domain}(s) * \text{range}(r).$$

$$\exists x[M]. \exists y[M]. \exists z[M]. xz = \langle x,z \rangle \ \& \ \langle x,y \rangle \in s \ \& \ \langle y,z \rangle \in r\}$$

apply (*simp add: comp_def*)
apply (*rule equalityI*)
apply *clarify*
apply *simp*
apply (*blast dest: transM*)
done

lemma (**in** *M_basic_no_repl*) *comp_closed [intro,simp]*:

$$[| M(r); M(s) |] ==> M(r \ O \ s)$$

apply (*simp add: M_comp_iff*)
apply (*insert comp_separation [of r s], simp*)
done

lemma (**in** *M_basic_no_repl*) *composition_abs [simp]*:

$$[| M(r); M(s); M(t) |] ==> \text{composition}(M,r,s,t) \longleftrightarrow t = r \ O \ s$$

apply *safe*

Proving $\text{composition}(M, r, s, r \ O \ s)$

prefer 2
apply (*simp add: composition_def comp_def*)
apply (*blast dest: transM*)

Opposite implication

apply (*rule M_equalityI*)
apply (*simp add: composition_def comp_def*)
apply (*blast del: allE dest: transM*)
done

no longer needed

lemma (**in** *M_basic_no_repl*) *restriction_is_function*:

$$[| \text{restriction}(M,f,A,z); \text{function}(f); M(f); M(A); M(z) |]$$

```

==> function(z)
apply (simp add: restriction_def ball_iff_equiv)
apply (unfold function_def, blast)
done

lemma (in M_basic_no_repl) restrict_closed [intro,simp]:
  [| M(A); M(r) |] ==> M(restrict(r,A))
apply (simp add: M_restrict_iff)
apply (insert restrict_separation [of A], simp)
done

lemma (in M_basic_no_repl) Inter_closed [intro,simp]:
  M(A) ==> M( $\bigcap$ (A))
by (insert Inter_separation, simp add: Inter_def)

lemma (in M_basic_no_repl) Int_closed [intro,simp]:
  [| M(A); M(B) |] ==> M(A  $\cap$  B)
apply (subgoal_tac M({A,B}))
apply (frule Inter_closed, force+)
done

lemma (in M_basic_no_repl) Diff_closed [intro,simp]:
  [| M(A); M(B) |] ==> M(A - B)
by (insert Diff_separation, simp add: Diff_def)

```

24.0.5 Some Facts About Separation Axioms

```

lemma (in M_basic_no_repl) separation_conj:
  [| separation(M,P); separation(M,Q) |] ==> separation(M,  $\lambda z. P(z) \ \& \ Q(z)$ )
by (simp del: separation_closed
      add: separation_iff Collect_Int_Collect_eq [symmetric])

lemma (in M_basic_no_repl) separation_disj:
  [| separation(M,P); separation(M,Q) |] ==> separation(M,  $\lambda z. P(z) \ | \ Q(z)$ )
by (simp del: separation_closed
      add: separation_iff Collect_Un_Collect_eq [symmetric])

lemma (in M_basic_no_repl) separation_neg:
  separation(M,P) ==> separation(M,  $\lambda z. \sim P(z)$ )
by (simp del: separation_closed
      add: separation_iff Diff_Collect_eq [symmetric])

lemma (in M_basic_no_repl) separation_imp:
  [| separation(M,P); separation(M,Q) |]
  ==> separation(M,  $\lambda z. P(z) \longrightarrow Q(z)$ )
by (simp add: separation_neg separation_disj not_disj_iff_imp [symmetric])

```

This result is a hint of how little can be done without the Reflection Theorem. The quantifier has to be bounded by a set. We also need another instance of Separation!

```

lemma (in M_basic_no_repl) separation_rall:
  [|M(Y);  $\forall y[M]. \text{separation}(M, \lambda x. P(x,y));$ 
    $\forall z[M]. \text{strong\_replacement}(M, \lambda x y. y = \{u \in z . P(u,x)\})$ ]|]
  ==> separation(M,  $\lambda x. \forall y[M]. y \in Y \longrightarrow P(x,y)$ )
apply (simp del: separation_closed rall_abs
  add: separation_iff Collect_rall_eq)
apply (blast intro!: RepFun_closed dest: transM)
done

```

24.0.6 Functions and function space

```

lemma (in M_basic_no_repl) succ_fun_eq2:
  [|M(B); M(n->B)|] ==>
  succ(n) -> B =
   $\bigcup \{z. p \in (n \rightarrow B) * B, \exists f[M]. \exists b[M]. p = \langle f, b \rangle \ \& \ z = \{\text{cons}(\langle n, b \rangle, f)\}\}$ 
apply (simp add: succ_fun_eq)
apply (blast dest: transM)
done

```

```

lemma (in M_basic_no_repl) list_case'_closed [intro,simp]:
  [|M(k); M(a);  $\forall x[M]. \forall y[M]. M(b(x,y))$ ]|] ==> M(list_case'(a,b,k))
apply (case_tac quasilist(k))
apply (simp add: quasilist_def, force)
apply (simp add: non_list_case)
done

```

```

lemma (in M_basic_no_repl) tl'_closed: M(x) ==> M(tl'(x))
apply (simp add: tl'_def)
apply (force simp add: quasilist_def)
done

```

end

References

- [1] 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).
- [2] L.C. PAULSON, The relative consistency of the axiom of choice mechanized using Isabelle/ZF, *LMS J. Comput. Math.* **6**: 198–248 (2003). Appendix A available electronically at <http://www.lms.ac.uk/jcm/6/lms2003-001/appendix-a/>.