Transitive Models of Fragments of ZF

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

March 17, 2025

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 **7**

2 Auxiliary results on arithmetic **9**
2.1 Some results in ordinal arithmetic 12

3 Various results missing from ZF. **15**

4 Renaming of variables in internalized formulas **19**
4.1 Renaming of free variables 19
4.2 Renaming of formulas 27
4.2.1 image, preimage, domain, range 34
4.2.2 Domain, range and field 34
4.2.3 Relations, functions and application 35
4.2.4 Composition of relations 35
4.2.5 Some Facts About Separation Axioms 37
4.2.6 Functions and function space 37

*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.

5 Absoluteness Properties for Recursive Datatypes **38**
5.1 The lfp of a continuous function can be expressed as a union 38
5.1.1 Some Standard Datatype Constructions Preserve Continuity . . . 39
5.2 Absoluteness for "Iterates" . . . 40
5.3 lists without univ . . . 41
5.4 formulas without univ . . . 42
5.5 *M* Contains the List and Formula Datatypes . . . 43
5.5.1 Towards Absoluteness of *formula_rec* . . . 45
5.5.2 Absoluteness of the List Construction . . . 47
5.5.3 Absoluteness of Formulas . . . 47
5.6 Absoluteness for ε-Closure: the *eclose* Operator . . . 47
5.7 Absoluteness for *transrec* . . . 49
5.8 Absoluteness for the List Operator *length* . . . 50
5.9 Absoluteness for the List Operator *nth* . . . 50
5.10 Relativization and Absoluteness for the *formula* Constructors 50
5.11 Absoluteness for *formula_rec* . . . 51
5.11.1 Absoluteness for the Formula Operator *depth* . . . 52

6 Some enhanced theorems on recursion **53**

7 Absoluteness Properties for Recursive Datatypes **60**
7.0.1 Absoluteness of the List Construction . . . 60
7.0.2 Absoluteness of Formulas . . . 61
7.0.3 *is_formula_case*: relativization of *formula_case* . . . 63
7.0.4 Absoluteness for *formula_rec*: Final Results . . . 64
7.1 Internalized Forms of Data Structuring Operators . . . 66
7.1.1 The Formula *is_Inl*, Internalized . . . 66
7.1.2 The Formula *is_Inr*, Internalized . . . 67
7.1.3 The Formula *is_Nil*, Internalized . . . 67
7.1.4 The Formula *is_Cons*, Internalized . . . 68
7.1.5 The Formula *is_quasilist*, Internalized . . . 68
7.2 Absoluteness for the Function *nth* . . . 69
7.2.1 The Formula *is_hd*, Internalized . . . 69
7.2.2 The Formula *is_tl*, Internalized . . . 69
7.2.3 The Operator *is_bool_of_o* . . . 70
7.3 More Internalizations . . . 71
7.3.1 The Operator *is_lambda* . . . 71
7.3.2 The Operator *is_Member*, Internalized . . . 72
7.3.3 The Operator *is_Equal*, Internalized . . . 72
7.3.4 The Operator *is_Nand*, Internalized . . . 73
7.3.5 The Operator *is_Forall*, Internalized . . . 74
7.3.6 The Operator *is_and*, Internalized . . . 74
7.3.7 The Operator *is_or*, Internalized . . . 75

7.3.8 The Operator *is_not*, Internalized . . . 75
7.4 Well-Founded Recursion! . . . 76
7.4.1 The Operator *M_is_recfun* . . . 76
7.4.2 The Operator *is_wfrec* . . . 78
7.5 For Datatypes . . . 79
7.5.1 Binary Products, Internalized . . . 79
7.5.2 Binary Sums, Internalized . . . 80
7.5.3 The Operator *quasinat* . . . 80
7.5.4 The Operator *is_nat_case* . . . 81
7.6 The Operator *iterates_MH*, Needed for Iteration . . . 82
7.6.1 The Operator *is_iterates* . . . 83
7.6.2 The Formula *is_eclose_n*, Internalized . . . 85
7.6.3 Membership in *eclose*(A) . . . 85
7.6.4 The Predicate "Is *eclose*(A)" . . . 86
7.6.5 The List Functor, Internalized . . . 86
7.6.6 The Formula *is_list_N*, Internalized . . . 87
7.6.7 The Predicate "Is A List" . . . 88
7.6.8 The Predicate "Is *list*(A)" . . . 88
7.6.9 The Formula Functor, Internalized . . . 89
7.6.10 The Formula *is_formula_N*, Internalized . . . 90
7.6.11 The Predicate "Is A Formula" . . . 91
7.6.12 The Predicate "Is *formula*" . . . 91
7.6.13 The Operator *is_transrec* . . . 92

8 Separation for Facts About Recursion 93
8.1 The Locale *M_trancl* . . . 93
8.1.1 Separation for Reflexive/Transitive Closure . . . 93
8.1.2 Reflexive/Transitive Closure, Internalized . . . 94
8.1.3 Transitive Closure of a Relation, Internalized . . . 95
8.1.4 Separation for the Proof of *wellfounded_on_trancl* . . . 96
8.1.5 Instantiating the locale *M_trancl* . . . 96
8.2 L is Closed Under the Operator *list* . . . 96
8.2.1 Instances of Replacement for Lists . . . 96
8.3 L is Closed Under the Operator *formula* . . . 97
8.3.1 Instances of Replacement for Formulas . . . 97
8.3.2 The Formula *is_nth*, Internalized . . . 98
8.3.3 An Instance of Replacement for *nth* . . . 99
8.3.4 Instantiating the locale *M_datatypes* . . . 100
8.4 L is Closed Under the Operator *eclose* . . . 100
8.4.1 Instances of Replacement for *eclose* . . . 100
8.4.2 Instantiating the locale *M_eclose* . . . 101

9 Absoluteness for the Satisfies Relation on Formulas **101**
9.1 More Internalization . . . 101
9.1.1 The Formula *is_depth*, Internalized . . . 101
9.1.2 The Operator *is_formula_case* . . . 102
9.2 Absoluteness for the Function *satisfies* . . . 104
9.3 Internalizations Needed to Instantiate *M_satisfies* . . . 111
9.3.1 The Operator *is_depth_apply*, Internalized . . . 111
9.3.2 The Operator *satisfies_is_a*, Internalized . . . 112
9.3.3 The Operator *satisfies_is_b*, Internalized . . . 113
9.3.4 The Operator *satisfies_is_c*, Internalized . . . 113
9.3.5 The Operator *satisfies_is_d*, Internalized . . . 114
9.3.6 The Operator *satisfies_MH*, Internalized . . . 115
9.4 Lemmas for Instantiating the Locale *M_satisfies* . . . 116
9.4.1 The *Member* Case . . . 116
9.4.2 The *Equal* Case . . . 117
9.4.3 The *Nand* Case . . . 117
9.4.4 The *Forall* Case . . . 118
9.4.5 The *transrec_replacement* Case . . . 119
9.4.6 The Lambda Replacement Case . . . 119
9.5 Instantiating *M_satisfies* . . . 120

10 Absoluteness for the Definable Powerset Function **121**
10.1 Preliminary Internalizations . . . 121
10.1.1 The Operator *is_formula_rec* . . . 121
10.1.2 The Operator *is_satisfies* . . . 122
10.2 Relativization of the Operator *DPow'* . . . 123
10.2.1 The Operator *is_DPow_sats*, Internalized . . . 123
10.3 A Locale for Relativizing the Operator *DPow'* . . . 124
10.4 Instantiating the Locale *M_DPow* . . . 125
10.4.1 The Instance of Separation . . . 125
10.4.2 The Instance of Replacement . . . 125
10.4.3 Actually Instantiating the Locale . . . 126
10.4.4 The Operator *is_Collect* . . . 126
10.4.5 The Operator *is_Replace* . . . 127
10.4.6 The Operator *is_DPow'*, Internalized . . . 128
10.5 A Locale for Relativizing the Operator *Lset* . . . 129
10.6 Instantiating the Locale *M_Lset* . . . 130
10.6.1 The First Instance of Replacement . . . 130
10.6.2 The Second Instance of Replacement . . . 131
10.6.3 Actually Instantiating *M_Lset* . . . 132
10.7 The Notion of Constructible Set . . . 132

11 Automatic synthesis of formulas **132**

12 Aids to internalize formulas **133**

13 The binder *Least* **139**
13.1 Uniqueness, absoluteness and closure under *Least* 140

14 Fully relational versions of higher order constructs **142**

15 Automatic relativization of terms and formulas **145**
15.1 Discipline of relativization of basic concepts 147
15.2 Discipline for *Pow* 152
15.3 Discipline for *PiP* 154
15.4 Discipline for *Pi* 156
15.5 Auxiliary ported results on *Pi_rel*, now unused 159

16 Arities of internalized formulas **160**

17 Basic relativization of function spaces **166**
17.1 Discipline for *fst* and *snd* 166
17.2 Discipline for *minimum* 167
17.3 Discipline for $\lambda A\ B.\ A \to B$ 168
17.4 Discipline for *Collect* terms. 171
17.5 Discipline for *inj* 172
17.6 Discipline for *surj* 175
17.7 Discipline for *surj* 176
17.8 Discipline for *Inter* 178
17.9 Discipline for *bij* 179
17.10 Discipline for $(\approx)$ 181
17.11 Discipline for $(\lesssim)$ 182
17.12 Discipline for $(\prec)$ 183

18 Replacements using Lambdas **185**
18.1 Replacement instances obtained through Powerset 190
18.2 Some basic replacement lemmas 237

19 Basic relativization of cardinality **238**
19.1 Disicpline for $(\oplus)$ 240
19.2 Disicpline for $(\otimes)$ 241

20 Relativization of the cumulative hierarchy **242**
20.1 Formula synthesis 247

21 Relative, Choice-less Cardinal Numbers **251**
21.1 The Schroeder-Bernstein Theorem 253
21.2 Banach's Decomposition Theorem 254
21.3 lesspoll: contributions by Krzysztof Grabczewski 262

22 Porting from *ZF.Cardinal* **264**

23 Relative, Choice-less Cardinal Arithmetic **282**
23.1 Discipline for *csucc* . . . 284
23.2 Cardinal addition . . . 286
23.2.1 Cardinal addition is commutative . . . 286
23.2.2 Cardinal addition is associative . . . 287
23.2.3 0 is the identity for addition . . . 287
23.2.4 Addition by another cardinal . . . 288
23.2.5 Monotonicity of addition . . . 288
23.2.6 Addition of finite cardinals is "ordinary" addition . . . 289
23.3 Cardinal multiplication . . . 290
23.3.1 Cardinal multiplication is commutative . . . 290
23.3.2 Cardinal multiplication is associative . . . 290
23.3.3 Cardinal multiplication distributes over addition . . . 291
23.3.4 Multiplication by 0 yields 0 . . . 291
23.3.5 1 is the identity for multiplication . . . 291
23.4 Some inequalities for multiplication . . . 292
23.4.1 Multiplication by a non-zero cardinal . . . 292
23.4.2 Monotonicity of multiplication . . . 292
23.5 Multiplication of finite cardinals is "ordinary" multiplication . 293
23.6 Infinite Cardinals are Limit Ordinals . . . 294
23.6.1 Toward's Kunen's Corollary 10.13 (1) . . . 309
23.7 For Every Cardinal Number There Exists A Greater One . . 311
23.8 Basic Properties of Successor Cardinals . . . 315
23.8.1 Theorems by Krzysztof Grabczewski, proofs by lcp . . 316

24 Relative, Cardinal Arithmetic Using AC **326**
24.1 Strengthened Forms of Existing Theorems on Cardinals . . . 329
24.2 The relationship between cardinality and le-pollence . . . 330
24.3 Other Applications of AC . . . 331

25 Relativization of Finite Functions **335**
25.1 The set of finite binary sequences . . . 336
25.2 Representation of finite functions . . . 337

26 Library of basic *ZF* results **345**
26.1 Discipline for *cexp* . . . 351

27 Lambda-replacements required for cardinal inequalities **367**

28 Cardinal Arithmetic under Choice **378**
28.1 Miscellaneous . . . 378
28.2 Countable and uncountable sets . . . 383
28.3 Results on Aleph_rels . . . 387

28.4 Applications of transfinite recursive constructions 392
28.5 Results on relative cardinal exponentiation 402

29 The Delta System Lemma, Relativized **404**

30 Relative DC **412**

31 Cohen forcing notions **422**

1 Introduction

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

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 of the absolute concepts. 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 [3, Sect. 7.3].

In the theory `Lambda_Replacement` we introduce a new locale assuming two instances of separation and six instances of "lambda replacements" (i.e., replacements using definable functions of the form $\lambda xy.y = \langle x, f(x)\rangle$) that allow some form of compositionality of further instances of separations and replacements.

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 and ordertypes. 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.

Finally, there are some repetitions from *ZF-Constructible* in the present project. We had to duplicate some material from `Relative` in our theory `M_Basic_No_Repl` in order to eliminate one replacement instance appearing in the locale `M_Basic`. The same goes for theories `DPow_absolute`, `Datatype_absolute`, `Internalizations`, `Internalize`, `Rank_Separation`,

`Rec_Separation`, `Recursion_Thms`, `Relativization`, and `Satisfies_absolute`, that were modified to remodularize the locale structure (namely, by eliminating the dependence of locale `M_eclose` on `M_datatypes`, which defines lists and formulas). In order to get the rest of the top-level theories of *ZF-Constructible* to work, we added a new theory called `Eclose_Absolute`.

theory *Utils*
  imports *ZF-Constructible.Formula*
begin

This theory encapsulates some ML utilities

ML_file‹*Utils.ML*›

end

2 Auxiliary results on arithmetic

theory *Nat_Miscellanea*
  imports
    Delta_System_Lemma.ZF_Library
begin

hide_const (**open**) *Order.pred*
notation *add* (**infixl** ‹$+_\omega$› *65*)
notation *diff* (**infixl** ‹$-_\omega$› *65*)

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

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

lemma *nat_succD* : $m \in nat \Longrightarrow succ(n) \in succ(m) \Longrightarrow n \in m$
  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* : $m \in nat \Longrightarrow n \in m \Longrightarrow n \in nat$
  by(*drule ltI*[*of n*],*auto simp add*: *lt_nat_in_nat*)

lemma *ltI_neg* : $x \in nat \Longrightarrow j \leq x \Longrightarrow j \neq x \Longrightarrow j < x$
  by (*simp add*: *le_iff*)

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

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

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

lemma *succpred_leI* : $n \in nat \Longrightarrow n \leq succ(pred(n))$

by (*auto elim*: *natE*)

lemma *succpred_n0* : $succ(n) \in p \Longrightarrow p \neq 0$
by (*auto*)

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

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

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

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

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

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

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

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

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

lemma *succ_mono* : $m \in nat \Longrightarrow n \leq m \Longrightarrow succ(n) \leq succ(m)$
by *auto*

lemma *union_abs1* :
$[\![i \leq j]\!] \Longrightarrow i \cup j = j$
by (*rule Un_absorb1*,*erule le_imp_subset*)

lemma *union_abs2* :
$[\![i \leq j]\!] \Longrightarrow j \cup i = j$
by (*rule Un_absorb2*,*erule le_imp_subset*)

lemma *ord_un_max* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow i \cup j = max(i,j)$
using *max_def union_abs1 not_lt_iff_le leI union_abs2*
by *auto*

lemma *ord_max_ty* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow Ord(max(i,j))$
unfolding *max_def* **by** *simp*

lemmas *ord_simp_union* = *ord_un_max ord_max_ty max_def*

lemma *le_succ* : $x \in nat \Longrightarrow x \leq succ(x)$ **by** *simp*

lemma *le_pred* : $x \in nat \Longrightarrow pred(x) \leq x$
  using *pred_le*[*OF _ le_succ*] *pred_succ_eq*
  by *simp*

lemma *not_le_anti_sym* : $x \in nat \Longrightarrow y \in nat \Longrightarrow \neg\, x \leq y \Longrightarrow \neg y \leq x \Longrightarrow y = x$
  using *Ord_linear not_le_iff_lt ltD lt_trans*
  by *auto*

lemma *Un_le_compat* : $o \leq p \Longrightarrow q \leq r \Longrightarrow Ord(o) \Longrightarrow Ord(p) \Longrightarrow Ord(q) \Longrightarrow Ord(r) \Longrightarrow o \cup q \leq p \cup r$
  using *le_trans*[*of q r p∪r*,*OF _ Un_upper2_le*] *le_trans*[*of o p p∪r*,*OF _ Un_upper1_le*]
    ord_simp_union
  by *auto*

lemma *Un_le* : $p \leq r \Longrightarrow q \leq r \Longrightarrow$
  $Ord(p) \Longrightarrow Ord(q) \Longrightarrow Ord(r) \Longrightarrow$
  $p \cup q \leq r$
  using *ord_simp_union* **by** *auto*

lemma *Un_leI3* : $o \leq r \Longrightarrow p \leq r \Longrightarrow q \leq r \Longrightarrow$
  $Ord(o) \Longrightarrow Ord(p) \Longrightarrow Ord(q) \Longrightarrow Ord(r) \Longrightarrow$
  $o \cup p \cup q \leq r$
  using *ord_simp_union* **by** *auto*

lemma *diff_mono* :
  assumes $m \in nat\ n \in nat\ p \in nat\ m < n\ p \leq m$
  shows $m\#\text{-}p < n\#\text{-}p$
proof -
  from *assms*
  have $m\#\text{-}p \in nat\ m\#\text{-}p +_\omega p = m$
    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*:
  $x \in nat \Longrightarrow y \in nat \Longrightarrow pred(succ(x) \cup y) = x \cup pred(y)$
  $x \in nat \Longrightarrow y \in nat \Longrightarrow pred(x \cup succ(y)) = pred(x) \cup y$
  using *pred_Un_distrib pred_succ_eq* **by** *simp_all*

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

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

```
  by(rule ltE[of j n],simp+)

lemma leD : assumes n∈nat j ≤ n
  shows j < n | j = n
  using leE[OF ‹j≤n›,of j<n | j = n] by auto

lemma pred_nat_eq :
  assumes n∈nat
  shows pred(n) = ⋃ n
  using assms
proof(induct)
  case 0
  then show ?case by simp
next
  case (succ x)
  then show ?case using 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 *frecR*

```
lemma max_cong :
  assumes x ≤ y Ord(y) Ord(z)
  shows max(x,y) ≤ 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 ≤ 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 ≤ y Ord(y) Ord(z) Ord(x)
  shows max(x,z) ≤ max(y,z)
proof -
  from assms
  have  x ∪ z ≤ y ∪ z
```

```
  using lt_Ord Ord_Un Un_mono[OF le_imp_subset[OF ‹x≤y›]] subset_imp_le
by auto
 then show ?thesis
   using ord_simp_union ‹Ord(x)› ‹Ord(z)› ‹Ord(y)› 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≤y
proof -
 from assms
 have w <  x ∪ w using Un_upper2_lt[OF ‹w<z›] 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 ∪ 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 ∨ 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 ∪ y using Un_upper2_lt[OF ‹x<y›] 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 ‹x<y›] by simp
     next
       case False
       then
```

```
      have w ≤ y
        using not_lt_iff_le[OF assms(5) assms(4)] by simp
      with ‹w=z›
      have max(z,y) = y  unfolding max_def using assms by simp
      with assms
      have ... = x ∪ 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(α) Ord(β) α < β x < n y < n 0 <n
  shows n ** α + x < n **β + y
proof -
  consider (0) β=0 | (s) γ where  Ord(γ) β = succ(γ) | (l) Limit(β)
    using Ord_cases[OF ‹Ord(β)›,of ?thesis] by force
  then show ?thesis
  proof cases
    case 0
    then show ?thesis using ‹α<β› by auto
  next
    case s
    then
    have α≤γ using ‹α<β› using leI by auto
    then
    have n ** α ≤ n ** γ using omult_le_mono[OF _ ‹α≤γ›] ‹Ord(n)› by simp
    then
    have n ** α + x < n ** γ + n using oadd_lt_mono[OF _ ‹x<n›] by simp
    also
    have ... = n ** β using ‹β=succ(_)› omult_succ ‹Ord(β)› ‹Ord(n)› by simp
    finally
    have n ** α + x < n ** β by auto
    then
    show ?thesis using oadd_le_self ‹Ord(β)› lt_trans2 ‹Ord(n)› by auto
  next
    case l
    have Ord(x) using ‹x<n› lt_Ord by simp
    with l
    have succ(α) < β using Limit_has_succ ‹α<β› by simp
    have n ** α + x < n ** α + n
        using oadd_lt_mono[OF le_refl[OF Ord_omult[OF _ ‹Ord(α)›]] ‹x<n›]
‹Ord(n)› by simp
    also
    have ... = n ** succ(α) using omult_succ ‹Ord(α)› ‹Ord(n)› by simp
```

```
    finally
    have n ** α + x < n ** succ(α) by simp
    with ‹succ(α) < β›
     have n ** α + x < n ** β using lt_trans omult_lt_mono ‹Ord(n)› ‹0<n›
by auto
    then show ?thesis using oadd_le_self ‹Ord(β)› lt_trans2 ‹Ord(n)› by auto
  qed
qed
end
```

3 Various results missing from ZF.

```
theory ZF_Miscellanea
  imports
    ZF
    Nat_Miscellanea
begin

lemma rex_mono :
  assumes ∃ d ∈ A . P(d) A⊆B
  shows ∃ d ∈ B. P(d)
  using assms by auto

lemma function_subset:
  function(f) ⟹ g⊆f ⟹ function(g)
  unfolding function_def subset_def by auto

lemma converse_refl : refl(A,r) ⟹ refl(A,converse(r))
  unfolding refl_def by simp

lemma Ord_lt_subset : Ord(b) ⟹ a<b ⟹ a⊆b
  by(intro subsetI,frule ltD,rule_tac Ord_trans,simp_all)

lemma funcI : f ∈ A → B ⟹ a ∈ A ⟹ b= f ‘ a ⟹ ⟨a, b⟩ ∈ f
  by(simp_all add: apply_Pair)

lemma vimage_fun_sing:
  assumes f∈A→B b∈B
  shows {a∈A . f‘a=b} = f-‘‘{b}
  using assms vimage_singleton_iff function_apply_equality Pi_iff funcI by auto

lemma image_fun_subset: S∈A→B ⟹ C⊆A⟹ {S ‘ x . x∈ C} = S‘‘C
  using image_function[symmetric,of S C] domain_of_fun Pi_iff by auto

lemma inj_range_Diff:
  assumes f ∈ inj(A,A′)
  shows f‘‘A - f‘‘T = f‘‘(A - T)
    using inj_equality[OF _ _ assms] by auto
```

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

lemma *Diff_bij*:
assumes $\forall A{\in}F.\ X \subseteq A$ **shows** $(\lambda A{\in}F.\ A\text{-}X) \in bij(F, \{A\text{-}X.\ A{\in}F\})$
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 $(\lambda x{\in}A.\ b) : A \to B$
using *assms lam_type* **by** *force*

lemma *lam_constant_eq_cartprod*: $(\lambda_{\in}A.\ y) = A \times \{y\}$
unfolding *lam_def* **by** *auto*

lemma *vimage_lam*: $(\lambda x{\in}A.\ f(x))$ -`` $B = \{\ x{\in}A\ .\ f(x) \in B\ \}$
using *lam_funtype*[*of A f*, *THEN* [*2*] *domain_type*]
lam_funtype[*of A f*, *THEN* [*2*] *apply_equality*] *lamI*[*of _ A f*]
by *auto blast*

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

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

lemma *Pi_range_eq*: $f \in Pi(A,B) \Longrightarrow range(f) = \{f\ `\ x\ .\ x \in A\}$
using *Pi_rangeD*[*of f A B*] *apply_rangeI*[*of f A B*]
by *blast*

lemma *Pi_vimage_subset* : $f \in Pi(A,B) \Longrightarrow f\text{-``}C \subseteq A$
unfolding *Pi_def* **by** *auto*

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

lemma *minimum_in'*: $minimum(r,B) \in B \cup \{0\}$
using *the_0 first_is_elem* **unfolding** *minimum_def*
by (*cases* $\exists!b.\ first(b,\ B,\ r)$)
(*auto dest*!:*theI*[*of* $\lambda b.\ first(b,\ B,\ r)$])

lemma *minimum_in*: $[\![\ well_ord(A,r);\ B{\subseteq}A;\ B{\neq}0\ ]\!] \Longrightarrow minimum(r,B) \in B$

```
  using the_first_in unfolding minimum_def by simp

lemma well_ord_surj_imp_inj_inverse:
  assumes well_ord(A,r) h ∈ surj(A,B)
  shows (λb∈B. minimum(r, {a∈A. h`a=b})) ∈ inj(B,A)
proof -
  let ?f=λb∈B. minimum(r, {a∈A. h`a=b})
  have minimum(r, {a ∈ A . h ` a = b}) ∈ {a∈A. h`a=b} if b∈B for b
  proof -
    from ‹h ∈ surj(A,B)› that
    have {a∈A. h`a=b} ≠ 0
      unfolding surj_def by blast
    with ‹well_ord(A,r)›
    show minimum(r,{a∈A. h`a=b}) ∈ {a∈A. h`a=b}
      using minimum_in by blast
  qed
  moreover from this
  have ?f : B → A
    using lam_type[of B _ λ_.A] by simp
  moreover
  have ?f ` w = ?f ` x ⟹ w = x if w∈B x∈B for w x
  proof -
    from calculation that
    have w = h ` minimum(r,{a∈A. h`a=w})
      x = h ` minimum(r,{a∈A. h`a=x})
      by simp_all
    moreover
    assume ?f ` w = ?f ` x
    moreover from this and that
    have minimum(r, {a ∈ A . h ` a = w}) = minimum(r, {a ∈ 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 well_ord(A,r) h ∈ surj(A,B)
  shows B≲A
  unfolding lepoll_def using well_ord_surj_imp_inj_inverse[OF assms]
  by blast

— New result
lemma surj_imp_well_ord:
  assumes well_ord(A,r) h ∈ surj(A,B)
  shows ∃ s. well_ord(B,s)
```

```
  using assms lepoll_well_ord[OF well_ord_surj_imp_lepoll]
  by force

lemma Pow_sing : Pow({a}) = {0,{a}}
proof(intro equalityI,simp_all)
  have z ∈ {0,{a}} if z ⊆ {a} for z
    using that by auto
  then
  show  Pow({a}) ⊆ {0, {a}} by auto
qed

lemma Pow_cons:
  shows Pow(cons(a,A)) = Pow(A) ∪ {{a} ∪ X . X: Pow(A)}
  using Un_Pow_subset Pow_sing
proof(intro equalityI,auto simp add:Un_Pow_subset)
  {
    fix C D
    assume ⋀ B . B∈Pow(A) ⟹ C ≠ {a} ∪ B C ⊆ {a} ∪ A D ∈ C
    moreover from this
    have ∀ x∈C . x=a ∨ x∈A by auto
    moreover from calculation
    consider (a) D=a | (b) D∈A by auto
    from this
    have D∈A
    proof(cases)
      case a
      with calculation show ?thesis by auto
    next
      case b
      then show ?thesis by simp
    qed
  }
  then show ⋀x xa. (∀ xa∈Pow(A). x ≠ {a} ∪ xa) ⟹ x ⊆ cons(a, A) ⟹ xa ∈
x ⟹ xa ∈ A
    by auto
qed

lemma app_nm :
  assumes n∈nat m∈nat f∈n→m x ∈ nat
  shows f‘x ∈ nat
proof(cases x∈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: Upair(a,b) = {a,b}
```

unfolding *cons_def* **by** *auto*

lemma *converse_apply_eq* : *converse*(f) ‘ $x = \bigcup(f$ -‘‘ $\{x\})$
unfolding *apply_def vimage_def* **by** *simp*

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

lemma *Finite_imp_lesspoll_nat*:
assumes *Finite*(A)
shows $A \prec nat$
using *assms subset_imp_lepoll*[*OF naturals_subset_nat*] *eq_lepoll_trans*
n_lesspoll_nat eq_lesspoll_trans
unfolding *Finite_def lesspoll_def* **by** *auto*

definition *curry* :: $[i,i,i] \Rightarrow i$ **where**
curry(A,B,f) $\equiv \lambda x{\in}A\ .\ \lambda y{\in}B\ .\ f$‘$\langle x,y\rangle$

lemma *curry_type* :
assumes $f \in A{\times}B \rightarrow C$
shows *curry*(A,B,f) $\in A \rightarrow (B \rightarrow C)$
using *assms lam_funtype*
unfolding *curry_def*
by *auto*

end

4 Renaming of variables in internalized formulas

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

4.1 Renaming of free variables

definition
union_fun :: $[i,i,i,i] \Rightarrow i$ **where**
union_fun(f,g,m,p) $\equiv \lambda j \in m \cup p$. *if* $j{\in}m$ *then* f‘j *else* g‘j

lemma *union_fun_type*:
assumes $f \in m \rightarrow n$
$g \in p \rightarrow q$
shows *union_fun*(f,g,m,p) $\in m \cup p \rightarrow n \cup q$
proof -
let *?h*=*union_fun*(f,g,m,p)
have
D: *?h*‘$x \in n \cup q$ **if** $x \in m \cup p$ **for** x
proof (*cases* $x \in m$)

```
    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 m ∪ p _ x ∈ (m ∪ p) × (n ∪ q)] D unfolding
union_fun_def
    by auto
  then have B:?h ⊆ (m ∪ p) × (n ∪ q) ..
  have m ∪ p ⊆ 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 ∈ list(M)
    env′ ∈ list(M)
    length(env) = m ∪ p
    ∀ i . i ∈ m ⟶  nth(f‘i,env′) = nth(i,env)
    ∀ j . j ∈ p ⟶ nth(g‘j,env′) = nth(j,env)
  shows ∀ i . i ∈ m ∪ p ⟶
          nth(i,env) = nth(union_fun(f,g,m,p)‘i,env′)
proof -
  let ?h = union_fun(f,g,m,p)
  have nth(x, env) = nth(?h‘x,env′) if x ∈ m ∪ p for x
    using that
  proof (cases x∈m)
    case True
    with ‹x∈m›
```

```
    have ?h`x = f`x
      unfolding union_fun_def  beta by simp
    with assms ‹x∈m›
    have nth(x,env) = nth(?h`x,env') by simp
    then show ?thesis .
  next
    case False
    with ‹x ∈ m ∪ p›
    have
      x ∈ p x∉m  by auto
    then
    have ?h`x = g`x
      unfolding union_fun_def beta by simp
    with assms ‹x∈p›
    have nth(x,env) = nth(?h`x,env') by simp
    then show ?thesis .
  qed
  then show ?thesis by simp
qed


lemma id_fn_type :
  assumes n ∈ nat
  shows id(n) ∈ n → n
  unfolding id_def using ‹n∈nat› by simp

lemma id_fn_action:
  assumes n ∈ nat env∈list(M)
  shows ⋀ j . j < n ⟹ 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 ‹n∈nat› ltD by
simp
qed


definition
  rsum :: [i,i,i,i,i] ⇒ i where
  rsum(f,g,m,n,p) ≡ λj ∈ m+ω p  . if j<m then f`j else (g`(j#-m))+ω n

lemma sum_inl:
  assumes m ∈ nat n∈nat
    f ∈ m→n x ∈ m
  shows rsum(f,g,m,n,p)`x = f`x
proof -
  from ‹m∈nat›
  have m≤m+ω p
    using add_le_self[of m] by simp
  with assms
  have x∈m+ω p
```

```
    using ltI[of x m] lt_trans2[of x m m+ω p] ltD by simp
  from assms
  have x<m
    using ltI by simp
  with ‹x∈m+ω p›
  show ?thesis unfolding rsum_def by simp
qed

lemma sum_inr:
  assumes m ∈ nat n∈nat p∈nat
    g∈p→q m ≤ x x < m+ω p
  shows rsum(f,g,m,n,p)‘x = g‘(x#-m)+ω n
proof -
  from assms
  have x∈nat
    using in_n_in_nat[of m+ω p] ltD
    by simp
  with assms
  have ¬ x<m
    using not_lt_iff_le[THEN iffD2] by simp
  from assms
  have x∈m+ω p
    using ltD by simp
  with ‹¬ x<m›
  show ?thesis unfolding rsum_def by simp
qed


lemma sum_action :
  assumes m ∈ nat n∈nat p∈nat q∈nat
    f ∈ m→n g∈p→q
    env ∈ list(M)
    env′ ∈ list(M)
    env1 ∈ list(M)
    env2 ∈ list(M)
    length(env) = m
    length(env1) = p
    length(env′) = n
    ⋀ i . i < m ⟹ nth(i,env) = nth(f‘i,env′)
    ⋀ j. j < p ⟹ nth(j,env1) = nth(g‘j,env2)
  shows ∀ i . i < m+ω p ⟶
          nth(i,env@env1) = nth(rsum(f,g,m,n,p)‘i,env′@env2)
proof -
  let ?h = rsum(f,g,m,n,p)
  from ‹m∈nat› ‹n∈nat› ‹q∈nat›
  have m≤m+ω p n≤n+ω q q≤n+ω q
    using add_le_self[of m]  add_le_self2[of n q] by simp_all
  from ‹p∈nat›
  have p = (m+ω p)#-m using diff_add_inverse2 by simp
```

```
have nth(x, env @ env1) = nth(?h`x,env'@env2) if x<m+ω p for x
proof (cases x<m)
  case True
  then
  have 2: ?h`x= f`x x∈m f`x ∈ n x∈nat
    using assms sum_inl ltD apply_type[of f m _ x] in_n_in_nat by simp_all
  with ‹x<m› assms
  have f`x < n f`x<length(env')  f`x∈nat
    using ltI in_n_in_nat by simp_all
  with 2 ‹x<m› assms
  have nth(x,env@env1) = nth(x,env)
    using nth_append[OF ‹env∈list(M)›] ‹x∈nat› by simp
  also
  have
    ... = nth(f`x,env')
    using 2 ‹x<m› assms by simp
  also
  have ... = nth(f`x,env'@env2)
   using nth_append[OF ‹env'∈list(M)›] ‹f`x<length(env')› ‹f`x ∈nat› 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 .
next
  case False
  have x∈nat
    using that in_n_in_nat[of m+ω 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)+ω n  ¬ x <length(env)
    unfolding rsum_def
    using  sum_inr that beta ltD by simp_all
  from assms ‹x∈nat› ‹p=m+ω p#-m›
  have x#-m < p
    using diff_mono[OF _ _ _ ‹x<m+ω 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))+ω n <n+ω q n ≤ g`(x#-m)+ω n ¬ g`(x#-m)+ω 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)+ω n)#-length(env′),env2)
      using  ‹length(env′) = n›
        diff_add_inverse2 ‹g‘(x#-m)∈nat›
      by simp
    also
    have ... = nth((g‘(x#-m)+ω n),env′@env2)
    using nth_append[OF ‹env′∈list(M)›] ‹n∈nat› ‹¬ g‘(x#-m)+ω 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 ∈ nat n∈nat p∈nat q∈nat
    f ∈ m→n g∈p→q
  shows rsum(f,g,m,n,p) ∈ (m+ω p) → (n+ω q)
proof -
  let ?h = rsum(f,g,m,n,p)
  from ‹m∈nat› ‹n∈nat› ‹q∈nat›
  have m≤m+ω p n≤n+ω q q≤n+ω q
    using add_le_self[of m]  add_le_self2[of n q] by simp_all
  from ‹p∈nat›
  have p = (m+ω p)#-m using diff_add_inverse2 by simp
  {fix x
    assume 1: x∈m+ω p x<m
    with 1 have ?h‘x= f‘x x∈m
      using assms sum_inl ltD by simp_all
    with ‹f∈m→n›
    have ?h‘x ∈ n by simp
    with ‹n∈nat› have ?h‘x < n using ltI by simp
    with ‹n≤n+ω q›
    have ?h‘x < n+ω q using lt_trans2 by simp
    then
    have ?h‘x ∈ n+ω q  using ltD by simp
  }
  then have 1:?h‘x ∈ n+ω q if x∈m+ω p x<m for x using that .
```

```
  {fix x
    assume 1: x∈m+ωp m≤x
    then have x<m+ωp x∈nat using ltI in_n_in_nat[of m+ωp] ltD by simp_all
    with 1
    have 2 : ?h‘x= g‘(x#-m)+ωn
      using assms sum_inr ltD by simp_all
    from assms ‹x∈nat› ‹p=m+ωp#-m›
    have x#-m < p using diff_mono[OF _ _ _ ‹x<m+ωp› ‹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› have g‘(x#-m) < q using ltI by simp
    with ‹q∈nat›
     have (g‘(x#-m))+ωn <n+ωq using add_lt_mono1[of g‘(x#-m) _ n,OF _
‹q∈nat›] by simp
    with 2
    have ?h‘x ∈ n+ωq  using ltD by simp
  }
  then have 2:?h‘x ∈ n+ωq if x∈m+ωp m≤x for x using that .
  have
    D: ?h‘x ∈ n+ωq if x∈m+ωp 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+ωp] 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 +ω p) × (n +ω q) if x∈ ?h for x
     using that lamE[of x m+ωp _ x ∈ (m +ω p) × (n +ω q)] D unfolding
rsum_def
    by auto
  then have B:?h ⊆ (m +ω p) × (n +ω q) ..
  have m +ω 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
$rsum(f,id(length(env1)),length(env),length(env'),length(env1)) \in$
$(length(env)+_\omega length(env1)) \rightarrow (length(env')+_\omega length(env1))$
using *assms length_type id_fn_type sum_type*
by *simp*

lemma *sum_type_id_aux2* :
assumes
$f \in m \rightarrow n$
$m \in nat$ $n \in nat$
$env1 \in list(M)$
shows
$rsum(f,id(length(env1)),m,n,length(env1)) \in$
$(m+_\omega length(env1)) \rightarrow (n+_\omega length(env1))$
using *assms id_fn_type sum_type*
by *auto*

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

lemma *sum_action_id_aux* :
assumes

$f \in m \to n$
$env \in list(M)$
$env' \in list(M)$
$env1 \in list(M)$
$length(env) = m$
$length(env') = n$
$length(env1) = p$
$\bigwedge i \,.\, i < m \Longrightarrow nth(i,env) = nth(f`i,env')$
shows $\bigwedge i \,.\, i < m +_\omega length(env1) \Longrightarrow$
$nth(i,env@env1) = nth(rsum(f,id(length(env1)),m,n,length(env1))`i,env'@env1)$
using *assms length_type id_fn_type sum_action_id*
by *auto*

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

lemma $sum_id0 : m{\in}nat \Longrightarrow sum_id(m,f)`0 = 0$
by(*unfold sum_id_def,subst sum_inl,auto*)

lemma $sum_idS : p{\in}nat \Longrightarrow q{\in}nat \Longrightarrow f{\in}p{\to}q \Longrightarrow x \in p \Longrightarrow sum_id(p,f)`(succ(x))$
$= succ(f`x)$
by(*subgoal_tac* $x{\in}nat$,*unfold sum_id_def,subst sum_inr,*
simp_all add:ltI,simp_all add: app_nm in_n_in_nat)

lemma $sum_id_tc_aux$:
$p \in nat \Longrightarrow q \in nat \Longrightarrow f \in p \to q \Longrightarrow sum_id(p,f) \in 1+_\omega p \to 1+_\omega q$
by (*unfold sum_id_def,rule sum_type,simp_all*)

lemma sum_id_tc :
$n \in nat \Longrightarrow m \in nat \Longrightarrow f \in n \to m \Longrightarrow sum_id(n,f) \in succ(n) \to succ(m)$
by(*rule ssubst*[*of* $succ(n) \to succ(m)$ $1+_\omega n \to 1+_\omega m$],
simp,rule sum_id_tc_aux,simp_all)

4.2 Renaming of formulas

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

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

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

$ren(Forall(p)) =$

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

lemma *arity_meml* : $l \in nat \Longrightarrow Member(x,y) \in formula \Longrightarrow arity(Member(x,y)) \leq l \Longrightarrow x \in l$
by(*simp,rule subsetD,rule le_imp_subset,assumption,simp*)
lemma *arity_memr* : $l \in nat \Longrightarrow Member(x,y) \in formula \Longrightarrow arity(Member(x,y)) \leq l \Longrightarrow y \in l$
by(*simp,rule subsetD,rule le_imp_subset,assumption,simp*)
lemma *arity_eql* : $l \in nat \Longrightarrow Equal(x,y) \in formula \Longrightarrow arity(Equal(x,y)) \leq l \Longrightarrow x \in l$
by(*simp,rule subsetD,rule le_imp_subset,assumption,simp*)
lemma *arity_eqr* : $l \in nat \Longrightarrow Equal(x,y) \in formula \Longrightarrow arity(Equal(x,y)) \leq l \Longrightarrow y \in l$
by(*simp,rule subsetD,rule le_imp_subset,assumption,simp*)
lemma *nand_ar1* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(p) \leq arity(Nand(p,q))$
by (*simp,rule Un_upper1_le,simp+*)
lemma *nand_ar2* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(q) \leq arity(Nand(p,q))$
by (*simp,rule Un_upper2_le,simp+*)

lemma *nand_ar1D* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(Nand(p,q)) \leq n \Longrightarrow arity(p) \leq n$
by(*auto simp add*: *le_trans*[*OF Un_upper1_le*[*of arity*(*p*) *arity*(*q*)]])
lemma *nand_ar2D* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(Nand(p,q)) \leq n \Longrightarrow arity(q) \leq n$
by(*auto simp add*: *le_trans*[*OF Un_upper2_le*[*of arity*(*p*) *arity*(*q*)]])

lemma *ren_tc* : $p \in formula \Longrightarrow$
$(\bigwedge\ n\ m\ f\ .\ n \in nat \Longrightarrow m \in nat \Longrightarrow f \in n \to m \Longrightarrow\ ren(p)`n`m`f \in formula)$
by (*induct set*:*formula*,*auto simp add*: *app_nm sum_id_tc*)

lemma *arity_ren* :
fixes *p*
assumes $p \in formula$
shows $\bigwedge\ n\ m\ f\ .\ n \in nat \Longrightarrow m \in nat \Longrightarrow f \in n \to m \Longrightarrow arity(p) \leq n \Longrightarrow arity(ren(p)`n`m`f) \leq m$
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 arity(p)≤arity(Nand(p,q))
    arity(q)≤arity(Nand(p,q))
    by (subst  nand_ar1,simp,simp,simp,subst nand_ar2,simp+)
  then have arity(p)≤n
    and arity(q)≤n using Nand
    by (rule_tac j=arity(Nand(p,q)) in le_trans,simp,simp)+
  then have arity(ren(p)`n`m`f) ≤ m and  arity(ren(q)`n`m`f) ≤ m
    using Nand by auto
  then show ?case using Nand by (simp add:Un_least_lt)
next
  case (Forall p)
  from Forall have succ(n)∈nat  succ(m)∈nat by auto
  from Forall have 2: sum_id(n,f) ∈ succ(n)→succ(m) by (simp add:sum_id_tc)
  from Forall have 3:arity(p) ≤ succ(n) by (rule_tac n=arity(p) in natE,simp+)
  then have arity(ren(p)`succ(n)`succ(m)`sum_id(n,f))≤succ(m) using
      Forall ‹succ(n)∈nat› ‹succ(m)∈nat› 2 by force
  then show ?case using Forall 2 3 ren_tc arity_type pred_le by auto
qed

lemma arity_forallE : p ∈ formula ⟹ m ∈ nat ⟹ arity(Forall(p)) ≤ m ⟹
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_meml by simp
  moreover
  have y ∈ n using Member arity_memr by simp
  ultimately
  show ?case using Member ltI by simp
next
  case (Equal x y)
  have ren(Equal(x,y))`n`m`f = Equal(f`x,f`y) using Equal assms arity_type by
force
  moreover
  have x ∈ n using Equal arity_eql by simp
  moreover
  have y ∈ n using Equal arity_eqr by simp
  ultimately show ?case using Equal ltI by simp
next
  case (Nand p q)
  have ren(Nand(p,q))`n`m`f = Nand(ren(p)`n`m`f,ren(q)`n`m`f) using Nand by
simp
  moreover
  have arity(p) ≤ n using Nand nand_ar1D by simp
  moreover from this
 have i ∈ arity(p) ⟹ i ∈ n for i using subsetD[OF le_imp_subset[OF ‹arity(p)
≤ n›]] by simp
  moreover from this
  have i ∈ arity(p) ⟹ nth(i,ϱ) = nth(f`i,ϱ') for i using Nand ltI by simp
  moreover from this
 have sats(M,p,ϱ) ⟷ sats(M,ren(p)`n`m`f,ϱ') using ‹arity(p)≤n› Nand by simp
```

```
  have arity(q) ≤ n using Nand nand_ar2D by simp
  moreover from this
  have i ∈ arity(q) ⟹ i ∈ n for i using subsetD[OF le_imp_subset[OF ‹arity(q)
≤ n›]] by simp
  moreover from this
  have i ∈ arity(q) ⟹ nth(i,ϱ) = nth(f`i,ϱ') for i using Nand ltI by simp
  moreover from this
  have sats(M,q,ϱ) ⟷ sats(M,ren(q)`n`m`f,ϱ') using assms ‹arity(q)≤n› Nand
by simp
  ultimately
  show ?case using Nand by simp
next
  case (Forall p)
  have 0:ren(Forall(p))`n`m`f = Forall(ren(p)`succ(n)`succ(m)`sum_id(n,f))
    using Forall by simp
  have 1:sum_id(n,f) ∈ succ(n) → succ(m) (is ?g ∈ _) using sum_id_tc Forall
by simp
  then have 2: arity(p) ≤ succ(n)
    using Forall le_trans[of _ succ(pred(arity(p)))] succpred_leI by simp
  have succ(n)∈nat succ(m)∈nat using Forall by auto
  then have A:⋀ j .j < succ(n) ⟹ nth(j, Cons(a, ϱ)) = nth(?g`j, Cons(a, ϱ'))
if a∈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 Renaming_Auto
  imports
    Utils
    Renaming
keywords
  rename :: thy_decl % ML
and
  simple_rename :: thy_decl % ML
and
  src
and
  tgt
```

abbrevs
simple_rename =

begin

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

lemmas *nat_succI* = *nat_succ_iff*[*THEN iffD2*]
ML_file‹*Renaming_ML.ML*›

end
theory *M_Basic_No_Repl*
imports *ZF-Constructible.Relative*
begin

This locale is exactly *M_basic* without its only replacement instance.

locale *M_basic_no_repl* = *M_trivial* +
assumes *Inter_separation*:
$M(A)$ ==> $separation(M, \lambda x.\ \forall y[M].\ y{\in}A \longrightarrow x{\in}y)$
and *Diff_separation*:
$M(B)$ ==> $separation(M, \lambda x.\ x \notin B)$
and *cartprod_separation*:
[| $M(A)$; $M(B)$ |]
==> $separation(M, \lambda z.\ \exists x[M].\ x{\in}A\ \&\ (\exists y[M].\ y{\in}B\ \&\ pair(M,x,y,z)))$
and *image_separation*:
[| $M(A)$; $M(r)$ |]
==> $separation(M, \lambda y.\ \exists p[M].\ p{\in}r\ \&\ (\exists x[M].\ x{\in}A\ \&\ pair(M,x,y,p)))$
and *converse_separation*:
$M(r)$ ==> $separation(M,$
$\lambda z.\ \exists p[M].\ p{\in}r\ \&\ (\exists x[M].\ \exists y[M].\ pair(M,x,y,p)\ \&\ pair(M,y,x,z)))$
and *restrict_separation*:
$M(A)$ ==> $separation(M, \lambda z.\ \exists x[M].\ x{\in}A\ \&\ (\exists y[M].\ pair(M,x,y,z)))$
and *comp_separation*:
[| $M(r)$; $M(s)$ |]
==> $separation(M, \lambda xz.\ \exists x[M].\ \exists y[M].\ \exists z[M].\ \exists xy[M].\ \exists yz[M].$
$pair(M,x,z,xz)\ \&\ pair(M,x,y,xy)\ \&\ pair(M,y,z,yz)\ \&$
$xy{\in}s\ \&\ yz{\in}r)$
and *pred_separation*:
[| $M(r)$; $M(x)$ |] ==> $separation(M, \lambda y.\ \exists p[M].\ p{\in}r\ \&\ pair(M,y,x,p))$
and *Memrel_separation*:
$separation(M, \lambda z.\ \exists x[M].\ \exists y[M].\ pair(M,x,y,z)\ \&\ x \in y)$
and *is_recfun_separation*:
— for well-founded recursion: used to prove *is_recfun_equal*
[| $M(r)$; $M(f)$; $M(g)$; $M(a)$; $M(b)$ |]
==> $separation(M,$
$\lambda x.\ \exists xa[M].\ \exists xb[M].$
$pair(M,x,a,xa)\ \&\ xa \in r\ \&\ pair(M,x,b,xb)\ \&\ xb \in r\ \&$
$(\exists fx[M].\ \exists gx[M].\ fun_apply(M,f,x,fx)\ \&\ fun_apply(M,g,x,gx)\ \&$
$fx \neq gx))$

```
  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=λx. 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) |] ==> M(A*B)
 by (frule cartprod_closed_lemma, assumption, force)

lemma (in M_basic_no_repl) sum_closed [intro,simp]:
 [| M(A); M(B) |] ==> M(A+B)
 by (simp add: sum_def)

lemma (in M_basic_no_repl) sum_abs [simp]:
 [| M(A); M(B); M(Z) |] ==> is_sum(M,A,B,Z) ⟷ (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) ==>
    converse(r) =
```

```
    {z ∈ ⋃(⋃(r)) * ⋃(⋃(r)).
     ∃ p∈r. ∃ x[M]. ∃ y[M]. p = ⟨x,y⟩ & z = ⟨y,x⟩}
  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) ==> 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) |] ==> is_converse(M,r,z) ⟷ 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
```

4.2.1 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)
```

4.2.2 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*]:
  [| $M(r)$; $M(z)$ |] ==> $is_field(M,r,z) \longleftrightarrow z = field(r)$
  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*)

4.2.3 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*]:
  [| $M(f)$; $M(x)$; $M(y)$ |] ==> $fun_apply(M,f,x,y) \longleftrightarrow f`x = y$
  apply (*simp add*: *fun_apply_def apply_def*, *blast*)
  done

lemma (**in** *M_basic_no_repl*) *injection_abs* [*simp*]:
  [| $M(A)$; $M(f)$ |] ==> $injection(M,A,B,f) \longleftrightarrow f \in inj(A,B)$
  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*]:
  [| $M(A)$; $M(B)$; $M(f)$ |] ==> $surjection(M,A,B,f) \longleftrightarrow f \in surj(A,B)$
  by (*simp add*: *surjection_def surj_def*)

lemma (**in** *M_basic_no_repl*) *bijection_abs* [*simp*]:
  [| $M(A)$; $M(B)$; $M(f)$ |] ==> $bijection(M,A,B,f) \longleftrightarrow f \in bij(A,B)$
  by (*simp add*: *bijection_def bij_def*)

4.2.4 Composition of relations

lemma (**in** *M_basic_no_repl*) *M_comp_iff*:
  [| $M(r)$; $M(s)$ |]
    ==> $r\ O\ s =$
      $\{xz \in domain(s) * range(r).$
        $\exists x[M].\ \exists y[M].\ \exists z[M].\ xz = \langle x,z\rangle\ \&\ \langle x,y\rangle \in s\ \&\ \langle y,z\rangle \in r\}$
  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*]:
[| $M(r)$; $M(s)$; $M(t)$ |] ==> $composition(M,r,s,t) \longleftrightarrow t = r\ O\ s$
apply *safe*

Proving $composition(M,\ r,\ s,\ r\ O\ s)$

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*:
[| $restriction(M,f,A,z)$; $function(f)$; $M(f)$; $M(A)$; $M(z)$ |]
==> $function(z)$
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*]:
[|$M(A)$; $M(B)$|] ==> $M(A\text{-}B)$
by (*insert Diff_separation*, *simp add*: *Diff_def*)

4.2.5 Some Facts About Separation Axioms

lemma (**in** *M_basic_no_repl*) *separation_conj*:
[|*separation*(*M*,*P*); *separation*(*M*,*Q*)|] ==> *separation*(*M*, λ*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*, λ*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*, λ*z*. ~*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*, λ*z*. *P*(*z*) ⟶ *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*); ∀ *y*[*M*]. *separation*(*M*, λ*x*. *P*(*x*,*y*));
∀ *z*[*M*]. *strong_replacement*(*M*, λ*x y*. *y* = {*u* ∈ *z* . *P*(*u*,*x*)})|]
==> *separation*(*M*, λ*x*. ∀ *y*[*M*]. *y*∈*Y* ⟶ *P*(*x*,*y*))
apply (*simp del*: *separation_closed rall_abs*
add: *separation_iff Collect_rall_eq*)
apply (*blast intro*!: *RepFun_closed dest*: *transM*)
done

4.2.6 Functions and function space

lemma (**in** *M_basic_no_repl*) *succ_fun_eq2*:
[|*M*(*B*); *M*(*n*->*B*)|] ==>
succ(*n*) -> *B* =
⋃{*z*. *p* ∈ (*n*->*B*)∗*B*, ∃ *f*[*M*]. ∃ *b*[*M*]. *p* = <*f*,*b*> & *z* = {*cons*(<*n*,*b*>, *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*); ∀ *x*[*M*]. ∀ *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

sublocale *M_basic* ⊆ *mbnr*:*M_basic_no_repl*
using *Inter_separation Diff_separation cartprod_separation image_separation converse_separation restrict_separation comp_separation pred_separation Memrel_separation is_recfun_separation power_ax* **by** *unfold_locales*

end

5 Absoluteness Properties for Recursive Datatypes

theory *Eclose_Absolute* **imports** *ZF-Constructible.Formula ZF-Constructible.WF_absolute*
begin

5.1 The lfp of a continuous function can be expressed as a union

definition
directed :: *i=>o* **where**
$directed(A) == A \neq 0 \ \& \ (\forall x \in A.\ \forall y \in A.\ x \cup y \in A)$

definition
contin :: *(i=>i) => o* **where**
$contin(h) == (\forall A.\ directed(A) \longrightarrow h(\bigcup A) = (\bigcup X \in A.\ h(X)))$

lemma *bnd_mono_iterates_subset*: $[|bnd_mono(D,\ h);\ n \in nat|] ==> h\hat{}n\ (0) \subseteq D$
apply (*induct_tac n*)
apply (*simp_all add*: *bnd_mono_def*, *blast*)
done

lemma *bnd_mono_increasing* [*rule_format*]:
$[|i \in nat;\ j \in nat;\ bnd_mono(D,h)|] ==> i \leq j \longrightarrow h\hat{}i(0) \subseteq h\hat{}j(0)$
apply (*rule_tac m=i* **and** *n=j* **in** *diff_induct*, *simp_all*)
apply (*blast del*: *subsetI*
intro: *bnd_mono_iterates_subset bnd_monoD2* [*of concl*: *h*])
done

lemma *directed_iterates*: $bnd_mono(D,h) ==> directed(\{h\hat{}n\ (0).\ n \in nat\})$
apply (*simp add*: *directed_def*, *clarify*)
apply (*rename_tac i j*)
apply (*rule_tac x=i ∪ j* **in** *bexI*)
apply (*rule_tac i = i* **and** *j = j* **in** *Ord_linear_le*)
apply (*simp_all add*: *subset_Un_iff* [*THEN iffD1*] *le_imp_subset*
subset_Un_iff2 [*THEN iffD1*])

```
apply (simp_all add: subset_Un_iff [THEN iff_sym] bnd_mono_increasing
                     subset_Un_iff2 [THEN iff_sym])
done


lemma contin_iterates_eq:
    [|bnd_mono(D, h); contin(h)|]
     ==> h(⋃n∈nat. h^n (0)) = (⋃n∈nat. h^n (0))
apply (simp add: contin_def directed_iterates)
apply (rule trans)
apply (rule equalityI)
 apply (simp_all add: UN_subset_iff)
 apply safe
 apply (erule_tac [2] natE)
  apply (rule_tac a=succ(x) in UN_I)
   apply simp_all
apply blast
done

lemma lfp_subset_Union:
     [|bnd_mono(D, h); contin(h)|] ==> lfp(D,h) ⊆ (⋃n∈nat. h^n(0))
apply (rule lfp_lowerbound)
 apply (simp add: contin_iterates_eq)
apply (simp add: contin_def bnd_mono_iterates_subset UN_subset_iff)
done

lemma Union_subset_lfp:
     bnd_mono(D,h) ==> (⋃n∈nat. h^n(0)) ⊆ lfp(D,h)
apply (simp add: UN_subset_iff)
apply (rule ballI)
apply (induct_tac n, simp_all)
apply (rule subset_trans [of _ h(lfp(D,h))])
 apply (blast dest: bnd_monoD2 [OF _ _ lfp_subset])
apply (erule lfp_lemma2)
done

lemma lfp_eq_Union:
     [|bnd_mono(D, h); contin(h)|] ==> lfp(D,h) = (⋃n∈nat. h^n(0))
by (blast del: subsetI
          intro: lfp_subset_Union Union_subset_lfp)
```

5.1.1 Some Standard Datatype Constructions Preserve Continuity

```
lemma contin_imp_mono: [|X⊆Y; contin(F)|] ==> F(X) ⊆ F(Y)
apply (simp add: contin_def)
apply (drule_tac x={X,Y} in spec)
apply (simp add: directed_def subset_Un_iff2 Un_commute)
done
```

lemma *sum_contin*: [|*contin*(*F*); *contin*(*G*)|] ==> *contin*(λ*X*. *F*(*X*) + *G*(*X*))
by (*simp add*: *contin_def*, *blast*)

lemma *prod_contin*: [|*contin*(*F*); *contin*(*G*)|] ==> *contin*(λ*X*. *F*(*X*) ∗ *G*(*X*))
apply (*subgoal_tac* ∀ *B C*. *F*(*B*) ⊆ *F*(*B* ∪ *C*))
 prefer *2* **apply** (*simp add*: *Un_upper1 contin_imp_mono*)
apply (*subgoal_tac* ∀ *B C*. *G*(*C*) ⊆ *G*(*B* ∪ *C*))
 prefer *2* **apply** (*simp add*: *Un_upper2 contin_imp_mono*)
apply (*simp add*: *contin_def*, *clarify*)
apply (*rule equalityI*)
 prefer *2* **apply** *blast*
apply *clarify*
apply (*rename_tac B C*)
apply (*rule_tac a*=*B* ∪ *C* **in** *UN_I*)
 apply (*simp add*: *directed_def*, *blast*)
done

lemma *const_contin*: *contin*(λ*X*. *A*)
by (*simp add*: *contin_def directed_def*)

lemma *id_contin*: *contin*(λ*X*. *X*)
by (*simp add*: *contin_def*)

5.2 Absoluteness for "Iterates"

definition
  iterates_MH :: [*i*=>*o*, [*i*,*i*]=>*o*, *i*, *i*, *i*, *i*] => *o* **where**
   iterates_MH(*M*,*isF*,*v*,*n*,*g*,*z*) ==
      is_nat_case(*M*, *v*, λ*m u*. ∃ *gm*[*M*]. *fun_apply*(*M*,*g*,*m*,*gm*) & *isF*(*gm*,*u*),
            n, *z*)

definition
  is_iterates :: [*i*=>*o*, [*i*,*i*]=>*o*, *i*, *i*, *i*] => *o* **where**
   is_iterates(*M*,*isF*,*v*,*n*,*Z*) ==
    ∃ *sn*[*M*]. ∃ *msn*[*M*]. *successor*(*M*,*n*,*sn*) & *membership*(*M*,*sn*,*msn*) &
          is_wfrec(*M*, *iterates_MH*(*M*,*isF*,*v*), *msn*, *n*, *Z*)

definition
  iterates_replacement :: [*i*=>*o*, [*i*,*i*]=>*o*, *i*] => *o* **where**
   iterates_replacement(*M*,*isF*,*v*) ==
    ∀ *n*[*M*]. *n*∈*nat* ⟶
      wfrec_replacement(*M*, *iterates_MH*(*M*,*isF*,*v*), *Memrel*(*succ*(*n*)))

lemma (**in** *M_basic*) *iterates_MH_abs*:
  [| *relation1*(*M*,*isF*,*F*); *M*(*n*); *M*(*g*); *M*(*z*) |]
  ==> *iterates_MH*(*M*,*isF*,*v*,*n*,*g*,*z*) ⟷ *z* = *nat_case*(*v*, λ*m*. *F*(*g*‘*m*), *n*)
by (*simp add*: *nat_case_abs* [*of* _ λ*m*. *F*(*g* ‘ *m*)]
        relation1_def iterates_MH_def)

lemma (**in** *M_trancl*) *iterates_imp_wfrec_replacement*:
[|*relation1*(*M*,*isF*,*F*); *n* ∈ *nat*; *iterates_replacement*(*M*,*isF*,*v*)|]
==> *wfrec_replacement*(*M*, λ*n f z*. *z* = *nat_case*(*v*, λ*m*. *F*(*f*‘*m*), *n*),
Memrel(*succ*(*n*)))
by (*simp add*: *iterates_replacement_def iterates_MH_abs*)

theorem (**in** *M_trancl*) *iterates_abs*:
[| *iterates_replacement*(*M*,*isF*,*v*); *relation1*(*M*,*isF*,*F*);
n ∈ *nat*; *M*(*v*); *M*(*z*); ∀ *x*[*M*]. *M*(*F*(*x*)) |]
==> *is_iterates*(*M*,*isF*,*v*,*n*,*z*) ⟷ *z* = *iterates*(*F*,*n*,*v*)
apply (*frule iterates_imp_wfrec_replacement*, *assumption*+)
apply (*simp add*: *wf_Memrel trans_Memrel relation_Memrel*
is_iterates_def relation2_def iterates_MH_abs
iterates_nat_def recursor_def transrec_def
eclose_sing_Ord_eq nat_into_M
trans_wfrec_abs [*of* _ _ _ _ λ*n g*. *nat_case*(*v*, λ*m*. *F*(*g*‘*m*), *n*)])
done

lemma (**in** *M_trancl*) *iterates_closed* [*intro*,*simp*]:
[| *iterates_replacement*(*M*,*isF*,*v*); *relation1*(*M*,*isF*,*F*);
n ∈ *nat*; *M*(*v*); ∀ *x*[*M*]. *M*(*F*(*x*)) |]
==> *M*(*iterates*(*F*,*n*,*v*))
apply (*frule iterates_imp_wfrec_replacement*, *assumption*+)
apply (*simp add*: *wf_Memrel trans_Memrel relation_Memrel*
relation2_def iterates_MH_abs
iterates_nat_def recursor_def transrec_def
eclose_sing_Ord_eq nat_into_M
trans_wfrec_closed [*of* _ _ _ λ*n g*. *nat_case*(*v*, λ*m*. *F*(*g*‘*m*), *n*)])
done

5.3 lists without univ

lemmas *datatype_univs* = *Inl_in_univ Inr_in_univ*
Pair_in_univ nat_into_univ A_into_univ

lemma *list_fun_bnd_mono*: *bnd_mono*(*univ*(*A*), λ*X*. {*0*} + *A*∗*X*)
apply (*rule bnd_monoI*)
apply (*intro subset_refl zero_subset_univ A_subset_univ*
sum_subset_univ Sigma_subset_univ)
apply (*rule subset_refl sum_mono Sigma_mono* | *assumption*)+
done

lemma *list_fun_contin*: *contin*(λ*X*. {*0*} + *A*∗*X*)
by (*intro sum_contin prod_contin id_contin const_contin*)

Re-expresses lists using sum and product

lemma *list_eq_lfp2*: *list*(*A*) = *lfp*(*univ*(*A*), λ*X*. {*0*} + *A*∗*X*)

apply (*simp add*: *list_def*)
apply (*rule equalityI*)
 apply (*rule lfp_lowerbound*)
  prefer *2* **apply** (*rule lfp_subset*)
 apply (*clarify*, *subst lfp_unfold* [*OF list_fun_bnd_mono*])
 apply (*simp add*: *Nil_def Cons_def*)
 apply *blast*

Opposite inclusion

apply (*rule lfp_lowerbound*)
 prefer *2* **apply** (*rule lfp_subset*)
apply (*clarify*, *subst lfp_unfold* [*OF list.bnd_mono*])
apply (*simp add*: *Nil_def Cons_def*)
apply (*blast intro*: *datatype_univs*
 dest: *lfp_subset* [*THEN subsetD*])
done

Re-expresses lists using "iterates", no univ.

lemma *list_eq_Union*:
 $list(A) = (\bigcup n{\in}nat.\ (\lambda X.\ \{0\} + A{*}X)\ \hat{}\ n\ (0))$
by (*simp add*: *list_eq_lfp2 lfp_eq_Union list_fun_bnd_mono list_fun_contin*)

definition
 is_list_functor :: [*i*=>*o*,*i*,*i*,*i*] => *o* **where**
  $is_list_functor(M,A,X,Z) ==$
   $\exists\, n1[M].\ \exists\, AX[M].$
    $number1(M,n1)\ \&\ cartprod(M,A,X,AX)\ \&\ is_sum(M,n1,AX,Z)$

lemma (**in** *M_basic*) *list_functor_abs* [*simp*]:
 $[|\ M(A);\ M(X);\ M(Z)\ |] ==> is_list_functor(M,A,X,Z) \longleftrightarrow (Z = \{0\} + A{*}X)$
by (*simp add*: *is_list_functor_def singleton_0 nat_into_M*)

5.4 formulas without univ

lemma *formula_fun_bnd_mono*:
 $bnd_mono(univ(0),\ \lambda X.\ ((nat{*}nat) + (nat{*}nat)) + (X{*}X + X))$
apply (*rule bnd_monoI*)
 apply (*intro subset_refl zero_subset_univ A_subset_univ*
  sum_subset_univ Sigma_subset_univ nat_subset_univ)
apply (*rule subset_refl sum_mono Sigma_mono* | *assumption*)+
done

lemma *formula_fun_contin*:
 $contin(\lambda X.\ ((nat{*}nat) + (nat{*}nat)) + (X{*}X + X))$
by (*intro sum_contin prod_contin id_contin const_contin*)

Re-expresses formulas using sum and product

lemma *formula_eq_lfp2*:
  formula = *lfp*(*univ*(*0*), λ*X*. ((*nat*∗*nat*) + (*nat*∗*nat*)) + (*X*∗*X* + *X*))
apply (*simp add*: *formula_def*)
apply (*rule equalityI*)
 apply (*rule lfp_lowerbound*)
  prefer *2* **apply** (*rule lfp_subset*)
 apply (*clarify*, *subst lfp_unfold* [*OF formula_fun_bnd_mono*])
 apply (*simp add*: *Member_def Equal_def Nand_def Forall_def*)
 apply *blast*

Opposite inclusion

apply (*rule lfp_lowerbound*)
 prefer *2* **apply** (*rule lfp_subset*, *clarify*)
apply (*subst lfp_unfold* [*OF formula.bnd_mono*, *simplified*])
apply (*simp add*: *Member_def Equal_def Nand_def Forall_def*)
apply (*elim sumE SigmaE*, *simp_all*)
apply (*blast intro*: *datatype_univs dest*: *lfp_subset* [*THEN subsetD*])+
done

Re-expresses formulas using "iterates", no univ.

lemma *formula_eq_Union*:
  formula =
  (⋃*n*∈*nat*. (λ*X*. ((*nat*∗*nat*) + (*nat*∗*nat*)) + (*X*∗*X* + *X*)) ^ *n* (*0*))
by (*simp add*: *formula_eq_lfp2 lfp_eq_Union formula_fun_bnd_mono*
        formula_fun_contin)

definition
 is_formula_functor :: [*i*=>*o*,*i*,*i*] => *o* **where**
  is_formula_functor(*M*,*X*,*Z*) ==
    ∃ *nat'*[*M*]. ∃ *natnat*[*M*]. ∃ *natnatsum*[*M*]. ∃ *XX*[*M*]. ∃ *X3*[*M*].
      omega(*M*,*nat'*) & *cartprod*(*M*,*nat'*,*nat'*,*natnat*) &
      is_sum(*M*,*natnat*,*natnat*,*natnatsum*) &
      cartprod(*M*,*X*,*X*,*XX*) & *is_sum*(*M*,*XX*,*X*,*X3*) &
      is_sum(*M*,*natnatsum*,*X3*,*Z*)

lemma (**in** *M_trancl*) *formula_functor_abs* [*simp*]:
  [| *M*(*X*); *M*(*Z*) |]
  ==> *is_formula_functor*(*M*,*X*,*Z*) ⟷
    Z = ((*nat*∗*nat*) + (*nat*∗*nat*)) + (*X*∗*X* + *X*)
by (*simp add*: *is_formula_functor_def*)

5.5 *M* Contains the List and Formula Datatypes

definition
 list_N :: [*i*,*i*] => *i* **where**
  list_N(*A*,*n*) == (λ*X*. {*0*} + *A* ∗ *X*)^*n* (*0*)

lemma *Nil_in_list_N* [*simp*]: [] ∈ *list_N*(*A*,*succ*(*n*))

by (*simp add*: *list_N_def Nil_def*)

lemma *Cons_in_list_N* [*simp*]:
  Cons(*a*,*l*) ∈ *list_N*(*A*,*succ*(*n*)) ⟷ *a*∈*A* & *l* ∈ *list_N*(*A*,*n*)
by (*simp add*: *list_N_def Cons_def*)

These two aren't simprules because they reveal the underlying list representation.

lemma *list_N_0*: *list_N*(*A*,*0*) = *0*
by (*simp add*: *list_N_def*)

lemma *list_N_succ*: *list_N*(*A*,*succ*(*n*)) = {*0*} + *A* ∗ (*list_N*(*A*,*n*))
by (*simp add*: *list_N_def*)

lemma *list_N_imp_list*:
  [| *l* ∈ *list_N*(*A*,*n*); *n* ∈ *nat* |] ==> *l* ∈ *list*(*A*)
by (*force simp add*: *list_eq_Union list_N_def*)

lemma *list_N_imp_length_lt* [*rule_format*]:
  n ∈ *nat* ==> ∀ *l* ∈ *list_N*(*A*,*n*). *length*(*l*) < *n*
apply (*induct_tac n*)
apply (*auto simp add*: *list_N_0 list_N_succ*
                Nil_def [*symmetric*] *Cons_def* [*symmetric*])
done

lemma *list_imp_list_N* [*rule_format*]:
  l ∈ *list*(*A*) ==> ∀ *n*∈*nat*. *length*(*l*) < *n* ⟶ *l* ∈ *list_N*(*A*, *n*)
apply (*induct_tac l*)
apply (*force elim*: *natE*)+
done

lemma *list_N_imp_eq_length*:
  [|*n* ∈ *nat*; *l* ∉ *list_N*(*A*, *n*); *l* ∈ *list_N*(*A*, *succ*(*n*))|]
  ==> *n* = *length*(*l*)
apply (*rule le_anti_sym*)
prefer *2* **apply** (*simp add*: *list_N_imp_length_lt*)
apply (*frule list_N_imp_list*, *simp*)
apply (*simp add*: *not_lt_iff_le* [*symmetric*])
apply (*blast intro*: *list_imp_list_N*)
done

Express *list_rec* without using *rank* or *Vset*, neither of which is absolute.

lemma (**in** *M_trivial*) *list_rec_eq*:
 l ∈ *list*(*A*) ==>
  list_rec(*a*,*g*,*l*) =
  transrec (*succ*(*length*(*l*)),
    λ*x h*. *Lambda* (*list*(*A*),
              list_case′ (*a*,
                λ*a l*. *g*(*a*, *l*, *h* ‘ *succ*(*length*(*l*)) ‘ *l*)))) ‘ *l*

apply (*induct_tac l*)
apply (*subst transrec*, *simp*)
apply (*subst transrec*)
apply (*simp add*: *list_imp_list_N*)
done

definition
is_list_N :: [*i*=>*o*,*i*,*i*,*i*] => *o* **where**
is_list_N(*M*,*A*,*n*,*Z*) ==
∃ *zero*[*M*]. *empty*(*M*,*zero*) &
is_iterates(*M*, *is_list_functor*(*M*,*A*), *zero*, *n*, *Z*)

definition
mem_list :: [*i*=>*o*,*i*,*i*] => *o* **where**
mem_list(*M*,*A*,*l*) ==
∃ *n*[*M*]. ∃ *listn*[*M*].
finite_ordinal(*M*,*n*) & *is_list_N*(*M*,*A*,*n*,*listn*) & *l* ∈ *listn*

definition
is_list :: [*i*=>*o*,*i*,*i*] => *o* **where**
is_list(*M*,*A*,*Z*) == ∀ *l*[*M*]. *l* ∈ *Z* ⟷ *mem_list*(*M*,*A*,*l*)

5.5.1 Towards Absoluteness of *formula_rec*

consts *depth* :: *i*=>*i*
primrec
depth(*Member*(*x*,*y*)) = *0*
depth(*Equal*(*x*,*y*)) = *0*
depth(*Nand*(*p*,*q*)) = *succ*(*depth*(*p*) ∪ *depth*(*q*))
depth(*Forall*(*p*)) = *succ*(*depth*(*p*))

lemma *depth_type* [*TC*]: *p* ∈ *formula* ==> *depth*(*p*) ∈ *nat*
by (*induct_tac p*, *simp_all*)

definition
formula_N :: *i* => *i* **where**
formula_N(*n*) == (λ*X*. ((*nat*∗*nat*) + (*nat*∗*nat*)) + (*X*∗*X* + *X*)) ^ *n* (*0*)

lemma *Member_in_formula_N* [*simp*]:
Member(*x*,*y*) ∈ *formula_N*(*succ*(*n*)) ⟷ *x* ∈ *nat* & *y* ∈ *nat*
by (*simp add*: *formula_N_def Member_def*)

lemma *Equal_in_formula_N* [*simp*]:
Equal(*x*,*y*) ∈ *formula_N*(*succ*(*n*)) ⟷ *x* ∈ *nat* & *y* ∈ *nat*
by (*simp add*: *formula_N_def Equal_def*)

lemma *Nand_in_formula_N* [*simp*]:
Nand(*x*,*y*) ∈ *formula_N*(*succ*(*n*)) ⟷ *x* ∈ *formula_N*(*n*) & *y* ∈ *formula_N*(*n*)

by (*simp add*: *formula_N_def Nand_def*)

lemma *Forall_in_formula_N* [*simp*]:
  $Forall(x) \in formula_N(succ(n)) \longleftrightarrow x \in formula_N(n)$
by (*simp add*: *formula_N_def Forall_def*)

These two aren't simprules because they reveal the underlying formula representation.

lemma *formula_N_0*: $formula_N(0) = 0$
by (*simp add*: *formula_N_def*)

lemma *formula_N_succ*:
  $formula_N(succ(n)) =$
  $((nat*nat) + (nat*nat)) + (formula_N(n) * formula_N(n) + formula_N(n))$
by (*simp add*: *formula_N_def*)

lemma *formula_N_imp_formula*:
  $[|\ p \in formula_N(n);\ n \in nat\ |] ==> p \in formula$
by (*force simp add*: *formula_eq_Union formula_N_def*)

lemma *formula_N_imp_depth_lt* [*rule_format*]:
  $n \in nat ==> \forall p \in formula_N(n).\ depth(p) < n$
apply (*induct_tac n*)
apply (*auto simp add*: *formula_N_0 formula_N_succ*
  depth_type formula_N_imp_formula Un_least_lt_iff
  Member_def [*symmetric*] *Equal_def* [*symmetric*]
  Nand_def [*symmetric*] *Forall_def* [*symmetric*])
done

lemma *formula_imp_formula_N* [*rule_format*]:
  $p \in formula ==> \forall n{\in}nat.\ depth(p) < n \longrightarrow p \in formula_N(n)$
apply (*induct_tac p*)
apply (*simp_all add*: *succ_Un_distrib Un_least_lt_iff*)
apply (*force elim*: *natE*)+
done

lemma *formula_N_imp_eq_depth*:
  $[|n \in nat;\ p \notin formula_N(n);\ p \in formula_N(succ(n))|]$
  $==> n = depth(p)$
apply (*rule le_anti_sym*)
prefer *2* **apply** (*simp add*: *formula_N_imp_depth_lt*)
apply (*frule formula_N_imp_formula*, *simp*)
apply (*simp add*: *not_lt_iff_le* [*symmetric*])
apply (*blast intro*: *formula_imp_formula_N*)
done

This result and the next are unused.

lemma *formula_N_mono* [*rule_format*]:
  $[|\ m \in nat;\ n \in nat\ |] ==> m{\le}n \longrightarrow formula_N(m) \subseteq formula_N(n)$

apply (*rule_tac m = m* **and** *n = n* **in** *diff_induct*)
apply (*simp_all add*: *formula_N_0 formula_N_succ*, *blast*)
done

lemma *formula_N_distrib*:
[| $m \in nat$; $n \in nat$ |] ==> $formula_N(m \cup n) = formula_N(m) \cup formula_N(n)$
apply (*rule_tac i = m* **and** *j = n* **in** *Ord_linear_le*, *auto*)
apply (*simp_all add*: *subset_Un_iff* [*THEN iffD1*] *subset_Un_iff2* [*THEN iffD1*]

le_imp_subset formula_N_mono)
done

definition
is_formula_N :: [*i*=>*o*,*i*,*i*] => *o* **where**
is_formula_N(*M*,*n*,*Z*) ==
∃ *zero*[*M*]. *empty*(*M*,*zero*) &
is_iterates(*M*, *is_formula_functor*(*M*), *zero*, *n*, *Z*)

definition
mem_formula :: [*i*=>*o*,*i*] => *o* **where**
mem_formula(*M*,*p*) ==
∃ *n*[*M*]. ∃ *formn*[*M*].
finite_ordinal(*M*,*n*) & *is_formula_N*(*M*,*n*,*formn*) & $p \in formn$

definition
is_formula :: [*i*=>*o*,*i*] => *o* **where**
is_formula(*M*,*Z*) == ∀ *p*[*M*]. $p \in Z \longleftrightarrow mem_formula(M,p)$

5.5.2 Absoluteness of the List Construction

5.5.3 Absoluteness of Formulas

5.6 Absoluteness for ε-Closure: the *eclose* Operator

Re-expresses eclose using "iterates"

lemma *eclose_eq_Union*:
$eclose(A) = (\bigcup n{\in}nat.\ Union\hat{}n\ (A))$
apply (*simp add*: *eclose_def*)
apply (*rule UN_cong*)
apply (*rule refl*)
apply (*induct_tac n*)
apply (*simp add*: *nat_rec_0*)
apply (*simp add*: *nat_rec_succ*)
done

definition
is_eclose_n :: [*i*=>*o*,*i*,*i*,*i*] => *o* **where**
is_eclose_n(*M*,*A*,*n*,*Z*) == *is_iterates*(*M*, *big_union*(*M*), *A*, *n*, *Z*)

definition
mem_eclose :: [*i*=>*o*,*i*,*i*] => *o* **where**
mem_eclose(*M*,*A*,*l*) ==
∃ *n*[*M*]. ∃ *eclosen*[*M*].
finite_ordinal(*M*,*n*) & *is_eclose_n*(*M*,*A*,*n*,*eclosen*) & *l* ∈ *eclosen*

definition
is_eclose :: [*i*=>*o*,*i*,*i*] => *o* **where**
is_eclose(*M*,*A*,*Z*) == ∀ *u*[*M*]. *u* ∈ *Z* ⟷ *mem_eclose*(*M*,*A*,*u*)

locale *M_eclose* = *M_trancl* +
assumes *eclose_replacement1*:
M(*A*) ==> *iterates_replacement*(*M*, *big_union*(*M*), *A*)
and *eclose_replacement2*:
M(*A*) ==> *strong_replacement*(*M*,
λ*n y*. *n*∈*nat* & *is_iterates*(*M*, *big_union*(*M*), *A*, *n*, *y*))

lemma (**in** *M_eclose*) *eclose_replacement2′*:
M(*A*) ==> *strong_replacement*(*M*, λ*n y*. *n*∈*nat* & *y* = *Union*^*n* (*A*))
apply (*insert eclose_replacement2* [*of A*])
apply (*rule strong_replacement_cong* [*THEN iffD1*])
apply (*rule conj_cong* [*OF iff_refl iterates_abs* [*of big_union*(*M*)]])
apply (*simp_all add*: *eclose_replacement1 relation1_def*)
done

lemma (**in** *M_eclose*) *eclose_closed* [*intro*,*simp*]:
M(*A*) ==> *M*(*eclose*(*A*))
apply (*insert eclose_replacement1*)
by (*simp add*: *RepFun_closed2 eclose_eq_Union*
eclose_replacement2′ relation1_def
iterates_closed [*of big_union*(*M*)])

lemma (**in** *M_eclose*) *is_eclose_n_abs* [*simp*]:
[|*M*(*A*); *n*∈*nat*; *M*(*Z*)|] ==> *is_eclose_n*(*M*,*A*,*n*,*Z*) ⟷ *Z* = *Union*^*n* (*A*)
apply (*insert eclose_replacement1*)
apply (*simp add*: *is_eclose_n_def relation1_def nat_into_M*
iterates_abs [*of big_union*(*M*) _ *Union*])
done

lemma (**in** *M_eclose*) *mem_eclose_abs* [*simp*]:
M(*A*) ==> *mem_eclose*(*M*,*A*,*l*) ⟷ *l* ∈ *eclose*(*A*)
apply (*insert eclose_replacement1*)
apply (*simp add*: *mem_eclose_def relation1_def eclose_eq_Union*
iterates_closed [*of big_union*(*M*)])
done

lemma (**in** *M_eclose*) *eclose_abs* [*simp*]:

$[|M(A);\ M(Z)|] ==> is_eclose(M,A,Z) \longleftrightarrow Z = eclose(A)$
apply (*simp add*: *is_eclose_def*, *safe*)
apply (*rule M_equalityI*, *simp_all*)
done

5.7 Absoluteness for *transrec*

$transrec(a,\ H) \equiv wfrec(Memrel(eclose(\{a\})),\ a,\ H)$

definition
is_transrec :: $[i=>o,\ [i,i,i]=>o,\ i,\ i] => o$ **where**
is_transrec(M,MH,a,z) ==
$\exists\, sa[M].\ \exists\, esa[M].\ \exists\, mesa[M].$
upair(M,a,a,sa) & *is_eclose*(M,sa,esa) & *membership*$(M,esa,mesa)$ &
is_wfrec$(M,MH,mesa,a,z)$

definition
transrec_replacement :: $[i=>o,\ [i,i,i]=>o,\ i] => o$ **where**
transrec_replacement(M,MH,a) ==
$\exists\, sa[M].\ \exists\, esa[M].\ \exists\, mesa[M].$
upair(M,a,a,sa) & *is_eclose*(M,sa,esa) & *membership*$(M,esa,mesa)$ &
wfrec_replacement$(M,MH,mesa)$

The condition $Ord(i)$ lets us use the simpler *trans_wfrec_abs* rather than *trans_wfrec_abs*, which I haven't even proved yet.

theorem (**in** *M_eclose*) *transrec_abs*:
$[|$*transrec_replacement*(M,MH,i); *relation2*(M,MH,H);
$Ord(i)$; $M(i)$; $M(z)$;
$\forall\, x[M].\ \forall\, g[M].\ function(g) \longrightarrow M(H(x,g))|]$
$==>$ *is_transrec*$(M,MH,i,z) \longleftrightarrow z = transrec(i,H)$
by (*simp add*: *trans_wfrec_abs transrec_replacement_def is_transrec_def transrec_def eclose_sing_Ord_eq wf_Memrel trans_Memrel relation_Memrel*)

theorem (**in** *M_eclose*) *transrec_closed*:
$[|$*transrec_replacement*(M,MH,i); *relation2*(M,MH,H);
$Ord(i)$; $M(i)$;
$\forall\, x[M].\ \forall\, g[M].\ function(g) \longrightarrow M(H(x,g))|]$
$==>$ $M(transrec(i,H))$
by (*simp add*: *trans_wfrec_closed transrec_replacement_def is_transrec_def transrec_def eclose_sing_Ord_eq wf_Memrel trans_Memrel relation_Memrel*)

Helps to prove instances of *transrec_replacement*

lemma (**in** *M_eclose*) *transrec_replacementI*:
$[|M(a);$
strong_replacement $(M,$
$\lambda x\ z.\ \exists\, y[M].\ pair(M,\ x,\ y,\ z)$ &
is_wfrec$(M,MH,Memrel(eclose(\{a\})),x,y))|]$
$==>$ *transrec_replacement*(M,MH,a)
by (*simp add*: *transrec_replacement_def wfrec_replacement_def*)

5.8 Absoluteness for the List Operator *length*

But it is never used.

definition
is_length :: [*i*=>*o*,*i*,*i*,*i*] => *o* **where**
is_length(*M*,*A*,*l*,*n*) ==
∃ *sn*[*M*]. ∃ *list_n*[*M*]. ∃ *list_sn*[*M*].
is_list_N(*M*,*A*,*n*,*list_n*) & *l* ∉ *list_n* &
successor(*M*,*n*,*sn*) & *is_list_N*(*M*,*A*,*sn*,*list_sn*) & *l* ∈ *list_sn*

Proof is trivial since *length* returns natural numbers.

lemma (**in** *M_trivial*) *length_closed* [*intro*,*simp*]:
l ∈ *list*(*A*) ==> *M*(*length*(*l*))
by (*simp add*: *nat_into_M*)

5.9 Absoluteness for the List Operator *nth*

lemma *nth_eq_hd_iterates_tl* [*rule_format*]:
xs ∈ *list*(*A*) ==> ∀ *n* ∈ *nat*. *nth*(*n*,*xs*) = *hd'* (*tl'*^*n* (*xs*))
apply (*induct_tac xs*)
apply (*simp add*: *iterates_tl_Nil hd'_Nil*, *clarify*)
apply (*erule natE*)
apply (*simp add*: *hd'_Cons*)
apply (*simp add*: *tl'_Cons iterates_commute*)
done

lemma (**in** *M_basic*) *iterates_tl'_closed*:
[|*n* ∈ *nat*; *M*(*x*)|] ==> *M*(*tl'*^*n* (*x*))
apply (*induct_tac n*, *simp*)
apply (*simp add*: *tl'_Cons tl'_closed*)
done

Immediate by type-checking

lemma (**in** *M_trancl*) *nth_closed* [*intro*,*simp*]:
[|*xs* ∈ *list*(*A*); *n* ∈ *nat*; *M*(*A*)|] ==> *M*(*nth*(*n*,*xs*))
apply (*case_tac n* < *length*(*xs*))
apply (*blast intro*: *nth_type transM*)
apply (*simp add*: *not_lt_iff_le nth_eq_0*)
done

5.10 Relativization and Absoluteness for the *formula* Constructors

definition
is_Member :: [*i*=>*o*,*i*,*i*,*i*] => *o* **where**
— because *Member*(*x*, *y*) ≡ *Inl*(*Inl*(⟨*x*, *y*⟩))
is_Member(*M*,*x*,*y*,*Z*) ==
∃ *p*[*M*]. ∃ *u*[*M*]. *pair*(*M*,*x*,*y*,*p*) & *is_Inl*(*M*,*p*,*u*) & *is_Inl*(*M*,*u*,*Z*)

lemma (**in** *M_trivial*) *Member_abs* [*simp*]:
$[|M(x);\ M(y);\ M(Z)|] ==> is_Member(M,x,y,Z) \longleftrightarrow (Z = Member(x,y))$
by (*simp add*: *is_Member_def Member_def*)

lemma (**in** *M_trivial*) *Member_in_M_iff* [*iff*]:
$M(Member(x,y)) \longleftrightarrow M(x)\ \&\ M(y)$
by (*simp add*: *Member_def*)

definition
is_Equal :: $[i=>o,i,i,i] => o$ **where**
— because $Equal(x,\ y) \equiv Inl(Inr(\langle x,\ y\rangle))$
$is_Equal(M,x,y,Z) ==$
$\exists\, p[M].\ \exists\, u[M].\ pair(M,x,y,p)\ \&\ is_Inr(M,p,u)\ \&\ is_Inl(M,u,Z)$

lemma (**in** *M_trivial*) *Equal_abs* [*simp*]:
$[|M(x);\ M(y);\ M(Z)|] ==> is_Equal(M,x,y,Z) \longleftrightarrow (Z = Equal(x,y))$
by (*simp add*: *is_Equal_def Equal_def*)

lemma (**in** *M_trivial*) *Equal_in_M_iff* [*iff*]: $M(Equal(x,y)) \longleftrightarrow M(x)\ \&\ M(y)$
by (*simp add*: *Equal_def*)

definition
is_Nand :: $[i=>o,i,i,i] => o$ **where**
— because $Nand(x,\ y) \equiv Inr(Inl(\langle x,\ y\rangle))$
$is_Nand(M,x,y,Z) ==$
$\exists\, p[M].\ \exists\, u[M].\ pair(M,x,y,p)\ \&\ is_Inl(M,p,u)\ \&\ is_Inr(M,u,Z)$

lemma (**in** *M_trivial*) *Nand_abs* [*simp*]:
$[|M(x);\ M(y);\ M(Z)|] ==> is_Nand(M,x,y,Z) \longleftrightarrow (Z = Nand(x,y))$
by (*simp add*: *is_Nand_def Nand_def*)

lemma (**in** *M_trivial*) *Nand_in_M_iff* [*iff*]: $M(Nand(x,y)) \longleftrightarrow M(x)\ \&\ M(y)$
by (*simp add*: *Nand_def*)

definition
is_Forall :: $[i=>o,i,i] => o$ **where**
— because $Forall(x) \equiv Inr(Inr(p))$
$is_Forall(M,p,Z) == \exists\, u[M].\ is_Inr(M,p,u)\ \&\ is_Inr(M,u,Z)$

lemma (**in** *M_trivial*) *Forall_abs* [*simp*]:
$[|M(x);\ M(Z)|] ==> is_Forall(M,x,Z) \longleftrightarrow (Z = Forall(x))$
by (*simp add*: *is_Forall_def Forall_def*)

lemma (**in** *M_trivial*) *Forall_in_M_iff* [*iff*]: $M(Forall(x)) \longleftrightarrow M(x)$
by (*simp add*: *Forall_def*)

5.11 Absoluteness for *formula_rec*

definition

formula_rec_case :: [[*i*,*i*]=>*i*, [*i*,*i*]=>*i*, [*i*,*i*,*i*,*i*]=>*i*, [*i*,*i*]=>*i*, *i*, *i*] => *i* **where**
— the instance of *formula_case* in *formula_rec*
formula_rec_case(*a*,*b*,*c*,*d*,*h*) ==
formula_case (*a*, *b*,
λ*u v*. *c*(*u*, *v*, *h* ‘ *succ*(*depth*(*u*)) ‘ *u*,
h ‘ *succ*(*depth*(*v*)) ‘ *v*),
λ*u*. *d*(*u*, *h* ‘ *succ*(*depth*(*u*)) ‘ *u*))

Unfold *formula_rec* to *formula_rec_case*. Express *formula_rec* without using *rank* or *Vset*, neither of which is absolute.

lemma (**in** *M_trivial*) *formula_rec_eq*:
p ∈ *formula* ==>
formula_rec(*a*,*b*,*c*,*d*,*p*) =
transrec (*succ*(*depth*(*p*)),
λ*x h*. *Lambda* (*formula*, *formula_rec_case*(*a*,*b*,*c*,*d*,*h*))) ‘ *p*
apply (*simp add*: *formula_rec_case_def*)
apply (*induct_tac p*)

Base case for *Member*

apply (*subst transrec*, *simp add*: *formula.intros*)

Base case for *Equal*

apply (*subst transrec*, *simp add*: *formula.intros*)

Inductive step for *Nand*

apply (*subst transrec*)
apply (*simp add*: *succ_Un_distrib formula.intros*)

Inductive step for *Forall*

apply (*subst transrec*)
apply (*simp add*: *formula_imp_formula_N formula.intros*)
done

5.11.1 Absoluteness for the Formula Operator *depth*

definition
is_depth :: [*i*=>*o*,*i*,*i*] => *o* **where**
is_depth(*M*,*p*,*n*) ==
∃ *sn*[*M*]. ∃ *formula_n*[*M*]. ∃ *formula_sn*[*M*].
is_formula_N(*M*,*n*,*formula_n*) & *p* ∉ *formula_n* &
successor(*M*,*n*,*sn*) & *is_formula_N*(*M*,*sn*,*formula_sn*) & *p* ∈ *formula_sn*

Proof is trivial since *depth* returns natural numbers.

lemma (**in** *M_trivial*) *depth_closed* [*intro*,*simp*]:
p ∈ *formula* ==> *M*(*depth*(*p*))
by (*simp add*: *nat_into_M*)

end

6 Some enhanced theorems on recursion

theory *Recursion_Thms*
  imports *Eclose_Absolute*

begin

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

— 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 $R\hat{}*$

lemma *fld_restrict_eq* : $a \in A \Longrightarrow (r \cap A{\times}A)\text{-``}\{a\} = (r\text{-``}\{a\} \cap A)$
  by(*force*)

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

lemma *fld_restrict_dom* :
  assumes $relation(r)$ $domain(r) \subseteq A$ $range(r) \subseteq A$
  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 $x \in r \Longrightarrow x \in r \cap A{\times}A$ **for** x **.**
qed

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

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

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

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

r_into_trancl trancl_trans)

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

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

lemma *wfrec_restr* :
  assumes *rr*: *relation*(r) **and** *wfr*:*wf*(r)
  shows $a \in A \Longrightarrow$ *tr_down*$(r,a) \subseteq A \Longrightarrow$ *wfrec*$(r,a,H) =$ *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 -‘‘ $\{a\} \subseteq$ *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 ‹r-‘‘$\{a\} \subseteq A$›
    have $x \in A$ **..**
    from *pred_down rr*
    have *b* : r -‘‘$\{x\} \subseteq$ *tr_down*(r,x) **.**
    then
    have *tr_down*$(r,x) \subseteq$ *tr_down*(r,a)
      using *tr_down_mono x_a rr* **by** *simp*
    with *1*
    have *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* ‹*tr_down*$(r,x) \subseteq A$› ‹$x \in A$›
    have *wfrec*$(r,x,H) =$ *wfrec*$[A](r,x,H)$ **by** *simp*

```
  }
  then
  have x∈ r-``{a} ⟹ wfrec(r,x,H) =  wfrec[A](r,x,H) for x  .
  then
  have Eq1 :(λ x ∈ r-``{a} . wfrec(r,x,H)) = (λ x ∈ r-``{a} . wfrec[A](r,x,H))
    using lam_cong by simp

  from assms
  have wfrec(r,a,H) = H(a,λ x ∈ r-``{a} . wfrec(r,x,H)) by (simp add:wfrec)
  also
  have ... = H(a,λ x ∈ r-``{a} . wfrec[A](r,x,H))
    using assms Eq1 by simp
  also from 1 ‹r-``{a} ⊆ A›
  have ... = H(a,λ x ∈ (r∩A×A)-``{a} . wfrec[A](r,x,H))
    using assms rest_eq  by simp
  also from ‹a∈A›
  have ... = H(a,λ x ∈ (r-``{a})∩A . wfrec[A](r,x,H))
    using fld_restrict_eq by simp
  also from ‹a∈A› ‹wf[A](r)›
  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) ⟹ wf(r) ⟹ trans(r) ⟹ r-``{a}⊆A ⟹
a ∈ A ⟹
  wfrec(r, a, H) = wfrec[A](r, a, H)
   by(subgoal_tac tr_down(r,a) ⊆ 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⇒i⇒o,i] ⇒ i where
  Rrel(R,A) ≡ {z∈A×A. ∃x y. z = ⟨x, y⟩ ∧ R(x,y)}

lemma RrelI : x ∈ A ⟹ y ∈ A ⟹ R(x,y) ⟹ ⟨x,y⟩ ∈ 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)) ⊆  d
```

```
  unfolding Rrel_def by auto

lemma Rrel_mono : A ⊆ B ⟹ Rrel(R,A) ⊆ Rrel(R,B)
  unfolding Rrel_def by blast

lemma Rrel_restr_eq : Rrel(R,A) ∩ B×B = Rrel(R,A∩B)
  unfolding Rrel_def by blast

— We obtain this lemmas as a consequence of the previous one; alternatively it can
be obtained using Ordinal.Memrel_type
lemma field_Memrel : field(Memrel(A)) ⊆ 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∈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 ∈ y
  shows restrict(f,Memrel(eclose({x}))-``{y})`w
      = restrict(f,(Memrel(eclose({x}))^+)-``{y})`w
  using assms restrict_trancl_Rrel[of mem ] Rrel_mem by (simp)

lemma wf_eq_trancl:
  assumes ⋀ f y . H(y,restrict(f,R-``{y})) = H(y,restrict(f,R^+-``{y}))
  shows  wfrec(R, x, H) = wfrec(R^+, x, H) (is wfrec(?r,_,_) = wfrec(?r',_,_))
proof -
  have wfrec(R, x, H) = wftrec(?r^+, x, λy f. H(y, restrict(f,?r-``{y})))
    unfolding wfrec_def ..
  also
  have  ... = wftrec(?r^+, x, λy f. H(y, restrict(f,(?r^+)-``{y})))
    using assms by simp
  also
  have  ... =  wfrec(?r^+, 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
    ⋀x f . Ord(x) ⟹ foo(x,f) = bar(x,f)
    Ord(α)
  shows
    transrec(α, foo) = transrec(α, bar)
proof -
  have transrec(β,foo) = transrec(β,bar) if Ord(β) for β
```

```
    using that
  proof (induct rule:trans_induct)
    case (step β)
    have transrec(β, foo) = foo(β, λx∈β. transrec(x, foo))
      using def_transrec[of λx. transrec(x, foo) foo] by blast
    also from assms and step
    have  ... = bar(β, λx∈β. transrec(x, foo))
      by simp
    also from step
    have  ... = bar(β, λx∈β. transrec(x, bar))
      by (auto)
    also
    have  ... = transrec(β, bar)
      using def_transrec[of λx. transrec(x, bar) bar, symmetric]
      by blast
    finally
    show transrec(β, foo) = transrec(β, bar) .
  qed
  with assms
  show ?thesis by simp
qed

— Next theorem is very similar to transrec_equal_on_Ord
lemma (in M_eclose) transrec_equal_on_M:
  assumes
    ⋀x f . M(x) ⟹ M(f) ⟹ foo(x,f) = bar(x,f)
    ⋀β. M(β) ⟹ transrec_replacement(M,is_foo,β) relation2(M,is_foo,foo)
    strong_replacement(M, λx y. y = ⟨x, transrec(x, foo)⟩)
    ∀x[M]. ∀g[M]. function(g) ⟶ M(foo(x,g))
    M(α) Ord(α)
  shows
    transrec(α, foo) = transrec(α, bar)
proof -
  have M(transrec(x, foo)) if Ord(x) and M(x) for x
    using that assms transrec_closed[of is_foo]
    by simp
  have transrec(β,foo) = transrec(β,bar) M(transrec(β,foo)) if Ord(β) M(β) for β
    using that
  proof (induct rule:trans_induct)
    case (step β)
    moreover
    assume M(β)
    moreover
    note ‹Ord(β)⟹ M(β) ⟹ M(transrec(β, foo))›
    ultimately
    show M(transrec(β, foo)) by blast
    with step ‹M(β)› ‹⋀x. Ord(x)⟹ M(x) ⟹ M(transrec(x, foo))›
      ‹strong_replacement(M, λx y. y = ⟨x, transrec(x, foo)⟩)›
    have M(λx∈β. transrec(x, foo))
```

```
      using Ord_in_Ord transM[of _ β]
      by (rule_tac lam_closed) auto
    have transrec(β, foo) = foo(β, λx∈β. transrec(x, foo))
      using def_transrec[of λx. transrec(x, foo) foo] by blast
    also from assms and ‹M(λx∈β. transrec(x, foo))› ‹M(β)›
    have  ... = bar(β, λx∈β. transrec(x, foo))
      by simp
    also from step and ‹M(β)›
    have  ... = bar(β, λx∈β. transrec(x, bar))
      using transM[of _ β] by (auto)
    also
    have  ... = transrec(β, bar)
      using def_transrec[of λx. transrec(x, bar) bar, symmetric]
      by blast
    finally
    show transrec(β, foo) = transrec(β, 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
```

7 Absoluteness Properties for Recursive Datatypes

```
theory Datatype_absolute imports Eclose_Absolute begin

locale M_datatypes = M_trancl +
 assumes list_replacement1:
   M(A) ==> iterates_replacement(M, is_list_functor(M,A), 0)
  and list_replacement2:
   M(A) ==> strong_replacement(M,
        λn y. n∈nat & is_iterates(M, is_list_functor(M,A), 0, n, y))
  and formula_replacement1:
   iterates_replacement(M, is_formula_functor(M), 0)
  and formula_replacement2:
   strong_replacement(M,
        λn y. n∈nat & is_iterates(M, is_formula_functor(M), 0, n, y))
  and nth_replacement:
   M(l) ==> iterates_replacement(M, %l t. is_tl(M,l,t), l)
```

7.0.1 Absoluteness of the List Construction

```
lemma (in M_datatypes) list_replacement2':
  M(A) ==> strong_replacement(M, λn y. n∈nat & y = (λX. {0} + A * X)^n
(0))
```

apply (*insert list_replacement2* [*of A*])
apply (*rule strong_replacement_cong* [*THEN iffD1*])
apply (*rule conj_cong* [*OF iff_refl iterates_abs* [*of is_list_functor*(*M*,*A*)]])
apply (*simp_all add*: *list_replacement1 relation1_def*)
done

lemma (**in** *M_datatypes*) *list_closed* [*intro*,*simp*]:
  M(*A*) ==> *M*(*list*(*A*))
apply (*insert list_replacement1*)
by (*simp add*: *RepFun_closed2 list_eq_Union*
  list_replacement2′ relation1_def
  iterates_closed [*of is_list_functor*(*M*,*A*)])

WARNING: use only with *dest*: or with variables fixed!

lemmas (**in** *M_datatypes*) *list_into_M* = *transM* [*OF _ list_closed*]

lemma (**in** *M_datatypes*) *list_N_abs* [*simp*]:
  [|*M*(*A*); *n*∈*nat*; *M*(*Z*)|]
  ==> *is_list_N*(*M*,*A*,*n*,*Z*) ⟷ *Z* = *list_N*(*A*,*n*)
apply (*insert list_replacement1*)
apply (*simp add*: *is_list_N_def list_N_def relation1_def nat_into_M*
  iterates_abs [*of is_list_functor*(*M*,*A*) _ λ*X*. {*0*} + *A*∗*X*])
done

lemma (**in** *M_datatypes*) *list_N_closed* [*intro*,*simp*]:
  [|*M*(*A*); *n*∈*nat*|] ==> *M*(*list_N*(*A*,*n*))
apply (*insert list_replacement1*)
apply (*simp add*: *is_list_N_def list_N_def relation1_def nat_into_M*
  iterates_closed [*of is_list_functor*(*M*,*A*)])
done

lemma (**in** *M_datatypes*) *mem_list_abs* [*simp*]:
  M(*A*) ==> *mem_list*(*M*,*A*,*l*) ⟷ *l* ∈ *list*(*A*)
apply (*insert list_replacement1*)
apply (*simp add*: *mem_list_def list_N_def relation1_def list_eq_Union*
  iterates_closed [*of is_list_functor*(*M*,*A*)])
done

lemma (**in** *M_datatypes*) *list_abs* [*simp*]:
  [|*M*(*A*); *M*(*Z*)|] ==> *is_list*(*M*,*A*,*Z*) ⟷ *Z* = *list*(*A*)
apply (*simp add*: *is_list_def*, *safe*)
apply (*rule M_equalityI*, *simp_all*)
done

7.0.2 Absoluteness of Formulas

lemma (**in** *M_datatypes*) *formula_replacement2′*:
  strong_replacement(*M*, λ*n y*. *n*∈*nat* & *y* = (λ*X*. ((*nat*∗*nat*) + (*nat*∗*nat*)) + (*X*∗*X* + *X*))^*n* (*0*))

apply (*insert formula_replacement2*)
apply (*rule strong_replacement_cong* [*THEN iffD1*])
apply (*rule conj_cong* [*OF iff_refl iterates_abs* [*of is_formula_functor*(*M*)]])
apply (*simp_all add*: *formula_replacement1 relation1_def*)
done

lemma (**in** *M_datatypes*) *formula_closed* [*intro*,*simp*]:
  M(*formula*)
apply (*insert formula_replacement1*)
apply (*simp add*: *RepFun_closed2 formula_eq_Union*
  formula_replacement2′ relation1_def
  iterates_closed [*of is_formula_functor*(*M*)])
done

lemmas (**in** *M_datatypes*) *formula_into_M* = *transM* [*OF _ formula_closed*]

lemma (**in** *M_datatypes*) *formula_N_abs* [*simp*]:
  [|*n*∈*nat*; *M*(*Z*)|]
  ==> *is_formula_N*(*M*,*n*,*Z*) ⟷ *Z* = *formula_N*(*n*)
apply (*insert formula_replacement1*)
apply (*simp add*: *is_formula_N_def formula_N_def relation1_def nat_into_M*
  iterates_abs [*of is_formula_functor*(*M*) _
  λ*X*. ((*nat*∗*nat*) + (*nat*∗*nat*)) + (*X*∗*X* + *X*)])
done

lemma (**in** *M_datatypes*) *formula_N_closed* [*intro*,*simp*]:
  n∈*nat* ==> *M*(*formula_N*(*n*))
apply (*insert formula_replacement1*)
apply (*simp add*: *is_formula_N_def formula_N_def relation1_def nat_into_M*
  iterates_closed [*of is_formula_functor*(*M*)])
done

lemma (**in** *M_datatypes*) *mem_formula_abs* [*simp*]:
  mem_formula(*M*,*l*) ⟷ *l* ∈ *formula*
apply (*insert formula_replacement1*)
apply (*simp add*: *mem_formula_def relation1_def formula_eq_Union formula_N_def*
  iterates_closed [*of is_formula_functor*(*M*)])
done

lemma (**in** *M_datatypes*) *formula_abs* [*simp*]:
  [|*M*(*Z*)|] ==> *is_formula*(*M*,*Z*) ⟷ *Z* = *formula*
apply (*simp add*: *is_formula_def*, *safe*)
apply (*rule M_equalityI*, *simp_all*)
done

lemma (**in** *M_datatypes*) *length_abs* [*simp*]:
  [|*M*(*A*); *l* ∈ *list*(*A*); *n* ∈ *nat*|] ==> *is_length*(*M*,*A*,*l*,*n*) ⟷ *n* = *length*(*l*)

apply (*subgoal_tac M*(*l*) & *M*(*n*))
 prefer *2* **apply** (*blast dest*: *transM*)
apply (*simp add*: *is_length_def*)
apply (*blast intro*: *list_imp_list_N nat_into_Ord list_N_imp_eq_length*
 dest: *list_N_imp_length_lt*)
done

definition
 is_nth :: [*i*=>*o*,*i*,*i*,*i*] => *o* **where**
 is_nth(*M*,*n*,*l*,*Z*) ==
 ∃ *X*[*M*]. *is_iterates*(*M*, *is_tl*(*M*), *l*, *n*, *X*) & *is_hd*(*M*,*X*,*Z*)

lemma (**in** *M_datatypes*) *nth_abs* [*simp*]:
 [|*M*(*A*); *n* ∈ *nat*; *l* ∈ *list*(*A*); *M*(*Z*)|]
 ==> *is_nth*(*M*,*n*,*l*,*Z*) ⟷ *Z* = *nth*(*n*,*l*)
apply (*subgoal_tac M*(*l*))
 prefer *2* **apply** (*blast intro*: *transM*)
apply (*simp add*: *is_nth_def nth_eq_hd_iterates_tl nat_into_M*
 tl'_closed iterates_tl'_closed
 iterates_abs [*OF _ relation1_tl*] *nth_replacement*)
done

lemma (**in** *M_datatypes*) *depth_abs* [*simp*]:
 [|*p* ∈ *formula*; *n* ∈ *nat*|] ==> *is_depth*(*M*,*p*,*n*) ⟷ *n* = *depth*(*p*)
apply (*subgoal_tac M*(*p*) & *M*(*n*))
 prefer *2* **apply** (*blast dest*: *transM*)
apply (*simp add*: *is_depth_def*)
apply (*blast intro*: *formula_imp_formula_N nat_into_Ord formula_N_imp_eq_depth*
 dest: *formula_N_imp_depth_lt*)
done

7.0.3 *is_formula_case*: relativization of *formula_case*

definition
 is_formula_case ::
 [*i*=>*o*, [*i*,*i*,*i*]=>*o*, [*i*,*i*,*i*]=>*o*, [*i*,*i*,*i*]=>*o*, [*i*,*i*]=>*o*, *i*, *i*] => *o* **where**
 — no constraint on non-formulas
 is_formula_case(*M*, *is_a*, *is_b*, *is_c*, *is_d*, *p*, *z*) ==
 (∀ *x*[*M*]. ∀ *y*[*M*]. *finite_ordinal*(*M*,*x*) ⟶ *finite_ordinal*(*M*,*y*) ⟶
 is_Member(*M*,*x*,*y*,*p*) ⟶ *is_a*(*x*,*y*,*z*)) &
 (∀ *x*[*M*]. ∀ *y*[*M*]. *finite_ordinal*(*M*,*x*) ⟶ *finite_ordinal*(*M*,*y*) ⟶
 is_Equal(*M*,*x*,*y*,*p*) ⟶ *is_b*(*x*,*y*,*z*)) &
 (∀ *x*[*M*]. ∀ *y*[*M*]. *mem_formula*(*M*,*x*) ⟶ *mem_formula*(*M*,*y*) ⟶
 is_Nand(*M*,*x*,*y*,*p*) ⟶ *is_c*(*x*,*y*,*z*)) &
 (∀ *x*[*M*]. *mem_formula*(*M*,*x*) ⟶ *is_Forall*(*M*,*x*,*p*) ⟶ *is_d*(*x*,*z*))

lemma (**in** *M_datatypes*) *formula_case_abs* [*simp*]:
[| *Relation2*(*M*,*nat*,*nat*,*is_a*,*a*); *Relation2*(*M*,*nat*,*nat*,*is_b*,*b*);
Relation2(*M*,*formula*,*formula*,*is_c*,*c*); *Relation1*(*M*,*formula*,*is_d*,*d*);
p ∈ *formula*; *M*(*z*) |]
==> *is_formula_case*(*M*,*is_a*,*is_b*,*is_c*,*is_d*,*p*,*z*) ⟷
z = *formula_case*(*a*,*b*,*c*,*d*,*p*)
apply (*simp add*: *formula_into_M is_formula_case_def*)
apply (*erule formula.cases*)
apply (*simp_all add*: *Relation1_def Relation2_def*)
done

lemma (**in** *M_datatypes*) *formula_case_closed* [*intro*,*simp*]:
[|*p* ∈ *formula*;
∀ *x*[*M*]. ∀ *y*[*M*]. *x*∈*nat* ⟶ *y*∈*nat* ⟶ *M*(*a*(*x*,*y*));
∀ *x*[*M*]. ∀ *y*[*M*]. *x*∈*nat* ⟶ *y*∈*nat* ⟶ *M*(*b*(*x*,*y*));
∀ *x*[*M*]. ∀ *y*[*M*]. *x*∈*formula* ⟶ *y*∈*formula* ⟶ *M*(*c*(*x*,*y*));
∀ *x*[*M*]. *x*∈*formula* ⟶ *M*(*d*(*x*))|] ==> *M*(*formula_case*(*a*,*b*,*c*,*d*,*p*))
by (*erule formula.cases*, *simp_all*)

7.0.4 Absoluteness for *formula_rec*: Final Results

definition
is_formula_rec :: [*i*=>*o*, [*i*,*i*,*i*]=>*o*, *i*, *i*] => *o* **where**
— predicate to relativize the functional *formula_rec*
is_formula_rec(*M*,*MH*,*p*,*z*) ==
∃ *dp*[*M*]. ∃ *i*[*M*]. ∃ *f*[*M*]. *finite_ordinal*(*M*,*dp*) & *is_depth*(*M*,*p*,*dp*) &
successor(*M*,*dp*,*i*) & *fun_apply*(*M*,*f*,*p*,*z*) & *is_transrec*(*M*,*MH*,*i*,*f*)

Sufficient conditions to relativize the instance of *formula_case* in *formula_rec*

lemma (**in** *M_datatypes*) *Relation1_formula_rec_case*:
[|*Relation2*(*M*, *nat*, *nat*, *is_a*, *a*);
Relation2(*M*, *nat*, *nat*, *is_b*, *b*);
Relation2 (*M*, *formula*, *formula*,
is_c, λ*u v*. *c*(*u*, *v*, *h*‘*succ*(*depth*(*u*))‘*u*, *h*‘*succ*(*depth*(*v*))‘*v*));
Relation1(*M*, *formula*,
is_d, λ*u*. *d*(*u*, *h* ‘ *succ*(*depth*(*u*)) ‘ *u*));
M(*h*) |]
==> *Relation1*(*M*, *formula*,
is_formula_case (*M*, *is_a*, *is_b*, *is_c*, *is_d*),
formula_rec_case(*a*, *b*, *c*, *d*, *h*))
apply (*simp* (*no_asm*) *add*: *formula_rec_case_def Relation1_def*)
apply (*simp*)
done

This locale packages the premises of the following theorems, which is the normal purpose of locales. It doesn't accumulate constraints on the class *M*, as in most of this development.

locale *Formula_Rec* = *M_eclose* + *M_datatypes* +

fixes *a* **and** *is_a* **and** *b* **and** *is_b* **and** *c* **and** *is_c* **and** *d* **and** *is_d* **and** *MH*
defines
MH(*u*::*i*,*f*,*z*) ==
∀ *fml*[*M*]. *is_formula*(*M*,*fml*) ⟶
is_lambda
(*M*, *fml*, *is_formula_case* (*M*, *is_a*, *is_b*, *is_c*(*f*), *is_d*(*f*)), *z*)

assumes *a_closed*: [|*x*∈*nat*; *y*∈*nat*|] ==> *M*(*a*(*x*,*y*))
and *a_rel*: *Relation2*(*M*, *nat*, *nat*, *is_a*, *a*)
and *b_closed*: [|*x*∈*nat*; *y*∈*nat*|] ==> *M*(*b*(*x*,*y*))
and *b_rel*: *Relation2*(*M*, *nat*, *nat*, *is_b*, *b*)
and *c_closed*: [|*x* ∈ *formula*; *y* ∈ *formula*; *M*(*gx*); *M*(*gy*)|]
==> *M*(*c*(*x*, *y*, *gx*, *gy*))
and *c_rel*:
M(*f*) ==>
Relation2 (*M*, *formula*, *formula*, *is_c*(*f*),
λ*u v*. *c*(*u*, *v*, *f* ‘ *succ*(*depth*(*u*)) ‘ *u*, *f* ‘ *succ*(*depth*(*v*)) ‘ *v*))
and *d_closed*: [|*x* ∈ *formula*; *M*(*gx*)|] ==> *M*(*d*(*x*, *gx*))
and *d_rel*:
M(*f*) ==>
Relation1(*M*, *formula*, *is_d*(*f*), λ*u*. *d*(*u*, *f* ‘ *succ*(*depth*(*u*)) ‘ *u*))
and *fr_replace*: *n* ∈ *nat* ==> *transrec_replacement*(*M*,*MH*,*n*)
and *fr_lam_replace*:
M(*g*) ==>
strong_replacement
(*M*, λ*x y*. *x* ∈ *formula* &
y = ⟨*x*, *formula_rec_case*(*a*,*b*,*c*,*d*,*g*,*x*)⟩)

lemma (**in** *Formula_Rec*) *formula_rec_case_closed*:
[|*M*(*g*); *p* ∈ *formula*|] ==> *M*(*formula_rec_case*(*a*, *b*, *c*, *d*, *g*, *p*))
by (*simp add*: *formula_rec_case_def a_closed b_closed c_closed d_closed*)

lemma (**in** *Formula_Rec*) *formula_rec_lam_closed*:
M(*g*) ==> *M*(*Lambda* (*formula*, *formula_rec_case*(*a*,*b*,*c*,*d*,*g*)))
by (*simp add*: *lam_closed2 fr_lam_replace formula_rec_case_closed*)

lemma (**in** *Formula_Rec*) *MH_rel2*:
relation2 (*M*, *MH*,
λ*x h*. *Lambda* (*formula*, *formula_rec_case*(*a*,*b*,*c*,*d*,*h*)))
apply (*simp add*: *relation2_def MH_def*, *clarify*)
apply (*rule lambda_abs2*)
apply (*rule Relation1_formula_rec_case*)
apply (*simp_all add*: *a_rel b_rel c_rel d_rel formula_rec_case_closed*)
done

lemma (**in** *Formula_Rec*) *fr_transrec_closed*:
n ∈ *nat*
==> *M*(*transrec*
(*n*, λ*x h*. *Lambda*(*formula*, *formula_rec_case*(*a*, *b*, *c*, *d*, *h*))))

by (*simp add*: *transrec_closed* [*OF fr_replace MH_rel2*]
nat_into_M formula_rec_lam_closed)

The main two results: *formula_rec* is absolute for *M*.

theorem (**in** *Formula_Rec*) *formula_rec_closed*:
$p \in formula$ ==> $M(formula_rec(a,b,c,d,p))$
by (*simp add*: *formula_rec_eq fr_transrec_closed*
transM [*OF _ formula_closed*])

theorem (**in** *Formula_Rec*) *formula_rec_abs*:
[| $p \in formula$; $M(z)$|]
==> $is_formula_rec(M,MH,p,z) \longleftrightarrow z = formula_rec(a,b,c,d,p)$
by (*simp add*: *is_formula_rec_def formula_rec_eq transM* [*OF _ formula_closed*]
transrec_abs [*OF fr_replace MH_rel2*] *depth_type*
fr_transrec_closed formula_rec_lam_closed eq_commute)

end

theory *Internalize* **imports** *ZF-Constructible.L_axioms Eclose_Absolute* **begin**

7.1 Internalized Forms of Data Structuring Operators

7.1.1 The Formula *is_Inl*, Internalized

definition
Inl_fm :: [*i*,*i*]=>*i* **where**
$Inl_fm(a,z) == Exists(And(empty_fm(0), pair_fm(0,succ(a),succ(z))))$

lemma *Inl_type* [*TC*]:
[| $x \in nat$; $z \in nat$ |] ==> $Inl_fm(x,z) \in formula$
by (*simp add*: *Inl_fm_def*)

lemma *sats_Inl_fm* [*simp*]:
[| $x \in nat$; $z \in nat$; $env \in list(A)$|]
==> $sats(A, Inl_fm(x,z), env) \longleftrightarrow is_Inl(\#\#A, nth(x,env), nth(z,env))$
by (*simp add*: *Inl_fm_def is_Inl_def*)

lemma *Inl_iff_sats*:
[| $nth(i,env) = x$; $nth(k,env) = z$;
$i \in nat$; $k \in nat$; $env \in list(A)$|]
==> $is_Inl(\#\#A, x, z) \longleftrightarrow sats(A, Inl_fm(i,k), env)$
by *simp*

theorem *Inl_reflection*:
$REFLECTS[\lambda x.\ is_Inl(L,f(x),h(x)),$
$\lambda i\ x.\ is_Inl(\#\#Lset(i),f(x),h(x))]$
apply (*simp only*: *is_Inl_def*)
apply (*intro FOL_reflections function_reflections*)

done

7.1.2 The Formula *is_Inr*, Internalized

definition
Inr_fm :: [*i*,*i*]=>*i* **where**
Inr_fm(*a*,*z*) == *Exists*(*And*(*number1_fm*(*0*), *pair_fm*(*0*,*succ*(*a*),*succ*(*z*))))

lemma *Inr_type* [*TC*]:
[| *x* ∈ *nat*; *z* ∈ *nat* |] ==> *Inr_fm*(*x*,*z*) ∈ *formula*
by (*simp add*: *Inr_fm_def*)

lemma *sats_Inr_fm* [*simp*]:
[| *x* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *Inr_fm*(*x*,*z*), *env*) ⟷ *is_Inr*(##*A*, *nth*(*x*,*env*), *nth*(*z*,*env*))
by (*simp add*: *Inr_fm_def is_Inr_def*)

lemma *Inr_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*k*,*env*) = *z*;
i ∈ *nat*; *k* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_Inr*(##*A*, *x*, *z*) ⟷ *sats*(*A*, *Inr_fm*(*i*,*k*), *env*)
by *simp*

theorem *Inr_reflection*:
REFLECTS[λ*x*. *is_Inr*(*L*,*f*(*x*),*h*(*x*)),
λ*i x*. *is_Inr*(##*Lset*(*i*),*f*(*x*),*h*(*x*))]
apply (*simp only*: *is_Inr_def*)
apply (*intro FOL_reflections function_reflections*)
done

7.1.3 The Formula *is_Nil*, Internalized

definition
Nil_fm :: *i*=>*i* **where**
Nil_fm(*x*) == *Exists*(*And*(*empty_fm*(*0*), *Inl_fm*(*0*,*succ*(*x*))))

lemma *Nil_type* [*TC*]: *x* ∈ *nat* ==> *Nil_fm*(*x*) ∈ *formula*
by (*simp add*: *Nil_fm_def*)

lemma *sats_Nil_fm* [*simp*]:
[| *x* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *Nil_fm*(*x*), *env*) ⟷ *is_Nil*(##*A*, *nth*(*x*,*env*))
by (*simp add*: *Nil_fm_def is_Nil_def*)

lemma *Nil_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *i* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_Nil*(##*A*, *x*) ⟷ *sats*(*A*, *Nil_fm*(*i*), *env*)
by *simp*

theorem *Nil_reflection*:

REFLECTS[λ*x*. *is_Nil*(*L*,*f*(*x*)),
λ*i x*. *is_Nil*(##*Lset*(*i*),*f*(*x*))]
apply (*simp only*: *is_Nil_def*)
apply (*intro FOL_reflections function_reflections Inl_reflection*)
done

7.1.4 The Formula *is_Cons*, Internalized

definition
Cons_fm :: [*i*,*i*,*i*]=>*i* **where**
Cons_fm(*a*,*l*,*Z*) ==
Exists(*And*(*pair_fm*(*succ*(*a*),*succ*(*l*),*0*), *Inr_fm*(*0*,*succ*(*Z*))))

lemma *Cons_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *Cons_fm*(*x*,*y*,*z*) ∈ *formula*
by (*simp add*: *Cons_fm_def*)

lemma *sats_Cons_fm* [*simp*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *Cons_fm*(*x*,*y*,*z*), *env*) ⟷
is_Cons(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
by (*simp add*: *Cons_fm_def is_Cons_def*)

lemma *Cons_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
i ∈ *nat*; *j* ∈ *nat*; *k* ∈ *nat*; *env* ∈ *list*(*A*)|]
==>*is_Cons*(##*A*, *x*, *y*, *z*) ⟷ *sats*(*A*, *Cons_fm*(*i*,*j*,*k*), *env*)
by *simp*

theorem *Cons_reflection*:
REFLECTS[λ*x*. *is_Cons*(*L*,*f*(*x*),*g*(*x*),*h*(*x*)),
λ*i x*. *is_Cons*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*))]
apply (*simp only*: *is_Cons_def*)
apply (*intro FOL_reflections pair_reflection Inr_reflection*)
done

7.1.5 The Formula *is_quasilist*, Internalized

definition
quasilist_fm :: *i*=>*i* **where**
quasilist_fm(*x*) ==
Or(*Nil_fm*(*x*), *Exists*(*Exists*(*Cons_fm*(*1*,*0*,*succ*(*succ*(*x*))))))

lemma *quasilist_type* [*TC*]: *x* ∈ *nat* ==> *quasilist_fm*(*x*) ∈ *formula*
by (*simp add*: *quasilist_fm_def*)

lemma *sats_quasilist_fm* [*simp*]:
[| *x* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *quasilist_fm*(*x*), *env*) ⟷ *is_quasilist*(##*A*, *nth*(*x*,*env*))
by (*simp add*: *quasilist_fm_def is_quasilist_def*)

lemma *quasilist_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *i* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_quasilist*(##*A*, *x*) ⟷ *sats*(*A*, *quasilist_fm*(*i*), *env*)
by *simp*

theorem *quasilist_reflection*:
REFLECTS[λ*x*. *is_quasilist*(*L*,*f*(*x*)),
λ*i x*. *is_quasilist*(##*Lset*(*i*),*f*(*x*))]
apply (*simp only*: *is_quasilist_def*)
apply (*intro FOL_reflections Nil_reflection Cons_reflection*)
done

7.2 Absoluteness for the Function *nth*

7.2.1 The Formula *is_hd*, Internalized

definition
hd_fm :: [*i*,*i*]=>*i* **where**
hd_fm(*xs*,*H*) ==
And(*Implies*(*Nil_fm*(*xs*), *empty_fm*(*H*)),
And(*Forall*(*Forall*(*Or*(*Neg*(*Cons_fm*(*1*,*0*,*xs*#+*2*)), *Equal*(*H*#+*2*,*1*)))),
Or(*quasilist_fm*(*xs*), *empty_fm*(*H*))))

lemma *hd_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat* |] ==> *hd_fm*(*x*,*y*) ∈ *formula*
by (*simp add*: *hd_fm_def*)

lemma *sats_hd_fm* [*simp*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *hd_fm*(*x*,*y*), *env*) ⟷ *is_hd*(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
by (*simp add*: *hd_fm_def is_hd_def*)

lemma *hd_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*;
i ∈ *nat*; *j* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_hd*(##*A*, *x*, *y*) ⟷ *sats*(*A*, *hd_fm*(*i*,*j*), *env*)
by *simp*

theorem *hd_reflection*:
REFLECTS[λ*x*. *is_hd*(*L*,*f*(*x*),*g*(*x*)),
λ*i x*. *is_hd*(##*Lset*(*i*),*f*(*x*),*g*(*x*))]
apply (*simp only*: *is_hd_def*)
apply (*intro FOL_reflections Nil_reflection Cons_reflection*
quasilist_reflection empty_reflection)
done

7.2.2 The Formula *is_tl*, Internalized

definition

```
  tl_fm :: [i,i]=>i where
    tl_fm(xs,T) ==
      And(Implies(Nil_fm(xs), Equal(T,xs)),
          And(Forall(Forall(Or(Neg(Cons_fm(1,0,xs#+2)), Equal(T#+2,0)))),
              Or(quasilist_fm(xs), empty_fm(T))))

lemma tl_type [TC]:
    [| x ∈ nat; y ∈ nat |] ==> tl_fm(x,y) ∈ formula
by (simp add: tl_fm_def)

lemma sats_tl_fm [simp]:
   [| x ∈ nat; y ∈ nat; env ∈ list(A)|]
    ==> sats(A, tl_fm(x,y), env) ⟷ is_tl(##A, nth(x,env), nth(y,env))
by (simp add: tl_fm_def is_tl_def)

lemma tl_iff_sats:
      [| nth(i,env) = x; nth(j,env) = y;
         i ∈ nat; j ∈ nat; env ∈ list(A)|]
       ==> is_tl(##A, x, y) ⟷ sats(A, tl_fm(i,j), env)
by simp

theorem tl_reflection:
    REFLECTS[λx. is_tl(L,f(x),g(x)),
             λi x. is_tl(##Lset(i),f(x),g(x))]
apply (simp only: is_tl_def)
apply (intro FOL_reflections Nil_reflection Cons_reflection
            quasilist_reflection empty_reflection)
done
```

7.2.3 The Operator *is_bool_of_o*

The formula p has no free variables.

```
definition
  bool_of_o_fm :: [i, i]=>i where
  bool_of_o_fm(p,z) ==
    Or(And(p,number1_fm(z)),
       And(Neg(p),empty_fm(z)))

lemma is_bool_of_o_type [TC]:
    [| p ∈ formula; z ∈ nat |] ==> bool_of_o_fm(p,z) ∈ formula
by (simp add: bool_of_o_fm_def)

lemma sats_bool_of_o_fm:
  assumes p_iff_sats: P ⟷ sats(A, p, env)
  shows
      [|z ∈ nat; env ∈ list(A)|]
       ==> sats(A, bool_of_o_fm(p,z), env) ⟷
           is_bool_of_o(##A, P, nth(z,env))
by (simp add: bool_of_o_fm_def is_bool_of_o_def p_iff_sats [THEN iff_sym])
```

lemma *is_bool_of_o_iff_sats*:
[| $P \longleftrightarrow sats(A,\ p,\ env)$; $nth(k,env) = z$; $k \in nat$; $env \in list(A)$|]
==> $is_bool_of_o(\#\#A,\ P,\ z) \longleftrightarrow sats(A,\ bool_of_o_fm(p,k),\ env)$
by (*simp add*: *sats_bool_of_o_fm*)

theorem *bool_of_o_reflection*:
$REFLECTS\ [P(L),\ \lambda i.\ P(\#\#Lset(i))]$ ==>
$REFLECTS[\lambda x.\ is_bool_of_o(L,\ P(L,x),\ f(x)),$
$\lambda i\ x.\ is_bool_of_o(\#\#Lset(i),\ P(\#\#Lset(i),x),\ f(x))]$
apply (*simp* (*no_asm*) *only*: *is_bool_of_o_def*)
apply (*intro FOL_reflections function_reflections*, *assumption*+)
done

7.3 More Internalizations

7.3.1 The Operator *is_lambda*

The two arguments of p are always 1, 0. Remember that p will be enclosed by three quantifiers.

definition
$lambda_fm :: [i,\ i,\ i]=>i$ **where**
$lambda_fm(p,A,z) ==$
$Forall(Iff(Member(0,succ(z)),$
$Exists(Exists(And(Member(1,A\#+3),$
$And(pair_fm(1,0,2),\ p))))))$

We call p with arguments x, y by equating them with the corresponding quantified variables with de Bruijn indices 1, 0.

lemma *is_lambda_type* [*TC*]:
[| $p \in formula$; $x \in nat$; $y \in nat$ |]
==> $lambda_fm(p,x,y) \in formula$
by (*simp add*: *lambda_fm_def*)

lemma *sats_lambda_fm*:
assumes *is_b_iff_sats*:
!!$a0\ a1\ a2$.
[|$a0 \in A$; $a1 \in A$; $a2 \in A$|]
==> $is_b(a1,\ a0) \longleftrightarrow sats(A,\ p,\ Cons(a0,Cons(a1,Cons(a2,env))))$
shows
[|$x \in nat$; $y \in nat$; $env \in list(A)$|]
==> $sats(A,\ lambda_fm(p,x,y),\ env) \longleftrightarrow$
$is_lambda(\#\#A,\ nth(x,env),\ is_b,\ nth(y,env))$
by (*simp add*: *lambda_fm_def is_lambda_def is_b_iff_sats* [*THEN iff_sym*])

theorem *is_lambda_reflection*:
assumes *is_b_reflection*:
!!$f\ g\ h$. $REFLECTS[\lambda x.\ is_b(L,\ f(x),\ g(x),\ h(x)),$
$\lambda i\ x.\ is_b(\#\#Lset(i),\ f(x),\ g(x),\ h(x))]$

shows *REFLECTS*[λx. *is_lambda*(*L*, *A*(*x*), *is_b*(*L*,*x*), *f*(*x*)),
$\lambda i\ x$. *is_lambda*(##*Lset*(*i*), *A*(*x*), *is_b*(##*Lset*(*i*),*x*), *f*(*x*))]
apply (*simp* (*no_asm_use*) *only*: *is_lambda_def*)
apply (*intro FOL_reflections is_b_reflection pair_reflection*)
done

7.3.2 The Operator *is_Member*, Internalized

definition
Member_fm :: [*i*,*i*,*i*]=>*i* **where**
Member_fm(*x*,*y*,*Z*) ==
Exists(*Exists*(*And*(*pair_fm*(*x*#+2,*y*#+2,1),
And(*Inl_fm*(1,0), *Inl_fm*(0,*Z*#+2)))))

lemma *is_Member_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *Member_fm*(*x*,*y*,*z*) ∈ *formula*
by (*simp add*: *Member_fm_def*)

lemma *sats_Member_fm* [*simp*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *Member_fm*(*x*,*y*,*z*), *env*) ⟷
is_Member(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
by (*simp add*: *Member_fm_def is_Member_def*)

lemma *Member_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
i ∈ *nat*; *j* ∈ *nat*; *k* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_Member*(##*A*, *x*, *y*, *z*) ⟷ *sats*(*A*, *Member_fm*(*i*,*j*,*k*), *env*)
by (*simp*)

theorem *Member_reflection*:
REFLECTS[λx. *is_Member*(*L*,*f*(*x*),*g*(*x*),*h*(*x*)),
$\lambda i\ x$. *is_Member*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*))]
apply (*simp only*: *is_Member_def*)
apply (*intro FOL_reflections pair_reflection Inl_reflection*)
done

7.3.3 The Operator *is_Equal*, Internalized

definition
Equal_fm :: [*i*,*i*,*i*]=>*i* **where**
Equal_fm(*x*,*y*,*Z*) ==
Exists(*Exists*(*And*(*pair_fm*(*x*#+2,*y*#+2,1),
And(*Inr_fm*(1,0), *Inl_fm*(0,*Z*#+2)))))

lemma *is_Equal_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *Equal_fm*(*x*,*y*,*z*) ∈ *formula*
by (*simp add*: *Equal_fm_def*)

lemma *sats_Equal_fm* [*simp*]:

[| $x \in nat$; $y \in nat$; $z \in nat$; $env \in list(A)$|]
==> $sats(A, Equal_fm(x,y,z), env) \longleftrightarrow$
$is_Equal(\#\#A, nth(x,env), nth(y,env), nth(z,env))$
by (*simp add*: *Equal_fm_def is_Equal_def*)

lemma *Equal_iff_sats*:
[| $nth(i,env) = x$; $nth(j,env) = y$; $nth(k,env) = z$;
$i \in nat$; $j \in nat$; $k \in nat$; $env \in list(A)$|]
==> $is_Equal(\#\#A, x, y, z) \longleftrightarrow sats(A, Equal_fm(i,j,k), env)$
by (*simp*)

theorem *Equal_reflection*:
$REFLECTS[\lambda x.\ is_Equal(L,f(x),g(x),h(x)),$
$\lambda i\ x.\ is_Equal(\#\#Lset(i),f(x),g(x),h(x))]$
apply (*simp only*: *is_Equal_def*)
apply (*intro FOL_reflections pair_reflection Inl_reflection Inr_reflection*)
done

7.3.4 The Operator *is_Nand*, Internalized

definition
$Nand_fm :: [i,i,i]=>i$ **where**
$Nand_fm(x,y,Z) ==$
$Exists(Exists(And(pair_fm(x\#+2,y\#+2,1),$
$And(Inl_fm(1,0), Inr_fm(0,Z\#+2)))))$

lemma *is_Nand_type* [*TC*]:
[| $x \in nat$; $y \in nat$; $z \in nat$ |] ==> $Nand_fm(x,y,z) \in formula$
by (*simp add*: *Nand_fm_def*)

lemma *sats_Nand_fm* [*simp*]:
[| $x \in nat$; $y \in nat$; $z \in nat$; $env \in list(A)$|]
==> $sats(A, Nand_fm(x,y,z), env) \longleftrightarrow$
$is_Nand(\#\#A, nth(x,env), nth(y,env), nth(z,env))$
by (*simp add*: *Nand_fm_def is_Nand_def*)

lemma *Nand_iff_sats*:
[| $nth(i,env) = x$; $nth(j,env) = y$; $nth(k,env) = z$;
$i \in nat$; $j \in nat$; $k \in nat$; $env \in list(A)$|]
==> $is_Nand(\#\#A, x, y, z) \longleftrightarrow sats(A, Nand_fm(i,j,k), env)$
by (*simp*)

theorem *Nand_reflection*:
$REFLECTS[\lambda x.\ is_Nand(L,f(x),g(x),h(x)),$
$\lambda i\ x.\ is_Nand(\#\#Lset(i),f(x),g(x),h(x))]$
apply (*simp only*: *is_Nand_def*)
apply (*intro FOL_reflections pair_reflection Inl_reflection Inr_reflection*)
done

7.3.5 The Operator *is_Forall*, Internalized

definition
  Forall_fm :: [*i*,*i*]=>*i* **where**
    Forall_fm(*x*,*Z*) ==
      Exists(*And*(*Inr_fm*(*succ*(*x*),*0*), *Inr_fm*(*0*,*succ*(*Z*))))

lemma *is_Forall_type* [*TC*]:
    [| *x* ∈ *nat*; *y* ∈ *nat* |] ==> *Forall_fm*(*x*,*y*) ∈ *formula*
by (*simp add*: *Forall_fm_def*)

lemma *sats_Forall_fm* [*simp*]:
  [| *x* ∈ *nat*; *y* ∈ *nat*; *env* ∈ *list*(*A*)|]
   ==> *sats*(*A*, *Forall_fm*(*x*,*y*), *env*) ⟷
      is_Forall(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
by (*simp add*: *Forall_fm_def is_Forall_def*)

lemma *Forall_iff_sats*:
    [| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*;
      i ∈ *nat*; *j* ∈ *nat*; *env* ∈ *list*(*A*)|]
     ==> *is_Forall*(##*A*, *x*, *y*) ⟷ *sats*(*A*, *Forall_fm*(*i*,*j*), *env*)
by (*simp*)

theorem *Forall_reflection*:
    REFLECTS[λ*x*. *is_Forall*(*L*,*f*(*x*),*g*(*x*)),
          λ*i x*. *is_Forall*(##*Lset*(*i*),*f*(*x*),*g*(*x*))]
apply (*simp only*: *is_Forall_def*)
apply (*intro FOL_reflections pair_reflection Inr_reflection*)
done

7.3.6 The Operator *is_and*, Internalized

definition
  and_fm :: [*i*,*i*,*i*]=>*i* **where**
    and_fm(*a*,*b*,*z*) ==
      Or(*And*(*number1_fm*(*a*), *Equal*(*z*,*b*)),
        And(*Neg*(*number1_fm*(*a*)),*empty_fm*(*z*)))

lemma *is_and_type* [*TC*]:
    [| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *and_fm*(*x*,*y*,*z*) ∈ *formula*
by (*simp add*: *and_fm_def*)

lemma *sats_and_fm* [*simp*]:
  [| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
   ==> *sats*(*A*, *and_fm*(*x*,*y*,*z*), *env*) ⟷
      is_and(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
by (*simp add*: *and_fm_def is_and_def*)

lemma *is_and_iff_sats*:
    [| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;

$i \in nat$; $j \in nat$; $k \in nat$; $env \in list(A)$|]
==> $is_and(\#\#A,\ x,\ y,\ z) \longleftrightarrow sats(A,\ and_fm(i,j,k),\ env)$
by *simp*

theorem *is_and_reflection*:
$REFLECTS[\lambda x.\ is_and(L,f(x),g(x),h(x)),$
$\lambda i\ x.\ is_and(\#\#Lset(i),f(x),g(x),h(x))]$
apply (*simp only*: *is_and_def*)
apply (*intro FOL_reflections function_reflections*)
done

7.3.7 The Operator *is_or*, Internalized

definition
$or_fm :: [i,i,i]=>i$ **where**
$or_fm(a,b,z) ==$
$Or(And(number1_fm(a),\ number1_fm(z)),$
$And(Neg(number1_fm(a)),\ Equal(z,b)))$

lemma *is_or_type* [*TC*]:
[| $x \in nat$; $y \in nat$; $z \in nat$ |] ==> $or_fm(x,y,z) \in formula$
by (*simp add*: *or_fm_def*)

lemma *sats_or_fm* [*simp*]:
[| $x \in nat$; $y \in nat$; $z \in nat$; $env \in list(A)$|]
==> $sats(A,\ or_fm(x,y,z),\ env) \longleftrightarrow$
$is_or(\#\#A,\ nth(x,env),\ nth(y,env),\ nth(z,env))$
by (*simp add*: *or_fm_def is_or_def*)

lemma *is_or_iff_sats*:
[| $nth(i,env) = x$; $nth(j,env) = y$; $nth(k,env) = z$;
$i \in nat$; $j \in nat$; $k \in nat$; $env \in list(A)$|]
==> $is_or(\#\#A,\ x,\ y,\ z) \longleftrightarrow sats(A,\ or_fm(i,j,k),\ env)$
by *simp*

theorem *is_or_reflection*:
$REFLECTS[\lambda x.\ is_or(L,f(x),g(x),h(x)),$
$\lambda i\ x.\ is_or(\#\#Lset(i),f(x),g(x),h(x))]$
apply (*simp only*: *is_or_def*)
apply (*intro FOL_reflections function_reflections*)
done

7.3.8 The Operator *is_not*, Internalized

definition
$not_fm :: [i,i]=>i$ **where**
$not_fm(a,z) ==$
$Or(And(number1_fm(a),\ empty_fm(z)),$
$And(Neg(number1_fm(a)),\ number1_fm(z)))$

lemma *is_not_type* [*TC*]:
[| $x \in nat$; $z \in nat$ |] ==> $not_fm(x,z) \in formula$
by (*simp add*: *not_fm_def*)

lemma *sats_is_not_fm* [*simp*]:
[| $x \in nat$; $z \in nat$; $env \in list(A)$|]
==> $sats(A,\ not_fm(x,z),\ env) \longleftrightarrow is_not(\#\#A,\ nth(x,env),\ nth(z,env))$
by (*simp add*: *not_fm_def is_not_def*)

lemma *is_not_iff_sats*:
[| $nth(i,env) = x$; $nth(k,env) = z$;
$i \in nat$; $k \in nat$; $env \in list(A)$|]
==> $is_not(\#\#A,\ x,\ z) \longleftrightarrow sats(A,\ not_fm(i,k),\ env)$
by *simp*

theorem *is_not_reflection*:
$REFLECTS[\lambda x.\ is_not(L,f(x),g(x)),$
$\lambda i\ x.\ is_not(\#\#Lset(i),f(x),g(x))]$
apply (*simp only*: *is_not_def*)
apply (*intro FOL_reflections function_reflections*)
done

lemmas *extra_reflections* =
Inl_reflection Inr_reflection Nil_reflection Cons_reflection
quasilist_reflection hd_reflection tl_reflection bool_of_o_reflection
is_lambda_reflection Member_reflection Equal_reflection Nand_reflection
Forall_reflection is_and_reflection is_or_reflection is_not_reflection

7.4 Well-Founded Recursion!

7.4.1 The Operator *M_is_recfun*

Alternative definition, minimizing nesting of quantifiers around MH

lemma *M_is_recfun_iff*:
$M_is_recfun(M,MH,r,a,f) \longleftrightarrow$
$(\forall z[M].\ z \in f \longleftrightarrow$
$(\exists x[M].\ \exists f_r_sx[M].\ \exists y[M].$
$MH(x,\ f_r_sx,\ y)\ \&\ pair(M,x,y,z)\ \&$
$(\exists xa[M].\ \exists sx[M].\ \exists r_sx[M].$
$pair(M,x,a,xa)\ \&\ upair(M,x,x,sx)\ \&$
$pre_image(M,r,sx,r_sx)\ \&\ restriction(M,f,r_sx,f_r_sx)\ \&$
$xa \in r)))$
apply (*simp add*: *M_is_recfun_def*)
apply (*rule rall_cong*, *blast*)
done

The three arguments of p are always 2, 1, 0 and z

definition

```
is_recfun_fm :: [i, i, i, i]=>i where
is_recfun_fm(p,r,a,f) ==
 Forall(Iff(Member(0,succ(f)),
  Exists(Exists(Exists(
   And(p,
    And(pair_fm(2,0,3),
     Exists(Exists(Exists(
      And(pair_fm(5,a#+7,2),
       And(upair_fm(5,5,1),
        And(pre_image_fm(r#+7,1,0),
         And(restriction_fm(f#+7,0,4), Member(2,r#+7))))))))))))))
```

lemma *is_recfun_type* [*TC*]:
```
    [| p ∈ formula; x ∈ nat; y ∈ nat; z ∈ nat |]
     ==> is_recfun_fm(p,x,y,z) ∈ formula
```
by (*simp add*: *is_recfun_fm_def*)

lemma *sats_is_recfun_fm*:
```
  assumes MH_iff_sats:
      !!a0 a1 a2 a3.
        [|a0∈A; a1∈A; a2∈A; a3∈A|]
     ==> MH(a2, a1, a0) ⟷ sats(A, p, Cons(a0,Cons(a1,Cons(a2,Cons(a3,env)))))
  shows
      [|x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)|]
       ==> sats(A, is_recfun_fm(p,x,y,z), env) ⟷
           M_is_recfun(##A, MH, nth(x,env), nth(y,env), nth(z,env))
```
by (*simp add*: *is_recfun_fm_def M_is_recfun_iff MH_iff_sats* [*THEN iff_sym*])

lemma *is_recfun_iff_sats*:
```
  assumes MH_iff_sats:
      !!a0 a1 a2 a3.
        [|a0∈A; a1∈A; a2∈A; a3∈A|]
     ==> MH(a2, a1, a0) ⟷ sats(A, p, Cons(a0,Cons(a1,Cons(a2,Cons(a3,env)))))
  shows
  [| nth(i,env) = x; nth(j,env) = y; nth(k,env) = z;
      i ∈ nat; j ∈ nat; k ∈ nat; env ∈ list(A)|]
   ==> M_is_recfun(##A, MH, x, y, z) ⟷ sats(A, is_recfun_fm(p,i,j,k), env)
```
by (*simp add*: *sats_is_recfun_fm* [*OF MH_iff_sats*])

The additional variable in the premise, namely f', is essential. It lets *MH* depend upon x, which seems often necessary. The same thing occurs in *is_wfrec_reflection*.

theorem *is_recfun_reflection*:
```
  assumes MH_reflection:
    !!f' f g h. REFLECTS[λx. MH(L, f'(x), f(x), g(x), h(x)),
                         λi x. MH(##Lset(i), f'(x), f(x), g(x), h(x))]
  shows REFLECTS[λx. M_is_recfun(L, MH(L,x), f(x), g(x), h(x)),
                 λi x. M_is_recfun(##Lset(i), MH(##Lset(i),x), f(x), g(x), h(x))]
```

apply (*simp* (*no_asm_use*) *only*: *M_is_recfun_def*)
apply (*intro FOL_reflections function_reflections*
restriction_reflection MH_reflection)
done

7.4.2 The Operator *is_wfrec*

The three arguments of p are always 2, 1, 0; p is enclosed by 5 quantifiers.

definition
is_wfrec_fm :: [*i*, *i*, *i*, *i*]=>*i* **where**
is_wfrec_fm(*p*,*r*,*a*,*z*) ==
Exists(*And*(*is_recfun_fm*(*p*, *succ*(*r*), *succ*(*a*), *0*),
Exists(*Exists*(*Exists*(*Exists*(
And(*Equal*(*2*,*a*#+*5*), *And*(*Equal*(*1*,*4*), *And*(*Equal*(*0*,*z*#+*5*), *p*))))))))))

We call p with arguments a, f, z by equating them with the corresponding quantified variables with de Bruijn indices 2, 1, 0.

There's an additional existential quantifier to ensure that the environments in both calls to MH have the same length.

lemma *is_wfrec_type* [*TC*]:
[| *p* ∈ *formula*; *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |]
==> *is_wfrec_fm*(*p*,*x*,*y*,*z*) ∈ *formula*
by (*simp add*: *is_wfrec_fm_def*)

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* < *length*(*env*); *z* < *length*(*env*); *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *is_wfrec_fm*(*p*,*x*,*y*,*z*), *env*) ⟷
is_wfrec(##*A*, *MH*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
apply (*frule_tac x*=*z* **in** *lt_length_in_nat*, *assumption*)
apply (*frule lt_length_in_nat*, *assumption*)
apply (*simp add*: *is_wfrec_fm_def sats_is_recfun_fm is_wfrec_def MH_iff_sats*
[*THEN iff_sym*], *blast*)
done

lemma *is_wfrec_iff_sats*:
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
[|*nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
i ∈ *nat*; *j* < *length*(*env*); *k* < *length*(*env*); *env* ∈ *list*(*A*)|]

==> *is_wfrec*(##*A*, *MH*, *x*, *y*, *z*) ⟷ *sats*(*A*, *is_wfrec_fm*(*p*,*i*,*j*,*k*), *env*)
by (*simp add*: *sats_is_wfrec_fm* [*OF MH_iff_sats*])

theorem *is_wfrec_reflection*:
assumes *MH_reflection*:
!!*f′ f g h*. *REFLECTS*[λ*x*. *MH*(*L*, *f′*(*x*), *f*(*x*), *g*(*x*), *h*(*x*)),
λ*i x*. *MH*(##*Lset*(*i*), *f′*(*x*), *f*(*x*), *g*(*x*), *h*(*x*))]
shows *REFLECTS*[λ*x*. *is_wfrec*(*L*, *MH*(*L*,*x*), *f*(*x*), *g*(*x*), *h*(*x*)),
λ*i x*. *is_wfrec*(##*Lset*(*i*), *MH*(##*Lset*(*i*),*x*), *f*(*x*), *g*(*x*), *h*(*x*))]
apply (*simp* (*no_asm_use*) *only*: *is_wfrec_def*)
apply (*intro FOL_reflections MH_reflection is_recfun_reflection*)
done

7.5 For Datatypes

7.5.1 Binary Products, Internalized

definition
cartprod_fm :: [*i*,*i*,*i*]=>*i* **where**

cartprod_fm(*A*,*B*,*z*) ==
Forall(*Iff*(*Member*(*0*,*succ*(*z*)),
Exists(*And*(*Member*(*0*,*succ*(*succ*(*A*))),
Exists(*And*(*Member*(*0*,*succ*(*succ*(*succ*(*B*)))),
pair_fm(*1*,*0*,*2*)))))))

lemma *cartprod_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *cartprod_fm*(*x*,*y*,*z*) ∈ *formula*
by (*simp add*: *cartprod_fm_def*)

lemma *sats_cartprod_fm* [*simp*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *cartprod_fm*(*x*,*y*,*z*), *env*) ⟷
cartprod(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
by (*simp add*: *cartprod_fm_def cartprod_def*)

lemma *cartprod_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
i ∈ *nat*; *j* ∈ *nat*; *k* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *cartprod*(##*A*, *x*, *y*, *z*) ⟷ *sats*(*A*, *cartprod_fm*(*i*,*j*,*k*), *env*)
by (*simp*)

theorem *cartprod_reflection*:
REFLECTS[λ*x*. *cartprod*(*L*,*f*(*x*),*g*(*x*),*h*(*x*)),
λ*i x*. *cartprod*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*))]
apply (*simp only*: *cartprod_def*)
apply (*intro FOL_reflections pair_reflection*)
done

7.5.2 Binary Sums, Internalized

definition
sum_fm :: [*i*,*i*,*i*]=>*i* **where**
sum_fm(*A*,*B*,*Z*) ==
Exists(*Exists*(*Exists*(*Exists*(
And(*number1_fm*(*2*),
And(*cartprod_fm*(*2*,*A*#+*4*,*3*),
And(*upair_fm*(*2*,*2*,*1*),
And(*cartprod_fm*(*1*,*B*#+*4*,*0*), *union_fm*(*3*,*0*,*Z*#+*4*))))))))

lemma *sum_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *sum_fm*(*x*,*y*,*z*) ∈ *formula*
by (*simp add*: *sum_fm_def*)

lemma *sats_sum_fm* [*simp*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *sum_fm*(*x*,*y*,*z*), *env*) ⟷
is_sum(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
by (*simp add*: *sum_fm_def is_sum_def*)

lemma *sum_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
i ∈ *nat*; *j* ∈ *nat*; *k* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_sum*(##*A*, *x*, *y*, *z*) ⟷ *sats*(*A*, *sum_fm*(*i*,*j*,*k*), *env*)
by *simp*

theorem *sum_reflection*:
REFLECTS[λ*x*. *is_sum*(*L*,*f*(*x*),*g*(*x*),*h*(*x*)),
λ*i x*. *is_sum*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*))]
apply (*simp only*: *is_sum_def*)
apply (*intro FOL_reflections function_reflections cartprod_reflection*)
done

7.5.3 The Operator *quasinat*

definition
quasinat_fm :: *i*=>*i* **where**
quasinat_fm(*z*) == *Or*(*empty_fm*(*z*), *Exists*(*succ_fm*(*0*,*succ*(*z*))))

lemma *quasinat_type* [*TC*]:
x ∈ *nat* ==> *quasinat_fm*(*x*) ∈ *formula*
by (*simp add*: *quasinat_fm_def*)

lemma *sats_quasinat_fm* [*simp*]:
[| *x* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *quasinat_fm*(*x*), *env*) ⟷ *is_quasinat*(##*A*, *nth*(*x*,*env*))
by (*simp add*: *quasinat_fm_def is_quasinat_def*)

lemma *quasinat_iff_sats*:

[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*;
i ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_quasinat*(##*A*, *x*) ⟷ *sats*(*A*, *quasinat_fm*(*i*), *env*)
by *simp*

theorem *quasinat_reflection*:
REFLECTS[λ*x*. *is_quasinat*(*L*,*f*(*x*)),
λ*i x*. *is_quasinat*(##*Lset*(*i*),*f*(*x*))]
apply (*simp only*: *is_quasinat_def*)
apply (*intro FOL_reflections function_reflections*)
done

7.5.4 The Operator *is_nat_case*

I could not get it to work with the more natural assumption that *is_b* takes two arguments. Instead it must be a formula where 1 and 0 stand for *m* and *b*, respectively.

The formula *is_b* has free variables 1 and 0.

definition
is_nat_case_fm :: [*i*, *i*, *i*, *i*]=>*i* **where**
is_nat_case_fm(*a*,*is_b*,*k*,*z*) ==
And(*Implies*(*empty_fm*(*k*), *Equal*(*z*,*a*)),
And(*Forall*(*Implies*(*succ_fm*(*0*,*succ*(*k*)),
Forall(*Implies*(*Equal*(*0*,*succ*(*succ*(*z*))), *is_b*)))),
Or(*quasinat_fm*(*k*), *empty_fm*(*z*))))

lemma *is_nat_case_type* [*TC*]:
[| *is_b* ∈ *formula*;
x ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |]
==> *is_nat_case_fm*(*x*,*is_b*,*y*,*z*) ∈ *formula*
by (*simp add*: *is_nat_case_fm_def*)

lemma *sats_is_nat_case_fm*:
assumes *is_b_iff_sats*:
!!*a*. *a* ∈ *A* ==> *is_b*(*a*,*nth*(*z*, *env*)) ⟷
sats(*A*, *p*, *Cons*(*nth*(*z*,*env*), *Cons*(*a*, *env*)))
shows
[|*x* ∈ *nat*; *y* ∈ *nat*; *z* < *length*(*env*); *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *is_nat_case_fm*(*x*,*p*,*y*,*z*), *env*) ⟷
is_nat_case(##*A*, *nth*(*x*,*env*), *is_b*, *nth*(*y*,*env*), *nth*(*z*,*env*))
apply (*frule lt_length_in_nat*, *assumption*)
apply (*simp add*: *is_nat_case_fm_def is_nat_case_def is_b_iff_sats* [*THEN iff_sym*])
done

lemma *is_nat_case_iff_sats*:
[| (!!*a*. *a* ∈ *A* ==> *is_b*(*a*,*z*) ⟷
sats(*A*, *p*, *Cons*(*z*, *Cons*(*a*,*env*))));

$nth(i,env) = x$; $nth(j,env) = y$; $nth(k,env) = z$;
$i \in nat$; $j \in nat$; $k < length(env)$; $env \in list(A)|]$
==> $is_nat_case(\#\#A,\ x,\ is_b,\ y,\ z) \longleftrightarrow sats(A,\ is_nat_case_fm(i,p,j,k),$
$env)$
by (*simp add*: *sats_is_nat_case_fm* [*of A is_b*])

The second argument of *is_b* gives it direct access to x, which is essential for handling free variable references. Without this argument, we cannot prove reflection for *iterates_MH*.

theorem *is_nat_case_reflection*:
assumes *is_b_reflection*:
$!!h\ f\ g.\ REFLECTS[\lambda x.\ is_b(L,\ h(x),\ f(x),\ g(x)),$
$\lambda i\ x.\ is_b(\#\#Lset(i),\ h(x),\ f(x),\ g(x))]$
shows $REFLECTS[\lambda x.\ is_nat_case(L,\ f(x),\ is_b(L,x),\ g(x),\ h(x)),$
$\lambda i\ x.\ is_nat_case(\#\#Lset(i),\ f(x),\ is_b(\#\#Lset(i),\ x),\ g(x),\ h(x))]$
apply (*simp* (*no_asm_use*) *only*: *is_nat_case_def*)
apply (*intro FOL_reflections function_reflections*
restriction_reflection is_b_reflection quasinat_reflection)
done

7.6 The Operator *iterates_MH*, Needed for Iteration

definition
iterates_MH_fm :: $[i,\ i,\ i,\ i,\ i]$=>i **where**
$iterates_MH_fm(isF,v,n,g,z) ==$
$is_nat_case_fm(v,$
$Exists(And(fun_apply_fm(succ(succ(succ(g))),2,0),$
$Forall(Implies(Equal(0,2),\ isF)))),$
$n,\ z)$

lemma *iterates_MH_type* [*TC*]:
$[|\ p \in formula;$
$v \in nat;\ x \in nat;\ y \in nat;\ z \in nat\ |]$
==> $iterates_MH_fm(p,v,x,y,z) \in formula$
by (*simp add*: *iterates_MH_fm_def*)

lemma *sats_iterates_MH_fm*:
assumes *is_F_iff_sats*:
$!!a\ b\ c\ d.\ [|\ a \in A;\ b \in A;\ c \in A;\ d \in A|]$
==> $is_F(a,b) \longleftrightarrow$
$sats(A,\ p,\ Cons(b,\ Cons(a,\ Cons(c,\ Cons(d,env)))))$
shows
$[|v \in nat;\ x \in nat;\ y \in nat;\ z < length(env);\ env \in list(A)|]$
==> $sats(A,\ iterates_MH_fm(p,v,x,y,z),\ env) \longleftrightarrow$
$iterates_MH(\#\#A,\ is_F,\ nth(v,env),\ nth(x,env),\ nth(y,env),\ nth(z,env))$
apply (*frule lt_length_in_nat*, *assumption*)
apply (*simp add*: *iterates_MH_fm_def iterates_MH_def sats_is_nat_case_fm*
is_F_iff_sats [*symmetric*])
apply (*rule is_nat_case_cong*)

apply (*simp_all add*: *setclass_def*)
done

lemma *iterates_MH_iff_sats*:
assumes *is_F_iff_sats*:
!!*a b c d*. [| $a \in A$; $b \in A$; $c \in A$; $d \in A$|]
==> *is_F*(*a*,*b*) $\longleftrightarrow$
sats(*A*, *p*, *Cons*(*b*, *Cons*(*a*, *Cons*(*c*, *Cons*(*d*,*env*)))))
shows
[| *nth*(*i'*,*env*) = *v*; *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
$i' \in nat$; $i \in nat$; $j \in nat$; $k < length(env)$; $env \in list(A)$|]
==> *iterates_MH*(##*A*, *is_F*, *v*, *x*, *y*, *z*) $\longleftrightarrow$
sats(*A*, *iterates_MH_fm*(*p*,*i'*,*i*,*j*,*k*), *env*)
by (*simp add*: *sats_iterates_MH_fm* [*OF is_F_iff_sats*])

The second argument of *p* gives it direct access to *x*, which is essential for handling free variable references. Without this argument, we cannot prove reflection for *list_N*.

theorem *iterates_MH_reflection*:
assumes *p_reflection*:
!!*f g h*. *REFLECTS*[λx. *p*(*L*, *h*(*x*), *f*(*x*), *g*(*x*)),
λi *x*. *p*(##*Lset*(*i*), *h*(*x*), *f*(*x*), *g*(*x*))]
shows *REFLECTS*[λx. *iterates_MH*(*L*, *p*(*L*,*x*), *e*(*x*), *f*(*x*), *g*(*x*), *h*(*x*)),
λi *x*. *iterates_MH*(##*Lset*(*i*), *p*(##*Lset*(*i*),*x*), *e*(*x*), *f*(*x*), *g*(*x*), *h*(*x*))]
apply (*simp* (*no_asm_use*) *only*: *iterates_MH_def*)
apply (*intro FOL_reflections function_reflections is_nat_case_reflection*
restriction_reflection p_reflection)
done

7.6.1 The Operator *is_iterates*

The three arguments of *p* are always 2, 1, 0; *p* is enclosed by 9 (??) quantifiers.

definition
is_iterates_fm :: [*i*, *i*, *i*, *i*]=>*i* **where**
is_iterates_fm(*p*,*v*,*n*,*Z*) ==
Exists(*Exists*(
And(*succ_fm*(*n*#+*2*,*1*),
And(*Memrel_fm*(*1*,*0*),
is_wfrec_fm(*iterates_MH_fm*(*p*, *v*#+*7*, *2*, *1*, *0*),
0, *n*#+*2*, *Z*#+*2*)))))

We call *p* with arguments a, f, z by equating them with the corresponding quantified variables with de Bruijn indices 2, 1, 0.

lemma *is_iterates_type* [*TC*]:
[| $p \in formula$; $x \in nat$; $y \in nat$; $z \in nat$ |]
==> *is_iterates_fm*(*p*,*x*,*y*,*z*) $\in$ *formula*
by (*simp add*: *is_iterates_fm_def*)

lemma *sats_is_iterates_fm*:
  assumes *is_F_iff_sats*:
    !!*a b c d e f g h i j k*.
      [| $a \in A$; $b \in A$; $c \in A$; $d \in A$; $e \in A$; $f \in A$;
        $g \in A$; $h \in A$; $i \in A$; $j \in A$; $k \in A$|]
      ==> *is_F*(*a*,*b*) $\longleftrightarrow$
        sats(*A*, *p*, *Cons*(*b*, *Cons*(*a*, *Cons*(*c*, *Cons*(*d*, *Cons*(*e*, *Cons*(*f*,
          Cons(*g*, *Cons*(*h*, *Cons*(*i*, *Cons*(*j*, *Cons*(*k*, *env*)))))))))))))
  shows
    [|$x \in$ *nat*; $y <$ *length*(*env*); $z <$ *length*(*env*); *env* $\in$ *list*(*A*)|]
     ==> *sats*(*A*, *is_iterates_fm*(*p*,*x*,*y*,*z*), *env*) $\longleftrightarrow$
      is_iterates(##*A*, *is_F*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
apply (*frule_tac x=z* **in** *lt_length_in_nat*, *assumption*)
apply (*frule lt_length_in_nat*, *assumption*)
apply (*simp add*: *is_iterates_fm_def is_iterates_def sats_is_nat_case_fm*
      is_F_iff_sats [*symmetric*] *sats_is_wfrec_fm sats_iterates_MH_fm*)
done

lemma *is_iterates_iff_sats*:
  assumes *is_F_iff_sats*:
    !!*a b c d e f g h i j k*.
      [| $a \in A$; $b \in A$; $c \in A$; $d \in A$; $e \in A$; $f \in A$;
        $g \in A$; $h \in A$; $i \in A$; $j \in A$; $k \in A$|]
      ==> *is_F*(*a*,*b*) $\longleftrightarrow$
        sats(*A*, *p*, *Cons*(*b*, *Cons*(*a*, *Cons*(*c*, *Cons*(*d*, *Cons*(*e*, *Cons*(*f*,
          Cons(*g*, *Cons*(*h*, *Cons*(*i*, *Cons*(*j*, *Cons*(*k*, *env*)))))))))))))
  shows
  [| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
    $i \in$ *nat*; $j <$ *length*(*env*); $k <$ *length*(*env*); *env* $\in$ *list*(*A*)|]
   ==> *is_iterates*(##*A*, *is_F*, *x*, *y*, *z*) $\longleftrightarrow$
    sats(*A*, *is_iterates_fm*(*p*,*i*,*j*,*k*), *env*)
by (*simp add*: *sats_is_iterates_fm* [*OF is_F_iff_sats*])

The second argument of p gives it direct access to x, which is essential for handling free variable references. Without this argument, we cannot prove reflection for *list_N*.

theorem *is_iterates_reflection*:
  assumes *p_reflection*:
   !!*f g h*. *REFLECTS*[λx. *p*(*L*, *h*(*x*), *f*(*x*), *g*(*x*)),
                $\lambda i\ x$. *p*(##*Lset*(*i*), *h*(*x*), *f*(*x*), *g*(*x*))]
 shows *REFLECTS*[λx. *is_iterates*(*L*, *p*(*L*,*x*), *f*(*x*), *g*(*x*), *h*(*x*)),
            $\lambda i\ x$. *is_iterates*(##*Lset*(*i*), *p*(##*Lset*(*i*),*x*), *f*(*x*), *g*(*x*), *h*(*x*))]
apply (*simp* (*no_asm_use*) *only*: *is_iterates_def*)
apply (*intro FOL_reflections function_reflections p_reflection*
      is_wfrec_reflection iterates_MH_reflection)
done

7.6.2 The Formula *is_eclose_n*, Internalized

definition
eclose_n_fm :: [*i*,*i*,*i*]=>*i* **where**
eclose_n_fm(*A*,*n*,*Z*) == *is_iterates_fm*(*big_union_fm*(*1*,*0*), *A*, *n*, *Z*)

lemma *eclose_n_fm_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *eclose_n_fm*(*x*,*y*,*z*) ∈ *formula*
by (*simp add*: *eclose_n_fm_def*)

lemma *sats_eclose_n_fm* [*simp*]:
[| *x* ∈ *nat*; *y* < *length*(*env*); *z* < *length*(*env*); *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *eclose_n_fm*(*x*,*y*,*z*), *env*) ⟷
is_eclose_n(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
apply (*frule_tac x=z* **in** *lt_length_in_nat*, *assumption*)
apply (*frule_tac x=y* **in** *lt_length_in_nat*, *assumption*)
apply (*simp add*: *eclose_n_fm_def is_eclose_n_def*
sats_is_iterates_fm)
done

lemma *eclose_n_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
i ∈ *nat*; *j* < *length*(*env*); *k* < *length*(*env*); *env* ∈ *list*(*A*)|]
==> *is_eclose_n*(##*A*, *x*, *y*, *z*) ⟷ *sats*(*A*, *eclose_n_fm*(*i*,*j*,*k*), *env*)
by (*simp*)

theorem *eclose_n_reflection*:
REFLECTS[λ*x*. *is_eclose_n*(*L*, *f*(*x*), *g*(*x*), *h*(*x*)),
λ*i x*. *is_eclose_n*(##*Lset*(*i*), *f*(*x*), *g*(*x*), *h*(*x*))]
apply (*simp only*: *is_eclose_n_def*)
apply (*intro FOL_reflections function_reflections is_iterates_reflection*)
done

7.6.3 Membership in *eclose*(*A*)

definition
mem_eclose_fm :: [*i*,*i*]=>*i* **where**
mem_eclose_fm(*x*,*y*) ==
Exists(*Exists*(
And(*finite_ordinal_fm*(*1*),
And(*eclose_n_fm*(*x*#+*2*,*1*,*0*), *Member*(*y*#+*2*,*0*)))))

lemma *mem_eclose_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat* |] ==> *mem_eclose_fm*(*x*,*y*) ∈ *formula*
by (*simp add*: *mem_eclose_fm_def*)

lemma *sats_mem_eclose_fm* [*simp*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *mem_eclose_fm*(*x*,*y*), *env*) ⟷ *mem_eclose*(##*A*, *nth*(*x*,*env*),
nth(*y*,*env*))

by (*simp add*: *mem_eclose_fm_def mem_eclose_def*)

lemma *mem_eclose_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*;
i ∈ *nat*; *j* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *mem_eclose*(##*A*, *x*, *y*) ⟷ *sats*(*A*, *mem_eclose_fm*(*i*,*j*), *env*)
by *simp*

theorem *mem_eclose_reflection*:
REFLECTS[λ*x*. *mem_eclose*(*L*,*f*(*x*),*g*(*x*)),
λ*i x*. *mem_eclose*(##*Lset*(*i*),*f*(*x*),*g*(*x*))]
apply (*simp only*: *mem_eclose_def*)
apply (*intro FOL_reflections finite_ordinal_reflection eclose_n_reflection*)
done

7.6.4 The Predicate "Is *eclose*(*A*)"

definition
is_eclose_fm :: [*i*,*i*]=>*i* **where**
is_eclose_fm(*A*,*Z*) ==
Forall(*Iff*(*Member*(*0*,*succ*(*Z*)), *mem_eclose_fm*(*succ*(*A*),*0*)))

lemma *is_eclose_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat* |] ==> *is_eclose_fm*(*x*,*y*) ∈ *formula*
by (*simp add*: *is_eclose_fm_def*)

lemma *sats_is_eclose_fm* [*simp*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *is_eclose_fm*(*x*,*y*), *env*) ⟷ *is_eclose*(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
by (*simp add*: *is_eclose_fm_def is_eclose_def*)

lemma *is_eclose_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*;
i ∈ *nat*; *j* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_eclose*(##*A*, *x*, *y*) ⟷ *sats*(*A*, *is_eclose_fm*(*i*,*j*), *env*)
by *simp*

theorem *is_eclose_reflection*:
REFLECTS[λ*x*. *is_eclose*(*L*,*f*(*x*),*g*(*x*)),
λ*i x*. *is_eclose*(##*Lset*(*i*),*f*(*x*),*g*(*x*))]
apply (*simp only*: *is_eclose_def*)
apply (*intro FOL_reflections mem_eclose_reflection*)
done

7.6.5 The List Functor, Internalized

definition
list_functor_fm :: [*i*,*i*,*i*]=>*i* **where**

list_functor_fm(*A*,*X*,*Z*) ==

```
    Exists(Exists(
     And(number1_fm(1),
        And(cartprod_fm(A#+2,X#+2,0), sum_fm(1,0,Z#+2)))))
```

lemma *list_functor_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *list_functor_fm*(*x*,*y*,*z*) ∈ *formula*
by (*simp add*: *list_functor_fm_def*)

lemma *sats_list_functor_fm* [*simp*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *list_functor_fm*(*x*,*y*,*z*), *env*) ⟷
is_list_functor(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
by (*simp add*: *list_functor_fm_def is_list_functor_def*)

lemma *list_functor_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
i ∈ *nat*; *j* ∈ *nat*; *k* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_list_functor*(##*A*, *x*, *y*, *z*) ⟷ *sats*(*A*, *list_functor_fm*(*i*,*j*,*k*), *env*)
by *simp*

theorem *list_functor_reflection*:
REFLECTS[λ*x*. *is_list_functor*(*L*,*f*(*x*),*g*(*x*),*h*(*x*)),
λ*i x*. *is_list_functor*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*))]
apply (*simp only*: *is_list_functor_def*)
apply (*intro FOL_reflections number1_reflection*
cartprod_reflection sum_reflection)
done

7.6.6 The Formula *is_list_N*, Internalized

definition
list_N_fm :: [*i*,*i*,*i*]=>*i* **where**
list_N_fm(*A*,*n*,*Z*) ==
Exists(
And(*empty_fm*(*0*),
is_iterates_fm(*list_functor_fm*(*A*#+*9*#+*3*,*1*,*0*), *0*, *n*#+*1*, *Z*#+*1*)))

lemma *list_N_fm_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *list_N_fm*(*x*,*y*,*z*) ∈ *formula*
by (*simp add*: *list_N_fm_def*)

lemma *sats_list_N_fm* [*simp*]:
[| *x* ∈ *nat*; *y* < *length*(*env*); *z* < *length*(*env*); *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *list_N_fm*(*x*,*y*,*z*), *env*) ⟷
is_list_N(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
apply (*frule_tac x*=*z* **in** *lt_length_in_nat*, *assumption*)
apply (*frule_tac x*=*y* **in** *lt_length_in_nat*, *assumption*)
apply (*simp add*: *list_N_fm_def is_list_N_def sats_is_iterates_fm*)
done

lemma *list_N_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
i ∈ *nat*; *j* < *length*(*env*); *k* < *length*(*env*); *env* ∈ *list*(*A*)|]
==> *is_list_N*(##*A*, *x*, *y*, *z*) ⟷ *sats*(*A*, *list_N_fm*(*i*,*j*,*k*), *env*)
by (*simp*)

theorem *list_N_reflection*:
REFLECTS[λ*x*. *is_list_N*(*L*, *f*(*x*), *g*(*x*), *h*(*x*)),
λ*i x*. *is_list_N*(##*Lset*(*i*), *f*(*x*), *g*(*x*), *h*(*x*))]
apply (*simp only*: *is_list_N_def*)
apply (*intro FOL_reflections function_reflections*
is_iterates_reflection list_functor_reflection)
done

7.6.7 The Predicate "Is A List"

definition
mem_list_fm :: [*i*,*i*]=>*i* **where**
mem_list_fm(*x*,*y*) ==
Exists(*Exists*(
And(*finite_ordinal_fm*(*1*),
And(*list_N_fm*(*x*#+*2*,*1*,*0*), *Member*(*y*#+*2*,*0*)))))

lemma *mem_list_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat* |] ==> *mem_list_fm*(*x*,*y*) ∈ *formula*
by (*simp add*: *mem_list_fm_def*)

lemma *sats_mem_list_fm* [*simp*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *mem_list_fm*(*x*,*y*), *env*) ⟷ *mem_list*(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
by (*simp add*: *mem_list_fm_def mem_list_def*)

lemma *mem_list_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*;
i ∈ *nat*; *j* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *mem_list*(##*A*, *x*, *y*) ⟷ *sats*(*A*, *mem_list_fm*(*i*,*j*), *env*)
by *simp*

theorem *mem_list_reflection*:
REFLECTS[λ*x*. *mem_list*(*L*,*f*(*x*),*g*(*x*)),
λ*i x*. *mem_list*(##*Lset*(*i*),*f*(*x*),*g*(*x*))]
apply (*simp only*: *mem_list_def*)
apply (*intro FOL_reflections finite_ordinal_reflection list_N_reflection*)
done

7.6.8 The Predicate "Is *list*(*A*)"

definition
is_list_fm :: [*i*,*i*]=>*i* **where**

is_list_fm(*A*,*Z*) ==
Forall(*Iff*(*Member*(*0*,*succ*(*Z*)), *mem_list_fm*(*succ*(*A*),*0*)))

lemma *is_list_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat* |] ==> *is_list_fm*(*x*,*y*) ∈ *formula*
by (*simp add*: *is_list_fm_def*)

lemma *sats_is_list_fm* [*simp*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *is_list_fm*(*x*,*y*), *env*) ⟷ *is_list*(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
by (*simp add*: *is_list_fm_def is_list_def*)

lemma *is_list_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*;
i ∈ *nat*; *j* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_list*(##*A*, *x*, *y*) ⟷ *sats*(*A*, *is_list_fm*(*i*,*j*), *env*)
by *simp*

theorem *is_list_reflection*:
REFLECTS[λ*x*. *is_list*(*L*,*f*(*x*),*g*(*x*)),
λ*i x*. *is_list*(##*Lset*(*i*),*f*(*x*),*g*(*x*))]
apply (*simp only*: *is_list_def*)
apply (*intro FOL_reflections mem_list_reflection*)
done

7.6.9 The Formula Functor, Internalized

definition *formula_functor_fm* :: [*i*,*i*]=>*i* **where**

formula_functor_fm(*X*,*Z*) ==
Exists(*Exists*(*Exists*(*Exists*(*Exists*(
And(*omega_fm*(*4*),
And(*cartprod_fm*(*4*,*4*,*3*),
And(*sum_fm*(*3*,*3*,*2*),
And(*cartprod_fm*(*X*#+*5*,*X*#+*5*,*1*),
And(*sum_fm*(*1*,*X*#+*5*,*0*), *sum_fm*(*2*,*0*,*Z*#+*5*))))))))))

lemma *formula_functor_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat* |] ==> *formula_functor_fm*(*x*,*y*) ∈ *formula*
by (*simp add*: *formula_functor_fm_def*)

lemma *sats_formula_functor_fm* [*simp*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *formula_functor_fm*(*x*,*y*), *env*) ⟷
is_formula_functor(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
by (*simp add*: *formula_functor_fm_def is_formula_functor_def*)

lemma *formula_functor_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*;

$i \in nat$; $j \in nat$; $env \in list(A)$|]
==> *is_formula_functor*(##*A*, *x*, *y*) ⟷ *sats*(*A*, *formula_functor_fm*(*i*,*j*), *env*)
by *simp*

theorem *formula_functor_reflection*:
REFLECTS[λ*x*. *is_formula_functor*(*L*,*f*(*x*),*g*(*x*)),
λ*i x*. *is_formula_functor*(##*Lset*(*i*),*f*(*x*),*g*(*x*))]
apply (*simp only*: *is_formula_functor_def*)
apply (*intro FOL_reflections omega_reflection*
cartprod_reflection sum_reflection)
done

7.6.10 The Formula *is_formula_N*, Internalized

definition
formula_N_fm :: [*i*,*i*]=>*i* **where**
formula_N_fm(*n*,*Z*) ==
Exists(
And(*empty_fm*(*0*),
is_iterates_fm(*formula_functor_fm*(*1*,*0*), *0*, *n*#+*1*, *Z*#+*1*)))

lemma *formula_N_fm_type* [*TC*]:
[| $x \in nat$; $y \in nat$ |] ==> *formula_N_fm*(*x*,*y*) ∈ *formula*
by (*simp add*: *formula_N_fm_def*)

lemma *sats_formula_N_fm* [*simp*]:
[| $x < length(env)$; $y < length(env)$; $env \in list(A)$|]
==> *sats*(*A*, *formula_N_fm*(*x*,*y*), *env*) ⟷
is_formula_N(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
apply (*frule_tac x*=*y* **in** *lt_length_in_nat*, *assumption*)
apply (*frule lt_length_in_nat*, *assumption*)
apply (*simp add*: *formula_N_fm_def is_formula_N_def sats_is_iterates_fm*)
done

lemma *formula_N_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*;
$i < length(env)$; $j < length(env)$; $env \in list(A)$|]
==> *is_formula_N*(##*A*, *x*, *y*) ⟷ *sats*(*A*, *formula_N_fm*(*i*,*j*), *env*)
by (*simp*)

theorem *formula_N_reflection*:
REFLECTS[λ*x*. *is_formula_N*(*L*, *f*(*x*), *g*(*x*)),
λ*i x*. *is_formula_N*(##*Lset*(*i*), *f*(*x*), *g*(*x*))]
apply (*simp only*: *is_formula_N_def*)
apply (*intro FOL_reflections function_reflections*
is_iterates_reflection formula_functor_reflection)
done

7.6.11 The Predicate "Is A Formula"

definition
mem_formula_fm :: *i*=>*i* **where**
mem_formula_fm(*x*) ==
Exists(*Exists*(
And(*finite_ordinal_fm*(*1*),
And(*formula_N_fm*(*1*,*0*), *Member*(*x*#+*2*,*0*)))))

lemma *mem_formula_type* [*TC*]:
x ∈ *nat* ==> *mem_formula_fm*(*x*) ∈ *formula*
by (*simp add*: *mem_formula_fm_def*)

lemma *sats_mem_formula_fm* [*simp*]:
[| *x* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *mem_formula_fm*(*x*), *env*) ⟷ *mem_formula*(##*A*, *nth*(*x*,*env*))
by (*simp add*: *mem_formula_fm_def mem_formula_def*)

lemma *mem_formula_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *i* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *mem_formula*(##*A*, *x*) ⟷ *sats*(*A*, *mem_formula_fm*(*i*), *env*)
by *simp*

theorem *mem_formula_reflection*:
REFLECTS[λ*x*. *mem_formula*(*L*,*f*(*x*)),
λ*i x*. *mem_formula*(##*Lset*(*i*),*f*(*x*))]
apply (*simp only*: *mem_formula_def*)
apply (*intro FOL_reflections finite_ordinal_reflection formula_N_reflection*)
done

7.6.12 The Predicate "Is *formula*"

definition
is_formula_fm :: *i*=>*i* **where**
is_formula_fm(*Z*) == *Forall*(*Iff*(*Member*(*0*,*succ*(*Z*)), *mem_formula_fm*(*0*)))

lemma *is_formula_type* [*TC*]:
x ∈ *nat* ==> *is_formula_fm*(*x*) ∈ *formula*
by (*simp add*: *is_formula_fm_def*)

lemma *sats_is_formula_fm* [*simp*]:
[| *x* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *is_formula_fm*(*x*), *env*) ⟷ *is_formula*(##*A*, *nth*(*x*,*env*))
by (*simp add*: *is_formula_fm_def is_formula_def*)

lemma *is_formula_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *i* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_formula*(##*A*, *x*) ⟷ *sats*(*A*, *is_formula_fm*(*i*), *env*)
by *simp*

theorem *is_formula_reflection*:
  REFLECTS[λx. *is_formula*(*L*,*f*(*x*)),
    $\lambda i\ x$. *is_formula*(##*Lset*(*i*),*f*(*x*))]
apply (*simp only*: *is_formula_def*)
apply (*intro FOL_reflections mem_formula_reflection*)
done

7.6.13 The Operator *is_transrec*

The three arguments of *p* are always 2, 1, 0. It is buried within eight quantifiers! We call *p* with arguments a, f, z by equating them with the corresponding quantified variables with de Bruijn indices 2, 1, 0.

definition
 is_transrec_fm :: [*i*, *i*, *i*]=>*i* **where**
 is_transrec_fm(*p*,*a*,*z*) ==
  Exists(*Exists*(*Exists*(
   And(*upair_fm*(*a*#+*3*,*a*#+*3*,*2*),
    And(*is_eclose_fm*(*2*,*1*),
     And(*Memrel_fm*(*1*,*0*), *is_wfrec_fm*(*p*,*0*,*a*#+*3*,*z*#+*3*)))))))

lemma *is_transrec_type* [*TC*]:
  [| *p* ∈ *formula*; *x* ∈ *nat*; *z* ∈ *nat* |]
  ==> *is_transrec_fm*(*p*,*x*,*z*) ∈ *formula*
by (*simp add*: *is_transrec_fm_def*)

lemma *sats_is_transrec_fm*:
 assumes *MH_iff_sats*:
  !!*a0 a1 a2 a3 a4 a5 a6 a7*.
   [|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*; *a3*∈*A*; *a4*∈*A*; *a5*∈*A*; *a6*∈*A*; *a7*∈*A*|]
   ==> *MH*(*a2*, *a1*, *a0*) ⟷
    sats(*A*, *p*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*Cons*(*a3*,
      Cons(*a4*,*Cons*(*a5*,*Cons*(*a6*,*Cons*(*a7*,*env*))))))))))
 shows
  [|*x* < *length*(*env*); *z* < *length*(*env*); *env* ∈ *list*(*A*)|]
   ==> *sats*(*A*, *is_transrec_fm*(*p*,*x*,*z*), *env*) ⟷
    is_transrec(##*A*, *MH*, *nth*(*x*,*env*), *nth*(*z*,*env*))
apply (*frule_tac x=z* **in** *lt_length_in_nat*, *assumption*)
apply (*frule_tac x=x* **in** *lt_length_in_nat*, *assumption*)
apply (*simp add*: *is_transrec_fm_def sats_is_wfrec_fm is_transrec_def MH_iff_sats* [*THEN iff_sym*])
done

lemma *is_transrec_iff_sats*:
 assumes *MH_iff_sats*:
  !!*a0 a1 a2 a3 a4 a5 a6 a7*.
   [|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*; *a3*∈*A*; *a4*∈*A*; *a5*∈*A*; *a6*∈*A*; *a7*∈*A*|]
   ==> *MH*(*a2*, *a1*, *a0*) ⟷

```
        sats(A, p, Cons(a0,Cons(a1,Cons(a2,Cons(a3,
                    Cons(a4,Cons(a5,Cons(a6,Cons(a7,env)))))))))
  shows
  [|nth(i,env) = x; nth(k,env) = z;
      i < length(env); k < length(env); env ∈ list(A)|]
   ==> is_transrec(##A, MH, x, z) ⟷ sats(A, is_transrec_fm(p,i,k), env)
by (simp add: sats_is_transrec_fm [OF MH_iff_sats])

theorem is_transrec_reflection:
  assumes MH_reflection:
    !!f' f g h. REFLECTS[λx. MH(L, f'(x), f(x), g(x), h(x)),
                    λi x. MH(##Lset(i), f'(x), f(x), g(x), h(x))]
  shows REFLECTS[λx. is_transrec(L, MH(L,x), f(x), h(x)),
              λi x. is_transrec(##Lset(i), MH(##Lset(i),x), f(x), h(x))]
apply (simp (no_asm_use) only: is_transrec_def)
apply (intro FOL_reflections function_reflections MH_reflection
            is_wfrec_reflection is_eclose_reflection)
done

end
```

8 Separation for Facts About Recursion

theory *Rec_Separation* **imports** *ZF-Constructible.Separation Internalize Datatype_absolute*
begin

This theory proves all instances needed for locales *M_trancl* and *M_datatypes*

```
lemma eq_succ_imp_lt: [|i = succ(j); Ord(i)|] ==> j<i
by simp
```

8.1 The Locale *M_trancl*

8.1.1 Separation for Reflexive/Transitive Closure

First, The Defining Formula

```
definition
  rtran_closure_mem_fm :: [i,i,i]=>i where
 rtran_closure_mem_fm(A,r,p) ==
   Exists(Exists(Exists(
    And(omega_fm(2),
     And(Member(1,2),
      And(succ_fm(1,0),
       Exists(And(typed_function_fm(1, A#+4, 0),
        And(Exists(Exists(Exists(
              And(pair_fm(2,1,p#+7),
               And(empty_fm(0),
                And(fun_apply_fm(3,0,2), fun_apply_fm(3,5,1)))))))),
           Forall(Implies(Member(0,3),
```

```
Exists(Exists(Exists(Exists(
 And(fun_apply_fm(5,4,3),
  And(succ_fm(4,2),
   And(fun_apply_fm(5,2,1),
    And(pair_fm(3,1,0), Member(0,r#+9))))))))))))))))))))
```

lemma *rtran_closure_mem_type* [*TC*]:
[| $x \in nat$; $y \in nat$; $z \in nat$ |] ==> *rtran_closure_mem_fm*(*x*,*y*,*z*) $\in$ *formula*
by (*simp add*: *rtran_closure_mem_fm_def*)

lemma *sats_rtran_closure_mem_fm* [*simp*]:
[| $x \in nat$; $y \in nat$; $z \in nat$; $env \in list(A)$|]
==> *sats*(*A*, *rtran_closure_mem_fm*(*x*,*y*,*z*), *env*) $\longleftrightarrow$
rtran_closure_mem(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
by (*simp add*: *rtran_closure_mem_fm_def rtran_closure_mem_def*)

lemma *rtran_closure_mem_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
$i \in nat$; $j \in nat$; $k \in nat$; $env \in list(A)$|]
==> *rtran_closure_mem*(##*A*, *x*, *y*, *z*) $\longleftrightarrow$ *sats*(*A*, *rtran_closure_mem_fm*(*i*,*j*,*k*), *env*)
by (*simp*)

lemma *rtran_closure_mem_reflection*:
REFLECTS[λx. *rtran_closure_mem*(*L*,*f*(*x*),*g*(*x*),*h*(*x*)),
$\lambda i\ x$. *rtran_closure_mem*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*))]
apply (*simp only*: *rtran_closure_mem_def*)
apply (*intro FOL_reflections function_reflections fun_plus_reflections*)
done

Separation for *r*^∗.

lemma *rtrancl_separation*:
[| *L*(*r*); *L*(*A*) |] ==> *separation* (*L*, *rtran_closure_mem*(*L*,*A*,*r*))
apply (*rule gen_separation_multi* [*OF rtran_closure_mem_reflection*, *of* {*r*,*A*}],
auto)
apply (*rule_tac env*=[*r*,*A*] **in** *DPow_LsetI*)
apply (*rule rtran_closure_mem_iff_sats sep_rules* | *simp*)+
done

8.1.2 Reflexive/Transitive Closure, Internalized

definition
rtran_closure_fm :: [*i*,*i*]=>*i* **where**
rtran_closure_fm(*r*,*s*) ==
Forall(*Implies*(*field_fm*(*succ*(*r*),*0*),
Forall(*Iff*(*Member*(*0*,*succ*(*succ*(*s*))),
rtran_closure_mem_fm(*1*,*succ*(*succ*(*r*)),*0*)))))

lemma *rtran_closure_type* [*TC*]:
[| $x \in nat$; $y \in nat$ |] ==> $rtran_closure_fm(x,y) \in formula$
by (*simp add*: *rtran_closure_fm_def*)

lemma *sats_rtran_closure_fm* [*simp*]:
[| $x \in nat$; $y \in nat$; $env \in list(A)$|]
==> $sats(A,\ rtran_closure_fm(x,y),\ env) \longleftrightarrow$
$rtran_closure(\#\#A,\ nth(x,env),\ nth(y,env))$
by (*simp add*: *rtran_closure_fm_def rtran_closure_def*)

lemma *rtran_closure_iff_sats*:
[| $nth(i,env) = x$; $nth(j,env) = y$;
$i \in nat$; $j \in nat$; $env \in list(A)$|]
==> $rtran_closure(\#\#A,\ x,\ y) \longleftrightarrow sats(A,\ rtran_closure_fm(i,j),\ env)$
by *simp*

theorem *rtran_closure_reflection*:
$REFLECTS[\lambda x.\ rtran_closure(L,f(x),g(x)),$
$\lambda i\ x.\ rtran_closure(\#\#Lset(i),f(x),g(x))]$
apply (*simp only*: *rtran_closure_def*)
apply (*intro FOL_reflections function_reflections rtran_closure_mem_reflection*)
done

8.1.3 Transitive Closure of a Relation, Internalized

definition
tran_closure_fm :: $[i,i]$=>i **where**
$tran_closure_fm(r,s)$ ==
$Exists(And(rtran_closure_fm(succ(r),0),\ composition_fm(succ(r),0,succ(s))))$

lemma *tran_closure_type* [*TC*]:
[| $x \in nat$; $y \in nat$ |] ==> $tran_closure_fm(x,y) \in formula$
by (*simp add*: *tran_closure_fm_def*)

lemma *sats_tran_closure_fm* [*simp*]:
[| $x \in nat$; $y \in nat$; $env \in list(A)$|]
==> $sats(A,\ tran_closure_fm(x,y),\ env) \longleftrightarrow$
$tran_closure(\#\#A,\ nth(x,env),\ nth(y,env))$
by (*simp add*: *tran_closure_fm_def tran_closure_def*)

lemma *tran_closure_iff_sats*:
[| $nth(i,env) = x$; $nth(j,env) = y$;
$i \in nat$; $j \in nat$; $env \in list(A)$|]
==> $tran_closure(\#\#A,\ x,\ y) \longleftrightarrow sats(A,\ tran_closure_fm(i,j),\ env)$
by *simp*

theorem *tran_closure_reflection*:
$REFLECTS[\lambda x.\ tran_closure(L,f(x),g(x)),$
$\lambda i\ x.\ tran_closure(\#\#Lset(i),f(x),g(x))]$

apply (*simp only*: *tran_closure_def*)
apply (*intro FOL_reflections function_reflections*
rtran_closure_reflection composition_reflection)
done

8.1.4 Separation for the Proof of *wellfounded_on_trancl*

lemma *wellfounded_trancl_reflects*:
REFLECTS[λx. $\exists$ *w*[*L*]. $\exists$ *wx*[*L*]. $\exists$ *rp*[*L*].
w $\in$ *Z* & *pair*(*L*,*w*,*x*,*wx*) & *tran_closure*(*L*,*r*,*rp*) & *wx* $\in$ *rp*,
λi *x*. $\exists$ *w* $\in$ *Lset*(*i*). $\exists$ *wx* $\in$ *Lset*(*i*). $\exists$ *rp* $\in$ *Lset*(*i*).
w $\in$ *Z* & *pair*(##*Lset*(*i*),*w*,*x*,*wx*) & *tran_closure*(##*Lset*(*i*),*r*,*rp*) &
wx $\in$ *rp*]
by (*intro FOL_reflections function_reflections fun_plus_reflections*
tran_closure_reflection)

lemma *wellfounded_trancl_separation*:
[| *L*(*r*); *L*(*Z*) |] ==>
separation (*L*, λx.
$\exists$ *w*[*L*]. $\exists$ *wx*[*L*]. $\exists$ *rp*[*L*].
w $\in$ *Z* & *pair*(*L*,*w*,*x*,*wx*) & *tran_closure*(*L*,*r*,*rp*) & *wx* $\in$ *rp*)
apply (*rule gen_separation_multi* [*OF wellfounded_trancl_reflects*, *of* {*r*,*Z*}],
auto)
apply (*rule_tac env*=[*r*,*Z*] **in** *DPow_LsetI*)
apply (*rule sep_rules tran_closure_iff_sats* | *simp*)+
done

8.1.5 Instantiating the locale *M_trancl*

lemma *M_trancl_axioms_L*: *M_trancl_axioms*(*L*)
apply (*rule M_trancl_axioms.intro*)
apply (*assumption* | *rule rtrancl_separation wellfounded_trancl_separation*
L_nat)+
done

theorem *M_trancl_L*: *M_trancl*(*L*)
by (*rule M_trancl.intro* [*OF M_basic_L M_trancl_axioms_L*])

interpretation *L*: *M_trancl L* **by** (*rule M_trancl_L*)

8.2 *L* is Closed Under the Operator *list*

8.2.1 Instances of Replacement for Lists

lemma *list_replacement1_Reflects*:
REFLECTS
[λx. $\exists$ *u*[*L*]. *u* $\in$ *B* $\wedge$ ($\exists$ *y*[*L*]. *pair*(*L*,*u*,*y*,*x*) $\wedge$
is_wfrec(*L*, *iterates_MH*(*L*, *is_list_functor*(*L*,*A*), *0*), *memsn*, *u*, *y*)),
λi *x*. $\exists$ *u* $\in$ *Lset*(*i*). *u* $\in$ *B* $\wedge$ ($\exists$ *y* $\in$ *Lset*(*i*). *pair*(##*Lset*(*i*), *u*, *y*, *x*) $\wedge$
is_wfrec(##*Lset*(*i*),

```
                iterates_MH(##Lset(i),
                        is_list_functor(##Lset(i), A), 0), memsn, u, y))]
by (intro FOL_reflections function_reflections is_wfrec_reflection
        iterates_MH_reflection list_functor_reflection)


lemma list_replacement1:
  L(A) ==> iterates_replacement(L, is_list_functor(L,A), 0)
apply (unfold iterates_replacement_def wfrec_replacement_def, clarify)
apply (rule strong_replacementI)
apply (rule_tac u={B,A,n,0,Memrel(succ(n))}
        in gen_separation_multi [OF list_replacement1_Reflects],
      auto)
apply (rule_tac env=[B,A,n,0,Memrel(succ(n))] in DPow_LsetI)
apply (rule sep_rules is_nat_case_iff_sats list_functor_iff_sats
          is_wfrec_iff_sats iterates_MH_iff_sats quasinat_iff_sats | simp)+
done


lemma list_replacement2_Reflects:
 REFLECTS
  [λx. ∃u[L]. u ∈ B & u ∈ nat &
              is_iterates(L, is_list_functor(L, A), 0, u, x),
   λi x. ∃u ∈ Lset(i). u ∈ B & u ∈ nat &
              is_iterates(##Lset(i), is_list_functor(##Lset(i), A), 0, u, x)]
by (intro FOL_reflections
        is_iterates_reflection list_functor_reflection)

lemma list_replacement2:
  L(A) ==> strong_replacement(L,
        λn y. n∈nat & is_iterates(L, is_list_functor(L,A), 0, n, y))
apply (rule strong_replacementI)
apply (rule_tac u={A,B,0,nat}
        in gen_separation_multi [OF list_replacement2_Reflects],
      auto)
apply (rule_tac env=[A,B,0,nat] in DPow_LsetI)
apply (rule sep_rules list_functor_iff_sats is_iterates_iff_sats | simp)+
done
```

8.3 L is Closed Under the Operator *formula*

8.3.1 Instances of Replacement for Formulas

```
lemma formula_replacement1_Reflects:
 REFLECTS
  [λx. ∃u[L]. u ∈ B & (∃y[L]. pair(L,u,y,x) &
        is_wfrec(L, iterates_MH(L, is_formula_functor(L), 0), memsn, u, y)),
   λi x. ∃u ∈ Lset(i). u ∈ B & (∃y ∈ Lset(i). pair(##Lset(i), u, y, x) &
        is_wfrec(##Lset(i),
                iterates_MH(##Lset(i),
```

```
                    is_formula_functor(##Lset(i)), 0), memsn, u, y))]
by (intro FOL_reflections function_reflections is_wfrec_reflection
        iterates_MH_reflection formula_functor_reflection)

lemma formula_replacement1:
  iterates_replacement(L, is_formula_functor(L), 0)
apply (unfold iterates_replacement_def wfrec_replacement_def, clarify)
apply (rule strong_replacementI)
apply (rule_tac u={B,n,0,Memrel(succ(n))}
        in gen_separation_multi [OF formula_replacement1_Reflects],
      auto)
apply (rule_tac env=[n,B,0,Memrel(succ(n))] in DPow_LsetI)
apply (rule sep_rules is_nat_case_iff_sats formula_functor_iff_sats
          is_wfrec_iff_sats iterates_MH_iff_sats quasinat_iff_sats | simp)+
done

lemma formula_replacement2_Reflects:
 REFLECTS
  [λx. ∃u[L]. u ∈ B & u ∈ nat &
              is_iterates(L, is_formula_functor(L), 0, u, x),
   λi x. ∃u ∈ Lset(i). u ∈ B & u ∈ nat &
              is_iterates(##Lset(i), is_formula_functor(##Lset(i)), 0, u, x)]
by (intro FOL_reflections
        is_iterates_reflection formula_functor_reflection)

lemma formula_replacement2:
  strong_replacement(L,
        λn y. n∈nat & is_iterates(L, is_formula_functor(L), 0, n, y))
apply (rule strong_replacementI)
apply (rule_tac u={B,0,nat}
        in gen_separation_multi [OF formula_replacement2_Reflects],
      auto)
apply (rule_tac env=[B,0,nat] in DPow_LsetI)
apply (rule sep_rules formula_functor_iff_sats is_iterates_iff_sats | simp)+
done
```

NB The proofs for type *formula* are virtually identical to those for *list*(A). It was a cut-and-paste job!

8.3.2 The Formula *is_nth*, Internalized

```
definition
  nth_fm :: [i,i,i]=>i where
    nth_fm(n,l,Z) ==
      Exists(And(is_iterates_fm(tl_fm(1,0), succ(l), succ(n), 0),
            hd_fm(0,succ(Z))))

lemma nth_fm_type [TC]:
 [| x ∈ nat; y ∈ nat; z ∈ nat |] ==> nth_fm(x,y,z) ∈ formula
```

by (*simp add*: *nth_fm_def*)

lemma *sats_nth_fm* [*simp*]:
[| $x < length(env)$; $y \in nat$; $z \in nat$; $env \in list(A)$|]
==> $sats(A, nth_fm(x,y,z), env) \longleftrightarrow$
$is_nth(\#\#A, nth(x,env), nth(y,env), nth(z,env))$
apply (*frule lt_length_in_nat*, *assumption*)
apply (*simp add*: *nth_fm_def is_nth_def sats_is_iterates_fm*)
done

lemma *nth_iff_sats*:
[| $nth(i,env) = x$; $nth(j,env) = y$; $nth(k,env) = z$;
$i < length(env)$; $j \in nat$; $k \in nat$; $env \in list(A)$|]
==> $is_nth(\#\#A, x, y, z) \longleftrightarrow sats(A, nth_fm(i,j,k), env)$
by (*simp*)

theorem *nth_reflection*:
$REFLECTS[\lambda x.\ is_nth(L, f(x), g(x), h(x)),$
$\lambda i\ x.\ is_nth(\#\#Lset(i), f(x), g(x), h(x))]$
apply (*simp only*: *is_nth_def*)
apply (*intro FOL_reflections is_iterates_reflection*
hd_reflection tl_reflection)
done

8.3.3 An Instance of Replacement for *nth*

lemma *nth_replacement_Reflects*:
REFLECTS
$[\lambda x.\ \exists u[L].\ u \in B\ \&\ (\exists y[L].\ pair(L,u,y,x)\ \&$
$is_wfrec(L, iterates_MH(L, is_tl(L), z), memsn, u, y)),$
$\lambda i\ x.\ \exists u \in Lset(i).\ u \in B\ \&\ (\exists y \in Lset(i).\ pair(\#\#Lset(i), u, y, x)\ \&$
$is_wfrec(\#\#Lset(i),$
$iterates_MH(\#\#Lset(i),$
$is_tl(\#\#Lset(i)), z), memsn, u, y))]$
by (*intro FOL_reflections function_reflections is_wfrec_reflection*
iterates_MH_reflection tl_reflection)

lemma *nth_replacement*:
$L(w)$ ==> $iterates_replacement(L, is_tl(L), w)$
apply (*unfold iterates_replacement_def wfrec_replacement_def*, *clarify*)
apply (*rule strong_replacementI*)
apply (*rule_tac* $u=\{B,w,Memrel(succ(n))\}$
in *gen_separation_multi* [*OF nth_replacement_Reflects*],
auto)
apply (*rule_tac* $env=[B,w,Memrel(succ(n))]$ **in** *DPow_LsetI*)
apply (*rule sep_rules is_nat_case_iff_sats tl_iff_sats*
is_wfrec_iff_sats iterates_MH_iff_sats quasinat_iff_sats | *simp*)+
done

8.3.4 Instantiating the locale *M_datatypes*

lemma *M_datatypes_axioms_L*: *M_datatypes_axioms*(*L*)
  apply (*rule M_datatypes_axioms.intro*)
    apply (*assumption* | *rule*
      list_replacement1 list_replacement2
      formula_replacement1 formula_replacement2
      nth_replacement)+
  done

theorem *M_datatypes_L*: *M_datatypes*(*L*)
  apply (*rule M_datatypes.intro*)
  apply (*rule M_trancl_L*)
  apply (*rule M_datatypes_axioms_L*)
  done

interpretation *L*: *M_datatypes L* **by** (*rule M_datatypes_L*)

8.4 *L* is Closed Under the Operator *eclose*

8.4.1 Instances of Replacement for *eclose*

lemma *eclose_replacement1_Reflects*:
 REFLECTS
  [λ*x*. ∃ *u*[*L*]. *u* ∈ *B* & (∃ *y*[*L*]. *pair*(*L*,*u*,*y*,*x*) &
      is_wfrec(*L*, *iterates_MH*(*L*, *big_union*(*L*), *A*), *memsn*, *u*, *y*)),
   λ*i x*. ∃ *u* ∈ *Lset*(*i*). *u* ∈ *B* & (∃ *y* ∈ *Lset*(*i*). *pair*(##*Lset*(*i*), *u*, *y*, *x*) &
      is_wfrec(##*Lset*(*i*),
        iterates_MH(##*Lset*(*i*), *big_union*(##*Lset*(*i*)), *A*),
        memsn, *u*, *y*))]
by (*intro FOL_reflections function_reflections is_wfrec_reflection*
      iterates_MH_reflection)

lemma *eclose_replacement1*:
  L(*A*) ==> *iterates_replacement*(*L*, *big_union*(*L*), *A*)
apply (*unfold iterates_replacement_def wfrec_replacement_def*, *clarify*)
apply (*rule strong_replacementI*)
apply (*rule_tac u*={*B*,*A*,*n*,*Memrel*(*succ*(*n*))}
      in *gen_separation_multi* [*OF eclose_replacement1_Reflects*], *auto*)
apply (*rule_tac env*=[*B*,*A*,*n*,*Memrel*(*succ*(*n*))] **in** *DPow_LsetI*)
apply (*rule sep_rules iterates_MH_iff_sats is_nat_case_iff_sats*
        is_wfrec_iff_sats big_union_iff_sats quasinat_iff_sats | *simp*)+
done

lemma *eclose_replacement2_Reflects*:
 REFLECTS
  [λ*x*. ∃ *u*[*L*]. *u* ∈ *B* & *u* ∈ *nat* &
        is_iterates(*L*, *big_union*(*L*), *A*, *u*, *x*),
   λ*i x*. ∃ *u* ∈ *Lset*(*i*). *u* ∈ *B* & *u* ∈ *nat* &

```
            is_iterates(##Lset(i), big_union(##Lset(i)), A, u, x)]
by (intro FOL_reflections function_reflections is_iterates_reflection)

lemma eclose_replacement2:
  L(A) ==> strong_replacement(L,
        λn y. n∈nat & is_iterates(L, big_union(L), A, n, y))
apply (rule strong_replacementI)
apply (rule_tac u={A,B,nat}
        in gen_separation_multi [OF eclose_replacement2_Reflects],
      auto)
apply (rule_tac env=[A,B,nat] in DPow_LsetI)
apply (rule sep_rules is_iterates_iff_sats big_union_iff_sats | simp)+
done
```

8.4.2 Instantiating the locale *M_eclose*

```
lemma M_eclose_axioms_L: M_eclose_axioms(L)
  apply (rule M_eclose_axioms.intro)
   apply (assumption | rule eclose_replacement1 eclose_replacement2)+
  done

theorem M_eclose_L: M_eclose(L)
  apply (rule M_eclose.intro)
   apply (rule M_trancl_L)
  apply (rule M_eclose_axioms_L)
  done

interpretation L: M_eclose L by (rule M_eclose_L)


end
```

9 Absoluteness for the Satisfies Relation on Formulas

theory *Satisfies_absolute* **imports** *Datatype_absolute Rec_Separation* **begin**

9.1 More Internalization

9.1.1 The Formula *is_depth*, Internalized

```
definition
  depth_fm :: [i,i]=>i where
  depth_fm(p,n) ==
     Exists(Exists(Exists(
       And(formula_N_fm(n#+3,1),
         And(Neg(Member(p#+3,1)),
          And(succ_fm(n#+3,2),
           And(formula_N_fm(2,0), Member(p#+3,0))))))))
```

lemma *depth_fm_type* [*TC*]:
[| $x \in nat$; $y \in nat$ |] ==> $depth_fm(x,y) \in formula$
by (*simp add*: *depth_fm_def*)

lemma *sats_depth_fm* [*simp*]:
[| $x \in nat$; $y < length(env)$; $env \in list(A)$|]
==> $sats(A,\ depth_fm(x,y),\ env) \longleftrightarrow$
$is_depth(\#\#A,\ nth(x,env),\ nth(y,env))$
apply (*frule_tac x=y* **in** *lt_length_in_nat*, *assumption*)
apply (*simp add*: *depth_fm_def is_depth_def*)
done

lemma *depth_iff_sats*:
[| $nth(i,env) = x$; $nth(j,env) = y$;
$i \in nat$; $j < length(env)$; $env \in list(A)$|]
==> $is_depth(\#\#A,\ x,\ y) \longleftrightarrow sats(A,\ depth_fm(i,j),\ env)$
by (*simp*)

theorem *depth_reflection*:
$REFLECTS[\lambda x.\ is_depth(L,\ f(x),\ g(x)),$
$\lambda i\ x.\ is_depth(\#\#Lset(i),\ f(x),\ g(x))]$
apply (*simp only*: *is_depth_def*)
apply (*intro FOL_reflections function_reflections formula_N_reflection*)
done

9.1.2 The Operator *is_formula_case*

The arguments of *is_a* are always 2, 1, 0, and the formula will be enclosed by three quantifiers.

definition
formula_case_fm :: [*i*, *i*, *i*, *i*, *i*, *i*]=>*i* **where**
formula_case_fm(*is_a*, *is_b*, *is_c*, *is_d*, *v*, *z*) ==
And(*Forall*(*Forall*(*Implies*(*finite_ordinal_fm*(*1*),
Implies(*finite_ordinal_fm*(*0*),
Implies(*Member_fm*(*1*,*0*,*v*#+*2*),
Forall(*Implies*(*Equal*(*0*,*z*#+*3*), *is_a*))))))),
And(*Forall*(*Forall*(*Implies*(*finite_ordinal_fm*(*1*),
Implies(*finite_ordinal_fm*(*0*),
Implies(*Equal_fm*(*1*,*0*,*v*#+*2*),
Forall(*Implies*(*Equal*(*0*,*z*#+*3*), *is_b*))))))),
And(*Forall*(*Forall*(*Implies*(*mem_formula_fm*(*1*),
Implies(*mem_formula_fm*(*0*),
Implies(*Nand_fm*(*1*,*0*,*v*#+*2*),
Forall(*Implies*(*Equal*(*0*,*z*#+*3*), *is_c*))))))),
Forall(*Implies*(*mem_formula_fm*(*0*),
Implies(*Forall_fm*(*0*,*succ*(*v*)),
Forall(*Implies*(*Equal*(*0*,*z*#+*2*), *is_d*)))))))

lemma *is_formula_case_type* [*TC*]:
[| *is_a* ∈ *formula*; *is_b* ∈ *formula*; *is_c* ∈ *formula*; *is_d* ∈ *formula*;
x ∈ *nat*; *y* ∈ *nat* |]
==> *formula_case_fm*(*is_a*, *is_b*, *is_c*, *is_d*, *x*, *y*) ∈ *formula*
by (*simp add*: *formula_case_fm_def*)

lemma *sats_formula_case_fm*:
assumes *is_a_iff_sats*:
!!*a0 a1 a2*.
[|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*|]
==> *ISA*(*a2*, *a1*, *a0*) ⟷ *sats*(*A*, *is_a*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*env*))))
and *is_b_iff_sats*:
!!*a0 a1 a2*.
[|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*|]
==> *ISB*(*a2*, *a1*, *a0*) ⟷ *sats*(*A*, *is_b*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*env*))))
and *is_c_iff_sats*:
!!*a0 a1 a2*.
[|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*|]
==> *ISC*(*a2*, *a1*, *a0*) ⟷ *sats*(*A*, *is_c*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*env*))))
and *is_d_iff_sats*:
!!*a0 a1*.
[|*a0*∈*A*; *a1*∈*A*|]
==> *ISD*(*a1*, *a0*) ⟷ *sats*(*A*, *is_d*, *Cons*(*a0*,*Cons*(*a1*,*env*)))
shows
[|*x* ∈ *nat*; *y* < *length*(*env*); *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *formula_case_fm*(*is_a*,*is_b*,*is_c*,*is_d*,*x*,*y*), *env*) ⟷
is_formula_case(##*A*, *ISA*, *ISB*, *ISC*, *ISD*, *nth*(*x*,*env*), *nth*(*y*,*env*))
apply (*frule_tac x=y* **in** *lt_length_in_nat*, *assumption*)
apply (*simp add*: *formula_case_fm_def is_formula_case_def*
is_a_iff_sats [*THEN iff_sym*] *is_b_iff_sats* [*THEN iff_sym*]
is_c_iff_sats [*THEN iff_sym*] *is_d_iff_sats* [*THEN iff_sym*])
done

lemma *formula_case_iff_sats*:
assumes *is_a_iff_sats*:
!!*a0 a1 a2*.
[|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*|]
==> *ISA*(*a2*, *a1*, *a0*) ⟷ *sats*(*A*, *is_a*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*env*))))
and *is_b_iff_sats*:
!!*a0 a1 a2*.
[|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*|]
==> *ISB*(*a2*, *a1*, *a0*) ⟷ *sats*(*A*, *is_b*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*env*))))
and *is_c_iff_sats*:
!!*a0 a1 a2*.
[|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*|]
==> *ISC*(*a2*, *a1*, *a0*) ⟷ *sats*(*A*, *is_c*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*env*))))
and *is_d_iff_sats*:
!!*a0 a1*.

$[|a0 \in A;\ a1 \in A|]$
$==> ISD(a1,\ a0) \longleftrightarrow sats(A,\ is_d,\ Cons(a0,Cons(a1,env)))$
shows
$[|nth(i,env) = x;\ nth(j,env) = y;$
$i \in nat;\ j < length(env);\ env \in list(A)|]$
$==> is_formula_case(\#\#A,\ ISA,\ ISB,\ ISC,\ ISD,\ x,\ y) \longleftrightarrow$
$sats(A,\ formula_case_fm(is_a,is_b,is_c,is_d,i,j),\ env)$
by (*simp add*: *sats_formula_case_fm* [*OF is_a_iff_sats is_b_iff_sats*
is_c_iff_sats is_d_iff_sats])

The second argument of *is_a* gives it direct access to x, which is essential for handling free variable references. Treatment is based on that of *is_nat_case_reflection*.

theorem *is_formula_case_reflection*:
assumes *is_a_reflection*:
$!!h\ f\ g\ g'.\ REFLECTS[\lambda x.\ is_a(L,\ h(x),\ f(x),\ g(x),\ g'(x)),$
$\lambda i\ x.\ is_a(\#\#Lset(i),\ h(x),\ f(x),\ g(x),\ g'(x))]$
and *is_b_reflection*:
$!!h\ f\ g\ g'.\ REFLECTS[\lambda x.\ is_b(L,\ h(x),\ f(x),\ g(x),\ g'(x)),$
$\lambda i\ x.\ is_b(\#\#Lset(i),\ h(x),\ f(x),\ g(x),\ g'(x))]$
and *is_c_reflection*:
$!!h\ f\ g\ g'.\ REFLECTS[\lambda x.\ is_c(L,\ h(x),\ f(x),\ g(x),\ g'(x)),$
$\lambda i\ x.\ is_c(\#\#Lset(i),\ h(x),\ f(x),\ g(x),\ g'(x))]$
and *is_d_reflection*:
$!!h\ f\ g\ g'.\ REFLECTS[\lambda x.\ is_d(L,\ h(x),\ f(x),\ g(x)),$
$\lambda i\ x.\ is_d(\#\#Lset(i),\ h(x),\ f(x),\ g(x))]$
shows $REFLECTS[\lambda x.\ is_formula_case(L,\ is_a(L,x),\ is_b(L,x),\ is_c(L,x),$
$is_d(L,x),\ g(x),\ h(x)),$
$\lambda i\ x.\ is_formula_case(\#\#Lset(i),\ is_a(\#\#Lset(i),\ x),\ is_b(\#\#Lset(i),$
$x),\ is_c(\#\#Lset(i),\ x),\ is_d(\#\#Lset(i),\ x),\ g(x),\ h(x))]$
apply (*simp* (*no_asm_use*) *only*: *is_formula_case_def*)
apply (*intro FOL_reflections function_reflections finite_ordinal_reflection*
mem_formula_reflection
Member_reflection Equal_reflection Nand_reflection Forall_reflection
is_a_reflection is_b_reflection is_c_reflection is_d_reflection)
done

9.2 Absoluteness for the Function *satisfies*

definition
is_depth_apply :: $[i=>o,i,i,i] => o$ **where**
— Merely a useful abbreviation for the sequel.
$is_depth_apply(M,h,p,z) ==$
$\exists\, dp[M].\ \exists\, sdp[M].\ \exists\, hsdp[M].$
$finite_ordinal(M,dp)\ \&\ is_depth(M,p,dp)\ \&\ successor(M,dp,sdp)\ \&$
$fun_apply(M,h,sdp,hsdp)\ \&\ fun_apply(M,hsdp,p,z)$

lemma (**in** *M_datatypes*) *is_depth_apply_abs* [*simp*]:
$[|M(h);\ p \in formula;\ M(z)|]$

$==> is_depth_apply(M,h,p,z) \longleftrightarrow z = h \text{ ‘ } succ(depth(p)) \text{ ‘ } p$
by (*simp add*: *is_depth_apply_def formula_into_M depth_type eq_commute*)

There is at present some redundancy between the relativizations in e.g. *satisfies_is_a* and those in e.g. *Member_replacement*.

These constants let us instantiate the parameters *a*, *b*, *c*, *d*, etc., of the locale *Formula_Rec*.

definition
$satisfies_a :: [i,i,i]=>i$ **where**
$satisfies_a(A) ==$
$\lambda x\ y.\ \lambda env \in list(A).\ bool_of_o\ (nth(x,env) \in nth(y,env))$

definition
$satisfies_is_a :: [i=>o,i,i,i,i]=>o$ **where**
$satisfies_is_a(M,A) ==$
$\lambda x\ y\ zz.\ \forall lA[M].\ is_list(M,A,lA) \longrightarrow$
$is_lambda(M,\ lA,$
$\lambda env\ z.\ is_bool_of_o(M,$
$\exists nx[M].\ \exists ny[M].$
$is_nth(M,x,env,nx)\ \&\ is_nth(M,y,env,ny)\ \&\ nx \in ny,\ z),$
$zz)$

definition
$satisfies_b :: [i,i,i]=>i$ **where**
$satisfies_b(A) ==$
$\lambda x\ y.\ \lambda env \in list(A).\ bool_of_o\ (nth(x,env) = nth(y,env))$

definition
$satisfies_is_b :: [i=>o,i,i,i,i]=>o$ **where**
— We simplify the formula to have just *nx* rather than introducing *ny* with *nx* = *ny*
$satisfies_is_b(M,A) ==$
$\lambda x\ y\ zz.\ \forall lA[M].\ is_list(M,A,lA) \longrightarrow$
$is_lambda(M,\ lA,$
$\lambda env\ z.\ is_bool_of_o(M,$
$\exists nx[M].\ is_nth(M,x,env,nx)\ \&\ is_nth(M,y,env,nx),\ z),$
$zz)$

definition
$satisfies_c :: [i,i,i,i,i]=>i$ **where**
$satisfies_c(A) == \lambda p\ q\ rp\ rq.\ \lambda env \in list(A).\ not(rp \text{ ‘ } env\ and\ rq \text{ ‘ } env)$

definition
$satisfies_is_c :: [i=>o,i,i,i,i,i]=>o$ **where**
$satisfies_is_c(M,A,h) ==$
$\lambda p\ q\ zz.\ \forall lA[M].\ is_list(M,A,lA) \longrightarrow$
$is_lambda(M,\ lA,\ \lambda env\ z.\ \exists hp[M].\ \exists hq[M].$
$(\exists rp[M].\ is_depth_apply(M,h,p,rp)\ \&\ fun_apply(M,rp,env,hp))\ \&$

$(\exists\, rq[M].\ is_depth_apply(M,h,q,rq)\ \&\ fun_apply(M,rq,env,hq))\ \&$
$(\exists\, pq[M].\ is_and(M,hp,hq,pq)\ \&\ is_not(M,pq,z)),$
$zz)$

definition
$satisfies_d :: [i,i,i]{=>}i$ **where**
$satisfies_d(A)$
$== \lambda p\ rp.\ \lambda env \in list(A).\ bool_of_o\ (\forall\, x{\in}A.\ rp\ `\ (Cons(x,env)) = 1)$

definition
$satisfies_is_d :: [i{=>}o,i,i,i,i]{=>}o$ **where**
$satisfies_is_d(M,A,h) ==$
$\lambda p\ zz.\ \forall\, lA[M].\ is_list(M,A,lA) \longrightarrow$
$is_lambda(M,\ lA,$
$\lambda env\ z.\ \exists\, rp[M].\ is_depth_apply(M,h,p,rp)\ \&$
$is_bool_of_o(M,$
$\forall\, x[M].\ \forall\, xenv[M].\ \forall\, hp[M].$
$x{\in}A \longrightarrow is_Cons(M,x,env,xenv) \longrightarrow$
$fun_apply(M,rp,xenv,hp) \longrightarrow number1(M,hp),$
$z),$
$zz)$

definition
$satisfies_MH :: [i{=>}o,i,i,i,i]{=>}o$ **where**
— The variable u is unused, but gives *satisfies_MH* the correct arity.
$satisfies_MH ==$
$\lambda M\ A\ u\ f\ z.$
$\forall\, fml[M].\ is_formula(M,fml) \longrightarrow$
$is_lambda\ (M,\ fml,$
$is_formula_case\ (M,\ satisfies_is_a(M,A),$
$satisfies_is_b(M,A),$
$satisfies_is_c(M,A,f),\ satisfies_is_d(M,A,f)),$
$z)$

definition
$is_satisfies :: [i{=>}o,i,i,i]{=>}o$ **where**
$is_satisfies(M,A) == is_formula_rec\ (M,\ satisfies_MH(M,A))$

This lemma relates the fragments defined above to the original primitive recursion in *satisfies*. Induction is not required: the definitions are directly equal!

lemma *satisfies_eq*:
$satisfies(A,p) =$
$formula_rec\ (satisfies_a(A),\ satisfies_b(A),$
$satisfies_c(A),\ satisfies_d(A),\ p)$
by (*simp add*: *satisfies_formula_def satisfies_a_def satisfies_b_def satisfies_c_def satisfies_d_def*)

Further constraints on the class M in order to prove absoluteness for the con-

stants defined above. The ultimate goal is the absoluteness of the function *satisfies*.

```
locale M_satisfies = M_eclose + M_datatypes +
 assumes
   Member_replacement:
    [|M(A); x ∈ nat; y ∈ nat|]
     ==> strong_replacement
         (M, λenv z. ∃ bo[M]. ∃ nx[M]. ∃ ny[M].
              env ∈ list(A) & is_nth(M,x,env,nx) & is_nth(M,y,env,ny) &
              is_bool_of_o(M, nx ∈ ny, bo) &
              pair(M, env, bo, z))
 and
   Equal_replacement:
    [|M(A); x ∈ nat; y ∈ nat|]
     ==> strong_replacement
         (M, λenv z. ∃ bo[M]. ∃ nx[M]. ∃ ny[M].
              env ∈ list(A) & is_nth(M,x,env,nx) & is_nth(M,y,env,ny) &
              is_bool_of_o(M, nx = ny, bo) &
              pair(M, env, bo, z))
 and
   Nand_replacement:
    [|M(A); M(rp); M(rq)|]
     ==> strong_replacement
         (M, λenv z. ∃ rpe[M]. ∃ rqe[M]. ∃ andpq[M]. ∃ notpq[M].
               fun_apply(M,rp,env,rpe) & fun_apply(M,rq,env,rqe) &
               is_and(M,rpe,rqe,andpq) & is_not(M,andpq,notpq) &
               env ∈ list(A) & pair(M, env, notpq, z))
 and
  Forall_replacement:
   [|M(A); M(rp)|]
    ==> strong_replacement
        (M, λenv z. ∃ bo[M].
              env ∈ list(A) &
              is_bool_of_o (M,
                          ∀ a[M]. ∀ co[M]. ∀ rpco[M].
                             a∈A ⟶ is_Cons(M,a,env,co) ⟶
                             fun_apply(M,rp,co,rpco) ⟶ number1(M, rpco),
                          bo) &
              pair(M,env,bo,z))
 and
  formula_rec_replacement:
      — For the transrec
   [|n ∈ nat; M(A)|] ==> transrec_replacement(M, satisfies_MH(M,A), n)
 and
  formula_rec_lambda_replacement:
      — For the λ-abstraction in the transrec body
   [|M(g); M(A)|] ==>
    strong_replacement (M,
       λx y. mem_formula(M,x) &
```

$(\exists\, c[M].\ is_formula_case(M,\ satisfies_is_a(M,A),$
$satisfies_is_b(M,A),$
$satisfies_is_c(M,A,g),$
$satisfies_is_d(M,A,g),\ x,\ c)\ \&$
$pair(M,\ x,\ c,\ y)))$

lemma (**in** *M_satisfies*) *Member_replacement'*:
$[|M(A);\ x \in nat;\ y \in nat|]$
$==>\ strong_replacement$
$(M,\ \lambda env\ z.\ env \in list(A)\ \&$
$z = \langle env,\ bool_of_o(nth(x,\ env) \in nth(y,\ env))\rangle)$
by (*insert Member_replacement*, *simp*)

lemma (**in** *M_satisfies*) *Equal_replacement'*:
$[|M(A);\ x \in nat;\ y \in nat|]$
$==>\ strong_replacement$
$(M,\ \lambda env\ z.\ env \in list(A)\ \&$
$z = \langle env,\ bool_of_o(nth(x,\ env) = nth(y,\ env))\rangle)$
by (*insert Equal_replacement*, *simp*)

lemma (**in** *M_satisfies*) *Nand_replacement'*:
$[|M(A);\ M(rp);\ M(rq)|]$
$==>\ strong_replacement$
$(M,\ \lambda env\ z.\ env \in list(A)\ \&\ z = \langle env,\ not(rp`env\ and\ rq`env)\rangle)$
by (*insert Nand_replacement*, *simp*)

lemma (**in** *M_satisfies*) *Forall_replacement'*:
$[|M(A);\ M(rp)|]$
$==>\ strong_replacement$
$(M,\ \lambda env\ z.$
$env \in list(A)\ \&$
$z = \langle env,\ bool_of_o\ (\forall\, a{\in}A.\ rp\ `\ Cons(a,env) = 1)\rangle)$
by (*insert Forall_replacement*, *simp*)

lemma (**in** *M_satisfies*) *a_closed*:
$[|M(A);\ x{\in}nat;\ y{\in}nat|]\ ==>\ M(satisfies_a(A,x,y))$
apply (*simp add*: *satisfies_a_def*)
apply (*blast intro*: *lam_closed2 Member_replacement'*)
done

lemma (**in** *M_satisfies*) *a_rel*:
$M(A)\ ==>\ Relation2(M,\ nat,\ nat,\ satisfies_is_a(M,A),\ satisfies_a(A))$
apply (*simp add*: *Relation2_def satisfies_is_a_def satisfies_a_def*)
apply (*auto del*: *iffI intro*!: *lambda_abs2 simp add*: *Relation1_def*)
done

lemma (**in** *M_satisfies*) *b_closed*:
$[|M(A);\ x{\in}nat;\ y{\in}nat|]\ ==>\ M(satisfies_b(A,x,y))$

apply (*simp add*: *satisfies_b_def*)
apply (*blast intro*: *lam_closed2 Equal_replacement'*)
done

lemma (**in** *M_satisfies*) *b_rel*:
  M(*A*) ==> *Relation2*(*M*, *nat*, *nat*, *satisfies_is_b*(*M*,*A*), *satisfies_b*(*A*))
apply (*simp add*: *Relation2_def satisfies_is_b_def satisfies_b_def*)
apply (*auto del*: *iffI intro*!: *lambda_abs2 simp add*: *Relation1_def*)
done

lemma (**in** *M_satisfies*) *c_closed*:
  [|*M*(*A*); *x* ∈ *formula*; *y* ∈ *formula*; *M*(*rx*); *M*(*ry*)|]
  ==> *M*(*satisfies_c*(*A*,*x*,*y*,*rx*,*ry*))
apply (*simp add*: *satisfies_c_def*)
apply (*rule lam_closed2*)
apply (*rule Nand_replacement'*)
apply (*simp_all add*: *formula_into_M list_into_M* [*of* _ *A*])
done

lemma (**in** *M_satisfies*) *c_rel*:
 [|*M*(*A*); *M*(*f*)|] ==>
  Relation2 (*M*, *formula*, *formula*,
          satisfies_is_c(*M*,*A*,*f*),
          λ*u v*. *satisfies_c*(*A*, *u*, *v*, *f* ‘ *succ*(*depth*(*u*)) ‘ *u*,
                f ‘ *succ*(*depth*(*v*)) ‘ *v*))
apply (*simp add*: *Relation2_def satisfies_is_c_def satisfies_c_def*)
apply (*auto del*: *iffI intro*!: *lambda_abs2*
      simp add: *Relation1_def formula_into_M*)
done

lemma (**in** *M_satisfies*) *d_closed*:
  [|*M*(*A*); *x* ∈ *formula*; *M*(*rx*)|] ==> *M*(*satisfies_d*(*A*,*x*,*rx*))
apply (*simp add*: *satisfies_d_def*)
apply (*rule lam_closed2*)
apply (*rule Forall_replacement'*)
apply (*simp_all add*: *formula_into_M list_into_M* [*of* _ *A*])
done

lemma (**in** *M_satisfies*) *d_rel*:
 [|*M*(*A*); *M*(*f*)|] ==>
  Relation1(*M*, *formula*, *satisfies_is_d*(*M*,*A*,*f*),
    λ*u*. *satisfies_d*(*A*, *u*, *f* ‘ *succ*(*depth*(*u*)) ‘ *u*))
apply (*simp del*: *rall_abs*
      add: *Relation1_def satisfies_is_d_def satisfies_d_def*)
apply (*auto del*: *iffI intro*!: *lambda_abs2 simp add*: *Relation1_def*)
done

lemma (**in** *M_satisfies*) *fr_replace*:

```
    [|n ∈ nat; M(A)|] ==> transrec_replacement(M,satisfies_MH(M,A),n)
by (blast intro: formula_rec_replacement)

lemma (in M_satisfies) formula_case_satisfies_closed:
 [|M(g); M(A); x ∈ formula|] ==>
  M(formula_case (satisfies_a(A), satisfies_b(A),
      λu v. satisfies_c(A, u, v,
                    g ` succ(depth(u)) ` u, g ` succ(depth(v)) ` v),
      λu. satisfies_d (A, u, g ` succ(depth(u)) ` u),
      x))
by (blast intro: a_closed b_closed c_closed d_closed)

lemma (in M_satisfies) fr_lam_replace:
  [|M(g); M(A)|] ==>
   strong_replacement (M, λx y. x ∈ formula &
          y = ⟨x,
               formula_rec_case(satisfies_a(A),
                                satisfies_b(A),
                                satisfies_c(A),
                                satisfies_d(A), g, x)⟩)
apply (insert formula_rec_lambda_replacement)
apply (simp add: formula_rec_case_def formula_case_satisfies_closed
               formula_case_abs [OF a_rel b_rel c_rel d_rel])
done
```

Instantiate locale *Formula_Rec* for the Function *satisfies*

```
lemma (in M_satisfies) Formula_Rec_axioms_M:
  M(A) ==>
   Formula_Rec_axioms(M, satisfies_a(A), satisfies_is_a(M,A),
                      satisfies_b(A), satisfies_is_b(M,A),
                      satisfies_c(A), satisfies_is_c(M,A),
                      satisfies_d(A), satisfies_is_d(M,A))
apply (rule Formula_Rec_axioms.intro)
apply (assumption |
      rule a_closed a_rel b_closed b_rel c_closed c_rel d_closed d_rel
      fr_replace [unfolded satisfies_MH_def]
      fr_lam_replace) +
done


theorem (in M_satisfies) Formula_Rec_M:
   M(A) ==>
    Formula_Rec(M, satisfies_a(A), satisfies_is_a(M,A),
                satisfies_b(A), satisfies_is_b(M,A),
                satisfies_c(A), satisfies_is_c(M,A),
                satisfies_d(A), satisfies_is_d(M,A))
 apply (rule Formula_Rec.intro)
 apply (rule M_satisfies.axioms, rule M_satisfies_axioms, rule M_datatypes_axioms)
 apply (erule Formula_Rec_axioms_M)
```

done

lemmas (**in** *M_satisfies*)
  satisfies_closed′ = *Formula_Rec.formula_rec_closed* [*OF Formula_Rec_M*]
and *satisfies_abs′* = *Formula_Rec.formula_rec_abs* [*OF Formula_Rec_M*]

lemma (**in** *M_satisfies*) *satisfies_closed*:
  [|*M*(*A*); *p* ∈ *formula*|] ==> *M*(*satisfies*(*A*,*p*))
by (*simp add*: *Formula_Rec.formula_rec_closed* [*OF Formula_Rec_M*]
        satisfies_eq)

lemma (**in** *M_satisfies*) *satisfies_abs*:
  [|*M*(*A*); *M*(*z*); *p* ∈ *formula*|]
  ==> *is_satisfies*(*M*,*A*,*p*,*z*) ⟷ *z* = *satisfies*(*A*,*p*)
by (*simp only*: *Formula_Rec.formula_rec_abs* [*OF Formula_Rec_M*]
        satisfies_eq is_satisfies_def satisfies_MH_def)

9.3 Internalizations Needed to Instantiate *M_satisfies*

9.3.1 The Operator *is_depth_apply*, Internalized

definition
  depth_apply_fm :: [*i*,*i*,*i*]=>*i* **where**
  depth_apply_fm(*h*,*p*,*z*) ==
    Exists(*Exists*(*Exists*(
      And(*finite_ordinal_fm*(*2*),
       And(*depth_fm*(*p*#+*3*,*2*),
        And(*succ_fm*(*2*,*1*),
         And(*fun_apply_fm*(*h*#+*3*,*1*,*0*), *fun_apply_fm*(*0*,*p*#+*3*,*z*#+*3*))))))))

lemma *depth_apply_type* [*TC*]:
  [| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *depth_apply_fm*(*x*,*y*,*z*) ∈ *formula*
by (*simp add*: *depth_apply_fm_def*)

lemma *sats_depth_apply_fm* [*simp*]:
  [| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
  ==> *sats*(*A*, *depth_apply_fm*(*x*,*y*,*z*), *env*) ⟷
    is_depth_apply(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
by (*simp add*: *depth_apply_fm_def is_depth_apply_def*)

lemma *depth_apply_iff_sats*:
  [| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
    i ∈ *nat*; *j* ∈ *nat*; *k* ∈ *nat*; *env* ∈ *list*(*A*)|]
  ==> *is_depth_apply*(##*A*, *x*, *y*, *z*) ⟷ *sats*(*A*, *depth_apply_fm*(*i*,*j*,*k*), *env*)
by *simp*

lemma *depth_apply_reflection*:
  REFLECTS[λ*x*. *is_depth_apply*(*L*,*f*(*x*),*g*(*x*),*h*(*x*)),
      λ*i x*. *is_depth_apply*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*))]

apply (*simp only*: *is_depth_apply_def*)
apply (*intro FOL_reflections function_reflections depth_reflection*
finite_ordinal_reflection)
done

9.3.2 The Operator *satisfies_is_a*, Internalized

definition
satisfies_is_a_fm :: [*i*,*i*,*i*,*i*]=>*i* **where**
satisfies_is_a_fm(*A*,*x*,*y*,*z*) ==
Forall(
Implies(*is_list_fm*(*succ*(*A*),*0*),
lambda_fm(
bool_of_o_fm(*Exists*(
Exists(*And*(*nth_fm*(*x*#+*6*,*3*,*1*),
And(*nth_fm*(*y*#+*6*,*3*,*0*),
Member(*1*,*0*))))), *0*),
0, *succ*(*z*))))

lemma *satisfies_is_a_type* [*TC*]:
[| *A* ∈ *nat*; *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |]
==> *satisfies_is_a_fm*(*A*,*x*,*y*,*z*) ∈ *formula*
by (*simp add*: *satisfies_is_a_fm_def*)

lemma *sats_satisfies_is_a_fm* [*simp*]:
[| *u* ∈ *nat*; *x* < *length*(*env*); *y* < *length*(*env*); *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *satisfies_is_a_fm*(*u*,*x*,*y*,*z*), *env*) ⟷
satisfies_is_a(##*A*, *nth*(*u*,*env*), *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
apply (*frule_tac x=x* **in** *lt_length_in_nat*, *assumption*)
apply (*frule_tac x=y* **in** *lt_length_in_nat*, *assumption*)
apply (*simp add*: *satisfies_is_a_fm_def satisfies_is_a_def sats_lambda_fm*
sats_bool_of_o_fm)
done

lemma *satisfies_is_a_iff_sats*:
[| *nth*(*u*,*env*) = *nu*; *nth*(*x*,*env*) = *nx*; *nth*(*y*,*env*) = *ny*; *nth*(*z*,*env*) = *nz*;
u ∈ *nat*; *x* < *length*(*env*); *y* < *length*(*env*); *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *satisfies_is_a*(##*A*,*nu*,*nx*,*ny*,*nz*) ⟷
sats(*A*, *satisfies_is_a_fm*(*u*,*x*,*y*,*z*), *env*)
by *simp*

theorem *satisfies_is_a_reflection*:
REFLECTS[λ*x*. *satisfies_is_a*(*L*,*f*(*x*),*g*(*x*),*h*(*x*),*g*′(*x*)),
λ*i x*. *satisfies_is_a*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*),*g*′(*x*))]
apply (*unfold satisfies_is_a_def*)
apply (*intro FOL_reflections is_lambda_reflection bool_of_o_reflection*
nth_reflection is_list_reflection)
done

9.3.3 The Operator *satisfies_is_b*, Internalized

```
definition
  satisfies_is_b_fm :: [i,i,i,i]=>i where
 satisfies_is_b_fm(A,x,y,z) ==
   Forall(
     Implies(is_list_fm(succ(A),0),
       lambda_fm(
         bool_of_o_fm(Exists(And(nth_fm(x#+5,2,0), nth_fm(y#+5,2,0))), 0),
         0, succ(z))))

lemma satisfies_is_b_type [TC]:
     [| A ∈ nat; x ∈ nat; y ∈ nat; z ∈ nat |]
      ==> satisfies_is_b_fm(A,x,y,z) ∈ formula
by (simp add: satisfies_is_b_fm_def)

lemma sats_satisfies_is_b_fm [simp]:
   [| u ∈ nat; x < length(env); y < length(env); z ∈ nat; env ∈ list(A)|]
    ==> sats(A, satisfies_is_b_fm(u,x,y,z), env) ⟷
        satisfies_is_b(##A, nth(u,env), nth(x,env), nth(y,env), nth(z,env))
apply (frule_tac x=x in lt_length_in_nat, assumption)
apply (frule_tac x=y in lt_length_in_nat, assumption)
apply (simp add: satisfies_is_b_fm_def satisfies_is_b_def sats_lambda_fm
                 sats_bool_of_o_fm)
done

lemma satisfies_is_b_iff_sats:
  [| nth(u,env) = nu; nth(x,env) = nx; nth(y,env) = ny; nth(z,env) = nz;
      u ∈ nat; x < length(env); y < length(env); z ∈ nat; env ∈ list(A)|]
   ==> satisfies_is_b(##A,nu,nx,ny,nz) ⟷
       sats(A, satisfies_is_b_fm(u,x,y,z), env)
by simp

theorem satisfies_is_b_reflection:
     REFLECTS[λx. satisfies_is_b(L,f(x),g(x),h(x),g'(x)),
              λi x. satisfies_is_b(##Lset(i),f(x),g(x),h(x),g'(x))]
apply (unfold satisfies_is_b_def)
apply (intro FOL_reflections is_lambda_reflection bool_of_o_reflection
            nth_reflection is_list_reflection)
done
```

9.3.4 The Operator *satisfies_is_c*, Internalized

```
definition
  satisfies_is_c_fm :: [i,i,i,i,i]=>i where
 satisfies_is_c_fm(A,h,p,q,zz) ==
   Forall(
     Implies(is_list_fm(succ(A),0),
       lambda_fm(
         Exists(Exists(
```

```
        And(Exists(And(depth_apply_fm(h#+7,p#+7,0), fun_apply_fm(0,4,2))),
        And(Exists(And(depth_apply_fm(h#+7,q#+7,0), fun_apply_fm(0,4,1))),
              Exists(And(and_fm(2,1,0), not_fm(0,3))))))),
         0, succ(zz))))
```

lemma *satisfies_is_c_type* [*TC*]:
```
    [| A ∈ nat; h ∈ nat; x ∈ nat; y ∈ nat; z ∈ nat |]
     ==> satisfies_is_c_fm(A,h,x,y,z) ∈ formula
```
by (*simp add*: *satisfies_is_c_fm_def*)

lemma *sats_satisfies_is_c_fm* [*simp*]:
```
   [| u ∈ nat; v ∈ nat; x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)|]
    ==> sats(A, satisfies_is_c_fm(u,v,x,y,z), env) ⟷
        satisfies_is_c(##A, nth(u,env), nth(v,env), nth(x,env),
                       nth(y,env), nth(z,env))
```
by (*simp add*: *satisfies_is_c_fm_def satisfies_is_c_def sats_lambda_fm*)

lemma *satisfies_is_c_iff_sats*:
```
  [| nth(u,env) = nu; nth(v,env) = nv; nth(x,env) = nx; nth(y,env) = ny;
      nth(z,env) = nz;
      u ∈ nat; v ∈ nat; x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)|]
   ==> satisfies_is_c(##A,nu,nv,nx,ny,nz) ⟷
       sats(A, satisfies_is_c_fm(u,v,x,y,z), env)
```
by *simp*

theorem *satisfies_is_c_reflection*:
```
     REFLECTS[λx. satisfies_is_c(L,f(x),g(x),h(x),g'(x),h'(x)),
              λi x. satisfies_is_c(##Lset(i),f(x),g(x),h(x),g'(x),h'(x))]
```
apply (*unfold satisfies_is_c_def*)
apply (*intro FOL_reflections function_reflections is_lambda_reflection
extra_reflections nth_reflection depth_apply_reflection
is_list_reflection*)
done

9.3.5 The Operator *satisfies_is_d*, Internalized

definition
```
  satisfies_is_d_fm :: [i,i,i,i]=>i where
 satisfies_is_d_fm(A,h,p,zz) ==
   Forall(
     Implies(is_list_fm(succ(A),0),
       lambda_fm(
         Exists(
           And(depth_apply_fm(h#+5,p#+5,0),
               bool_of_o_fm(
                Forall(Forall(Forall(
                 Implies(Member(2,A#+8),
                  Implies(Cons_fm(2,5,1),
                   Implies(fun_apply_fm(3,1,0), number1_fm(0))))))), 1))),
```

```
        0, succ(zz))))

lemma satisfies_is_d_type [TC]:
    [| A ∈ nat; h ∈ nat; x ∈ nat; z ∈ nat |]
     ==> satisfies_is_d_fm(A,h,x,z) ∈ formula
by (simp add: satisfies_is_d_fm_def)

lemma sats_satisfies_is_d_fm [simp]:
   [| u ∈ nat; x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)|]
    ==> sats(A, satisfies_is_d_fm(u,x,y,z), env) ⟷
        satisfies_is_d(##A, nth(u,env), nth(x,env), nth(y,env), nth(z,env))
by (simp add: satisfies_is_d_fm_def satisfies_is_d_def sats_lambda_fm
              sats_bool_of_o_fm)

lemma satisfies_is_d_iff_sats:
  [| nth(u,env) = nu; nth(x,env) = nx; nth(y,env) = ny; nth(z,env) = nz;
     u ∈ nat; x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)|]
   ==> satisfies_is_d(##A,nu,nx,ny,nz) ⟷
       sats(A, satisfies_is_d_fm(u,x,y,z), env)
by simp

theorem satisfies_is_d_reflection:
    REFLECTS[λx. satisfies_is_d(L,f(x),g(x),h(x),g'(x)),
             λi x. satisfies_is_d(##Lset(i),f(x),g(x),h(x),g'(x))]
apply (unfold satisfies_is_d_def)
apply (intro FOL_reflections function_reflections is_lambda_reflection
            extra_reflections nth_reflection depth_apply_reflection
            is_list_reflection)
done
```

9.3.6 The Operator *satisfies_MH*, Internalized

```
definition
  satisfies_MH_fm :: [i,i,i,i]=>i where
 satisfies_MH_fm(A,u,f,zz) ==
   Forall(
     Implies(is_formula_fm(0),
       lambda_fm(
         formula_case_fm(satisfies_is_a_fm(A#+7,2,1,0),
                         satisfies_is_b_fm(A#+7,2,1,0),
                         satisfies_is_c_fm(A#+7,f#+7,2,1,0),
                         satisfies_is_d_fm(A#+6,f#+6,1,0),
                         1, 0),
         0, succ(zz))))

lemma satisfies_MH_type [TC]:
    [| A ∈ nat; u ∈ nat; x ∈ nat; z ∈ nat |]
     ==> satisfies_MH_fm(A,u,x,z) ∈ formula
by (simp add: satisfies_MH_fm_def)
```

lemma *sats_satisfies_MH_fm* [*simp*]:
[| $u \in nat$; $x \in nat$; $y \in nat$; $z \in nat$; $env \in list(A)$|]
==> $sats(A,\ satisfies_MH_fm(u,x,y,z),\ env) \longleftrightarrow$
$satisfies_MH(\#\#A,\ nth(u,env),\ nth(x,env),\ nth(y,env),\ nth(z,env))$
by (*simp add*: *satisfies_MH_fm_def satisfies_MH_def sats_lambda_fm*
sats_formula_case_fm)

lemma *satisfies_MH_iff_sats*:
[| $nth(u,env) = nu$; $nth(x,env) = nx$; $nth(y,env) = ny$; $nth(z,env) = nz$;
$u \in nat$; $x \in nat$; $y \in nat$; $z \in nat$; $env \in list(A)$|]
==> $satisfies_MH(\#\#A,nu,nx,ny,nz) \longleftrightarrow$
$sats(A,\ satisfies_MH_fm(u,x,y,z),\ env)$
by *simp*

lemmas *satisfies_reflections* =
is_lambda_reflection is_formula_reflection
is_formula_case_reflection
satisfies_is_a_reflection satisfies_is_b_reflection
satisfies_is_c_reflection satisfies_is_d_reflection

theorem *satisfies_MH_reflection*:
$REFLECTS[\lambda x.\ satisfies_MH(L,f(x),g(x),h(x),g'(x)),$
$\lambda i\ x.\ satisfies_MH(\#\#Lset(i),f(x),g(x),h(x),g'(x))]$
apply (*unfold satisfies_MH_def*)
apply (*intro FOL_reflections satisfies_reflections*)
done

9.4 Lemmas for Instantiating the Locale *M_satisfies*

9.4.1 The *Member* Case

lemma *Member_Reflects*:
$REFLECTS[\lambda u.\ \exists v[L].\ v \in B \wedge (\exists bo[L].\ \exists nx[L].\ \exists ny[L].$
$v \in lstA \wedge is_nth(L,x,v,nx) \wedge is_nth(L,y,v,ny) \wedge$
$is_bool_of_o(L,\ nx \in ny,\ bo) \wedge pair(L,v,bo,u)),$
$\lambda i\ u.\ \exists v \in Lset(i).\ v \in B \wedge (\exists bo \in Lset(i).\ \exists nx \in Lset(i).\ \exists ny \in Lset(i).$
$v \in lstA \wedge is_nth(\#\#Lset(i),\ x,\ v,\ nx) \wedge$
$is_nth(\#\#Lset(i),\ y,\ v,\ ny) \wedge$
$is_bool_of_o(\#\#Lset(i),\ nx \in ny,\ bo) \wedge pair(\#\#Lset(i),\ v,\ bo,\ u))]$
by (*intro FOL_reflections function_reflections nth_reflection*
bool_of_o_reflection)

lemma *Member_replacement*:
[|$L(A)$; $x \in nat$; $y \in nat$|]
==> *strong_replacement*
$(L,\ \lambda env\ z.\ \exists bo[L].\ \exists nx[L].\ \exists ny[L].$
$env \in list(A)\ \&\ is_nth(L,x,env,nx)\ \&\ is_nth(L,y,env,ny)\ \&$
$is_bool_of_o(L,\ nx \in ny,\ bo)\ \&$

```
            pair(L, env, bo, z))
apply (rule strong_replacementI)
apply (rule_tac u={list(A),B,x,y}
        in gen_separation_multi [OF Member_Reflects],
      auto)
apply (rule_tac env=[list(A),B,x,y] in DPow_LsetI)
apply (rule sep_rules nth_iff_sats is_bool_of_o_iff_sats | simp)+
done
```

9.4.2 The *Equal* Case

```
lemma Equal_Reflects:
 REFLECTS[λu. ∃ v[L]. v ∈ B ∧ (∃ bo[L]. ∃ nx[L]. ∃ ny[L].
        v ∈ lstA ∧ is_nth(L, x, v, nx) ∧ is_nth(L, y, v, ny) ∧
        is_bool_of_o(L, nx = ny, bo) ∧ pair(L, v, bo, u)),
   λi u. ∃ v ∈ Lset(i). v ∈ B ∧ (∃ bo ∈ Lset(i). ∃ nx ∈ Lset(i). ∃ ny ∈ Lset(i).
           v ∈ lstA ∧ is_nth(##Lset(i), x, v, nx) ∧
           is_nth(##Lset(i), y, v, ny) ∧
        is_bool_of_o(##Lset(i), nx = ny, bo) ∧ pair(##Lset(i), v, bo, u))]
by (intro FOL_reflections function_reflections nth_reflection
        bool_of_o_reflection)


lemma Equal_replacement:
   [|L(A); x ∈ nat; y ∈ nat|]
    ==> strong_replacement
        (L, λenv z. ∃ bo[L]. ∃ nx[L]. ∃ ny[L].
            env ∈ list(A) & is_nth(L,x,env,nx) & is_nth(L,y,env,ny) &
            is_bool_of_o(L, nx = ny, bo) &
            pair(L, env, bo, z))
apply (rule strong_replacementI)
apply (rule_tac u={list(A),B,x,y}
        in gen_separation_multi [OF Equal_Reflects],
      auto)
apply (rule_tac env=[list(A),B,x,y] in DPow_LsetI)
apply (rule sep_rules nth_iff_sats is_bool_of_o_iff_sats | simp)+
done
```

9.4.3 The *Nand* Case

```
lemma Nand_Reflects:
   REFLECTS [λx. ∃ u[L]. u ∈ B ∧
            (∃ rpe[L]. ∃ rqe[L]. ∃ andpq[L]. ∃ notpq[L].
              fun_apply(L, rp, u, rpe) ∧ fun_apply(L, rq, u, rqe) ∧
              is_and(L, rpe, rqe, andpq) ∧ is_not(L, andpq, notpq) ∧
              u ∈ list(A) ∧ pair(L, u, notpq, x)),
   λi x. ∃ u ∈ Lset(i). u ∈ B ∧
    (∃ rpe ∈ Lset(i). ∃ rqe ∈ Lset(i). ∃ andpq ∈ Lset(i). ∃ notpq ∈ Lset(i).
      fun_apply(##Lset(i), rp, u, rpe) ∧ fun_apply(##Lset(i), rq, u, rqe) ∧
      is_and(##Lset(i), rpe, rqe, andpq) ∧ is_not(##Lset(i), andpq, notpq) ∧
```

```
      u ∈ list(A) ∧ pair(##Lset(i), u, notpq, x))]
apply (unfold is_and_def is_not_def)
apply (intro FOL_reflections function_reflections)
done

lemma Nand_replacement:
   [|L(A); L(rp); L(rq)|]
    ==> strong_replacement
        (L, λenv z. ∃ rpe[L]. ∃ rqe[L]. ∃ andpq[L]. ∃ notpq[L].
              fun_apply(L,rp,env,rpe) & fun_apply(L,rq,env,rqe) &
              is_and(L,rpe,rqe,andpq) & is_not(L,andpq,notpq) &
              env ∈ list(A) & pair(L, env, notpq, z))
apply (rule strong_replacementI)
apply (rule_tac u={list(A),B,rp,rq}
        in gen_separation_multi [OF Nand_Reflects],
      auto)
apply (rule_tac env=[list(A),B,rp,rq] in DPow_LsetI)
apply (rule sep_rules is_and_iff_sats is_not_iff_sats | simp)+
done
```

9.4.4 The *Forall* Case

```
lemma Forall_Reflects:
 REFLECTS [λx. ∃ u[L]. u ∈ B ∧ (∃ bo[L]. u ∈ list(A) ∧
                is_bool_of_o (L,
    ∀ a[L]. ∀ co[L]. ∀ rpco[L]. a ∈ A ⟶
              is_Cons(L,a,u,co) ⟶ fun_apply(L,rp,co,rpco) ⟶
              number1(L,rpco),
                        bo) ∧ pair(L,u,bo,x)),
 λi x. ∃ u ∈ Lset(i). u ∈ B ∧ (∃ bo ∈ Lset(i). u ∈ list(A) ∧
        is_bool_of_o (##Lset(i),
 ∀ a ∈ Lset(i). ∀ co ∈ Lset(i). ∀ rpco ∈ Lset(i). a ∈ A ⟶
           is_Cons(##Lset(i),a,u,co) ⟶ fun_apply(##Lset(i),rp,co,rpco) ⟶
           number1(##Lset(i),rpco),
                     bo) ∧ pair(##Lset(i),u,bo,x))]
apply (unfold is_bool_of_o_def)
apply (intro FOL_reflections function_reflections Cons_reflection)
done

lemma Forall_replacement:
   [|L(A); L(rp)|]
    ==> strong_replacement
        (L, λenv z. ∃ bo[L].
              env ∈ list(A) &
              is_bool_of_o (L,
                          ∀ a[L]. ∀ co[L]. ∀ rpco[L].
                             a∈A ⟶ is_Cons(L,a,env,co) ⟶
                             fun_apply(L,rp,co,rpco) ⟶ number1(L, rpco),
                          bo) &
```

```
            pair(L,env,bo,z))
apply (rule strong_replacementI)
apply (rule_tac u={A,list(A),B,rp}
        in gen_separation_multi [OF Forall_Reflects],
      auto)
apply (rule_tac env=[A,list(A),B,rp] in DPow_LsetI)
apply (rule sep_rules is_bool_of_o_iff_sats Cons_iff_sats | simp)+
done
```

9.4.5 The *transrec_replacement* Case

```
lemma formula_rec_replacement_Reflects:
 REFLECTS [λx. ∃u[L]. u ∈ B ∧ (∃y[L]. pair(L, u, y, x) ∧
            is_wfrec (L, satisfies_MH(L,A), mesa, u, y)),
    λi x. ∃u ∈ Lset(i). u ∈ B ∧ (∃y ∈ Lset(i). pair(##Lset(i), u, y, x) ∧
            is_wfrec (##Lset(i), satisfies_MH(##Lset(i),A), mesa, u, y))]
by (intro FOL_reflections function_reflections satisfies_MH_reflection
         is_wfrec_reflection)

lemma formula_rec_replacement:
      — For the transrec
   [|n ∈ nat; L(A)|] ==> transrec_replacement(L, satisfies_MH(L,A), n)
apply (rule L.transrec_replacementI, simp add: L.nat_into_M)
apply (rule strong_replacementI)
apply (rule_tac u={B,A,n,Memrel(eclose({n}))}
        in gen_separation_multi [OF formula_rec_replacement_Reflects],
      auto simp add: L.nat_into_M)
apply (rule_tac env=[B,A,n,Memrel(eclose({n}))] in DPow_LsetI)
apply (rule sep_rules satisfies_MH_iff_sats is_wfrec_iff_sats | simp)+
done
```

9.4.6 The Lambda Replacement Case

```
lemma formula_rec_lambda_replacement_Reflects:
 REFLECTS [λx. ∃u[L]. u ∈ B &
     mem_formula(L,u) &
     (∃c[L].
         is_formula_case
          (L, satisfies_is_a(L,A), satisfies_is_b(L,A),
           satisfies_is_c(L,A,g), satisfies_is_d(L,A,g),
           u, c) &
         pair(L,u,c,x)),
  λi x. ∃u ∈ Lset(i). u ∈ B & mem_formula(##Lset(i),u) &
     (∃c ∈ Lset(i).
         is_formula_case
          (##Lset(i), satisfies_is_a(##Lset(i),A), satisfies_is_b(##Lset(i),A),
           satisfies_is_c(##Lset(i),A,g), satisfies_is_d(##Lset(i),A,g),
           u, c) &
         pair(##Lset(i),u,c,x))]
by (intro FOL_reflections function_reflections mem_formula_reflection
```

```
        is_formula_case_reflection satisfies_is_a_reflection
        satisfies_is_b_reflection satisfies_is_c_reflection
        satisfies_is_d_reflection)

lemma formula_rec_lambda_replacement:
    — For the transrec
  [|L(g); L(A)|] ==>
   strong_replacement (L,
     λx y. mem_formula(L,x) &
          (∃ c[L]. is_formula_case(L, satisfies_is_a(L,A),
                                   satisfies_is_b(L,A),
                                   satisfies_is_c(L,A,g),
                                   satisfies_is_d(L,A,g), x, c) &
          pair(L, x, c, y)))
apply (rule strong_replacementI)
apply (rule_tac u={B,A,g}
       in gen_separation_multi [OF formula_rec_lambda_replacement_Reflects],

      auto)
apply (rule_tac env=[A,g,B] in DPow_LsetI)
apply (rule sep_rules mem_formula_iff_sats
        formula_case_iff_sats satisfies_is_a_iff_sats
        satisfies_is_b_iff_sats satisfies_is_c_iff_sats
        satisfies_is_d_iff_sats | simp)+
done
```

9.5 Instantiating *M_satisfies*

```
lemma M_satisfies_axioms_L: M_satisfies_axioms(L)
  apply (rule M_satisfies_axioms.intro)
      apply (assumption | rule
        Member_replacement Equal_replacement
        Nand_replacement Forall_replacement
        formula_rec_replacement formula_rec_lambda_replacement)+
  done

theorem M_satisfies_L: M_satisfies(L)
  apply (rule M_satisfies.intro)
   apply (rule M_eclose_L)
   apply (rule M_datatypes_L)
  apply (rule M_satisfies_axioms_L)
  done
```

Finally: the point of the whole theory!

```
lemmas satisfies_closed = M_satisfies.satisfies_closed [OF M_satisfies_L]
   and satisfies_abs = M_satisfies.satisfies_abs [OF M_satisfies_L]

end
```

10 Absoluteness for the Definable Powerset Function

theory *DPow_absolute* **imports** *Satisfies_absolute* **begin**

10.1 Preliminary Internalizations

10.1.1 The Operator *is_formula_rec*

The three arguments of p are always 2, 1, 0. It is buried within 11 quantifiers!!

definition
formula_rec_fm :: [*i*, *i*, *i*]=>*i* **where**
formula_rec_fm(*mh*,*p*,*z*) ==
Exists(*Exists*(*Exists*(
And(*finite_ordinal_fm*(*2*),
And(*depth_fm*(*p*#+*3*,*2*),
And(*succ_fm*(*2*,*1*),
And(*fun_apply_fm*(*0*,*p*#+*3*,*z*#+*3*), *is_transrec_fm*(*mh*,*1*,*0*))))))))

lemma *is_formula_rec_type* [*TC*]:
[| *p* ∈ *formula*; *x* ∈ *nat*; *z* ∈ *nat* |]
==> *formula_rec_fm*(*p*,*x*,*z*) ∈ *formula*
by (*simp add*: *formula_rec_fm_def*)

lemma *sats_formula_rec_fm*:
assumes *MH_iff_sats*:
!!*a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10*.
[|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*; *a3*∈*A*; *a4*∈*A*; *a5*∈*A*; *a6*∈*A*; *a7*∈*A*; *a8*∈*A*; *a9*∈*A*; *a10*∈*A*|]
==> *MH*(*a2*, *a1*, *a0*) ⟷
sats(*A*, *p*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*Cons*(*a3*,
Cons(*a4*,*Cons*(*a5*,*Cons*(*a6*,*Cons*(*a7*,
Cons(*a8*,*Cons*(*a9*,*Cons*(*a10*,*env*))))))))))))
shows
[|*x* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *formula_rec_fm*(*p*,*x*,*z*), *env*) ⟷
is_formula_rec(##*A*, *MH*, *nth*(*x*,*env*), *nth*(*z*,*env*))
by (*simp add*: *formula_rec_fm_def sats_is_transrec_fm is_formula_rec_def*
MH_iff_sats [*THEN iff_sym*])

lemma *formula_rec_iff_sats*:
assumes *MH_iff_sats*:
!!*a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10*.
[|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*; *a3*∈*A*; *a4*∈*A*; *a5*∈*A*; *a6*∈*A*; *a7*∈*A*; *a8*∈*A*; *a9*∈*A*; *a10*∈*A*|]
==> *MH*(*a2*, *a1*, *a0*) ⟷
sats(*A*, *p*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*Cons*(*a3*,
Cons(*a4*,*Cons*(*a5*,*Cons*(*a6*,*Cons*(*a7*,

Cons(*a8*,*Cons*(*a9*,*Cons*(*a10*,*env*)))))))))))
shows
[|*nth*(*i*,*env*) = *x*; *nth*(*k*,*env*) = *z*;
i ∈ *nat*; *k* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_formula_rec*(##*A*, *MH*, *x*, *z*) ⟷ *sats*(*A*, *formula_rec_fm*(*p*,*i*,*k*), *env*)
by (*simp add*: *sats_formula_rec_fm* [*OF MH_iff_sats*])

theorem *formula_rec_reflection*:
assumes *MH_reflection*:
!!*f′ f g h*. *REFLECTS*[λ*x*. *MH*(*L*, *f′*(*x*), *f*(*x*), *g*(*x*), *h*(*x*)),
λ*i x*. *MH*(##*Lset*(*i*), *f′*(*x*), *f*(*x*), *g*(*x*), *h*(*x*))]
shows *REFLECTS*[λ*x*. *is_formula_rec*(*L*, *MH*(*L*,*x*), *f*(*x*), *h*(*x*)),
λ*i x*. *is_formula_rec*(##*Lset*(*i*), *MH*(##*Lset*(*i*),*x*), *f*(*x*), *h*(*x*))]
apply (*simp* (*no_asm_use*) *only*: *is_formula_rec_def*)
apply (*intro FOL_reflections function_reflections fun_plus_reflections*
depth_reflection is_transrec_reflection MH_reflection)
done

10.1.2 The Operator *is_satisfies*

definition
satisfies_fm :: [*i*,*i*,*i*]=>*i* **where**
satisfies_fm(*x*) == *formula_rec_fm* (*satisfies_MH_fm*(*x*#+5#+6, *2*, *1*, *0*))

lemma *is_satisfies_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *satisfies_fm*(*x*,*y*,*z*) ∈ *formula*
by (*simp add*: *satisfies_fm_def*)

lemma *sats_satisfies_fm* [*simp*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *satisfies_fm*(*x*,*y*,*z*), *env*) ⟷
is_satisfies(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
by (*simp add*: *satisfies_fm_def is_satisfies_def sats_formula_rec_fm*)

lemma *satisfies_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
i ∈ *nat*; *j* ∈ *nat*; *k* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_satisfies*(##*A*, *x*, *y*, *z*) ⟷ *sats*(*A*, *satisfies_fm*(*i*,*j*,*k*), *env*)
by (*simp*)

theorem *satisfies_reflection*:
REFLECTS[λ*x*. *is_satisfies*(*L*,*f*(*x*),*g*(*x*),*h*(*x*)),
λ*i x*. *is_satisfies*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*))]
apply (*simp only*: *is_satisfies_def*)
apply (*intro formula_rec_reflection satisfies_MH_reflection*)
done

10.2 Relativization of the Operator *DPow′*

lemma *DPow′_eq*:
DPow′(*A*) = {*z* . *ep* ∈ *list*(*A*) ∗ *formula*,
∃ *env* ∈ *list*(*A*). ∃ *p* ∈ *formula*.
ep = <*env*,*p*> & *z* = {*x*∈*A*. *sats*(*A*, *p*, *Cons*(*x*,*env*))}}
by (*simp add*: *DPow′_def*, *blast*)

Relativize the use of λ*A p env*. *sats*(*A*, *p*, *env*) within *DPow′* (the comprehension).

definition
is_DPow_sats :: [*i*=>*o*,*i*,*i*,*i*,*i*] => *o* **where**
is_DPow_sats(*M*,*A*,*env*,*p*,*x*) ==
∀ *n1*[*M*]. ∀ *e*[*M*]. ∀ *sp*[*M*].
is_satisfies(*M*,*A*,*p*,*sp*) ⟶ *is_Cons*(*M*,*x*,*env*,*e*) ⟶
fun_apply(*M*, *sp*, *e*, *n1*) ⟶ *number1*(*M*, *n1*)

lemma (**in** *M_satisfies*) *DPow_sats_abs*:
[| *M*(*A*); *env* ∈ *list*(*A*); *p* ∈ *formula*; *M*(*x*) |]
==> *is_DPow_sats*(*M*,*A*,*env*,*p*,*x*) ⟷ *sats*(*A*, *p*, *Cons*(*x*,*env*))
apply (*subgoal_tac M*(*env*))
apply (*simp add*: *is_DPow_sats_def satisfies_closed satisfies_abs*)
apply (*blast dest*: *transM*)
done

lemma (**in** *M_satisfies*) *Collect_DPow_sats_abs*:
[| *M*(*A*); *env* ∈ *list*(*A*); *p* ∈ *formula* |]
==> *Collect*(*A*, *is_DPow_sats*(*M*,*A*,*env*,*p*)) =
{*x* ∈ *A*. *sats*(*A*, *p*, *Cons*(*x*,*env*))}
by (*simp add*: *DPow_sats_abs transM* [*of* _ *A*])

10.2.1 The Operator *is_DPow_sats*, Internalized

definition
DPow_sats_fm :: [*i*,*i*,*i*,*i*]=>*i* **where**
DPow_sats_fm(*A*,*env*,*p*,*x*) ==
Forall(*Forall*(*Forall*(
Implies(*satisfies_fm*(*A*#+*3*,*p*#+*3*,*0*),
Implies(*Cons_fm*(*x*#+*3*,*env*#+*3*,*1*),
Implies(*fun_apply_fm*(*0*,*1*,*2*), *number1_fm*(*2*)))))))

lemma *is_DPow_sats_type* [*TC*]:
[| *A* ∈ *nat*; *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |]
==> *DPow_sats_fm*(*A*,*x*,*y*,*z*) ∈ *formula*
by (*simp add*: *DPow_sats_fm_def*)

lemma *sats_DPow_sats_fm* [*simp*]:
[| *u* ∈ *nat*; *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *DPow_sats_fm*(*u*,*x*,*y*,*z*), *env*) ⟷
is_DPow_sats(##*A*, *nth*(*u*,*env*), *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))

by (*simp add*: *DPow_sats_fm_def is_DPow_sats_def*)

lemma *DPow_sats_iff_sats*:
[| *nth*(*u*,*env*) = *nu*; *nth*(*x*,*env*) = *nx*; *nth*(*y*,*env*) = *ny*; *nth*(*z*,*env*) = *nz*;
u ∈ *nat*; *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_DPow_sats*(##*A*,*nu*,*nx*,*ny*,*nz*) ⟷
sats(*A*, *DPow_sats_fm*(*u*,*x*,*y*,*z*), *env*)
by *simp*

theorem *DPow_sats_reflection*:
REFLECTS[λ*x*. *is_DPow_sats*(*L*,*f*(*x*),*g*(*x*),*h*(*x*),*g*′(*x*)),
λ*i x*. *is_DPow_sats*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*),*g*′(*x*))]
apply (*unfold is_DPow_sats_def*)
apply (*intro FOL_reflections function_reflections extra_reflections*
satisfies_reflection)
done

10.3 A Locale for Relativizing the Operator *DPow′*

locale *M_DPow* = *M_satisfies* +
assumes *sep*:
[| *M*(*A*); *env* ∈ *list*(*A*); *p* ∈ *formula* |]
==> *separation*(*M*, λ*x*. *is_DPow_sats*(*M*,*A*,*env*,*p*,*x*))
and *rep*:
M(*A*)
==> *strong_replacement* (*M*,
λ*ep z*. ∃ *env*[*M*]. ∃ *p*[*M*]. *mem_formula*(*M*,*p*) & *mem_list*(*M*,*A*,*env*) &
pair(*M*,*env*,*p*,*ep*) &
is_Collect(*M*, *A*, λ*x*. *is_DPow_sats*(*M*,*A*,*env*,*p*,*x*), *z*))

lemma (**in** *M_DPow*) *sep′*:
[| *M*(*A*); *env* ∈ *list*(*A*); *p* ∈ *formula* |]
==> *separation*(*M*, λ*x*. *sats*(*A*, *p*, *Cons*(*x*,*env*)))
by (*insert sep* [*of A env p*], *simp add*: *DPow_sats_abs*)

lemma (**in** *M_DPow*) *rep′*:
M(*A*)
==> *strong_replacement* (*M*,
λ*ep z*. ∃ *env*∈*list*(*A*). ∃ *p*∈*formula*.
ep = <*env*,*p*> & *z* = {*x* ∈ *A* . *sats*(*A*, *p*, *Cons*(*x*, *env*))})
by (*insert rep* [*of A*], *simp add*: *Collect_DPow_sats_abs*)

lemma *univalent_pair_eq*:
univalent (*M*, *A*, λ*xy z*. ∃ *x*∈*B*. ∃ *y*∈*C*. *xy* = ⟨*x*,*y*⟩ ∧ *z* = *f*(*x*,*y*))
by (*simp add*: *univalent_def*, *blast*)

lemma (**in** *M_DPow*) *DPow′_closed*: *M*(*A*) ==> *M*(*DPow′*(*A*))
apply (*simp add*: *DPow′_eq*)

apply (*fast intro*: *rep′ sep′ univalent_pair_eq*)
done

Relativization of the Operator *DPow′*

definition
is_DPow′ :: [*i*=>*o*,*i*,*i*] => *o* **where**
is_DPow′(*M*,*A*,*Z*) ==
∀ *X*[*M*]. *X* ∈ *Z* ⟷
subset(*M*,*X*,*A*) &
(∃ *env*[*M*]. ∃ *p*[*M*]. *mem_formula*(*M*,*p*) & *mem_list*(*M*,*A*,*env*) &
is_Collect(*M*, *A*, *is_DPow_sats*(*M*,*A*,*env*,*p*), *X*))

lemma (**in** *M_DPow*) *DPow′_abs*:
[|*M*(*A*); *M*(*Z*)|] ==> *is_DPow′*(*M*,*A*,*Z*) ⟷ *Z* = *DPow′*(*A*)
apply (*rule iffI*)
prefer *2* **apply** (*simp add*: *is_DPow′_def DPow′_def Collect_DPow_sats_abs*)

apply (*rule M_equalityI*)
apply (*simp add*: *is_DPow′_def DPow′_def Collect_DPow_sats_abs*, *assumption*)
apply (*erule DPow′_closed*)
done

10.4 Instantiating the Locale *M_DPow*

10.4.1 The Instance of Separation

lemma *DPow_separation*:
[| *L*(*A*); *env* ∈ *list*(*A*); *p* ∈ *formula* |]
==> *separation*(*L*, λ*x*. *is_DPow_sats*(*L*,*A*,*env*,*p*,*x*))
apply (*rule gen_separation_multi* [*OF DPow_sats_reflection*, *of* {*A*,*env*,*p*}],
auto intro: *transL*)
apply (*rule_tac env*=[*A*,*env*,*p*] **in** *DPow_LsetI*)
apply (*rule DPow_sats_iff_sats sep_rules* | *simp*)+
done

10.4.2 The Instance of Replacement

lemma *DPow_replacement_Reflects*:
REFLECTS [λ*x*. ∃ *u*[*L*]. *u* ∈ *B* &
(∃ *env*[*L*]. ∃ *p*[*L*].
mem_formula(*L*,*p*) & *mem_list*(*L*,*A*,*env*) & *pair*(*L*,*env*,*p*,*u*) &
is_Collect (*L*, *A*, *is_DPow_sats*(*L*,*A*,*env*,*p*), *x*)),
λ*i x*. ∃ *u* ∈ *Lset*(*i*). *u* ∈ *B* &
(∃ *env* ∈ *Lset*(*i*). ∃ *p* ∈ *Lset*(*i*).
mem_formula(##*Lset*(*i*),*p*) & *mem_list*(##*Lset*(*i*),*A*,*env*) &
pair(##*Lset*(*i*),*env*,*p*,*u*) &
is_Collect (##*Lset*(*i*), *A*, *is_DPow_sats*(##*Lset*(*i*),*A*,*env*,*p*), *x*))]
apply (*unfold is_Collect_def*)
apply (*intro FOL_reflections function_reflections mem_formula_reflection*

```
        mem_list_reflection DPow_sats_reflection)
done

lemma DPow_replacement:
   L(A)
   ==> strong_replacement (L,
       λep z. ∃ env[L]. ∃ p[L]. mem_formula(L,p) & mem_list(L,A,env) &
                pair(L,env,p,ep) &
                is_Collect(L, A, λx. is_DPow_sats(L,A,env,p,x), z))
apply (rule strong_replacementI)
apply (rule_tac u={A,B}
        in gen_separation_multi [OF DPow_replacement_Reflects],
      auto)
apply (unfold is_Collect_def)
apply (rule_tac env=[A,B] in DPow_LsetI)
apply (rule sep_rules mem_formula_iff_sats mem_list_iff_sats
          DPow_sats_iff_sats | simp)+
done
```

10.4.3 Actually Instantiating the Locale

```
lemma M_DPow_axioms_L: M_DPow_axioms(L)
  apply (rule M_DPow_axioms.intro)
   apply (assumption | rule DPow_separation DPow_replacement)+
  done

theorem M_DPow_L: M_DPow(L)
  apply (rule M_DPow.intro)
   apply (rule M_satisfies_L)
  apply (rule M_DPow_axioms_L)
  done

lemmas DPow'_closed [intro, simp] = M_DPow.DPow'_closed [OF M_DPow_L]
  and DPow'_abs [intro, simp] = M_DPow.DPow'_abs [OF M_DPow_L]
```

10.4.4 The Operator *is_Collect*

The formula *is_P* has one free variable, 0, and it is enclosed within a single quantifier.

```
definition
  Collect_fm :: [i, i, i]=>i where
 Collect_fm(A,is_P,z) ==
       Forall(Iff(Member(0,succ(z)),
                 And(Member(0,succ(A)), is_P)))

lemma is_Collect_type [TC]:
     [| is_P ∈ formula; x ∈ nat; y ∈ nat |]
      ==> Collect_fm(x,is_P,y) ∈ formula
by (simp add: Collect_fm_def)
```

lemma *sats_Collect_fm*:
  assumes *is_P_iff_sats*:
    $!!a.\ a \in A ==> is_P(a) \longleftrightarrow sats(A,\ p,\ Cons(a,\ env))$
  shows
    $[|x \in nat;\ y \in nat;\ env \in list(A)|]$
    $==> sats(A,\ Collect_fm(x,p,y),\ env) \longleftrightarrow$
      $is_Collect(\#\#A,\ nth(x,env),\ is_P,\ nth(y,env))$
by (*simp add*: *Collect_fm_def is_Collect_def is_P_iff_sats* [*THEN iff_sym*])

lemma *Collect_iff_sats*:
  assumes *is_P_iff_sats*:
    $!!a.\ a \in A ==> is_P(a) \longleftrightarrow sats(A,\ p,\ Cons(a,\ env))$
  shows
  $[|\ nth(i,env) = x;\ nth(j,env) = y;$
    $i \in nat;\ j \in nat;\ env \in list(A)|]$
  $==> is_Collect(\#\#A,\ x,\ is_P,\ y) \longleftrightarrow sats(A,\ Collect_fm(i,p,j),\ env)$
by (*simp add*: *sats_Collect_fm* [*OF is_P_iff_sats*])

The second argument of *is_P* gives it direct access to x, which is essential for handling free variable references.

theorem *Collect_reflection*:
  assumes *is_P_reflection*:
    $!!h\ f\ g.\ REFLECTS[\lambda x.\ is_P(L,\ f(x),\ g(x)),$
      $\lambda i\ x.\ is_P(\#\#Lset(i),\ f(x),\ g(x))]$
  shows $REFLECTS[\lambda x.\ is_Collect(L,\ f(x),\ is_P(L,x),\ g(x)),$
      $\lambda i\ x.\ is_Collect(\#\#Lset(i),\ f(x),\ is_P(\#\#Lset(i),\ x),\ g(x))]$
apply (*simp* (*no_asm_use*) *only*: *is_Collect_def*)
apply (*intro FOL_reflections is_P_reflection*)
done

10.4.5 The Operator *is_Replace*

BEWARE! The formula *is_P* has free variables 0, 1 and not the usual 1, 0! It is enclosed within two quantifiers.

definition
  $Replace_fm :: [i,\ i,\ i]=>i$ **where**
  $Replace_fm(A,is_P,z) ==$
    $Forall(Iff(Member(0,succ(z)),$
      $Exists(And(Member(0,A\#+2),\ is_P))))$

lemma *is_Replace_type* [*TC*]:
  $[|\ is_P \in formula;\ x \in nat;\ y \in nat\ |]$
  $==> Replace_fm(x,is_P,y) \in formula$
by (*simp add*: *Replace_fm_def*)

lemma *sats_Replace_fm*:
  assumes *is_P_iff_sats*:

```
    !!a b. [|a ∈ A; b ∈ A|]
            ==> is_P(a,b) ⟷ sats(A, p, Cons(a,Cons(b,env)))
  shows
    [|x ∈ nat; y ∈ nat; env ∈ list(A)|]
     ==> sats(A, Replace_fm(x,p,y), env) ⟷
         is_Replace(##A, nth(x,env), is_P, nth(y,env))
by (simp add: Replace_fm_def is_Replace_def is_P_iff_sats [THEN iff_sym])

lemma Replace_iff_sats:
  assumes is_P_iff_sats:
    !!a b. [|a ∈ A; b ∈ A|]
            ==> is_P(a,b) ⟷ sats(A, p, Cons(a,Cons(b,env)))
  shows
  [| nth(i,env) = x; nth(j,env) = y;
     i ∈ nat; j ∈ nat; env ∈ list(A)|]
   ==> is_Replace(##A, x, is_P, y) ⟷ sats(A, Replace_fm(i,p,j), env)
by (simp add: sats_Replace_fm [OF is_P_iff_sats])
```

The second argument of *is_P* gives it direct access to x, which is essential for handling free variable references.

```
theorem Replace_reflection:
  assumes is_P_reflection:
   !!h f g. REFLECTS[λx. is_P(L, f(x), g(x), h(x)),
                     λi x. is_P(##Lset(i), f(x), g(x), h(x))]
  shows REFLECTS[λx. is_Replace(L, f(x), is_P(L,x), g(x)),
                 λi x. is_Replace(##Lset(i), f(x), is_P(##Lset(i), x), g(x))]
apply (simp (no_asm_use) only: is_Replace_def)
apply (intro FOL_reflections is_P_reflection)
done
```

10.4.6 The Operator *is_DPow′*, Internalized

```
definition
  DPow′_fm :: [i,i]=>i where
   DPow′_fm(A,Z) ==
     Forall(
      Iff(Member(0,succ(Z)),
       And(subset_fm(0,succ(A)),
        Exists(Exists(
         And(mem_formula_fm(0),
          And(mem_list_fm(A#+3,1),
           Collect_fm(A#+3,
                      DPow_sats_fm(A#+4, 2, 1, 0), 2))))))))

lemma is_DPow′_type [TC]:
    [| x ∈ nat; y ∈ nat |] ==> DPow′_fm(x,y) ∈ formula
by (simp add: DPow′_fm_def)

lemma sats_DPow′_fm [simp]:
```

[| $x \in nat$; $y \in nat$; $env \in list(A)$|]
==> $sats(A,\ DPow'_fm(x,y),\ env) \longleftrightarrow$
$is_DPow'(\#\#A,\ nth(x,env),\ nth(y,env))$
by (*simp add*: *DPow'_fm_def is_DPow'_def sats_subset_fm' sats_Collect_fm*)

lemma *DPow'_iff_sats*:
[| $nth(i,env) = x$; $nth(j,env) = y$;
$i \in nat$; $j \in nat$; $env \in list(A)$|]
==> $is_DPow'(\#\#A,\ x,\ y) \longleftrightarrow sats(A,\ DPow'_fm(i,j),\ env)$
by (*simp*)

theorem *DPow'_reflection*:
$REFLECTS[\lambda x.\ is_DPow'(L,f(x),g(x)),$
$\lambda i\ x.\ is_DPow'(\#\#Lset(i),f(x),g(x))]$
apply (*simp only*: *is_DPow'_def*)
apply (*intro FOL_reflections function_reflections mem_formula_reflection mem_list_reflection Collect_reflection DPow_sats_reflection*)
done

10.5 A Locale for Relativizing the Operator *Lset*

definition
transrec_body :: $[i{=>}o,i,i,i,i]$ => o **where**
$transrec_body(M,g,x)$ ==
$\lambda y\ z.\ \exists gy[M].\ y \in x\ \&\ fun_apply(M,g,y,gy)\ \&\ is_DPow'(M,gy,z)$

lemma (**in** *M_DPow*) *transrec_body_abs*:
[|$M(x)$; $M(g)$; $M(z)$|]
==> $transrec_body(M,g,x,y,z) \longleftrightarrow y \in x\ \&\ z = DPow'(g`y)$
by (*simp add*: *transrec_body_def DPow'_abs transM* [*of _ x*])

locale *M_Lset* = *M_DPow* +
assumes *strong_rep*:
[|$M(x)$; $M(g)$|] ==> $strong_replacement(M,\ \lambda y\ z.\ transrec_body(M,g,x,y,z))$
and *transrec_rep*:
$M(i)$ ==> $transrec_replacement(M,\ \lambda x\ f\ u.$
$\exists r[M].\ is_Replace(M,\ x,\ transrec_body(M,f,x),\ r)\ \&$
$big_union(M,\ r,\ u),\ i)$

lemma (**in** *M_Lset*) *strong_rep'*:
[|$M(x)$; $M(g)$|]
==> $strong_replacement(M,\ \lambda y\ z.\ y \in x\ \&\ z = DPow'(g`y))$
by (*insert strong_rep* [*of x g*], *simp add*: *transrec_body_abs*)

lemma (**in** *M_Lset*) *DPow_apply_closed*:
[|$M(f)$; $M(x)$; $y{\in}x$|] ==> $M(DPow'(f`y))$
by (*blast intro*: *DPow'_closed dest*: *transM*)

lemma (**in** *M_Lset*) *RepFun_DPow_apply_closed*:
[|$M(f)$; $M(x)$|] ==> $M(\{DPow'(f`y).\ y{\in}x\})$
by (*blast intro*: *DPow_apply_closed RepFun_closed2 strong_rep'*)

lemma (**in** *M_Lset*) *RepFun_DPow_abs*:
[|$M(x)$; $M(f)$; $M(r)$ |]
==> *is_Replace*(M, x, $\lambda y\ z.$ *transrec_body*(M,f,x,y,z), r) $\longleftrightarrow$
$r = \{DPow'(f`y).\ y{\in}x\}$
apply (*simp add*: *transrec_body_abs RepFun_def*)
apply (*rule iff_trans*)
apply (*rule Replace_abs*)
apply (*simp_all add*: *DPow_apply_closed strong_rep'*)
done

lemma (**in** *M_Lset*) *transrec_rep'*:
$M(i)$ ==> *transrec_replacement*(M, $\lambda x\ f\ u.\ u = (\bigcup y{\in}x.\ DPow'(f\ `\ y))$, i)
apply (*insert transrec_rep* [*of i*])
apply (*simp add*: *RepFun_DPow_apply_closed RepFun_DPow_abs*
transrec_replacement_def)
done

Relativization of the Operator *Lset*

definition
is_Lset :: [i=>o, i, i] => o **where**
— We can use the term language below because *is_Lset* will not have to be internalized: it isn't used in any instance of separation.
is_Lset(M,a,z) == *is_transrec*(M, %$x\ f\ u.\ u = (\bigcup y{\in}x.\ DPow'(f`y))$, a, z)

lemma (**in** *M_Lset*) *Lset_abs*:
[|$Ord(i)$; $M(i)$; $M(z)$|]
==> *is_Lset*(M,i,z) $\longleftrightarrow$ z = *Lset*(i)
apply (*simp add*: *is_Lset_def Lset_eq_transrec_DPow'*)
apply (*rule transrec_abs*)
apply (*simp_all add*: *transrec_rep' relation2_def RepFun_DPow_apply_closed*)
done

lemma (**in** *M_Lset*) *Lset_closed*:
[|$Ord(i)$; $M(i)$|] ==> M(*Lset*(i))
apply (*simp add*: *Lset_eq_transrec_DPow'*)
apply (*rule transrec_closed* [*OF transrec_rep'*])
apply (*simp_all add*: *relation2_def RepFun_DPow_apply_closed*)
done

10.6 Instantiating the Locale *M_Lset*

10.6.1 The First Instance of Replacement

lemma *strong_rep_Reflects*:
REFLECTS [$\lambda u.\ \exists v[L].\ v \in B$ & ($\exists gy[L]$.
$v \in x$ & *fun_apply*(L,g,v,gy) & *is_DPow'*(L,gy,u)),

$\lambda i\ u.\ \exists v \in Lset(i).\ v \in B\ \&\ (\exists gy \in Lset(i).$
$v \in x\ \&\ fun_apply(\#\#Lset(i),g,v,gy)\ \&\ is_DPow'(\#\#Lset(i),gy,u))]$
by (*intro FOL_reflections function_reflections DPow'_reflection*)

lemma *strong_rep*:
$[|L(x);\ L(g)|] ==> strong_replacement(L,\ \lambda y\ z.\ transrec_body(L,g,x,y,z))$
apply (*unfold transrec_body_def*)
apply (*rule strong_replacementI*)
apply (*rule_tac u={x,g,B}*
in *gen_separation_multi* [*OF strong_rep_Reflects*], *auto*)
apply (*rule_tac env=[x,g,B]* **in** *DPow_LsetI*)
apply (*rule sep_rules DPow'_iff_sats* | *simp*)+
done

10.6.2 The Second Instance of Replacement

lemma *transrec_rep_Reflects*:
$REFLECTS\ [\lambda x.\ \exists v[L].\ v \in B\ \&$
$(\exists y[L].\ pair(L,v,y,x)\ \&$
$is_wfrec\ (L,\ \lambda x\ f\ u.\ \exists r[L].$
$is_Replace\ (L,\ x,\ \lambda y\ z.$
$\exists gy[L].\ y \in x\ \&\ fun_apply(L,f,y,gy)\ \&$
$is_DPow'(L,gy,z),\ r)\ \&\ big_union(L,r,u),\ mr,\ v,\ y)),$
$\lambda i\ x.\ \exists v \in Lset(i).\ v \in B\ \&$
$(\exists y \in Lset(i).\ pair(\#\#Lset(i),v,y,x)\ \&$
$is_wfrec\ (\#\#Lset(i),\ \lambda x\ f\ u.\ \exists r \in Lset(i).$
$is_Replace\ (\#\#Lset(i),\ x,\ \lambda y\ z.$
$\exists gy \in Lset(i).\ y \in x\ \&\ fun_apply(\#\#Lset(i),f,y,gy)\ \&$
$is_DPow'(\#\#Lset(i),gy,z),\ r)\ \&$
$big_union(\#\#Lset(i),r,u),\ mr,\ v,\ y))]$
apply (*simp only*: *rex_setclass_is_bex* [*symmetric*])
— Convert $\exists y{\in}Lset(i)$ to $\exists y[\#\#Lset(i)]$ within the body of the *is_wfrec* application.
apply (*intro FOL_reflections function_reflections*
is_wfrec_reflection Replace_reflection DPow'_reflection)
done

lemma *transrec_rep*:
$[|L(j)|]$
$==> transrec_replacement(L,\ \lambda x\ f\ u.$
$\exists r[L].\ is_Replace(L,\ x,\ transrec_body(L,f,x),\ r)\ \&$
$big_union(L,\ r,\ u),\ j)$
apply (*rule L.transrec_replacementI*, *assumption*)
apply (*unfold transrec_body_def*)
apply (*rule strong_replacementI*)
apply (*rule_tac u={j,B,Memrel(eclose({j}))}*
in *gen_separation_multi* [*OF transrec_rep_Reflects*], *auto*)
apply (*rule_tac env=[j,B,Memrel(eclose({j}))]* **in** *DPow_LsetI*)

apply (*rule sep_rules is_wfrec_iff_sats Replace_iff_sats DPow′_iff_sats* |
simp)+
done

10.6.3 Actually Instantiating *M_Lset*

lemma *M_Lset_axioms_L*: *M_Lset_axioms*(*L*)
apply (*rule M_Lset_axioms.intro*)
apply (*assumption* | *rule strong_rep transrec_rep*)+
done

theorem *M_Lset_L*: *M_Lset*(*L*)
apply (*rule M_Lset.intro*)
apply (*rule M_DPow_L*)
apply (*rule M_Lset_axioms_L*)
done

Finally: the point of the whole theory!

lemmas *Lset_closed* = *M_Lset.Lset_closed* [*OF M_Lset_L*]
and *Lset_abs* = *M_Lset.Lset_abs* [*OF M_Lset_L*]

10.7 The Notion of Constructible Set

definition
constructible :: [*i*=>*o*,*i*] => *o* **where**
constructible(*M*,*x*) ==
∃ *i*[*M*]. ∃ *Li*[*M*]. *ordinal*(*M*,*i*) & *is_Lset*(*M*,*i*,*Li*) & *x* ∈ *Li*

theorem *V_equals_L_in_L*:
L(*x*) ⟷ *constructible*(*L*,*x*)
apply (*simp add*: *constructible_def Lset_abs Lset_closed*)
apply (*simp add*: *L_def*)
apply (*blast intro*: *Ord_in_L*)
done

end

11 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 synthetized formulas.*

named_theorems *iff_sats Theorems for synthetising formulas.*

named_theorems *arity Theorems for arity of formulas.*

named_theorems *arity_aux Auxiliary theorems for calculating arities.*

ML_file‹*Synthetic_Definition.ML*›

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

end

12 Aids to internalize formulas

theory *Internalizations*
imports
DPow_absolute
Synthetic_Definition
Nat_Miscellanea
begin

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

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

definition
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*

Setting up notation for internalized formulas

abbreviation
dec10 :: *i* (‹*10*›) **where** *10* ≡ *succ*(*9*)
abbreviation
dec11 :: *i* (‹*11*›) **where** *11* ≡ *succ*(*10*)
abbreviation
dec12 :: *i* (‹*12*›) **where** *12* ≡ *succ*(*11*)
abbreviation
dec13 :: *i* (‹*13*›) **where** *13* ≡ *succ*(*12*)
abbreviation
dec14 :: *i* (‹*14*›) **where** *14* ≡ *succ*(*13*)
abbreviation
dec15 :: *i* (‹*15*›) **where** *15* ≡ *succ*(*14*)
abbreviation
dec16 :: *i* (‹*16*›) **where** *16* ≡ *succ*(*15*)
abbreviation
dec17 :: *i* (‹*17*›) **where** *17* ≡ *succ*(*16*)
abbreviation
dec18 :: *i* (‹*18*›) **where** *18* ≡ *succ*(*17*)
abbreviation
dec19 :: *i* (‹*19*›) **where** *19* ≡ *succ*(*18*)
abbreviation
dec20 :: *i* (‹*20*›) **where** *20* ≡ *succ*(*19*)
abbreviation
dec21 :: *i* (‹*21*›) **where** *21* ≡ *succ*(*20*)

abbreviation
dec22 :: *i* (‹*22*›) **where** *22* ≡ *succ*(*21*)
abbreviation
dec23 :: *i* (‹*23*›) **where** *23* ≡ *succ*(*22*)
abbreviation
dec24 :: *i* (‹*24*›) **where** *24* ≡ *succ*(*23*)
abbreviation
dec25 :: *i* (‹*25*›) **where** *25* ≡ *succ*(*24*)
abbreviation
dec26 :: *i* (‹*26*›) **where** *26* ≡ *succ*(*25*)
abbreviation
dec27 :: *i* (‹*27*›) **where** *27* ≡ *succ*(*26*)
abbreviation
dec28 :: *i* (‹*28*›) **where** *28* ≡ *succ*(*27*)
abbreviation
dec29 :: *i* (‹*29*›) **where** *29* ≡ *succ*(*28*)

notation *Member* (‹·_ ∈/ _·›)
notation *Equal* (‹·_ =/ _·›)
notation *Nand* (‹·¬′(_ ∧/ _′)·›)
notation *And* (‹·_ ∧/ _·›)
notation *Or* (‹·_ ∨/ _·›)
notation *Iff* (‹·_ ↔/ _·›)
notation *Implies* (‹·_ →/ _·›)
notation *Neg* (‹·¬_·›)
notation *Forall* (‹′(·∀(/_)·′)›)
notation *Exists* (‹′(·∃(/_)·′)›)

notation *subset_fm* (‹·_ ⊆/ _·›)
notation *succ_fm* (‹·*succ*′(_′) *is* _·›)
notation *empty_fm* (‹·_ *is empty*·›)
notation *fun_apply_fm* (‹·_‘_ *is* _·›)
notation *big_union_fm* (‹·⋃_ *is* _·›)
notation *upair_fm* (‹·{_,_} *is* _ ·›)
notation *ordinal_fm* (‹·_ *is ordinal*·›)

notation *pair_fm* (‹·⟨_,_⟩ *is* _ ·›)
notation *composition_fm* (‹·_ ∘ _ *is* _ ·›)
notation *domain_fm* (‹·*dom*′(_′) *is* _ ·›)
notation *range_fm* (‹·*ran*′(_′) *is* _ ·›)
notation *union_fm* (‹·_ ∪ _ *is* _ ·›)
notation *image_fm* (‹·_ ‘‘ _ *is* _ ·›)
notation *pre_image_fm* (‹·_ -‘‘ _ *is* _ ·›)
notation *field_fm* (‹·*fld*′(_′) *is* _ ·›)
notation *cons_fm* (‹·*cons*′(_,_′) *is* _ ·›)
notation *number1_fm* (‹·_ *is the number one*·›)
notation *function_fm* (‹·_ *is funct*·›)
notation *relation_fm* (‹·_ *is relat*·›)

notation *restriction_fm* (‹·_ ↾ _ *is* _ ·›)
notation *transset_fm* (‹·_ *is transitive*·›)
notation *limit_ordinal_fm* (‹·_ *is limit*·›)
notation *finite_ordinal_fm* (‹·_ *is finite ord*·›)
notation *omega_fm* (‹·_ *is* ω·›)
notation *cartprod_fm* (‹·_ × _ *is* _·›)
notation *Memrel_fm* (‹·*Memrel′*(_′) *is* _·›)
notation *quasinat_fm* (‹·_ *is qnat*·›)

notation *Inl_fm* (‹·*Inl′*(_′) *is* _ ·›)
notation *Inr_fm* (‹·*Inr′*(_′) *is* _ ·›)
notation *pred_set_fm* (‹·_-*predecessors of* _ *are* _·›)

abbreviation
fm_typedfun :: [*i*,*i*,*i*] ⇒ *i* (‹·_ : _ → _·›) **where**
fm_typedfun(*f*,*A*,*B*) ≡ *typed_function_fm*(*A*,*B*,*f*)

abbreviation
fm_surjection :: [*i*,*i*,*i*] ⇒ *i* (‹·_ *surjects* _ *to* _·›) **where**
fm_surjection(*f*,*A*,*B*) ≡ *surjection_fm*(*A*,*B*,*f*)

abbreviation
fm_injection :: [*i*,*i*,*i*] ⇒ *i* (‹·_ *injects* _ *to* _·›) **where**
fm_injection(*f*,*A*,*B*) ≡ *injection_fm*(*A*,*B*,*f*)

abbreviation
fm_bijection :: [*i*,*i*,*i*] ⇒ *i* (‹·_ *bijects* _ *to* _·›) **where**
fm_bijection(*f*,*A*,*B*) ≡ *bijection_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 *conj_setclass_model_iff_sats* [*iff_sats*]:
[| *0* ∈ *A*; *nth*(*i*,*env*) = *x*; *env* ∈ *list*(*A*);
P ⟷ *sats*(*A*,*p*,*env*); *env* ∈ *list*(*A*) |]
==> (*P* ∧ (##*A*)(*x*)) ⟷ *sats*(*A*, *p*, *env*)
[| *0* ∈ *A*; *nth*(*i*,*env*) = *x*; *env* ∈ *list*(*A*);
P ⟷ *sats*(*A*,*p*,*env*); *env* ∈ *list*(*A*) |]
==> ((##*A*)(*x*) ∧ *P*) ⟷ *sats*(*A*, *p*, *env*)
using *nth_closed*[*of env A i*]
by *auto*

lemma *conj_mem_model_iff_sats* [*iff_sats*]:
[| *0* ∈ *A*; *nth*(*i*,*env*) = *x*; *env* ∈ *list*(*A*);

$P \longleftrightarrow sats(A,p,env);\ env \in list(A)\ |]$
$==> (P \wedge x \in A) \longleftrightarrow sats(A,\ p,\ env)$
$[|\ 0 \in A;\ nth(i,env) = x;\ env \in list(A);$
$P \longleftrightarrow sats(A,p,env);\ env \in list(A)\ |]$
$==> (x \in A \wedge P) \longleftrightarrow sats(A,\ p,\ env)$
using *nth_closed*[*of env A i*]
by *auto*

lemma *mem_model_iff_sats* [*iff_sats*]:
$[|\ 0 \in A;\ nth(i,env) = x;\ env \in list(A)|]$
$==> (x \in A) \longleftrightarrow sats(A,\ Exists(Equal(0,0)),\ env)$
using *nth_closed*[*of env A i*]
by *auto*

lemma *subset_iff_sats*[*iff_sats*]:
$nth(i,\ env) = x \Longrightarrow nth(j,\ env) = y \Longrightarrow i \in nat \Longrightarrow j \in nat \Longrightarrow$
$env \in list(A) \Longrightarrow subset(\#\#A,\ x,\ y) \longleftrightarrow sats(A,\ subset_fm(i,\ j),\ env)$
using *sats_subset_fm'* **by** *simp*

lemma *not_mem_model_iff_sats* [*iff_sats*]:
$[|\ 0 \in A;\ nth(i,env) = x;\ env \in list(A)|]$
$==> (\forall\ x\ .\ x \notin A) \longleftrightarrow sats(A,\ Neg(Exists(Equal(0,0))),\ env)$
by *auto*

lemma *top_iff_sats* [*iff_sats*]:
$env \in list(A) \Longrightarrow 0 \in A \Longrightarrow sats(A,\ Exists(Equal(0,0)),\ env)$
by *auto*

lemma *prefix1_iff_sats*[*iff_sats*]:
assumes
$x \in nat\ env \in list(A)\ 0 \in A\ a \in A$
shows
$a = nth(x,env) \longleftrightarrow sats(A,\ Equal(0,x+_\omega 1),\ Cons(a,env))$
$nth(x,env) = a \longleftrightarrow sats(A,\ Equal(x+_\omega 1,0),\ Cons(a,env))$
$a \in nth(x,env) \longleftrightarrow sats(A,\ Member(0,x+_\omega 1),\ Cons(a,env))$
$nth(x,env) \in a \longleftrightarrow sats(A,\ Member(x+_\omega 1,0),\ Cons(a,env))$
using *assms nth_closed*
by *simp_all*

lemma *prefix2_iff_sats*[*iff_sats*]:
assumes
$x \in nat\ env \in list(A)\ 0 \in A\ a \in A\ b \in A$
shows
$b = nth(x,env) \longleftrightarrow sats(A,\ Equal(1,x+_\omega 2),\ Cons(a,Cons(b,env)))$
$nth(x,env) = b \longleftrightarrow sats(A,\ Equal(x+_\omega 2,1),\ Cons(a,Cons(b,env)))$
$b \in nth(x,env) \longleftrightarrow sats(A,\ Member(1,x+_\omega 2),\ Cons(a,Cons(b,env)))$
$nth(x,env) \in b \longleftrightarrow sats(A,\ Member(x+_\omega 2,1),\ Cons(a,Cons(b,env)))$

using *assms nth_closed*
by *simp_all*

lemma *prefix3_iff_sats*[*iff_sats*]:
assumes
$x \in nat\ env \in list(A)\ 0 \in A\ a \in A\ b \in A\ c \in A$
shows
$c = nth(x,env) \longleftrightarrow sats(A,\ Equal(2,x+_\omega 3),\ Cons(a,Cons(b,Cons(c,env))))$
$nth(x,env) = c \longleftrightarrow sats(A,\ Equal(x+_\omega 3,2),\ Cons(a,Cons(b,Cons(c,env))))$
$c \in nth(x,env) \longleftrightarrow sats(A,\ Member(2,x+_\omega 3),\ Cons(a,Cons(b,Cons(c,env))))$
$nth(x,env) \in c \longleftrightarrow sats(A,\ Member(x+_\omega 3,2),\ Cons(a,Cons(b,Cons(c,env))))$
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*: $[\![nth(n,l) = x;\ n \in nat]\!] \Longrightarrow nth(succ(n),\ Cons(a,l)) = x$
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 (term*sep_rules*) used in formula synthesis

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

lemmas *formulas_def* [*fm_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 quasilist_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*

lemmas *more_iff_sats* [*iff_sats*] = *rtran_closure_iff_sats tran_closure_iff_sats is_eclose_iff_sats Inl_iff_sats Inr_iff_sats fun_apply_iff_sats cartprod_iff_sats*

Collect_iff_sats

end

13 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 $\bigwedge y.\ Ord(y) \Longrightarrow R(y) \longleftrightarrow Q(y)$
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 $least_fm_type[TC]$ $: i \in nat \Longrightarrow q \in formula \Longrightarrow least_fm(q,i) \in formula$
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*:
$\bigwedge a.\ a \in A \Longrightarrow P(a) \longleftrightarrow sats(A,\ p,\ Cons(a,\ env))$

shows
$[\![y \in nat;\ env \in list(A)\ ;\ 0{\in}A]\!]$
$\Longrightarrow sats(A,\ least_fm(p,y),\ env) \longleftrightarrow$
$least(\#\#A,\ P,\ nth(y,env))$
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*:
$\bigwedge a.\ a \in A \Longrightarrow is_Q(a) \longleftrightarrow sats(A,\ q,\ Cons(a,env))$
shows
$[\![nth(j,env) = y;\ j \in nat;\ env \in list(A);\ 0{\in}A]\!]$
$\Longrightarrow least(\#\#A,\ is_Q,\ y) \longleftrightarrow sats(A,\ least_fm(q,j),\ env)$
using *sats_least_fm* [*OF is_Q_iff_sats*, *of j* , *symmetric*]
by *simp*

lemma *least_conj*: $a{\in}M \Longrightarrow least(\#\#M,\ \lambda x.\ x{\in}M \wedge Q(x),a) \longleftrightarrow least(\#\#M,Q,a)$
unfolding *least_def* **by** *simp*

context *M_trivial*
begin

13.1 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*) $a{\in}b$ | $a{=}b$ | (*ge*) $b{\in}a$
using *Ord_linear*[*OF* ‹$Ord(a)$› ‹$Ord(b)$›] **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 $\bigwedge x.\ Q(x) \Longrightarrow Ord(x) \Longrightarrow \exists\, y[M].\ Q(y) \wedge Ord(y)\ M(a)$

shows $least(M,Q,a) \longleftrightarrow a = (\mu\ x.\ Q(x))$
unfolding *least_def*
proof (*cases* $\forall\, b[M].\ Ord(b) \longrightarrow \neg\ Q(b)$; *intro iffI*; *simp add*:*assms*)
case *True*
with *assms*
have $\neg\ (\exists\, i.\ Ord(i) \wedge Q(i))$ **by** *blast*
then
show $0 = (\mu\ x.\ Q(x))$ **using** *Least_0* **by** *simp*
then
show $ordinal(M,\ \mu\ x.\ Q(x)) \wedge (empty(M,\ Least(Q)) \vee Q(Least(Q)))$
by *simp*
next
assume $\exists\, b[M].\ Ord(b) \wedge Q(b)$
then
obtain i **where** $M(i)\ Ord(i)\ Q(i)$ **by** *blast*
assume $a = (\mu\ x.\ Q(x))$
moreover
note ‹$M(a)$›
moreover from ‹$Q(i)$› ‹$Ord(i)$›
have $Q(\mu\ x.\ Q(x))$ (**is** *?G*)
by (*blast intro*:*LeastI*)
moreover
have $(\forall\, b[M].\ Ord(b) \wedge b \in (\mu\ x.\ Q(x)) \longrightarrow \neg\ Q(b))$ (**is** *?H*)
using *less_LeastE*[*of Q _ False*]
by (*auto*, *drule_tac ltI*, *simp*, *blast*)
ultimately
show $ordinal(M,\ \mu\ x.\ Q(x)) \wedge (empty(M,\ \mu\ x.\ Q(x)) \wedge (\forall\, b[M].\ Ord(b) \longrightarrow \neg\ Q(b)) \vee ?G \wedge ?H)$
by *simp*
next
assume *1*:$\exists\, b[M].\ Ord(b) \wedge Q(b)$
then
obtain i **where** $M(i)\ Ord(i)\ Q(i)$ **by** *blast*
assume $Ord(a) \wedge (a = 0 \wedge (\forall\, b[M].\ Ord(b) \longrightarrow \neg\ Q(b)) \vee Q(a) \wedge (\forall\, b[M].\ Ord(b) \wedge b \in a \longrightarrow \neg\ Q(b)))$
with *1*
have $Ord(a)\ Q(a)\ \forall\, b[M].\ Ord(b) \wedge b \in a \longrightarrow \neg\ Q(b)$
by *blast+*
moreover from *this* **and** *assms*
have $Ord(b) \Longrightarrow b \in a \Longrightarrow \neg\ Q(b)$ **for** b
by (*auto dest*:*transM*)
moreover from *this* **and** ‹$Ord(a)$›
have $b < a \Longrightarrow \neg\ Q(b)$ **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 $\bigwedge x.\ Q(x) \Longrightarrow Ord(x) \Longrightarrow \exists\, y[M].\ Q(y) \wedge Ord(y)$
shows $M(\mu\ x.\ Q(x))$
using *assms Least_le*[*of Q*] *Least_0*[*of Q*]
by (*cases* ($\exists\, i[M].\ Ord(i) \wedge Q(i)$)) (*force dest*:*transM ltD*)+

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

lemma *least_abs′*:
assumes $\bigwedge x.\ Q(x) \Longrightarrow M(x)\ M(a)$
shows $least(M,Q,a) \longleftrightarrow a = (\mu\ x.\ Q(x))$
using *assms least_abs*[*of Q*] **by** *auto*

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

end — *M_trivial*

end

14 Fully relational versions of higher order constructs

theory *Higher_Order_Constructs*
imports
Recursion_Thms
Least
begin

syntax
sats :: $[i,\ i,\ i] \Rightarrow o$ (‹(, _ ⊨ _)› [*36*,*36*,*36*] *25*)
syntax_consts
_sats ⇌ *sats*
translations
$(M,env \models \varphi) \rightleftharpoons CONST\ sats(M,\varphi,env)$

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*:
$is_If(M,b,t,f,r) \longleftrightarrow r = If(b,t,f)$
by (*simp add*: *is_If_def*)

definition
is_If_fm :: $[i,i,i,i] \Rightarrow i$ **where**

$is_If_fm(\varphi,t,f,r) \equiv Or(And(\varphi,Equal(t,r)),And(Neg(\varphi),Equal(f,r)))$

lemma *is_If_fm_type* [*TC*]: $\varphi \in formula \Longrightarrow t \in nat \Longrightarrow f \in nat \Longrightarrow r \in nat$
$\Longrightarrow$
$is_If_fm(\varphi,t,f,r) \in formula$
unfolding *is_If_fm_def* **by** *auto*

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

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

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

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

lemma (**in** *M_trans*) *The_abs*:
assumes $\bigwedge x.\ Q(x) \Longrightarrow M(x)$ $M(a)$
shows $is_The(M,Q,a) \longleftrightarrow a = (THE\ x.\ Q(x))$
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 ‹$\bigwedge x.\ Q(x) \Longrightarrow M(x)$›
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⇒o,i,[i,i,i]⇒o,i,i] ⇒o where
  is_recursor(M,a,is_b,k,r) ≡ is_transrec(M, λn f ntc. is_nat_case(M,a,
      λm bmfm.
      ∃fm[M]. fun_apply(M,f,m,fm) ∧ 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: ⋀m f bmf. M(m) ⟹ M(f) ⟹ M(bmf) ⟹ is_b(m,f,bmf) ⟷ bmf
= b(m,f) and
    b_closed: ⋀m f bmf. M(m) ⟹ M(f) ⟹ M(b(m,f)) and
    repl: transrec_replacement(M, λn f ntc. is_nat_case(M, a,
        λm bmfm. ∃fm[M]. fun_apply(M, f, m, fm) ∧ is_b( m, fm, bmfm), n, ntc),
k)
  shows
    is_recursor(M,a,is_b,k,r) ⟷ 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=λm bmfm.
      ∃fm[M]. fun_apply(M,_,m,fm) ∧ is_b(m,fm,bmfm)])
      apply (auto simp:relation1_def)
  apply (rule nat_case_abs[THEN iffD2, where is_b1=λm bmfm.
      ∃fm[M]. fun_apply(M,_,m,fm) ∧ is_b(m,fm,bmfm)])
     apply (auto simp:relation1_def)
  done

definition
  is_wfrec_on :: [i=>o,[i,i,i]=>o,i,i,i, i] => 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);
     ∀x[M]. ∀g[M]. function(g) ⟶ M(H(x,g));
     r-``{a}⊆A; a ∈ 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

15 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
Eclose_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 absoulte 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_eclose.is_eclose_n_abs
M_eclose.eclose_abs
M_trivial.Member_abs
M_trivial.Equal_abs
M_trivial.Nand_abs
M_trivial.Forall_abs

declare *relative_abs*[*absolut*]

ML_file‹*Relativization.ML*›

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
M_Basic_No_Repl
Relativization
ZF_Miscellanea

begin

hide_const (**open**) *Order.pred*
declare [[*syntax_ambiguity_warning* = *false*]]

15.1 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]. *empty*(A,c) $\wedge$ *is_cons*(A,x,c,z)

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

synthesize *singleton* **from_definition** *is_singleton*
notation *singleton_fm* (‹·{_} *is* _·›)

lemmas (**in** *M_trivial*) *singleton_closed* = *singleton_in_MI*

lemma (**in** *M_trivial*) *Upair_closed*[*simp*]: $M(a) \Longrightarrow M(b) \Longrightarrow M(Upair(a,b))$
using *Upair_eq_cons* **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**
$hcomp_fm(pf,pg,a,w) \equiv Exists(And(pg(succ(a),0),pf(0,succ(w))))$

lemma *sats_hcomp_fm*:
assumes
$f_iff_sats{:}\bigwedge a\ b\ z.\ a{\in}nat \Longrightarrow b{\in}nat \Longrightarrow z{\in}M \Longrightarrow$
$is_f(nth(a,Cons(z,env)),nth(b,Cons(z,env))) \longleftrightarrow sats(M,pf(a,b),Cons(z,env))$
and
$g_iff_sats{:}\bigwedge a\ b\ z.\ a{\in}nat \Longrightarrow b{\in}nat \Longrightarrow z{\in}M \Longrightarrow$
$is_g(nth(a,Cons(z,env)),nth(b,Cons(z,env))) \longleftrightarrow sats(M,pg(a,b),Cons(z,env))$
and
$a{\in}nat\ w{\in}nat\ env{\in}list(M)$
shows
$sats(M,hcomp_fm(pf,pg,a,w),env) \longleftrightarrow is_hcomp(\#\#M,is_f,is_g,nth(a,env),nth(w,env))$
proof -
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*⇒*o*,[*i*⇒*o*,*i*,*i*]⇒*o*,[*i*⇒*o*,*i*,*i*]⇒*o*,*i*,*i*] ⇒ *o* **where**
hcomp_r(*M*,*is_f*,*is_g*,*a*,*w*) ≡ ∃ *z*[*M*]. *is_g*(*M*,*a*,*z*) ∧ *is_f*(*M*,*z*,*w*)

definition
is_hcomp2_2 :: [*i*⇒*o*,[*i*⇒*o*,*i*,*i*,*i*]⇒*o*,[*i*⇒*o*,*i*,*i*,*i*]⇒*o*,[*i*⇒*o*,*i*,*i*,*i*]⇒*o*,*i*,*i*,*i*] ⇒ *o* **where**
is_hcomp2_2(*M*,*is_f*,*is_g1*,*is_g2*,*a*,*b*,*w*) ≡ ∃ *g1ab*[*M*]. ∃ *g2ab*[*M*].
is_g1(*M*,*a*,*b*,*g1ab*) ∧ *is_g2*(*M*,*a*,*b*,*g2ab*) ∧ *is_f*(*M*,*g1ab*,*g2ab*,*w*)

lemma (**in** *M_trivial*) *hcomp_abs*:
assumes
is_f_abs:⋀*a z*. *M*(*a*) ⟹ *M*(*z*) ⟹ *is_f*(*M*,*a*,*z*) ⟷ *z* = *f*(*a*) **and**
is_g_abs:⋀*a z*. *M*(*a*) ⟹ *M*(*z*) ⟹ *is_g*(*M*,*a*,*z*) ⟷ *z* = *g*(*a*) **and**
g_closed:⋀*a*. *M*(*a*) ⟹ *M*(*g*(*a*))
M(*a*) *M*(*w*)
shows
hcomp_r(*M*,*is_f*,*is_g*,*a*,*w*) ⟷ *w* = *f*(*g*(*a*))
unfolding *hcomp_r_def* **using** *assms* **by** *simp*

lemma *hcomp_uniqueness*:
assumes
uniq_is_f:
⋀*r d d′*. *M*(*r*) ⟹ *M*(*d*) ⟹ *M*(*d′*) ⟹ *is_f*(*M*, *r*, *d*) ⟹ *is_f*(*M*, *r*, *d′*) ⟹
d = *d′*
and
uniq_is_g:
⋀*r d d′*. *M*(*r*) ⟹ *M*(*d*) ⟹ *M*(*d′*) ⟹ *is_g*(*M*, *r*, *d*) ⟹ *is_g*(*M*, *r*, *d′*) ⟹
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** ‹*M*(*a*)›
have *z*=*z′* **by** *blast*
moreover note *uniq_is_f* **and** ‹*M*(*w*)› ‹*M*(*w′*)›

ultimately
show *?thesis* **by** *blast*
qed

lemma *hcomp_witness*:
assumes
wit_is_f: $\bigwedge r.\ M(r) \Longrightarrow \exists\, d[M].\ is_f(M,r,d)$ **and**
wit_is_g: $\bigwedge r.\ M(r) \Longrightarrow \exists\, d[M].\ is_g(M,r,d)$ **and**
$M(a)$
shows
$\exists\, w[M].\ hcomp_r(M,is_f,is_g,a,w)$
proof -
from ‹$M(a)$› **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

— In the next lemma, the absoluteness hypotheses on *g1* and *g2* can be restricted to the actual parameters.
lemma (**in** *M_trivial*) *hcomp2_2_abs*:
assumes
is_f_abs:$\bigwedge r1\ r2\ z.\ M(r1) \Longrightarrow M(r2) \Longrightarrow M(z) \Longrightarrow is_f(M,r1,r2,z) \longleftrightarrow z = f(r1,r2)$ **and**
is_g1_abs:$\bigwedge r1\ r2\ z.\ M(r1) \Longrightarrow M(r2) \Longrightarrow M(z) \Longrightarrow is_g1(M,r1,r2,z) \longleftrightarrow z = g1(r1,r2)$ **and**
is_g2_abs:$\bigwedge r1\ r2\ z.\ M(r1) \Longrightarrow M(r2) \Longrightarrow M(z) \Longrightarrow is_g2(M,r1,r2,z) \longleftrightarrow z = g2(r1,r2)$ **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*
by *simp*

lemma *hcomp2_2_uniqueness*:
assumes
uniq_is_f:
$\bigwedge r1\ r2\ d\ d'.\ M(r1) \Longrightarrow M(r2) \Longrightarrow M(d) \Longrightarrow M(d') \Longrightarrow$
$is_f(M,\ r1,\ r2\ ,\ d) \Longrightarrow is_f(M,\ r1,\ r2,\ d') \Longrightarrow\ d = d'$
and
uniq_is_g1:
$\bigwedge r1\ r2\ d\ d'.\ M(r1) \Longrightarrow M(r2) \Longrightarrow M(d) \Longrightarrow M(d') \Longrightarrow is_g1(M,\ r1,r2,\ d) \Longrightarrow is_g1(M,\ r1,r2,\ d') \Longrightarrow$
$d = d'$
and

uniq_is_g2:
$\bigwedge r1\ r2\ d\ d'.\ M(r1) \Longrightarrow M(r2) \Longrightarrow M(d) \Longrightarrow M(d') \Longrightarrow is_g2(M,\ r1,r2,\ d)$
$\Longrightarrow is_g2(M,\ r1,r2,\ d') \Longrightarrow$
$d = d'$
and
$M(a)\ M(b)\ M(w)\ M(w')$
$is_hcomp2_2(M,is_f,is_g1,is_g2,a,b,w)$
$is_hcomp2_2(M,is_f,is_g1,is_g2,a,b,w')$
shows
$w=w'$
proof -
from *assms*
obtain $z\ z'\ y\ y'$ **where** $is_g1(M,\ a,b,\ z)\ is_g1(M,\ a,b,\ z')$
$is_g2(M,\ a,b,\ y)\ is_g2(M,\ a,b,\ y')$
$is_f(M,z,y,w)\ is_f(M,z',y',w')$
$M(z)\ M(z')\ M(y)\ M(y')$
unfolding *is_hcomp2_2_def* **by** *force*
moreover from *this* **and** *uniq_is_g1 uniq_is_g2* **and** ‹$M(a)$› ‹$M(b)$›
have $z=z'\ y=y'$ **by** *blast+*
moreover note *uniq_is_f* **and** ‹$M(w)$› ‹$M(w')$›
ultimately
show *?thesis* **by** *blast*
qed

lemma *hcomp2_2_witness*:
assumes
wit_is_f: $\bigwedge r1\ r2.\ M(r1) \Longrightarrow M(r2) \Longrightarrow \exists\, d[M].\ is_f(M,r1,r2,d)$ **and**
wit_is_g1: $\bigwedge r1\ r2.\ M(r1) \Longrightarrow M(r2) \Longrightarrow \exists\, d[M].\ is_g1(M,r1,r2,d)$ **and**
wit_is_g2: $\bigwedge r1\ r2.\ M(r1) \Longrightarrow M(r2) \Longrightarrow \exists\, d[M].\ is_g2(M,r1,r2,d)$ **and**
$M(a)\ M(b)$
shows
$\exists\, w[M].\ is_hcomp2_2(M,is_f,is_g1,is_g2,a,b,w)$
proof -
from ‹$M(a)$› ‹$M(b)$› **and** *wit_is_g1*
obtain *g1a* **where** $is_g1(M,a,b,g1a)\ M(g1a)$ **by** *blast*
moreover from ‹$M(a)$› ‹$M(b)$› **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*)

15.2 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 :: $[i\Rightarrow o,i] \Rightarrow i$ (‹*Pow*–'(_')›) **where**
$Pow_rel(M,r) \equiv THE\ d.\ is_Pow(M,r,d)$

abbreviation
Pow_r_set :: $[i,i] \Rightarrow i$ (‹*Pow*–'(_')›) **where**
$Pow_r_set(M) \equiv Pow_rel(\#\#M)$

context *M_basic_no_repl*
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* : $[\![\ M(r);is_Pow(M,r,d)\ ]\!] \Longrightarrow M(d)$
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 _ 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 *Pow_iff*

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

lemma *Pow_rel_char*: $M(r) \Longrightarrow Pow_rel(M,r) = \{A{\in}Pow(r).\ M(A)\}$
proof -

assume $M(r)$
moreover from *this*
have $x \in Pow_rel(M,r) \Longrightarrow x \subseteq r$ $M(x) \Longrightarrow x \subseteq r \Longrightarrow x \in Pow_rel(M,r)$ **for** x
using *def_Pow_rel* **by** (*auto intro*!:*trans_Pow_rel_closed*)
ultimately
show *?thesis*
using *trans_Pow_rel_closed* **by** *blast*
qed

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

end — *M_basic_no_repl*

15.3 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
$Sigfun$ :: $[i,i \Rightarrow i] \Rightarrow i$ **where**
$Sigfun(x,B) \equiv \bigcup y \in B(x).\ \{\langle x,y \rangle\}$

lemma *SigFun_char* : $Sigfun(x,B) = \{x\} \times B(x)$
unfolding *Sigfun_def* **by** *auto*

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

definition — Note that B below is a higher order argument
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 $\bigcup\{z\ .\ y \in B(x),\ z = \{\langle x,\ y\rangle\}\} = (\bigcup y{\in}B(x).\ \{\langle x,\ y\rangle\})$ **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$(M,A,\lambda x\ z.\ z=Sigfun(x,B),RSf) \wedge big_union(M,RSf,S))$

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

$M(A)$
$\bigwedge x.\ x{\in}A \Longrightarrow\ M(B(x))$
$\forall x{\in}A.\ strong_replacement(M,\ \lambda y\ z.\ y \in B(x) \wedge z = \{\langle x,\ y\rangle\})$
$strong_replacement(M,\lambda x\ z.\ z{=}Sigfun(x,B))$
begin

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

15.4 Discipline for *Pi*

definition — completely relational
$is_Pi :: [i{\Rightarrow}o,i,i{\Rightarrow}i,i]{\Rightarrow}o$ **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
$Pi_rel :: [i{\Rightarrow}o,i,i{\Rightarrow}i] \Rightarrow i$ (‹*Pi*–′(_,_′)›) **where**
$Pi_rel(M,A,B) \equiv THE\ d.\ is_Pi(M,A,B,d)$

abbreviation

```
  Pi_r_set :: [i,i,i⇒i] ⇒ i (‹Pi–'(_,_')›) where
  Pi_r_set(M,A,B) ≡ Pi_rel(##M,A,B)

context M_basic
begin

lemmas Pow_rel_iff = mbnr.Pow_rel_iff
lemmas Pow_rel_char = mbnr.Pow_rel_char
lemmas mem_Pow_rel_abs = mbnr.mem_Pow_rel_abs
lemmas Pow_rel_closed = mbnr.Pow_rel_closed
lemmas def_Pow_rel = mbnr.def_Pow_rel
lemmas trans_Pow_rel_closed = mbnr.trans_Pow_rel_closed

end — M_basic

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: ∃ 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) ⟹ 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 B 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 ex1I*[*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 $f \in Pi_rel(M,A,\ B) \Longrightarrow A \subseteq domain(f) \wedge function(f)$ **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 $f \in Pow_rel(M,Sigma(A,B)) \Longrightarrow A \subseteq domain(f) \wedge function(f) \Longrightarrow f \in Pi_rel(M,A,\ B)$ **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. Σ_1^{ZF}, Π_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
$M_imp_N{:}M(x) \Longrightarrow N(x)$
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

15.5 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 $M(A)$ $\bigwedge x.\ x{\in}A \Longrightarrow\ M(B(x))$
    $\bigwedge x.\ x \in A \Longrightarrow b(x){\in}B(x)$ *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* _ ‹$M(A)$›]
   by (*rule_tac RepFun_closed*, *auto intro*!:*transM*[*OF* _ ‹$\bigwedge x.\ x{\in}A \Longrightarrow\ M(B(x))$›])
  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 $f \in Pi^M(A,C)$ $\bigwedge x.\ x \in A \Longrightarrow f`x \in B(x)$
    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 $f \in Pi^M(A,B)$ $\bigwedge x.\ x \in A \Longrightarrow B(x) \subseteq C(x)$
    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

16 Arities of internalized formulas

theory *Arities*
  imports
    Discipline_Base
begin

lemmas *FOL_arities* [*simp del*, *arity*] = *arity_And arity_Or arity_Implies ar-*

ity_Iff arity_Exists

declare *pred_Un_distrib*[*arity_aux*]

context
  notes *FOL_arities*[*simp*]
begin

lemma *arity_upair_fm* [*arity*] : ⟦ *t1*∈*nat* ; *t2*∈*nat* ; *up*∈*nat* ⟧ ⟹
  arity(*upair_fm*(*t1*,*t2*,*up*)) = ⋃ {*succ*(*t1*),*succ*(*t2*),*succ*(*up*)}
  unfolding *upair_fm_def*
  using *union_abs1 union_abs2 pred_Un*
  by *auto*

end

lemma *Un_trasposition_aux1*: $r \cup s \cup r = r \cup s$ **by** *auto*

lemma *Un_trasposition_aux2*:
  $r \cup (s \cup (r \cup u)) = r \cup (s \cup u)$
  $r \cup (s \cup (t \cup (r \cup u))) = r \cup (s \cup (t \cup u))$ **by** *auto*

We use the previous lemmas to guide automatic arity calculations.

context
  notes *Un_assoc*[*symmetric*,*simp*] *Un_trasposition_aux1*[*simp*]
begin

arity_theorem for *pair_fm*
arity_theorem for *composition_fm*
arity_theorem for *domain_fm*
arity_theorem for *range_fm*
arity_theorem for *union_fm*
arity_theorem for *image_fm*
arity_theorem for *pre_image_fm*
arity_theorem for *big_union_fm*
arity_theorem for *fun_apply_fm*
arity_theorem for *field_fm*
arity_theorem for *empty_fm*
arity_theorem for *cons_fm*
arity_theorem for *succ_fm*
arity_theorem for *number1_fm*
arity_theorem for *function_fm*
arity_theorem for *relation_fm*
arity_theorem for *restriction_fm*
arity_theorem for *typed_function_fm*
arity_theorem for *subset_fm*
arity_theorem for *transset_fm*
arity_theorem for *ordinal_fm*
arity_theorem for *limit_ordinal_fm*

arity_theorem for *finite_ordinal_fm*
arity_theorem for *omega_fm*
arity_theorem for *cartprod_fm*
arity_theorem for *singleton_fm*
arity_theorem for *Memrel_fm*
arity_theorem for *quasinat_fm*

end — context

context
  notes *FOL_arities*[*simp*]
begin

lemma *arity_is_recfun_fm* [*arity*]:
  $[\![p\in formula\ ;\ v\in nat\ ;\ n\in nat;\ Z\in nat; i\in nat]\!] \Longrightarrow\ arity(p) = i \Longrightarrow$
  $arity(is_recfun_fm(p,v,n,Z)) = succ(v) \cup succ(n) \cup succ(Z) \cup (pred\hat{}4(i))$
  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*]:
  $[\![p\in formula\ ;\ v\in nat\ ;\ n\in nat;\ Z\in nat\ ;\ i\in nat]\!] \Longrightarrow arity(p) = i \Longrightarrow$
    $arity(is_wfrec_fm(p,v,n,Z)) = succ(v) \cup succ(n) \cup succ(Z) \cup (pred\hat{}5(i))$
  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*]:
  $[\![p\in formula\ ;\ v\in nat\ ;\ n\in nat;\ Z\in nat;\ i\in nat]\!] \Longrightarrow arity(p) = i \Longrightarrow$
   $arity(is_nat_case_fm(v,p,n,Z)) = succ(v) \cup succ(n) \cup succ(Z) \cup pred(pred(i))$
  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∈formula v∈nat n∈nat g∈nat z∈nat i∈nat*
    $arity(isF) = i$
  shows $arity(iterates_MH_fm(isF,v,n,g,z)) =$
        $succ(v) \cup succ(n) \cup succ(g) \cup succ(z) \cup (pred\hat{}4(i))$
proof -
  let $?\varphi = (\cdot\exists\cdot\cdot g +_\omega 3`2\ is\ 0\cdot \wedge (\cdot\forall\cdot\cdot 0 = 2\cdot \rightarrow isF\cdot\cdot)\cdot\cdot)$
  let $?ar = (g +_\omega 3) \cup pred(pred(i))$
  from *assms*
  have $arity(?\varphi) = ?ar\ ?\varphi\in formula$
    using *arity_fun_apply_fm*
    union_abs1 union_abs2 pred_Un_distrib succ_Un_distrib Un_assoc[*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^11(i))
proof -
  let ?φ = iterates_MH_fm(p, 7+ω v, 2, 1, 0)
  let ?ψ = is_wfrec_fm(?φ, 0, n +ω 2, Z +ω 2)
  from ‹v∈_›
  have arity(?φ) = (8+ω v) ∪ (pred^4(i)) ?φ∈formula
    using assms arity_iterates_MH_fm union_abs2
    by simp_all
  then
  have arity(?ψ) = n +ω 3 ∪ (Z +ω 3) ∪ (3+ω v) ∪ (pred^9(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 auto
  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 $x\in nat$ $t\in nat$
shows $arity(mem_eclose_fm(x,t)) = succ(x) \cup succ(t)$
proof -
let $?\varphi=eclose_n_fm(x +_\omega 2, 1, 0)$
from ‹$x\in nat$›
have $arity(?\varphi) = x+_\omega 3$
using *arity_eclose_n_fm union_abs2*
by *simp*
with *assms*
show *?thesis*
unfolding *mem_eclose_fm_def*
using *arity_finite_ordinal_fm union_abs2 pred_Un_distrib*
by *simp*
qed

lemma *arity_is_eclose_fm* [*arity*]:
$[\![x\in nat ; t\in nat]\!] \Longrightarrow arity(is_eclose_fm(x,t)) = succ(x) \cup succ(t)$
unfolding *is_eclose_fm_def*
using *arity_mem_eclose_fm union_abs2 pred_Un_distrib*
by *auto*

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

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

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

lemma *arity_Replace_fm* [*arity*]:
$[\![p\in formula ; v\in nat ; n\in nat; i\in nat]\!] \Longrightarrow arity(p) = i \Longrightarrow$

arity(*Replace_fm*(*v*,*p*,*n*)) = *succ*(*n*) ∪ *succ*(*v*) ∪ *pred*(*pred*(*i*))
unfolding *Replace_fm_def*
using *union_abs2 pred_Un_distrib*
by *auto*

lemma *arity_lambda_fm* [*arity*]:
⟦*p*∈*formula*; *v*∈*nat* ; *n*∈*nat*; *i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
arity(*lambda_fm*(*p*,*v*,*n*)) = *succ*(*n*) ∪ (*succ*(*v*) ∪ (*pred*^3(*i*)))
unfolding *lambda_fm_def*
using *arity_pair_fm pred_Un_distrib union_abs1 union_abs2*
by *simp*

lemma *arity_transrec_fm* [*arity*]:
⟦*p*∈*formula* ; *v*∈*nat* ; *n*∈*nat*; *i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
arity(*is_transrec_fm*(*p*,*v*,*n*)) = *succ*(*v*) ∪ *succ*(*n*) ∪ (*pred*^8(*i*))
unfolding *is_transrec_fm_def*
using *arity Un_assoc*[*symmetric*] *pred_Un_distrib*
by *simp*

lemma *arity_wfrec_replacement_fm* :
⟦*p*∈*formula* ; *v*∈*nat* ; *n*∈*nat*; *Z*∈*nat* ; *i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
arity(*Exists*(*And*(*pair_fm*(*1*,*0*,*2*),*is_wfrec_fm*(*p*,*v*,*n*,*Z*))))
= *2* ∪ *v* ∪ *n* ∪ *Z* ∪ (*pred*^6(*i*))
unfolding *is_wfrec_fm_def*
using *arity_succ_fm arity_is_recfun_fm union_abs2 pred_Un_distrib arity_pair_fm*
by *auto*

end — *FOL_arities*

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

context
notes *Un_assoc*[*symmetric*,*simp*] *Un_trasposition_aux1*[*simp*]
begin
arity_theorem for *rtran_closure_mem_fm*
arity_theorem for *rtran_closure_fm*
arity_theorem for *tran_closure_fm*
end

context
notes *Un_assoc*[*simp*] *Un_trasposition_aux2*[*simp*]
begin
arity_theorem for *injection_fm*
arity_theorem for *surjection_fm*
arity_theorem for *bijection_fm*
arity_theorem for *order_isomorphism_fm*
end

arity_theorem for *Inl_fm*
arity_theorem for *Inr_fm*
arity_theorem for *pred_set_fm*

end

17 Basic relativization of function spaces

theory *Discipline_Function*
imports
Arities
begin

17.1 Discipline for *fst* and *snd*

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**
$is_fst(M,x,t) \equiv (\exists z[M].\ pair(M,t,z,x)) \vee$
$(\neg(\exists z[M].\ \exists w[M].\ pair(M,w,z,x)) \wedge empty(M,t))$
synthesize *fst* **from_definition** *is_fst*
notation *fst_fm* (‹·*fst*′(_′) *is* _·›)

arity_theorem for *fst_fm*

definition *fst_rel* :: $[i \Rightarrow o, i] \Rightarrow i$ **where**
$fst_rel(M,p) \equiv THE\ d.\ M(d) \wedge is_fst(M,p,d)$

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**
$is_snd(M,x,t) \equiv (\exists z[M].\ pair(M,z,t,x)) \vee$
$(\neg(\exists z[M].\ \exists w[M].\ pair(M,z,w,x)) \wedge empty(M,t))$
synthesize *snd* **from_definition** *is_snd*
notation *snd_fm* (‹·*snd*′(_′) *is* _·›)
arity_theorem for *snd_fm*

definition *snd_rel* :: $[i \Rightarrow o, i] \Rightarrow i$ **where**
$snd_rel(M,p) \equiv THE\ d.\ M(d) \wedge is_snd(M,p,d)$

reldb_add relational *snd is_snd*
reldb_add functional *snd snd_rel*

context *M_trans*

begin

lemma *fst_snd_closed*:
assumes $M(p)$
shows $M(fst(p)) \wedge M(snd(p))$
unfolding *fst_def snd_def* **using** *assms*
by (*cases* $\exists\, a.\ \exists\, b.\ p = \langle a,\ b\rangle$;*auto*)

lemma *fst_closed*[*intro*,*simp*]: $M(x) \Longrightarrow M(fst(x))$
using *fst_snd_closed* **by** *auto*

lemma *snd_closed*[*intro*,*simp*]: $M(x) \Longrightarrow M(snd(x))$
using *fst_snd_closed* **by** *auto*

lemma *fst_abs* [*absolut*]:
$[\![M(p);\ M(x)\ ]\!] \Longrightarrow is_fst(M,p,x) \longleftrightarrow x = fst(p)$
unfolding *is_fst_def fst_def*
by (*cases* $\exists\, a.\ \exists\, b.\ p = \langle a,\ b\rangle$;*auto*)

lemma *snd_abs* [*absolut*]:
$[\![M(p);\ M(y)\ ]\!] \Longrightarrow is_snd(M,p,y) \longleftrightarrow y = snd(p)$
unfolding *is_snd_def snd_def*
by (*cases* $\exists\, a.\ \exists\, b.\ p = \langle a,\ b\rangle$;*auto*)

lemma *empty_rel_abs* : $M(x) \Longrightarrow M(0) \Longrightarrow x = 0 \longleftrightarrow x = (THE\ d.\ M(d) \wedge empty(M,\ d))$
unfolding *the_def*
using *transM*
by *auto*

lemma *fst_rel_abs*:
assumes $M(p)$
shows $fst_rel(M,p) = fst(p)$
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 $snd_rel(M,p) = snd(p)$
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*

17.2 Discipline for *minimum*

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 _ ‹M(A)›]
  by(cases ∃x . first(x,A,r),auto simp:minimum_def)

lemma first_abs :
  assumes M(B)
  shows first(z,B,r) ⟷ 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) ⟷ M(b) ∧ first_rel(M,b,B,r) for b
    using first_abs
  proof (auto)
    fix b
    assume first_rel(M,b,B,r)
    with ‹M(B)›
    have b∈B using first_abs first_is_elem by simp
    with ‹M(B)›
    show M(b) using transM[OF ‹b∈B›] by simp
  qed
  with assms
  show ?thesis unfolding minimum_rel_def minimum_def
    by simp
qed

end — M_trans
```

17.3 Discipline for $\lambda A\ B.\ A \to B$

```
definition
  is_function_space :: [i⇒o,i,i,i] ⇒ o  where
  is_function_space(M,A,B,fs) ≡ M(fs) ∧ is_funspace(M,A,B,fs)

definition
  function_space_rel :: [i⇒o,i,i] ⇒ i  where
  function_space_rel(M,A,B) ≡ 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 :: $[i,i\Rightarrow o,i] \Rightarrow i$ (‹_ →– _› [61,1,61] 60) **where**
$A \rightarrow^M B \equiv$ *function_space_rel*(M,A,B)

abbreviation
function_space_r_set :: $[i,i,i] \Rightarrow i$ (‹_ →– _› [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$(M,A,B,d) \Longrightarrow M(d)$
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 is_function_space_iff:
  assumes M(x) M(y) M(d)
  shows is_function_space(M,x,y,d) ⟷ 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,λ_. y)
proof -
  from assms
  interpret M_Pi_assumptions M A λ_. y
    using Pi_replacement Pi_separation
    by unfold_locales (auto dest:transM simp add:Sigfun_def)
  from assms
  have x∈function_space_rel(M,A,y) ⟷ x∈Pi_rel(M,A,λ_. y) if M(x) for x
    using that
      is_function_space_iff[of A y, OF _ _ function_space_rel_closed, of A y]
```

def_Pi_rel Pi_rel_char mbnr.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 _* ‹*M*(*A*)› ‹*M*(*y*)›]
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* ∈ *A* → *y*. *M*(*f*)}
proof -
from *assms*
interpret *M_Pi_assumptions M A* λ_. *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* ∈ *function_space_rel*(*M*,*A*,*y*) ⟷ *f* ∈ *A* → *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*) ⟹ *N*(*x*)
begin

lemma *function_space_rel_transfer*: *M*(*A*) ⟹ *M*(*B*) ⟹
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 ≡ *fun_apply*
— It is not necessary to perform the Discipline for *is_apply* since it is absolute in this context

17.4 Discipline for *Collect* terms.

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

definition — completely relational

injP_rel:: $[i \Rightarrow o,i,i] \Rightarrow o$ **where**
injP_rel$(M,A,f) \equiv \forall w[M].\ \forall x[M].\ \forall fw[M].\ \forall fx[M].\ w \in A \wedge x \in A \wedge$
is_apply$(M,f,w,fw) \wedge$ *is_apply*$(M,f,x,fx) \wedge fw=fx \longrightarrow w=x$

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 *exI allI* to *rexI rallI* in some proofs. I wonder if this escalates well. Assuming that all terms appearing in the "def_" theorem are in M and using *transM*, it might do.
lemma *def_injP_rel*:
assumes
$M(A)$ $M(f)$
shows
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*

17.5 Discipline for *inj*

definition — completely relational
is_inj :: $[i \Rightarrow o,i,i,i] \Rightarrow o$ **where**
is_inj$(M,A,B,I) \equiv M(I) \wedge (\exists F[M].$ *is_function_space*$(M,A,B,F) \wedge$
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* (‹·_ → _ *is* _·›)

arity_theorem for *is_function_space_fm*

synthesize *is_inj* **from_definition assuming** *nonempty*
notation *is_inj_fm* (‹·*inj*′(_,_′) *is* _·›)

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

definition
inj_rel :: $[i \Rightarrow o,i,i] \Rightarrow i$ (‹*inj*–'(_,_')›) **where**
$inj_rel(M,A,B) \equiv THE\ d.\ is_inj(M,A,B,d)$

abbreviation
inj_r_set :: $[i,i,i] \Rightarrow i$ (‹*inj*–'(_,_')›) **where**
$inj_r_set(M) \equiv inj_rel(\#\#M)$

locale *M_inj* = *M_Pi* +
assumes
injP_separation: $M(r) \Longrightarrow separation(M,injP_rel(M,\ r))$
begin

lemma *is_inj_uniqueness*:
assumes
$M(r)\ M(B)$
$is_inj(M,r,B,d)\ is_inj(M,r,B,d')$
shows
$d=d'$
using *assms is_function_space_iff extensionality_trans*
unfolding *is_inj_def* **by** *simp*

lemma *is_inj_witness*: $M(r) \Longrightarrow M(B) \Longrightarrow \exists\, d[M].\ is_inj(M,r,B,d)$
using *injP_separation is_function_space_iff*
unfolding *is_inj_def* **by** *simp*

lemma *is_inj_closed* :
$is_inj(M,x,y,d) \Longrightarrow M(d)$
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). $\forall w[M]. \forall x[M]. w{\in}A \wedge x{\in}A \wedge f`w = f`x \longrightarrow w{=}x\}$
    (**is** _ = *Collect*(_,*?P*))
proof -
  from *assms*
  have *inj_rel*(M,A,B) $\subseteq$ *function_space_rel*(M,A,B)
    using *inj_rel_iff*[*of A B inj_rel*(M,A,B)] *is_function_space_iff*
    unfolding *is_inj_def* **by** *auto*
  moreover from *assms*
  have $f \in$ *inj_rel*(M,A,B) $\Longrightarrow$ *?P*(f) **for** f
    using *inj_rel_iff*[*of A B inj_rel*(M,A,B)] *is_function_space_iff*
     def_injP_rel transM[*OF _ function_space_rel_closed*, *OF _* ‹$M(A)$› ‹$M(B)$›]
    unfolding *is_inj_def* **by** *auto*
  moreover from *assms*
  have $f \in$ *function_space_rel*(M,A,B) $\Longrightarrow$ *?P*(f) $\Longrightarrow$ $f \in$ *inj_rel*(M,A,B) **for** f
    using *inj_rel_iff*[*of A B inj_rel*(M,A,B)] *is_function_space_iff*
     def_injP_rel transM[*OF _ function_space_rel_closed*, *OF _* ‹$M(A)$› ‹$M(B)$›]
    unfolding *is_inj_def* **by** *auto*
  ultimately
  show *?thesis* **by** *force*

qed

lemma *inj_rel_char*:
  assumes $M(A)$ $M(B)$
  shows $inj_rel(M,A,B) = \{f \in inj(A,B).\ M(f)\}$
proof -
  from *assms*
  interpret *M_Pi_assumptions M A* $\lambda_.\ B$
    using *Pi_replacement Pi_separation*
    by *unfold_locales* (*auto dest*:*transM simp add*:*Sigfun_def*)
  from *assms*
  show *?thesis*
    using *def_inj_rel*[*OF assms*] *def_function_space_rel*[*OF assms*]
      transM[*OF _* ‹$M(A)$›] *Pi_rel_char*
    unfolding *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:$M(x) \Longrightarrow N(x)$
begin

lemma *inj_rel_transfer*: $M(A) \Longrightarrow M(B) \Longrightarrow inj_rel(M,A,B) \subseteq inj_rel(N,A,B)$
  using *M.inj_rel_char N.inj_rel_char*
  by (*auto dest*!:*M_imp_N*)

end — *M_N_inj*

17.6 Discipline for *surj*

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*

17.7 Discipline for *surj*

definition — completely relational
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* (‹·*surj*′(_,_′) *is* _·›)

definition
surj_rel :: $[i \Rightarrow o,i,i] \Rightarrow i$ (‹*surj*–′(_,_′)›) **where**
surj_rel$(M,A,B) \equiv$ *THE d. is_surj*(M,A,B,d)

abbreviation
surj_r_set :: $[i,i,i] \Rightarrow i$ (‹*surj*–′(_,_′)›) **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 is_function_space_iff extensionality_trans*
unfolding *is_surj_def* **by** *simp*

lemma *is_surj_witness*: $M(r) \Longrightarrow M(B) \Longrightarrow \exists d[M].$ *is_surj*(M,r,B,d)
using *surjP_separation is_function_space_iff*
unfolding *is_surj_def* **by** *simp*

lemma *is_surj_closed* :
is_surj$(M,x,y,d) \Longrightarrow M(d)$
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*$(M,\ x,\ y,$ *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) ⟷ 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)] is_function_space_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)] is_function_space_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 $f \in$ *function_space_rel*(*M*,*A*,*B*) $\Longrightarrow$ *?P*(*f*) $\Longrightarrow$ $f \in$ *surj_rel*(*M*,*A*,*B*) **for** *f*
using *surj_rel_iff*[*of A B surj_rel*(*M*,*A*,*B*)] *is_function_space_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* $\in$ *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*) $\Longrightarrow$ *N*(*x*)
begin

lemma *surj_rel_transfer*: *M*(*A*) $\Longrightarrow$ *M*(*B*) $\Longrightarrow$ *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*

17.8 Discipline for *Inter*

definition
is_Int :: [*i*$\Rightarrow$*o*,*i*,*i*,*i*]$\Rightarrow$*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* (‹_ ∩ _ *is* _›)

context *M_basic*
begin

lemma *is_Int_closed* :
  is_Int(*M*,*A*,*B*,*I*) ⟹ *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*) ⟷ *I* = *A* ∩ *B*
  using *assms transM*[*OF* _ ‹*M*(*B*)›] *transM*[*OF* _ ‹*M*(*I*)›]
  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: *Int_closed* already in *ZF-Constructible.Relative*.

end — *M_basic*

17.9 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* (‹·*bij*′(_,_′) *is* _·›)

abbreviation

bij_r_class :: [*i*⇒*o*,*i*,*i*] ⇒ *i* (‹*bij*–′(_,_′)›) **where**
bij_r_class ≡ *bij_rel*

abbreviation
bij_r_set :: [*i*,*i*,*i*] ⇒ *i* (‹*bij*–′(_,_′)›) **where**
bij_r_set(*M*) ≡ *bij_rel*(##*M*)

locale *M_Perm* = *M_Pi* + *M_inj* + *M_surj*
begin

lemma *is_bij_closed* : *is_bij*(*M*,*f*,*y*,*d*) ⟹ *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*(*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*) ⟷ *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*) ∩ *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* ∈ *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*: $M(A) \Longrightarrow M(B) \Longrightarrow bij_rel(M,A,B) \subseteq bij_rel(N,A,B)$
  using *M.bij_rel_char N.bij_rel_char*
  by (*auto dest*!:*M_imp_N*)

end — *M_N_Perm*

context *M_Perm*
begin

lemma *mem_bij_rel*: $[\![f \in bij^M(A,B);\ M(A);\ M(B)]\!] \Longrightarrow f \in bij(A,B)$
  using *bij_rel_char* **by** *simp*

lemma *mem_inj_rel*: $[\![f \in inj^M(A,B);\ M(A);\ M(B)]\!] \Longrightarrow f \in inj(A,B)$
  using *inj_rel_char* **by** *simp*

lemma *mem_surj_rel*: $[\![f \in surj^M(A,B);\ M(A);\ M(B)]\!] \Longrightarrow f \in surj(A,B)$
  using *surj_rel_char* **by** *simp*

end — *M_Perm*

17.10 Discipline for (≈)

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* (‹·_ ≈ _·›)

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*⇒*o*,*i*] => *o* (‹_ ≈– _› [*51*,*1*,*51*] *50*) **where**
  $A \approx^M B \equiv eqpoll_rel(M,A,B)$

abbreviation
  eqpoll_r_set :: [*i*,*i*,*i*] ⇒ *o* (‹_ ≈– _› [*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

The next lemma is not part of the discipline

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*

17.11 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* (‹·_ $\lesssim$ _·›)
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*⇒*o*,*i*] => *o* (‹_ $\lesssim$– _› [*51*,*1*,*51*] *50*) **where**
$A \lesssim^M B \equiv$ *lepoll_rel*(*M*,*A*,*B*)

abbreviation
lepoll_r_set :: [*i*,*i*,*i*] ⇒ *o* (‹_ $\lesssim$– _› [*51*,*1*,*51*] *50*) **where**
lepoll_r_set(*A*,*M*) ≡ *lepoll_rel*(##*M*,*A*)

context *M_Perm*
begin

lemma *def_lepoll_rel*:
assumes
M(*A*) *M*(*B*)
shows
lepoll_rel(*M*,*A*,*B*) ⟷ (∃*f*[*M*]. *f* ∈ *inj_rel*(*M*,*A*,*B*))
using *assms inj_rel_iff*
unfolding *lepoll_rel_def* **by** *simp*

end — *M_Perm*

context *M_N_Perm*
begin

— This lemma is not part of the discipline.
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** $f \in inj^M(A,B)$ *N*(*f*)
using *M.def_lepoll_rel* **by** (*auto dest*!:*M_imp_N*)
moreover from *calculation*
have $f \in inj^N(A,B)$
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*

17.12 Discipline for (≺)

relativize functional *lesspoll lesspoll_rel* **external**
relationalize *lesspoll_rel is_lesspoll*

synthesize *is_lesspoll* **from_definition assuming** *nonempty*
notation *is_lesspoll_fm* (‹·_ ≺ _·›)
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 :: $[i,i \Rightarrow o,i]$ => *o* (‹_ ≺← _› [*51*,*1*,*51*] *50*) **where**
  $A \prec^M B \equiv$ *lesspoll_rel*(M,A,B)

abbreviation
  lesspoll_r_set :: $[i,i,i] \Rightarrow o$ (‹_ ≺← _› [*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.

definition
  Powapply :: $[i,i] \Rightarrow i$ **where**
  Powapply$(f,y) \equiv$ *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*

end

18 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 we introduce a new locale assuming two instances of separation and six instances of lambda replacements (ie, replacement of the form $\lambda xy.\ y = \langle x, f(x)\rangle$) and 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_trivial*
begin

lemma *lam_replacement_iff_lam_closed*:
assumes $\forall x[M].\ M(b(x))$
shows *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 *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 *lam_replacement*(M,f) $\forall x[M].\ f(x) = g(x)$ $\forall x[M].\ M(f(x))$
shows *lam_replacement*(M,g)
proof -
note *assms*

```
  moreover from this
  have ∀ A[M]. M(λx∈A. f(x))
    using lam_replacement_iff_lam_closed
    by simp
  moreover from calculation
  have (λx∈A . f(x)) = (λx∈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

end — M_trivial

context M_basic
begin

lemma separation_iff':
  assumes separation(M,λx . P(x)) separation(M,λx . Q(x))
  shows separation(M,λx . P(x) ⟷ Q(x))
  using assms separation_conj separation_imp iff_def
  by auto

lemma separation_in_constant :
  assumes M(a)
  shows separation(M,λx . x∈a)
proof -
  have {x∈A . x∈a} = A ∩ a for A by auto
  with ‹M(a)›
  show ?thesis using separation_iff Collect_abs
    by simp
qed

lemma separation_equal :
  shows separation(M,λx . x=a)
proof -
  have {x∈A . x=a} = (if a∈A then {a} else 0) for A
    by auto
  then
  have M({x∈A . x=a}) if M(A) for A
    using transM[OF _ ‹M(A)›] 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 $M(Memrel(A\cup\{a\})\ ``\ \{a\})$ **if** $M(A)$ **for** A
using *that* **by** *simp*
ultimately
show *?thesis*
using *separation_iff Collect_abs*
by *simp*
qed

—
lemma *lam_replacement_imp_strong_replacement_aux*:
assumes *lam_replacement*$(M,\ b)\ \forall x[M].\ M(b(x))$
shows *strong_replacement*$(M,\ \lambda x\ y.\ y = b(x))$
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 $M((\lambda x\in A.\ b(x))``A)\ \forall z[M].\ z \in (\lambda x\in A.\ b(x))``A \longleftrightarrow (\exists x\in A.\ z = b(x))$
by (*auto simp*:*lam_def*)
}
then
show *?thesis* **unfolding** *strong_replacement_def*
by *clarsimp* (*rule_tac* $x=(\lambda x\in A.\ b(x))``A$ **in** *rexI*, *auto*)
qed

— This lemma could be modularized into the instantiation (fixing X) and the projection of the result of f.
lemma *strong_lam_replacement_imp_strong_replacement* :
assumes *strong_replacement*$(M,\lambda\ x\ z\ .\ P(fst(x),snd(x)) \wedge z=\langle x,f(fst(x),snd(x))\rangle)$
$\bigwedge A\ .\ M(A) \Longrightarrow \forall x\in A.\ P(X,x) \longrightarrow M(f(X,x))\ M(X)$
shows *strong_replacement*$(M,\lambda\ x\ z\ .\ P(X,x) \wedge z=f(X,x))$
unfolding *strong_replacement_def*
proof(*clarsimp*)
fix A
assume $M(A)$
moreover from *this* ‹$M(X)$›
have $M(\{X\}\times A)$ (**is** $M(?A)$)
by *simp*

```
  moreover
  have fst(x) = X if x∈?A for x
    using that by auto
  moreover from calculation assms
  have M({z . x∈?A , P(fst(x),snd(x)) ∧ z=⟨x,f(fst(x),snd(x))⟩}) (is M(?F))
    using transM[of _ ?A]
    by(rule_tac strong_replacement_closed,simp_all)
  moreover
  have ?F=({⟨x,f(fst(x),snd(x))⟩ . x∈ {x∈?A .  P(fst(x),snd(x))}}) (is _=(?G))
    by auto
  moreover
  note ‹M(?A)›
  ultimately
  have M(?G``?A)
    by simp
  moreover
  have ?G``?A = {y . x∈?A , P(fst(x),snd(x)) ∧ y = f(fst(x),snd(x))} (is _=(?H))
    by auto
  ultimately
  show ∃ Y[M]. ∀ b[M]. b ∈ Y ⟷ (∃ x∈A. P(X,x) ∧ b = f(X,x))
    by(rule_tac rexI[of _ ?H],auto,force)
qed

lemma lam_replacement_imp_RepFun_Lam:
  assumes lam_replacement(M, f) M(A)
  shows M({y . x∈A , M(y) ∧ y=⟨x,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 calculation
  have Y = {y . x∈A , M(y) ∧ y = ⟨x,f(x)⟩} (is Y=?R)
  proof(intro equalityI subsetI)
    fix y
    assume y∈Y
    moreover from this 1
    obtain x where x∈A y=⟨x,f(x)⟩ M(y)
      using transM[OF _ ‹M(Y)›] by auto
    ultimately
    show y∈?R
      by auto
  next
    fix z
    assume z∈?R
    moreover from this
    obtain a where a∈A z=⟨a,f(a)⟩ M(a) M(f(a))
      using transM[OF _ ‹M(A)›]
      by auto
```

```
    ultimately
    show z∈Y using 1 by simp
  qed
  ultimately
  show ?thesis by auto
qed

lemma lam_closed_imp_closed:
  assumes ∀A[M]. M(λx∈A. f(x))
  shows ∀x[M]. M(f(x))
proof
  fix x
  assume M(x)
  moreover from this and assms
  have M(λx∈{x}. f(x)) by simp
  ultimately
  show M(f(x))
    using image_lam[of {x} {x} f]
      image_closed[of {x} (λx∈{x}. f(x))] by (auto dest:transM)
qed

lemma lam_replacement_if:
  assumes lam_replacement(M,f) lam_replacement(M,g) separation(M,b)
    ∀x[M]. M(f(x)) ∀x[M]. M(g(x))
  shows lam_replacement(M, λx. if b(x) then f(x) else g(x))
proof -
  let ?G=λx. if b(x) then f(x) else g(x)
  let ?b=λA . {x∈A. b(x)} and ?b'=λA . {x∈A. ¬b(x)}
  have eq:(λx∈A . ?G(x)) = (λx∈?b(A) . f(x)) ∪ (λx∈?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(λx∈?b(A). f(x)) M(λx∈?b'(A) . g(x)) if M(A) for A
    using lam_replacement_iff_lam_closed that
    by simp_all
  moreover from this
  have M((λx∈?b(A) . f(x)) ∪ (λx∈?b'(A).g(x))) if M(A) for A
    using that by simp
  ultimately
  have M(λx∈A. if b(x) then f(x) 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*: $M(b) \Longrightarrow$ *lam_replacement*$(M,\lambda_.\ b)$
  unfolding *lam_replacement_def strong_replacement_def*
  by *safe* (*rule_tac x*=_×{*b*} **in** *rexI*; *blast*)

18.1 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 $\bigwedge x\ b.\ x \in A \Longrightarrow P(x,b) \Longrightarrow b \in U\ \forall\, x{\in}A.\ \forall\, y\ y'.\ P(x,y) \wedge P(x,y') \longrightarrow y{=}y'$
    separation$(M,\ \lambda y.\ \exists\, x[M].\ x \in A \wedge P(x,\ y))$
    $M(U)\ M(A)$
  shows $\{y\ .\ x \in A,\ P(x,\ y)\} \in Pow^M(U)$
proof -
  from *assms*
  have $\{y\ .\ x \in A,\ P(x,\ y)\} \subseteq U$
    $z \in \{y\ .\ x \in A,\ P(x,\ y)\} \Longrightarrow M(z)$ **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 $\bigwedge b.\ P(b) \Longrightarrow b \in U$
    separation$(M,\ \lambda y.\ P(y))\ M(U)$
  shows $\{y\ .\ x \in \{0\},\ P(y)\} \in Pow^M(U)$
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 $\bigwedge b.\ P(b) \Longrightarrow b \in U$ *separation*$(M,\ \lambda y.\ P(y))\ M(U)$
  shows $(THE\ i.\ P(i)) \in Pow^M(\bigcup U)$
proof -
  note *assms*
  moreover from *this*
  have $(THE\ i.\ P(i)) \in Pow(\bigcup U)$
    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*$(M,\ \lambda y.\ Ord(y) \wedge P(y) \wedge (\forall j.\ j < y \longrightarrow \neg P(j)))$
  unfolding *separation_def*
proof
  fix z
  assume $M(z)$
  have $M(\{x \in z\ .\ x \in z \wedge Ord(x) \wedge P(x) \wedge (\forall j.\ j < x \longrightarrow \neg P(j))\})$
    (**is** $M(?y)$)
  proof (*cases* $\exists x{\in}z.\ Ord(x) \wedge P(x) \wedge (\forall j.\ j < x \longrightarrow \neg P(j))$)
    case *True*
    with ‹$M(z)$›
    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 $\{x \in z\ .\ x \in z \wedge Ord(x) \wedge P(x) \wedge (\forall j.\ j < x \longrightarrow \neg P(j))\} = 0$
      by *auto*
    then
    show *?thesis* **by** *auto*
  qed
  moreover from *this*
  have $\forall x[M].\ x \in ?y \longleftrightarrow x \in z \wedge Ord(x) \wedge P(x) \wedge (\forall j.\ j < x \longrightarrow \neg P(j))$ **by** *simp*
  ultimately
  show $\exists y[M].\ \forall x[M].\ x \in y \longleftrightarrow x \in z \wedge Ord(x) \wedge P(x) \wedge (\forall j.\ j < x \longrightarrow \neg P(j))$
    by *blast*

qed

lemma *Least_in_Pow_rel_Union*:
  assumes $\bigwedge b.\ P(b) \Longrightarrow b \in U$
    $M(U)$
  shows $(\mu\ i.\ P(i)) \in Pow^M(\bigcup U)$
  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*:$\forall A[M].\ separation(M,\lambda y.\ \exists x[M].\ x{\in}A \wedge y = \langle x, f(x)\rangle)$
    and *U_closed* [*intro*,*simp*]: $\bigwedge X.\ M(X) \Longrightarrow M(U(X))$
  shows $lam_replacement(M, f)$
proof -
  have $M(\lambda x{\in}A.\ f(x))$ **if** $M(A)$ **for** A
  proof -
    have $(\lambda x{\in}A.\ f(x)) = \{y{\in}\ Pow^M(Pow^M(A \cup U(A))).\ \exists x[M].\ x{\in}A \wedge y = \langle x, f(x)\rangle\}$
      using ‹$M(A)$› **unfolding** *lam_def*
    proof (*intro equalityI*, *auto*)
      fix x
      assume $x{\in}A$
      moreover
      note ‹$M(A)$›
      moreover from *calculation assms*
      have $f(x) \in U(A)$ **by** *simp*
      moreover from *calculation*
      have $\{x, f(x)\} \in Pow^M(A \cup U(A))$ $\{x,x\} \in Pow^M(A \cup U(A))$
        using $Pow_rel_char[of\ A \cup U(A)]$ **by** (*auto dest*:*transM*)
      ultimately
      show $\langle x, f(x)\rangle \in Pow^M(Pow^M(A \cup U(A)))$
        using $Pow_rel_char[of\ Pow^M(A \cup U(A))]$ **unfolding** *Pair_def*
        by (*auto dest*:*transM*)
    qed
    moreover from ‹$M(A)$›
    have $M(\{y{\in}\ Pow^M(Pow^M(A \cup U(A))).\ \exists x[M].\ x{\in}A \wedge y = \langle x, f(x)\rangle\})$
      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 fst_in_double_Union:
  assumes x∈X
  shows fst(x) ∈ {0} ∪ ⋃⋃X
proof -
  have fst(x) ∈ {0} ∪ ⋃x for x
    unfolding fst_def
    using the_0 fst_conv
    by(cases ∃!a.∃b. x = ⟨a,b⟩, auto, simp add:Pair_def)
  with assms
  show ?thesis by auto
qed

lemma snd_in_double_Union:
  assumes x∈X
  shows snd(x) ∈ {0} ∪ ⋃⋃X
proof -
  have snd(x) ∈ {0} ∪ ⋃x for x
    unfolding snd_def
    using the_0 snd_conv
    by(cases ∃!a.∃b. x = ⟨a,b⟩, auto, simp add:Pair_def)
  with assms
  show ?thesis by auto
qed

lemma bounded_lam_replacement_binary:
  fixes U
  assumes ∀X[M]. ∀x∈X. ∀y∈X. f(x,y) ∈ U(X)
   and separation_f:∀A[M]. separation(M,λy. ∃x[M]. x∈A ∧ y = ⟨x, f(fst(x),snd(x))⟩)
    and U_closed [intro,simp]: ⋀X. M(X) ⟹ M(U(X))
  shows lam_replacement(M, λr . f(fst(r),snd(r)))
proof -
  have M(λx∈A. f(fst(x),snd(x))) if M(A) for A
  proof -
    have (λx∈A. f(fst(x),snd(x))) =
    {y∈ Pow^M(Pow^M(A ∪ U({0} ∪ ⋃⋃A))). ∃x[M]. x∈A ∧ y = ⟨x, f(fst(x),snd(x))⟩}
      using ‹M(A)› unfolding lam_def
    proof (intro equalityI, auto)
      fix x
      assume x∈A
      moreover
      note ‹M(A)›
      moreover from calculation assms
      have f(fst(x),snd(x)) ∈ U({0} ∪ ⋃⋃A)
        using fst_in_double_Union snd_in_double_Union by simp
      moreover from calculation
```

have $\{x, f(fst(x),snd(x))\} \in Pow^M(A \cup U(\{0\} \cup \bigcup\bigcup A))$
$\{x,x\} \in Pow^M(A \cup U(\{0\} \cup \bigcup\bigcup A))$
using *Pow_rel_char*[*of* $A \cup U(\{0\} \cup \bigcup\bigcup A)$] **by** (*auto dest*:*transM*)
ultimately
show $\langle x, f(fst(x),snd(x))\rangle \in Pow^M(Pow^M(A \cup U(\{0\} \cup \bigcup\bigcup A)))$
using *Pow_rel_char*[*of* $Pow^M(A \cup U(\{0\} \cup \bigcup\bigcup A))$] **unfolding** *Pair_def*
by (*auto dest*:*transM*)
qed
moreover from ‹$M(A)$›
have $M(\{y\in Pow^M(Pow^M(A \cup U(\{0\} \cup \bigcup\bigcup A))). \exists x[M]. x\in A \wedge y = \langle x, f(fst(x),snd(x))\rangle\})$
using *separation_f*
by (*rule_tac separation_closed*) *simp_all*
ultimately
show *?thesis*
by *simp*
qed
moreover from *this*
have $\forall x[M]. M(f(fst(x),snd(x)))$
using *lam_closed_imp_closed*[*of* $\lambda x. f(fst(x),snd(x))$] **by** *simp*
ultimately
show *?thesis*
using *assms*
by (*rule_tac lam_replacement_iff_lam_closed*[*THEN iffD2*]) *simp_all*
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 $\forall A[M]. separation(M, \lambda y. \exists x[M]. x\in A \wedge y = \langle x, fst(x)\rangle)$
shows *lam_replacement*(M,*fst*)
using *fst_in_double_Union assms*
bounded_lam_replacement[*of fst* $\lambda X. \{0\} \cup \bigcup\bigcup X$] **by** *simp*

lemma *lam_replacement_snd'*:
assumes $\forall A[M]. separation(M, \lambda y. \exists x[M]. x\in A \wedge y = \langle x, snd(x)\rangle)$
shows *lam_replacement*(M,*snd*)
using *snd_in_double_Union assms*
bounded_lam_replacement[*of snd* $\lambda X. \{0\} \cup \bigcup\bigcup X$] **by** *simp*

— We can prove this separation in the locale introduced below.
lemma *lam_replacement_restrict*:
assumes $\forall A[M]. separation(M, \lambda y. \exists x[M]. x\in A \wedge y = \langle x, restrict(x,B)\rangle)$ $M(B)$
shows *lam_replacement*(M, λr . *restrict*(r,B))
proof -
from *assms*
have $restrict(r,B)\in Pow^M(\bigcup R)$ **if** $M(R)$ $r\in R$ **for** r R
using *Union_upper subset_Pow_Union subset_trans*[*OF restrict_subset*]
Pow_rel_char that transM[*of _ R*]

by *auto*
with *assms*
show *?thesis*
using *bounded_lam_replacement*[*of* λr . *restrict*(r,B) λX. $Pow^M(\bigcup X)$]
by *simp*
qed

lemma *lam_replacement_Union'* :
assumes $\forall A[M]$. *separation*(M, λy. $\exists x[M]$. $x \in A \wedge y = \langle x, \bigcup x\rangle$)
shows *lam_replacement*(M,*Union*)
proof -
have $\bigcup x \in Pow(\bigcup\bigcup A)$ **if** $x{\in}A$ **for** x A
using *that* **by** *auto*
then
have $\bigcup x \in Pow^M(\bigcup\bigcup A)$ **if** $M(A)$ $x{\in}A$ **for** x A
using *that transM*[*of* x A] *Pow_rel_char*
by *auto*
with *assms*
show *?thesis*
using *bounded_lam_replacement*[**where** $U{=}\lambda A$. $Pow^M(\bigcup\bigcup A)$]
by *auto*
qed

lemma *Image_in_Pow_rel_Union3*:
assumes $x{\in}X$ $y{\in}X$ $M(X)$
shows *Image*(x,y) $\in Pow^M(\bigcup\bigcup\bigcup X)$
proof -
from *assms*
have $\langle a, b\rangle \in x \Longrightarrow b \in \bigcup\bigcup x$ **for** a b
unfolding *Pair_def* **by** (*auto dest*:*transM*)
moreover from *this* **and** *assms*
have $\langle a, b\rangle \in x \Longrightarrow M(b)$ **for** a b
using *transM*[*of* _ X] *transM*[*of* _ x]
by *auto*
ultimately
show *?thesis*
using *assms transM*[*of* _ X] *transM*[*of* _ x] *mem_Pow_rel_abs*
by *auto fast*
qed

lemma *lam_replacement_Image'*:
assumes
$\forall A[M]$. *separation*(M,λy. $\exists x[M]$. $x{\in}A \wedge y = \langle x, Image(fst(x),snd(x))\rangle$)
shows
lam_replacement(M, λr . *Image*(*fst*(r),*snd*(r)))
using *assms Image_in_Pow_rel_Union3*
by (*rule_tac bounded_lam_replacement_binary*[*of* _ λX. $Pow^M(\bigcup\bigcup\bigcup X)$]) *auto*

lemma *minimum_in_Union*:

assumes $x \in X$ $y \in X$
shows $minimum(x,y) \in \{0\} \cup \bigcup X$
using *assms minimum_in'*
by *auto*

lemma *lam_replacement_minimum'*:
assumes
$\forall A[M].\ separation(M, \lambda y.\ \exists x[M].\ x \in A \wedge y = \langle x,\ minimum(fst(x), snd(x))\rangle)$
shows
$lam_replacement(M,\ \lambda r\ .\ minimum(fst(r), snd(r)))$
using *assms minimum_in_Union*
by (*rule_tac bounded_lam_replacement_binary*[*of* _ $\lambda X.\ \{0\} \cup \bigcup X$]) *auto*

lemma *lam_replacement_Pow_rel*:
assumes $\forall A[M].\ separation(M,\ \lambda y.\ \exists x[M].\ x \in A \wedge y = \langle x,\ Pow_rel(M,x)\rangle)$
shows $lam_replacement(M, Pow_rel(M))$
proof -
have $Pow_rel(M,x) \in Pow(Pow(\bigcup A))$ **if** $x \in A$ $M(A)$ **for** x A
using *that transM*[*of x A*] *def_Pow_rel*[*of* _ *x*] **by** (*auto dest*:*transM*)
then
have $Pow_rel(M,x) \in Pow(Pow^M(\bigcup A))$ **if** $M(A)$ $x \in A$ **for** x A
using *that transM*[*of x A*] *Pow_rel_char*
by *auto*
then
have $Pow_rel(M,x) \in Pow^M(Pow^M(\bigcup A))$ **if** $M(A)$ $x \in A$ **for** x A
using *that transM*[*of x A*] *Pow_rel_char*[*of* $Pow^M(\bigcup A)$]
by *auto*
with *assms*
show *?thesis*
using *bounded_lam_replacement*[**where** $U = \lambda A.\ Pow^M(Pow^M(\bigcup A))$]
by *auto*
qed

end — *M_basic*

definition
$middle_del :: i \Rightarrow i \Rightarrow i$ **where**
$middle_del(x,y) \equiv \langle fst(x), snd(y)\rangle$

relativize functional *middle_del middle_del_rel*
relationalize *middle_del_rel is_middle_del*
synthesize *is_middle_del* **from_definition**
arity_theorem for *is_middle_del_fm*

context *M_basic*
begin

lemma *middle_del_in_cartprod*:
assumes $x \in X$ $y \in X$ $M(X)$

shows *middle_del*$(x,y) \in (\{0\} \cup \bigcup\bigcup X) \times (\{0\} \cup \bigcup\bigcup X)$
using *assms Pow_rel_char transM*[*of _ X*] *fst_in_double_Union snd_in_double_Union*
unfolding *middle_del_def* **by** *auto*

lemma *lam_replacement_middle_del'*:
assumes
$\forall A[M].$ *separation*$(M,\lambda y.\ \exists x[M].\ x{\in}A \wedge y = \langle x,$ *middle_del*$(fst(x),snd(x))\rangle)$
shows
lam_replacement$(M,\ \lambda r\ .$ *middle_del*$(fst(r),snd(r)))$
using *assms middle_del_in_cartprod*
by (*rule_tac bounded_lam_replacement_binary*[*of _* $\lambda X.\ (\{0\} \cup \bigcup\bigcup X) \times (\{0\} \cup \bigcup\bigcup X)$]) *auto*

end — *M_basic*

definition
prodRepl :: $i{\Rightarrow}i{\Rightarrow}i$ **where**
prodRepl$(x,y) \equiv \langle snd(x),\langle fst(x),snd(y)\rangle\rangle$

relativize functional *prodRepl prodRepl_rel*
relationalize *prodRepl_rel is_prodRepl*
synthesize *is_prodRepl* **from_definition**
arity_theorem for *is_prodRepl_fm*

context *M_basic*
begin

lemma *prodRepl_in_cartprod*:
assumes $x{\in}X$ $y{\in}X$ $M(X)$
shows *prodRepl*$(x,y) \in (\{0\} \cup \bigcup\bigcup X) \times (\{0\} \cup \bigcup\bigcup X) \times (\{0\} \cup \bigcup\bigcup X)$
using *assms Pow_rel_char transM*[*of _ X*] *fst_in_double_Union snd_in_double_Union*
unfolding *prodRepl_def* **by** *auto*

lemma *lam_replacement_prodRepl'*:
assumes
$\forall A[M].$ *separation*$(M,\lambda y.\ \exists x[M].\ x{\in}A \wedge y = \langle x,$ *prodRepl*$(fst(x),snd(x))\rangle)$
shows
lam_replacement$(M,\ \lambda r\ .$ *prodRepl*$(fst(r),snd(r)))$
using *assms prodRepl_in_cartprod*
by (*rule_tac bounded_lam_replacement_binary*[*of _*
$\lambda X.\ (\{0\} \cup \bigcup\bigcup X) \times (\{0\} \cup \bigcup\bigcup X) \times (\{0\} \cup \bigcup\bigcup X)$]) *auto*

end — *M_basic*

context *M_Perm*
begin

lemma *inj_rel_in_Pow_rel_Pow_rel*:
assumes $x{\in}X$ $y{\in}X$ $M(X)$

shows $inj^M(x,y) \in Pow^M(Pow^M(\bigcup X \times \bigcup X))$
using *assms transM*[*of* _ *X*] *mem_Pow_rel_abs inj_def Pi_def*
by *clarsimp* (*use inj_rel_char* **in** *auto*)

lemma *lam_replacement_inj_rel'*:
assumes
$\forall A[M].\ separation(M,\lambda y.\ \exists x[M].\ x{\in}A \wedge y = \langle x,\ inj^M(fst(x),snd(x))\rangle)$
shows
$lam_replacement(M,\ \lambda r\ .\ inj^M(fst(r),snd(r)))$
using *assms inj_rel_in_Pow_rel_Pow_rel*
by (*rule_tac bounded_lam_replacement_binary*[*of* _ $\lambda X.\ Pow^M(Pow^M(\bigcup X \times \bigcup X))$]) *auto*

end — *M_Perm*

locale *M_replacement* = *M_basic* +
assumes
lam_replacement_fst: $lam_replacement(M,fst)$
and
lam_replacement_snd: $lam_replacement(M,snd)$
and
lam_replacement_Union: $lam_replacement(M,Union)$
and
lam_replacement_middle_del: $lam_replacement(M,\ \lambda r.\ middle_del(fst(r),snd(r)))$
and
lam_replacement_prodRepl: $lam_replacement(M,\ \lambda r.\ prodRepl(fst(r),snd(r)))$
and
lam_replacement_Image:$lam_replacement(M,\ \lambda p.\ fst(p)\ ``\ snd(p))$
and
middle_separation: $separation(M,\ \lambda x.\ snd(fst(x))=fst(snd(x)))$
and
separation_fst_in_snd: $separation(M,\ \lambda y.\ fst(snd(y)) \in snd(snd(y)))$
begin

— This lemma is similar to *strong_lam_replacement_imp_strong_replacement* and *lam_replacement_imp_strong_replacement_aux* but does not require g to be closed under M.
lemma *lam_replacement_imp_strong_replacement*:
assumes $lam_replacement(M,\ f)$
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 ∈ 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(f(x)) ∧ 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
    have ∃ Z[M]. ∀ z[M]. z∈Z ⟷ (∃ a[M]. a∈A ∧ z=f(a))
      by (rule_tac rexI[where x={snd(b) . b ∈ Y}],auto)
  }
  then
  show ?thesis unfolding strong_replacement_def by simp
qed

lemma Collect_middle: {p ∈ (λx∈A. f(x)) × (λx∈{f(x) . x∈A}. g(x)) . snd(fst(p))=fst(snd(p))}
    = { ⟨⟨x,f(x)⟩,⟨f(x),g(f(x))⟩⟩ . x∈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, λx. ⟨f(x),g(x)⟩)
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 $x{\in}C \Longrightarrow y{\in}C \Longrightarrow fst(x) = fst(y) \longrightarrow M(fst(y)) \wedge M(snd(x)) \wedge M(snd(y))$
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 $M(converse(?Y'){\times}?Z')$
by *simp*
moreover from *this*
have $M(\{p \in converse(?Y'){\times}?Z'\ .\ snd(fst(p)){=}fst(snd(p))\})$ (**is** $M(?P)$)
using *middle_separation* **by** *simp*
moreover from *calculation*
have $M(\{\ \langle snd(fst(p)),\langle fst(fst(p)),snd(snd(p))\rangle\rangle\ .\ p{\in}?P\ \})$ (**is** $M(?R)$)
using *RepFun_closed*[*OF lam_replacement_prodRepl*[*THEN*
lam_replacement_imp_strong_replacement] ‹$M(?P)$›]
unfolding *prodRepl_def* **by** *simp*
ultimately
have $b \in ?R \longleftrightarrow (\exists x[M].\ x \in A \wedge b = \langle x,\langle f(x),g(x)\rangle\rangle)$ **if** $M(b)$ **for** b
using *that*
apply(*intro iffI*) **apply**(*auto*)[*1*]
proof -
assume $\exists x[M].\ x \in A \wedge b = \langle x,\ f(x),\ g(x)\rangle$
moreover from *this*
obtain x **where** $M(x)\ x{\in}A\ b{=}\ \langle x,\ \langle f(x),\ g(x)\rangle\rangle$
by *auto*
moreover from *calculation that*
have $M(\langle x,f(x)\rangle)\ M(\langle x,g(x)\rangle)$ **by** *auto*
moreover from *calculation*
have $\langle f(x),x\rangle \in converse(?Y')\ \langle x,g(x)\rangle \in ?Z'$ **by** *auto*
moreover from *calculation*
have $\langle\langle f(x),x\rangle,\langle x,g(x)\rangle\rangle{\in}converse(?Y'){\times}?Z'$ **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 $b = \langle snd(fst(?p)),\langle fst(fst(?p)),snd(snd(?p))\rangle\rangle$ **by** *auto*
moreover from *calculation*
have $\langle snd(fst(?p)),\langle fst(fst(?p)),snd(snd(?p))\rangle\rangle{\in}?R$

```
      by(rule_tac RepFunI[of ?p ?P], simp)
    ultimately show b∈?R by simp
  qed
  with ‹M(?R)›
  have ∃ Y[M]. ∀ b[M]. b ∈ Y ⟷ (∃ x[M]. x ∈ A ∧ b = ⟨x,⟨f(x),g(x)⟩⟩)
    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) ∀ x[M]. M(f(x))
 shows lam_replacement(M, λx. g(f(x)))
proof -
 {
  fix A
  let ?Y={y . x∈A , y=f(x)}
  let ?Y'={y . x∈A , y=⟨x,f(x)⟩}
  have ∀ x∈C. M(⟨fst(fst(x)),snd(snd(x))⟩) if M(C) for C
    using transM[OF _ that] by auto
  moreover
  note assms
  moreover
  assume M(A)
  moreover from assms
  have eq:?Y = {y . x∈A ,M(y) ∧ y=f(x)}  ?Y' = {y . x∈A ,M(y) ∧ y=⟨x,f(x)⟩}
    using transM[OF _ ‹M(A)›] by auto
  moreover from ‹M(A)› 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∈?Y , M(z) ∧ z=⟨y,g(y)⟩}) (is M(?Z))
    using lam_replacement_imp_RepFun_Lam by auto
  moreover from calculation
  have M(?Y'×?Z)
    by simp
  moreover from this
  have M({p ∈ ?Y'×?Z . snd(fst(p))=fst(snd(p))}) (is M(?P))
    using middle_separation by simp
  moreover from calculation
  have M({ ⟨fst(fst(p)),snd(snd(p))⟩ . p∈?P }) (is M(?R))
    using RepFun_closed[OF lam_replacement_middle_del[THEN
          lam_replacement_imp_strong_replacement] ‹M(?P)›]
    unfolding middle_del_def by simp
  ultimately
  have b ∈ ?R ⟷ (∃ x[M]. x ∈ A ∧ b = ⟨x,g(f(x))⟩) if M(b) for b
    using that assms(3)
```

```
      apply(intro iffI) apply(auto)[1]
    proof -
      assume ∃x[M]. x ∈ A ∧ b = ⟨x, g(f(x))⟩
      moreover from this
      obtain x where M(x) x∈A b= ⟨x, g(f(x))⟩
        by auto
      moreover from calculation that assms(3)
      have M(f(x)) M(g(f(x))) by auto
      moreover from calculation
      have ⟨x,f(x)⟩ ∈ ?Y' by auto
      moreover from calculation
      have ⟨f(x),g(f(x))⟩∈?Z by auto
      moreover from calculation
      have ⟨⟨x,f(x)⟩,⟨f(x),g(f(x))⟩⟩ ∈ ?P
        (is ?p∈?P)
        by auto
      moreover from calculation
      have b = ⟨fst(fst(?p)),snd(snd(?p))⟩ by auto
      moreover from calculation
      have ⟨fst(fst(?p)),snd(snd(?p))⟩∈?R
        by(rule_tac RepFunI[of ?p ?P], simp)
      ultimately show b∈?R by simp
    qed
    with ‹M(?R)›
    have ∃ Y[M]. ∀ b[M]. b ∈ Y ⟷ (∃ x[M]. x ∈ A ∧ b = ⟨x,g(f(x))⟩)
      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_hcomp2:
  assumes lam_replacement(M,f) lam_replacement(M,g)
    ∀ x[M]. M(f(x)) ∀ x[M]. M(g(x))
    lam_replacement(M, λp. h(fst(p),snd(p)))
  shows lam_replacement(M, λx. h(f(x),g(x)))
  using assms lam_replacement_product[of f g]
    lam_replacement_hcomp[of λx. ⟨f(x), g(x)⟩ λ⟨x,y⟩. h(x,y)]
  unfolding split_def by simp

lemma lam_replacement_identity: lam_replacement(M,λx. x)
proof -
  {
    fix A
    assume M(A)
    moreover from this
    have id(A) = {⟨snd(fst(z)),fst(snd(z))⟩ . z∈ {z∈ (A×A)×(A×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∈ (A×A)×(A×A). snd(fst(z)) = fst(snd(z))}) (is M(?A′))
      using middle_separation by simp
    moreover from calculation
    have M({⟨snd(fst(z)),fst(snd(z))⟩ . z∈ ?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 strong_replacement_separation_aux :
  assumes strong_replacement(M,λ x y . y=f(x)) separation(M,P)
  shows strong_replacement(M, λx y . P(x) ∧ y=f(x))
proof -
  {
    fix A
    let ?Q=λX. ∀ b[M]. b ∈ X ⟷ (∃ x[M]. x ∈ A ∧ P(x) ∧ b = f(x))
    assume M(A)
    moreover from this
    have M({x∈A . P(x)}) (is M(?B)) using assms by simp
    moreover from calculation assms
    obtain Y where M(Y) ∀ b[M]. b ∈ Y ⟷ (∃ x[M]. x ∈ ?B ∧ b = f(x))
      unfolding strong_replacement_def by auto
    then
    have ∃ Y[M]. ∀ b[M]. b ∈ Y ⟷ (∃ x[M]. x ∈ A ∧ P(x) ∧ b = f(x))
      using rexI[of ?Q _ M] by simp
  }
  then
  show ?thesis
    unfolding strong_replacement_def by simp
qed

lemma separation_in:
  assumes ∀ x[M]. M(f(x)) lam_replacement(M,f)
    ∀ x[M]. M(g(x)) lam_replacement(M,g)
  shows separation(M,λx . f(x)∈g(x))
proof -
  let ?Z=λA. {⟨x,⟨f(x),g(x)⟩⟩. x∈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∈?Z(A) . fst(snd(u)) ∈snd(snd(u))}) (is M(?W(A))) if M(A) for A
    using that separation_fst_in_snd assms
    by auto
  then
  have M({fst(u) . u ∈ ?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∈A. f(x)∈g(x)} = {fst(u) . u∈?W(A)} for A
    by auto
  ultimately
  show ?thesis
    using separation_iff
    by auto
qed

lemma lam_replacement_swap: lam_replacement(M, λx. ⟨snd(x),fst(x)⟩)
  using lam_replacement_fst lam_replacement_snd
    lam_replacement_product[of snd fst] by simp


lemma relation_separation: separation(M, λz. ∃x y. z = ⟨x, y⟩)
  unfolding separation_def
proof (clarify)
  fix A
  assume M(A)
  moreover from this
  have {z∈A. ∃x y. z = ⟨x, y⟩} = {z∈A. ∃x∈domain(A). ∃y∈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 ∃y[M]. ∀x[M]. x ∈ y ⟷ x ∈ A ∧ (∃w y. x = ⟨w, y⟩)
    by (rule_tac x={z∈A. ∃x y. z = ⟨x, y⟩} in rexI) auto
qed

lemma separation_Pair:
  assumes separation(M, λy . 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_separation* :
assumes *lam_replacement*(M,f) *separation*(M,P)
shows *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*: $M(A) \Longrightarrow M(id(A))$
using *lam_replacement_identity lam_replacement_iff_lam_closed*
unfolding *id_def* **by** *simp*

lemma *lam_replacement_Pair*:
shows *lam_replacement*(M, $\lambda x.\ \langle fst(x),\ snd(x)\rangle$)
using *lam_replacement_product lam_replacement_fst lam_replacement_snd*
by *simp*

lemma *lam_replacement_Upair*: *lam_replacement*(M, $\lambda p.\ Upair(fst(p),\ snd(p))$)
proof -
have $(\neg(\exists\ a\ b\ .\ x = <a{,}b>)) \Longrightarrow fst(x) = 0 \wedge snd(x) = 0$ **for** x
unfolding *fst_def snd_def*
by (*simp add*:*the_0*)
then
have $Upair(fst(x){,}snd(x)) = (if\ (\exists\ w\ .\ \exists\ z\ .\ x{=}<w{,}z>)\ then\ (\bigcup x)\ else\ (\{0\}))$
for x

```
    using fst_conv snd_conv Upair_eq_cons
    by (auto simp add:Pair_def)
  moreover
  have lam_replacement(M, λx . (if (∃ w . ∃ z . x=<w,z>) then (⋃x) else ({0})))
    using lam_replacement_Union lam_replacement_constant relation_separation
lam_replacement_if
    by simp
  ultimately
  show ?thesis
    using lam_replacement_cong
    by simp
qed

lemma lam_replacement_Un: lam_replacement(M, λp. fst(p) ∪ snd(p))
  using lam_replacement_Upair lam_replacement_Union
    lam_replacement_hcomp[where g=Union and f=λp. Upair(fst(p),snd(p))]
  unfolding Un_def by simp

lemma lam_replacement_cons: lam_replacement(M, λp. cons(fst(p),snd(p)))
  using  lam_replacement_Upair
    lam_replacement_hcomp2[of _ _ (∪)]
    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_fun:
  assumes lam_replacement(M, f) ∀x[M]. M(f(x))
  shows lam_replacement(M, λx. {f(x)})
  using lam_replacement_constant lam_replacement_cons
    lam_replacement_hcomp2[of f λ_. 0 cons] assms
  by auto

lemma lam_replacement_sing: lam_replacement(M, λx. {x})
  using lam_replacement_sing_fun lam_replacement_identity
  by simp

lemma lam_replacement_CartProd:
  assumes lam_replacement(M,f) lam_replacement(M,g)
    ∀x[M]. M(f(x)) ∀x[M]. M(g(x))
  shows lam_replacement(M, λx. f(x) × g(x))
proof -
 note rep_closed = lam_replacement_imp_strong_replacement[THEN RepFun_closed]
  {
    fix A
    assume M(A)
    moreover
    note transM[OF _ ‹M(A)›]
    moreover from calculation assms
```

```
    have M({⟨x,⟨f(x),g(x)⟩⟩ . x∈A}) (is M(?A'))
    using lam_replacement_product[THEN lam_replacement_imp_lam_closed[unfolded
lam_def]]
      by simp
    moreover from calculation
    have M(⋃{f(x) . x∈A}) (is M(?F))
      using rep_closed[OF assms(1)] assms(3)
      by simp
    moreover from calculation
    have M(⋃{g(x) . x∈A}) (is M(?G))
      using rep_closed[OF assms(2)] assms(4)
      by simp
    moreover from calculation
    have M(?A' × (?F × ?G)) (is M(?T))
      by simp
    moreover from this
   have M({t ∈ ?T . fst(snd(t)) ∈ fst(snd(fst(t))) ∧ snd(snd(t)) ∈ snd(snd(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({⟨fst(fst(t)),snd(t)⟩ . t∈?Q}) (is M(?R))
      using rep_closed lam_replacement_product
         lam_replacement_hcomp[OF lam_replacement_fst lam_replacement_fst]
lam_replacement_snd
        transM[of _ ?Q]
      by simp
    moreover from calculation
    have M({⟨x,?R``{x}⟩ . x∈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``{x} = f(x)×g(x) if x∈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({⟨x,f(x) × g(x)⟩ . x∈A}) by auto
  }
  with assms
 show ?thesis using lam_replacement_iff_lam_closed[THEN iffD2,unfolded lam_def]
    by simp
qed
```

```
lemma lam_replacement_RepFun :
  assumes lam_replacement(M,λx . f(fst(x),snd(x))) lam_replacement(M,g)
    ∀x[M].∀y∈g(x).M(f(x,y)) ∀x[M].M(g(x))
  shows lam_replacement(M,λx .  {f(x,z) . z∈g(x)})
proof -
 note rep_closed = lam_replacement_imp_strong_replacement[THEN RepFun_closed]
  from assms
  have lam_replacement(M,λx.{x}×g(x))
    using lam_replacement_CartProd lam_replacement_sing
    by auto
  moreover from assms
  have M({f(x,z) . z ∈ g(x)}) if M(x) for x
    using that transM[of _ g(_)] rep_closed
      lam_replacement_product[OF lam_replacement_fst]
      lam_replacement_snd lam_replacement_identity
     assms(1)[THEN [5] lam_replacement_hcomp2] lam_replacement_constant[of
x]
    by simp
  moreover
  have M(λz∈A.{f(z,u) . u ∈ g(z)}) if M(A) for A
  proof -
    from that assms calculation
    have M(⋃{{x}×g(x) . x∈A}) (is M(?C))
      using rep_closed transM[of _ A]
      by simp
    with assms ‹M(A)›
    have M({⟨fst(x),f(fst(x),snd(x))⟩ . x ∈?C}) (is M(?B))
      using singleton_closed transM[of _ A] transM[of _ g(_)] rep_closed
        lam_replacement_product[OF lam_replacement_fst]
        lam_replacement_hcomp[OF lam_replacement_snd]
      by simp
    with ‹M(A)›
    have M({⟨x,?B``{x}⟩ . x∈A})
      using transM[of _ A] rep_closed
        lam_replacement_product[OF lam_replacement_identity]
        lam_replacement_Image[THEN [5] lam_replacement_hcomp2]
        lam_replacement_constant lam_replacement_sing
      by simp
    moreover
    have ?B``{z} = {f(z,u) . u ∈ g(z)} if z∈A for z
      using that
      by(intro equalityI subsetI,auto simp:imageI)
    moreover from this
    have {⟨x,?B``{x}⟩ . x∈A} = {⟨z,{f(z,u) . u ∈ g(z)}⟩ . z∈A}
      by auto
    ultimately
    show ?thesis
      unfolding lam_def by auto
```

```
  qed
  ultimately
  show ?thesis
    using lam_replacement_iff_lam_closed[THEN iffD2]
    by simp
qed

lemma lam_replacement_Collect :
  assumes lam_replacement(M,f) ∀x[M].M(f(x))
    separation(M,λx . F(fst(x),snd(x)))
  shows lam_replacement(M,λx. {y∈f(x) . F(x,y)})
proof -
 note rep_closed = lam_replacement_imp_strong_replacement[THEN RepFun_closed]
  from assms
  have 1:lam_replacement(M,λx.{x}×f(x))
    using lam_replacement_CartProd lam_replacement_sing
    by auto
  have M({u∈f(x) . F(x,u)}) if M(x) for x
  proof -
    from assms that
    have M({u ∈ {x}×f(x) . F(fst(u),snd(u))}) (is M(?F))
      by simp
    with assms that
    have M({snd(u) . u ∈ ?F})
      using rep_closed transM[of _ f(x)] lam_replacement_snd
      by simp
    moreover
    have {snd(u) . u ∈ ?F} = {u∈f(x) . F(x,u)}
      by auto
    ultimately show ?thesis by simp
  qed
  moreover
  have M(λz∈A.{y ∈ f(z) . F(z,y)}) if M(A) for A
  proof -
    from 1 that assms
    have M(⋃{{x}×f(x) . x∈A}) (is M(?C))
      using rep_closed transM[of _ A]
      by simp
    moreover from this assms
    have M({p ∈ ?C . F(fst(p),snd(p))}) (is M(?B))
      by(rule_tac separation_closed,simp_all)
    with ‹M(A)›
    have M({⟨x,?B``{x}⟩ . x∈A})
      using transM[of _ A] rep_closed
        lam_replacement_product[OF lam_replacement_identity]
        lam_replacement_Image[THEN [5] lam_replacement_hcomp2]
        lam_replacement_constant lam_replacement_sing
      by simp
    moreover
```

```
    have ?B``{z} = {u∈f(z) . F(z,u)} if z∈A for z
      using that
      by(intro equalityI subsetI,auto simp:imageI)
    moreover from this
    have {⟨x,?B``{x}⟩ . x∈A} = {⟨z,{u∈f(z) . F(z,u)}⟩ . z∈A}
      by auto
    ultimately
    show ?thesis
      unfolding lam_def by auto
  qed
  ultimately
  show ?thesis
    using lam_replacement_iff_lam_closed[THEN iffD2]
    by simp
qed


lemma separation_eq:
  assumes ∀x[M]. M(f(x)) lam_replacement(M,f)
    ∀x[M]. M(g(x)) lam_replacement(M,g)
  shows separation(M,λx . f(x) = g(x))
proof -
  let ?Z=λA. {⟨x,⟨f(x),⟨g(x),x⟩⟩⟩. x∈A}
  let ?Y=λA. {⟨⟨x,f(x)⟩,⟨g(x),x⟩⟩. x∈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_comp :
  assumes separation(M,P) lam_replacement(M,f) ∀x[M]. M(f(x))
  shows separation(M,λx. P(f(x)))
  unfolding separation_def
proof(clarify)
  fix A
  assume M(A)
  let ?B={f(a) . a ∈ A}
  let ?C=A×{b∈?B . P(b)}
  note ‹M(A)›
  moreover from calculation
  have M(?C)
   using lam_replacement_imp_strong_replacement assms RepFun_closed transM[of
_ A]
    by simp
  moreover from calculation
  have M({p∈?C . f(fst(p)) = snd(p)}) (is M(?Prod))
    using assms separation_eq lam_replacement_fst lam_replacement_snd
      lam_replacement_hcomp
    by simp
  moreover from calculation
  have M({fst(p) . p∈?Prod}) (is M(?L))
     using lam_replacement_imp_strong_replacement lam_replacement_fst Rep-
Fun_closed
      transM[of _ ?Prod]
    by simp
  moreover
  have ?L = {z∈A . P(f(z))}
    by(intro equalityI subsetI,auto)
  ultimately
```

```
  show  ∃ y[M]. ∀ z[M]. z ∈ y ⟷ z ∈ A ∧ P(f(z))
    by (rule_tac x=?L in rexI,simp_all)
qed

lemma lam_replacement_domain: lam_replacement(M,domain)
proof -
  have 1:{fst(z) . z∈ {u∈x . (∃ a b. u = <a,b>)}} =domain(x) for x
    unfolding domain_def
    by (intro equalityI subsetI,auto,rule_tac x=<xa,y> in bexI,auto)
  moreover
  have lam_replacement(M,λ x .{fst(z) . z∈ {u∈x . (∃ a b. u = <a,b>)}})
    using lam_replacement_hcomp lam_replacement_snd lam_replacement_fst
snd_closed[OF transM] fst_closed[OF transM] lam_replacement_identity
relation_separation relation_separation[THEN separation_comp,where f=snd]
lam_replacement_Collect
    by(rule_tac lam_replacement_RepFun,simp_all)
  ultimately
  show ?thesis
    using lam_replacement_cong
    by auto
qed

lemma separation_in_domain : M(a) ⟹ separation(M, λx. a∈domain(x))
  using lam_replacement_domain lam_replacement_constant separation_in
  by auto


lemma separation_all:
  assumes separation(M, λx . P(fst(fst(x)),snd(x)))
    lam_replacement(M,f) lam_replacement(M,g)
    ∀ x[M]. M(f(x)) ∀ x[M]. M(g(x))
  shows separation(M, λz. ∀ x∈f(z). P(g(z),x))
  unfolding separation_def
proof(clarify)
 note rep_closed = lam_replacement_imp_strong_replacement[THEN RepFun_closed]
 note lr_fs = lam_replacement_hcomp[OF lam_replacement_fst lam_replacement_snd]
  fix A
  assume M(A)
  with assms
  have M({f(x) . x∈A}) (is M(?F)) M({<g(x),f(x)> . x∈A}) (is M(?G))
    using rep_closed transM[of _ A]
      lam_replacement_product lam_replacement_imp_lam_closed
    unfolding lam_def
    by auto
  let ?B=⋃ ?F
  let ?C=?G×?B
  note ‹M(A)› ‹M(?F)› ‹M(?G)›
  moreover from calculation
  have M(?C)
```

```
    by simp
  moreover from calculation
  have M({p∈?C . P(fst(fst(p)),snd(p)) ∧ snd(p)∈snd(fst(p))}) (is M(?Prod))
   using assms separation_conj separation_in lam_replacement_fst lam_replacement_snd
      lam_replacement_hcomp
    by simp
  moreover from calculation
  have M({z∈?G . snd(z)=?Prod``{z}}) (is M(?L))
    using separation_eq
      lam_replacement_constant[of ?Prod] lam_replacement_constant[of 0]
      lam_replacement_hcomp[OF _ lam_replacement_sing]
      lam_replacement_Image[THEN [5] lam_replacement_hcomp2]
      lam_replacement_identity lam_replacement_snd
    by simp
  moreover from calculation assms
  have M({x∈A . <g(x),f(x)> ∈ ?L}) (is M(?N))
    using lam_replacement_product lam_replacement_constant
    by (rule_tac separation_closed,rule_tac separation_in,auto)
  moreover from calculation
  have ?N = {z∈A . ∀x∈f(z). P(g(z),x)}
  proof -
    have P(g(z),x) if z∈A x∈f(z) f(z) = ?Prod``{<g(z),f(z)>} for z x
      using that
      by(rule_tac imageE[of x ?Prod {<g(z),f(z)>}],simp_all)
    moreover
    have f(z) = ?Prod `` {<g(z),f(z)>} if z∈A ∀x∈f(z). P(g(z), x) for z
      using that
      by force
    ultimately
    show ?thesis
      by auto
  qed
  ultimately
  show ∃y[M]. ∀z[M]. z ∈ y ⟷ z ∈ A ∧ (∀x∈f(z) . P(g(z),x))
    by (rule_tac x=?N in rexI,simp_all)
qed

lemma separation_ex:
  assumes separation(M, λx . P(fst(fst(x)),snd(x)))
    lam_replacement(M,f) lam_replacement(M,g)
    ∀x[M]. M(f(x)) ∀x[M]. M(g(x))
  shows separation(M, λz. ∃x∈f(z). P(g(z),x))
  unfolding separation_def
proof(clarify)
 note rep_closed = lam_replacement_imp_strong_replacement[THEN RepFun_closed]
 note lr_fs = lam_replacement_hcomp[OF lam_replacement_fst lam_replacement_snd]
  fix A
  assume M(A)
  with assms
```

```
  have M({f(x) . x∈A}) (is M(?F)) M({<g(x),f(x)> . x∈A}) (is M(?G))
    using rep_closed transM[of _ A]
      lam_replacement_product lam_replacement_imp_lam_closed
    unfolding lam_def
    by auto
  let ?B=⋃ ?F
  let ?C=?G×?B
  note ‹M(A)› ‹M(?F)› ‹M(?G)›
  moreover from calculation
  have M(?C)
    by simp
  moreover from calculation
  have M({p∈?C . P(fst(fst(p)),snd(p)) ∧ snd(p)∈snd(fst(p))}) (is M(?Prod))
   using assms separation_conj separation_in lam_replacement_fst lam_replacement_snd
      lam_replacement_hcomp
    by simp
  moreover from calculation
  have M({z∈?G . 0≠?Prod``{z}}) (is M(?L))
    using separation_eq  separation_neg
      lam_replacement_constant[of ?Prod] lam_replacement_constant[of 0]
      lam_replacement_hcomp[OF _ lam_replacement_sing]
      lam_replacement_Image[THEN [5] lam_replacement_hcomp2]
      lam_replacement_identity
    by simp
  moreover from calculation assms
  have M({x∈A . <g(x),f(x)> ∈ ?L}) (is M(?N))
    using lam_replacement_product lam_replacement_constant
    by (rule_tac separation_closed,rule_tac separation_in,auto)
  moreover from calculation
  have ?N = {z∈A . ∃x∈f(z). P(g(z),x)}
    by(intro equalityI subsetI,simp_all,force) (rule ccontr,force)
  ultimately
  show  ∃y[M]. ∀z[M]. z ∈ y ⟷ z ∈ A ∧ (∃x∈f(z) . P(g(z),x))
    by (rule_tac x=?N in rexI,simp_all)
qed

lemma lam_replacement_converse: lam_replacement(M,converse)
proof-
  note lr_Un = lam_replacement_Union[THEN [2] lam_replacement_hcomp]
 note lr_fs = lam_replacement_hcomp[OF lam_replacement_fst lam_replacement_snd]
 note lr_ff = lam_replacement_hcomp[OF lam_replacement_fst lam_replacement_fst]
 note lr_ss = lam_replacement_hcomp[OF lam_replacement_snd lam_replacement_snd]
  note lr_sf = lam_replacement_hcomp[OF  lr_ff lam_replacement_snd]
  note lr_fff = lam_replacement_hcomp[OF lam_replacement_fst lr_ff]
  note lr_f4 = lam_replacement_hcomp[OF lr_ff lr_ff]
  have eq:converse(x) = {⟨snd(y),fst(y)⟩ . y ∈ {z ∈ x. ∃  u ∈⋃⋃x . ∃v∈⋃⋃x .
z=<u,v>}} for x
    unfolding converse_def
   by(intro equalityI subsetI,auto,rename_tac a b,rule_tac x=<a,b> in bexI,simp_all)
```

(*auto simp* : *Pair_def*)
have $M(x) \Longrightarrow M(\{z \in x.\ \exists\ u \in\bigcup\bigcup x\ .\ \exists\, v\in\bigcup\bigcup x\ .\ z=<u,v>\})$ **for** x
using *lam_replacement_identity lr_Un*[*OF lr_Un*] *lr_Un lr_Un*[*OF lam_replacement_Union*]
lam_replacement_snd lr_ff lam_replacement_fst lam_replacement_hcomp
lr_fff
lam_replacement_product[*OF lr_sf lam_replacement_snd*] *lr_f4*
lam_replacement_hcomp[*OF lr_f4 lam_replacement_snd*] *separation_eq*
lam_replacement_constant
by(*rule_tac separation_closed*,*rule_tac separation_ex*,*rule_tac separation_ex*,*simp_all*)
moreover
have *sep*:*separation*(M, $\lambda x.\ \exists\, u\in\bigcup\bigcup fst(x).\ \exists\, v\in\bigcup\bigcup fst(x).\ snd(x) = \langle u,\ v\rangle$)
using *lam_replacement_identity lr_Un*[*OF lr_Un*] *lr_Un*
lam_replacement_snd lr_ff lam_replacement_fst lam_replacement_hcomp
lr_fff
lam_replacement_product[*OF lr_sf lam_replacement_snd*]
lam_replacement_hcomp[*OF lr_f4 lam_replacement_snd*] *separation_eq*
by(*rule_tac separation_ex*,*rule_tac separation_ex*,*simp_all*)
moreover from *calculation*
have *lam_replacement*(M, $\lambda x.\ \{z \in x\ .\ \exists\, u\in\bigcup\bigcup x.\ \exists\, v\in\bigcup\bigcup x.\ z = \langle u,\ v\rangle\}$)
using *lam_replacement_identity*
by(*rule_tac lam_replacement_Collect*,*simp_all*)
ultimately
have *1*:*lam_replacement*(M, $\lambda x\ .\ \{\langle snd(y),fst(y)\rangle\ .\ y \in \{z \in x.\ \exists\, u \in\bigcup\bigcup x\ .$
$\exists\, v\in\bigcup\bigcup x.\ z=<u,v>\}\}$)
using *lam_replacement_product lam_replacement_fst lam_replacement_hcomp*
lam_replacement_identity lr_ff lr_ss
lam_replacement_snd lam_replacement_identity sep
by(*rule_tac lam_replacement_RepFun*,*simp_all*,*force dest*:*transM*)
with *eq*[*symmetric*]
show *?thesis*
by(*rule_tac lam_replacement_cong*[*OF 1*],*simp_all*)
qed

lemma *lam_replacement_Diff*: *lam_replacement*(M, $\lambda x\ .\ fst(x)$ - $snd(x)$)
proof -
have *eq*:X - $Y = \{x\in X\ .\ x\notin Y\}$ **for** X Y
by *auto*
moreover
have *lam_replacement*(M, $\lambda x\ .\ \{y \in fst(x)\ .\ y\notin snd(x)\}$)
using *lam_replacement_fst lam_replacement_hcomp*
lam_replacement_snd lam_replacement_identity separation_in separation_neg
by(*rule_tac lam_replacement_Collect*,*simp_all*)
then
show *?thesis*
by(*rule_tac lam_replacement_cong*,*auto*,*subst eq*[*symmetric*],*simp*)
qed

lemma *lam_replacement_range* : *lam_replacement*(M,*range*)
unfolding *range_def*

using *lam_replacement_hcomp*[*OF lam_replacement_converse lam_replacement_domain*]
by *auto*

lemma *separation_in_range* : $M(a) \Longrightarrow$ *separation*(M, $\lambda x.\ a \in$ *range*(x))
using *lam_replacement_range lam_replacement_constant separation_in*
by *auto*

lemmas *tag_replacement* = *lam_replacement_constant*[*unfolded lam_replacement_def*]

lemma *tag_lam_replacement* : $M(X) \Longrightarrow$ *lam_replacement*(M,$\lambda x.$ <X,x>)
using *lam_replacement_product*[*OF lam_replacement_constant*] *lam_replacement_identity*
by *simp*

lemma *strong_lam_replacement_imp_lam_replacement_RepFun* :
assumes *strong_replacement*(M,$\lambda\, x\, z$. P(*fst*(x),*snd*(x)) $\wedge$ z=$\langle x$,f(*fst*(x),*snd*(x))$\rangle$)
lam_replacement(M,g)
$\bigwedge A\ y$. $M(y) \Longrightarrow M(A) \Longrightarrow \forall\, x \in A.\ P(y,x) \longrightarrow M(f(y,x))$
$\forall\, x[M].\ M(g(x))$
separation(M, $\lambda x.$ P(*fst*(x),*snd*(x)))
shows *lam_replacement*(M, $\lambda x.$ $\{y$. $r \in g(x)$, $P(x,r) \wedge y$=$f(x,r)\}$)
proof -
note *rep_closed* = *lam_replacement_imp_strong_replacement*[*THEN RepFun_closed*]
moreover
have $\{f(x,\ xa)$. $xa \in \{xa \in g(x)$. $P(x,\ xa)\}\} = \{y$. $z \in g(x),\ P(x,\ z) \wedge y =$
$f(x,\ z)\}$ **for** x
by(*intro equalityI subsetI*,*auto*)
moreover from *assms*
have *0*:$M(\{xa \in g(x)$. $P(x,\ xa)\})$ **if** $M(x)$ **for** x
using *that separation_closed assms*(*5*)[*THEN separation_comp*,*OF tag_lam_replacement*]
by *simp*
moreover from *assms*
have *1*:*lam_replacement*(M,$\lambda x.\{x\}\times\{u \in g(x)$. $P(x,u)\}$) (**is** *lam_replacement*(M,$\lambda x.$*?R*(x)))
using *separation_closed assms*(*5*)[*THEN separation_comp*,*OF tag_lam_replacement*]
by(*rule_tac lam_replacement_CartProd*[*OF lam_replacement_sing lam_replacement_Collect*],*simp_all*)
moreover from *assms*
have $M(\{y$. $z \in g(x)$, $P(x,z) \wedge y$=$f(x,z)\})$ (**is** M(*?Q*(x))) **if** $M(x)$ **for** x
using *that transM*[*of _ g*(_)]
separation_closed assms(*5*)[*THEN separation_comp*,*OF tag_lam_replacement*]
assms(*3*)[*of x g*(x)] *strong_lam_replacement_imp_strong_replacement*
by *simp*
moreover
have $M(\lambda z \in A.\{f(z,r)$. $r \in \{u \in g(z)$. $P(z,u)\}\})$ **if** $M(A)$ **for** A
proof -
from *that assms calculation*
have $M(\bigcup\{$*?R*(x) . $x \in A\})$ (**is** M(*?C*))
using *transM*[*of _ A*] *rep_closed*
by *simp*
moreover from *assms* ‹$M(A)$›
have $x \in \{y\} \times \{x \in g(y)$. $P(y,\ x)\} \Longrightarrow M(x) \wedge M(f($*fst*($x$),*snd*($x$))) **if** $y \in A$

```
for y x
      using assms(3)[of y g(y)] transM[of _ A] transM[of _ g(y)] that
      by force
    moreover from this
    have ∃ y∈A . x ∈ {y} × {x ∈ g(y) . P(y, x)} ⟹ M(x) ∧ M(f(fst(x),snd(x)))
for x
      by auto
    moreover note assms ‹M(A)›
    ultimately
    have M({z . x∈?C , P(fst(x),snd(x)) ∧ z = ⟨x,f(fst(x),snd(x))⟩}) (is M(?B))
      using singleton_closed transM[of _ A] transM[of _ g(_)] rep_closed
        lam_replacement_product[OF lam_replacement_fst]
        lam_replacement_hcomp[OF lam_replacement_snd] transM[OF _ 0]
      by(rule_tac strong_replacement_closed,simp_all)
    then
    have M({⟨fst(fst(x)),snd(x)⟩ . x∈?B}) (is M(?D))
      using rep_closed transM[of _ ?B]
        lam_replacement_product[OF
          lam_replacement_hcomp[OF lam_replacement_fst lam_replacement_fst]
          lam_replacement_snd]
      by simp
    with ‹M(A)›
    have M({⟨x,?D``{x}⟩ . x∈A})
      using transM[of _ A] rep_closed
        lam_replacement_product[OF lam_replacement_identity]
        lam_replacement_Image[THEN [5] lam_replacement_hcomp2]
        lam_replacement_constant lam_replacement_sing
      by simp
    moreover from calculation
    have ?D``{z} = {f(z,r) . r ∈ {u∈ g(z) . P(z,u)}} if z∈A for z
      using that
      by (intro equalityI subsetI,auto,intro imageI,force,auto)
    moreover from this
    have {⟨x,?D``{x}⟩ . x∈A} = {⟨z,{f(z,r) . r ∈ {u∈ g(z) . P(z,u)}}⟩ . z∈A}
      by auto
    ultimately
    show ?thesis
      unfolding lam_def by auto
  qed
  ultimately
  show ?thesis
    using lam_replacement_iff_lam_closed[THEN iffD2]
    by simp
qed

lemma lam_replacement_Collect' :
  assumes M(A) separation(M,λp . F(fst(p),snd(p)))
  shows lam_replacement(M,λx. {y∈A . F(x,y)})
  using assms lam_replacement_constant
```

by(*rule_tac lam_replacement_Collect*, *simp_all*)

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, $\lambda p.\ fst(p)$ ‘ $snd(p)$)
using *lam_replacement_fst lam_replacement_sing_fun*[*OF lam_replacement_snd*]
lam_replacement_Image lam_replacement_Union
lam_replacement_hcomp[*of* _ *Union*] *lam_replacement_hcomp2*[*of* _ _ (‘‘)]
unfolding *apply_def*
by *simp*

lemma *lam_replacement_apply*:$M(S) \Longrightarrow$ *lam_replacement*(M, $\lambda x.\ \ S$ ‘ x)
using *lam_replacement_constant*[*of* S] *lam_replacement_identity*
lam_replacement_apply2[*THEN* [*5*] *lam_replacement_hcomp2*]
by *simp*

lemma *apply_replacement*:$M(S) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ y.\ y = S$ ‘ x)
using *lam_replacement_apply lam_replacement_imp_strong_replacement* **by** *simp*

lemma *lam_replacement_id_const*: $M(b) \Longrightarrow$ *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*: $M(b) \Longrightarrow$ *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*: $M(f) \Longrightarrow M(z) \Longrightarrow$
lam_replacement(M, $\lambda x.\ f$ ‘ $\langle z, x\rangle$)
using *lam_replacement_const_id*[*of* z] *lam_replacement_apply*
lam_replacement_hcomp[*of* $\lambda x.\ \langle z, x\rangle$] **by** *simp*

lemmas *apply_replacement2* = *lam_replacement_apply_const_id*[*unfolded lam_replacement_def*]

lemma *lam_replacement_Inl*: *lam_replacement*(M, *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*: *lam_replacement*(M, *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'*: $M(X) \Longrightarrow$ *lam_replacement*(M, $\lambda x.\ x$ - X)
using *lam_replacement_Diff*[*THEN* [*5*] *lam_replacement_hcomp2*]
lam_replacement_constant lam_replacement_identity
by *simp*

lemmas *Pair_diff_replacement* = *lam_replacement_Diff'*[*unfolded lam_replacement_def*]

lemma *diff_Pair_replacement*: $M(p) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ y$. y=$\langle x$,x-$\{p\}\rangle$)
using *Pair_diff_replacement* **by** *simp*

lemma *swap_replacement*:*strong_replacement*(M, $\lambda x\ y.\ y = \langle x, (\lambda\langle x,y\rangle.\ \langle y, x\rangle)(x)\rangle$)
using *lam_replacement_swap* **unfolding** *lam_replacement_def split_def* **by** *simp*

lemma *lam_replacement_Un_const*:$M(b) \Longrightarrow$ *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*: *lam_replacement*(M,$\lambda p.\ \langle fst(p) \cup snd(p), fst(p), snd(p)\rangle$)
using *lam_replacement_Un lam_replacement_fst lam_replacement_snd*
by (*fast intro*: *lam_replacement_product lam_replacement_hcomp2*)

lemma *csquare_lam_replacement*:*strong_replacement*(M, $\lambda x\ y.\ y = \langle x, (\lambda\langle x,y\rangle.\ \langle x \cup y, x, y\rangle)(x)\rangle$)
using *lam_replacement_csquare* **unfolding** *split_def lam_replacement_def* **.**

lemma *lam_replacement_assoc*:*lam_replacement*(M,$\lambda x.\ \langle fst(fst(x)), snd(fst(x)), snd(x)\rangle$)
using *lam_replacement_fst lam_replacement_snd*
by (*force intro*: *lam_replacement_product lam_replacement_hcomp*)

lemma *assoc_replacement*:*strong_replacement*(M, $\lambda x\ y.\ y = \langle x, (\lambda\langle\langle x,y\rangle,z\rangle.\ \langle x, y, z\rangle)(x)\rangle$)
using *lam_replacement_assoc* **unfolding** *split_def lam_replacement_def* **.**

lemma *lam_replacement_prod_fun*: $M(f) \Longrightarrow M(g) \Longrightarrow$ *lam_replacement*(M,$\lambda x.\ \langle f\ `\ fst(x), g\ `\ snd(x)\rangle$)
using *lam_replacement_fst lam_replacement_snd*
by (*force intro*: *lam_replacement_product lam_replacement_hcomp lam_replacement_apply*)

lemma *prod_fun_replacement*:$M(f) \Longrightarrow M(g) \Longrightarrow$
strong_replacement(M, $\lambda x\ y.\ y = \langle x, (\lambda\langle w,y\rangle.\ \langle f\ `\ w,\ g\ `\ y\rangle)(x)\rangle$)
using *lam_replacement_prod_fun* **unfolding** *split_def lam_replacement_def* **.**

lemma *separation_subset*:
assumes $\forall x[M].\ M(f(x))$ *lam_replacement*(M,f)
$\forall x[M].\ M(g(x))$ *lam_replacement*(M,g)
shows *separation*(M,$\lambda x\ .\ f(x) \subseteq g(x)$)
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 *lam_replacement_twist*: *lam_replacement*(M,$\lambda\langle\langle x,y\rangle,z\rangle.\ \langle x,y,z\rangle$)
using *lam_replacement_fst lam_replacement_snd*
lam_replacement_Pair[*THEN* [*5*] *lam_replacement_hcomp2*,
of $\lambda x.\ snd(fst(x))$ $\lambda x.\ snd(x)$, *THEN* [*2*] *lam_replacement_Pair*[
THEN [*5*] *lam_replacement_hcomp2*, *of* $\lambda x.\ fst(fst(x))$]]
lam_replacement_hcomp **unfolding** *split_def* **by** *simp*

lemma *twist_closed*[*intro*,*simp*]: $M(x) \Longrightarrow M((\lambda\langle\langle x,y\rangle,z\rangle.\ \langle x,y,z\rangle)(x))$
unfolding *split_def* **by** *simp*

lemma *lam_replacement_vimage* :
shows *lam_replacement*(M, $\lambda x.\ fst(x)$-``$snd(x)$)
unfolding *vimage_def* **using**
lam_replacement_Image[*THEN* [*5*] *lam_replacement_hcomp2*]
lam_replacement_hcomp[*OF lam_replacement_fst lam_replacement_converse*]
lam_replacement_snd
by *simp*

lemma *lam_replacement_vimage_sing*: *lam_replacement*(M, $\lambda p.\ fst(p)$ -``$\{snd(p)\}$)
using *lam_replacement_snd lam_replacement_sing_fun*
lam_replacement_vimage[*THEN* [*5*] *lam_replacement_hcomp2*] *lam_replacement_fst*
by *simp*

lemma *lam_replacement_vimage_sing_fun*: $M(f) \Longrightarrow$ *lam_replacement*(M, $\lambda x.$
f -``$\{x\}$)
using *lam_replacement_vimage_sing*[*THEN* [*5*] *lam_replacement_hcomp2*] *lam_replacement_constant*[*of*
f]
lam_replacement_identity
by *simp*

lemma *lam_replacement_image_sing_fun*: $M(f) \Longrightarrow$ *lam_replacement*(M, $\lambda x.\ f$ `` $\{x\}$)
using *lam_replacement_Image*[*THEN* [*5*] *lam_replacement_hcomp2*] *lam_replacement_constant*[*of f*]
lam_replacement_sing
by *simp*

lemma *converse_apply_projs*: $\forall x[M].\ \bigcup$ (*fst*(x) -`` {*snd*(x)}) = *converse*(*fst*(x)) ` (*snd*(x))
using *converse_apply_eq* **by** *auto*

lemma *lam_replacement_converse_app*: *lam_replacement*(M, $\lambda p.$ *converse*(*fst*(p)) ` *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*:
$M(x) \Longrightarrow$ *lam_replacement*(M, $\lambda y.\ \{\langle x,\ y\rangle\}$)
using *lam_replacement_const_id*[*of x*] *lam_replacement_sing_fun*
by *simp*

lemma *tag_singleton_closed*: $M(x) \Longrightarrow M(z) \Longrightarrow M(\{\{\langle z,\ y\rangle\}\ .\ y \in x\})$
using *RepFun_closed lam_replacement_imp_strong_replacement lam_replacement_sing_const_id transM*[*of* _ *x*]
by *simp*

lemma *separation_ball*:
assumes *separation*(M, $\lambda y.\ f$(*fst*(y),*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 ‹$M(X)$›
moreover from *calculation*
have $M(A{\times}X)$
by *simp*
then
have $M(\{p \in A{\times}X\ .\ f(fst(p),snd(p))\})$ (**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 ∀ a[M]. a ∈ ?A′ ⟷ a ∈ A ∧ (∀ x∈X. f(a, x))
    using image_singleton_iff
    by auto
  with ‹M(?A′)›
  show ∃ y[M]. ∀ a[M]. a ∈ y ⟷ a ∈ A ∧ (∀ x∈X. f(a, x))
    by (rule_tac x=?A′ in rexI,simp_all)
qed

lemma separation_bex:
  assumes separation(M, λy. f(fst(y),snd(y))) M(X)
  shows separation(M, λy. ∃ u∈X. f(y,u))
  unfolding separation_def
proof(clarify)
  fix A
  assume M(A)
  moreover
  note ‹M(X)›
  moreover from calculation
  have M(A×X)
    by simp
  then
  have M({p ∈ A×X . f(fst(p),snd(p))}) (is M(?P))
    using assms(1)
    by auto
  moreover from calculation
  have M({a∈A . ?P‘‘{a} ≠ 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 ∀ a[M]. a ∈ ?A′ ⟷ a ∈ A ∧ (∃ x∈X. f(a, x))
    using image_singleton_iff
    by auto
  with ‹M(?A′)›
  show ∃ y[M]. ∀ a[M]. a ∈ y ⟷ a ∈ A ∧ (∃ x∈X. f(a, x))
```

by (*rule_tac x=?A′* **in** *rexI*,*simp_all*)
qed

lemma *lam_replacement_Lambda*:
assumes *lam_replacement*(M, λy. b(*fst*(y), *snd*(y)))
$\forall w[M]$. $\forall y[M]$. $M(b(w, y))$ $M(W)$
shows *lam_replacement*(M, λx. $\lambda w \in W$. $b(x, w)$)
using *assms lam_replacement_constant lam_replacement_product lam_replacement_snd lam_replacement_RepFun transM*[*of _ W*]
unfolding *lam_def*
by *simp*

lemma *lam_replacement_apply_Pair*:
assumes $M(y)$
shows *lam_replacement*(M, λx. y ‘ $\langle$*fst*(x), *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, λw. *fst*(w) ‘ *fst*(*snd*(w)) ‘ *snd*(*snd*(w)))
using *lam_replacement_fst lam_replacement_snd lam_replacement_hcomp lam_replacement_apply2*[*THEN* [*5*] *lam_replacement_hcomp2*]
by *auto*

lemma *lam_replacement_RepFun_apply* :
assumes $M(f)$
shows *lam_replacement*(M, λx . $\{f‘y$. $y \in x\}$)
using *assms lam_replacement_identity lam_replacement_RepFun lam_replacement_hcomp lam_replacement_snd lam_replacement_apply*
by(*rule_tac lam_replacement_RepFun*,*auto dest*:*transM*)

lemma *separation_snd_in_fst*: *separation*(M, λx. *snd*(x) $\in$ *fst*(x))
using *separation_in lam_replacement_fst lam_replacement_snd*
by *auto*

lemma *lam_replacement_if_mem*:
lam_replacement(M, λx. *if snd*(x) $\in$ *fst*(x) *then 1 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, λx. $\lambda w \in X$. x ‘ *fst*(w) ‘ *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, $\lambda x.\ \lambda w{\in}X.\ y$ ‘ $\langle x,\ w\rangle$)
using *assms lam_replacement_apply_Pair lam_replacement_Lambda*
by *simp*

lemma *lam_replacement_Lambda_if_mem*:
assumes $M(X)$
shows *lam_replacement*(M, $\lambda x.\ \lambda xa{\in}X.$ *if* $xa \in x$ *then 1 else 0*)
using *assms lam_replacement_if_mem lam_replacement_Lambda*
by *simp*

lemma *case_closed* :
assumes $\forall x[M].\ M(f(x))$ $\forall x[M].\ M(g(x))$
shows $\forall x[M].\ M(case(f,g,x))$
unfolding *case_def split_def cond_def*
using *assms* **by** *simp*

lemma *separation_fst_equal* : $M(a) \Longrightarrow separation(M,\lambda x\ .\ fst(x){=}a)$
using *separation_eq lam_replacement_fst lam_replacement_constant*
by *auto*

lemma *lam_replacement_case* :
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, $\lambda x\ .\ case(f,g,x)$)
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*: $M(x) \Longrightarrow M(y) \Longrightarrow$ *strong_replacement*(M, $\lambda ya\ z.\ ya \in y \wedge z = \{\langle x,\ ya\rangle\}$)
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*:
$M(f) \Longrightarrow M(x) \Longrightarrow$ *strong_replacement*(M, $\lambda y\ z.\ y \in f$ -‘‘ $\{x\} \wedge z = \{\langle x,\ y\rangle\}$)
using *Pi_replacement1 vimage_closed singleton_closed*
by *simp*

lemmas *domain_replacement* = *lam_replacement_domain*[*unfolded lam_replacement_def*]

lemma *domain_replacement_simp*: *strong_replacement*(M, $\lambda x\ y.\ y{=}domain(x)$)
using *lam_replacement_domain lam_replacement_imp_strong_replacement* **by**

simp

lemma *un_Pair_replacement*: $M(p) \Longrightarrow$ *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*: $M(X) \Longrightarrow$ *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*:
lam_replacement$(M,\lambda z\ .\ succ(z))$
unfolding *succ_def*
using *lam_replacement_hcomp2*[*of* $\lambda x.\ x\ \lambda x.\ x$ *cons*]
lam_replacement_cons lam_replacement_identity
by *simp*

lemma *lam_replacement_hcomp_Least*:
assumes *lam_replacement*$(M,\ g)$ *lam_replacement*$(M,\lambda x.\ \mu\ i.\ x{\in}F(i,x))$
$\forall x[M].\ M(g(x))\ \bigwedge x\ i.\ M(x) \Longrightarrow i \in F(i,\ x) \Longrightarrow M(i)$
shows *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*: $M(A) \Longrightarrow$ *separation*$(M,\ \lambda x\ .\ domain(x){\in}A)$
using *separation_in lam_replacement_constant lam_replacement_domain*
by *auto*

lemma *domain_eq_separation*: $M(p) \Longrightarrow$ *separation*$(M,\ \lambda x\ .\ domain(x) = p)$
using *separation_eq lam_replacement_domain lam_replacement_constant*
by *auto*

lemma *lam_replacement_Int*:
shows *lam_replacement*$(M,\ \lambda x.\ fst(x) \cap snd(x))$
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 *restrict_eq_separation'*: $M(B) \Longrightarrow \forall A[M].\ separation(M,\ \lambda y.\ \exists x[M].\ x{\in}A$
$\wedge\ y = \langle x,\ restrict(x,\ B)\rangle)$

proof(*clarify*)
  fix A
  have $restrict(r,B) = r \cap (B \times range(r))$ **for** r
    unfolding *restrict_def* **by**(*rule equalityI subsetI*,*auto*)
  moreover
  assume $M(A)$ $M(B)$
  moreover from *this*
  have $separation(M, \lambda y.\ \exists x{\in}A.\ y = \langle x,\ x \cap (B \times range(x))\rangle)$
    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 $separation(M, \lambda y.\ \exists x[M].\ x{\in}A \wedge y = \langle x,\ restrict(x,\ B)\rangle)$
    by *simp*
qed

lemmas *lam_replacement_restrict′* = *lam_replacement_restrict*[*OF restrict_eq_separation′*]

lemma *restrict_strong_replacement*: $M(A) \Longrightarrow strong_replacement(M, \lambda x\ y.\ y{=}restrict(x,A))$
  using *lam_replacement_restrict restrict_eq_separation′*
    lam_replacement_imp_strong_replacement
  by *simp*

lemma *restrict_eq_separation*: $M(r) \Longrightarrow M(p) \Longrightarrow separation(M, \lambda x\ .\ restrict(x,r)$
$= p)$
  using *separation_eq lam_replacement_restrict′ lam_replacement_constant*
  by *auto*

lemma *separation_equal_fst2* : $M(a) \Longrightarrow separation(M,\lambda x\ .\ fst(fst(x))=a)$
 using *separation_eq lam_replacement_hcomp lam_replacement_fst lam_replacement_constant*
  by *auto*

lemma *separation_equal_apply*: $M(f) \Longrightarrow M(a) \Longrightarrow separation(M,\lambda x.\ f`x=a)$
  using *separation_eq lam_replacement_apply*[*of f*] *lam_replacement_constant*
  by *auto*

lemma *lam_apply_replacement*: $M(A) \Longrightarrow M(f) \Longrightarrow lam_replacement(M,\ \lambda x\ .$
$\lambda n{\in}A.\ f\ `\ \langle x,\ n\rangle)$
 using *lam_replacement_Lambda lam_replacement_hcomp*[*OF _ lam_replacement_apply*[*of*
f]] *lam_replacement_Pair*
  by *simp*

— We need a tactic to deal with composition of replacements!
lemma *lam_replacement_comp* :
  $lam_replacement(M,\ \lambda x\ .\ fst(x)\ O\ snd(x))$
proof -

note *lr_fs* = *lam_replacement_hcomp*[*OF lam_replacement_fst lam_replacement_snd*]
note *lr_ff* = *lam_replacement_hcomp*[*OF lam_replacement_fst lam_replacement_fst*]
note *lr_ss* = *lam_replacement_hcomp*[*OF lam_replacement_snd lam_replacement_snd*]
note *lr_sf* = *lam_replacement_hcomp*[*OF lam_replacement_snd lam_replacement_fst*]
note *lr_fff* = *lam_replacement_hcomp*[*OF lam_replacement_fst lr_ff*]
have *eq*:*R O S* =
$\{xz \in domain(S) \times range(R)$.
$\exists\, y \in range(S) \cap domain(R)$. $\langle fst(xz), y\rangle \in S \wedge \langle y, snd(xz)\rangle \in R\}$ **for** *R S*
unfolding *comp_def*
by(*intro equalityI subsetI*,*auto*)
moreover
have *lam_replacement*(M, λx . $\{xz \in domain(snd(x)) \times range(fst(x))$.
$\exists\, y \in range(snd(x)) \cap domain(fst(x))$. $\langle fst(xz), y\rangle \in snd(x) \wedge \langle y, snd(xz)\rangle \in fst(x)\}$)
using *lam_replacement_CartProd lam_replacement_hcomp*
lam_replacement_range lam_replacement_domain lam_replacement_fst lam_replacement_snd
lam_replacement_identity lr_fs lr_ff lr_ss lam_replacement_product
lam_replacement_Int[*THEN* [*5*] *lam_replacement_hcomp2*]
lam_replacement_hcomp[*OF lr_ff lam_replacement_domain*]
lam_replacement_hcomp[*OF lr_fs lam_replacement_range*]
lam_replacement_hcomp[*OF lr_ff lr_ff*]
lam_replacement_hcomp[*OF lr_ff lr_ss*]
lam_replacement_hcomp[*OF lr_ff lr_sf*]
lr_fff
lam_replacement_hcomp[*OF lr_fff lam_replacement_snd*]
separation_ex separation_conj separation_in
by(*rule_tac lam_replacement_Collect*,*simp_all*,*rule_tac separation_ex*,*rule_tac separation_conj*,*auto*)
ultimately
show *?thesis*
by(*rule_tac lam_replacement_cong*,*auto*,*subst eq*[*symmetric*],*rule_tac comp_closed*,*auto*)
qed

lemma *lam_replacement_comp′*:
$M(f) \Longrightarrow M(g) \Longrightarrow$ *lam_replacement*(M, λx . $f\ O\ x\ O\ g$)
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 *RepFun_SigFun_closed*: $M(x) \Longrightarrow M(z) \Longrightarrow M(\{\{\langle z, u\rangle\}\ .\ u \in x\})$
using *lam_replacement_sing_const_id lam_replacement_imp_strong_replacement RepFun_closed*
transM[*of _ x*] *singleton_in_M_iff pair_in_M_iff*
by *simp*

— Gives the orthogonal of x with respect to the relation Q.
lemma *separation_orthogonal*: $M(x) \Longrightarrow M(Q) \Longrightarrow$ *separation*(M, λa . $\forall\, s \in x$. $\langle s, a\rangle \in Q$)
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 *separation_Transset*: *separation*(*M*,*Transset*)
unfolding *Transset_def*
using *separation_subset lam_replacement_fst lam_replacement_snd*
lam_replacement_hcomp lam_replacement_identity
by(*rule_tac separation_all*,*auto*)

lemma *separation_Ord*: *separation*(*M*,*Ord*)
unfolding *Ord_def*
using *separation_Transset*
separation_comp separation_Transset lam_replacement_snd lam_replacement_identity
by(*rule_tac separation_conj*,*auto*,*rule_tac separation_all*,*auto*)

lemma *ord_iso_separation*: $M(A) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow$
$separation(M,\ \lambda f.\ \forall x{\in}A.\ \forall y{\in}A.\ \langle x,\ y\rangle \in r \longleftrightarrow \langle f\ `\ x,\ f\ `\ y\rangle \in s)$
using
lam_replacement_Pair[*THEN*[*5*] *lam_replacement_hcomp2*]
lam_replacement_apply2[*THEN*[*5*] *lam_replacement_hcomp2*]
lam_replacement_hcomp lam_replacement_fst lam_replacement_snd
lam_replacement_identity lam_replacement_constant
separation_in separation_ball[*of* _ *A*] *separation_iff'*
by *simp*

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 *separation_imp_lam_closed*:
assumes $\forall x{\in}A.\ F(x) \in B$ $separation(M,\ \lambda\langle x,y\rangle.\ F(x) = y)$
$M(A)$ $M(B)$
shows $M(\lambda x{\in}A.\ F(x))$
proof -
from ‹$\forall x{\in}A.\ F(x) \in B$›
have $(\lambda x{\in}A.\ F(x)) \subseteq A{\times}B$
using *times_subset_iff fun_is_rel*[*OF lam_funtype*, *of A F*]
by *auto*
moreover from *this*
have $(\lambda x{\in}A.\ F(x)) = \{\langle x,y\rangle \in A \times B.\ F(x) = y\}$
unfolding *lam_def* **by** *force*
moreover
note ‹$M(A)$› ‹$M(B)$› ‹$separation(M,\ \lambda\langle x,y\rangle.\ F(x) = y)$›
moreover from *this*
have $M(\{\langle x,y\rangle \in A \times B.\ F(x) = y\})$
by *simp*

```
  ultimately
  show ?thesis
    unfolding lam_def
    by auto
qed

lemma curry_closed :
  assumes M(f) M(A) M(B)
  shows M(curry(A,B,f))
  using assms lam_replacement_apply_const_id
    lam_apply_replacement lam_replacement_iff_lam_closed
  unfolding curry_def
  by auto

end — M_replacement

definition
  RepFun_cons :: i⇒i⇒i where
  RepFun_cons(x,y) ≡ RepFun(x, λz. {⟨y,z⟩})

context M_replacement
begin

lemma lam_replacement_RepFun_cons'':
  shows lam_replacement(M, λx . RepFun_cons(fst(x),snd(x)))
  unfolding RepFun_cons_def
  using lam_replacement_fst fst_closed snd_closed lam_replacement_product
lam_replacement_snd lam_replacement_hcomp lam_replacement_sing_fun
   transM[OF _ fst_closed]
lam_replacement_RepFun[of λ x z . {<snd(x),z>} fst]
  by simp

lemma RepFun_cons_replacement: lam_replacement(M, λp. RepFun(fst(p), λx.
{⟨snd(p),x⟩}))
  using lam_replacement_RepFun_cons''
  unfolding RepFun_cons_def
  by simp

lemma lam_replacement_Sigfun:
  assumes lam_replacement(M,f) ∀y[M]. M(f(y))
  shows lam_replacement(M, λx. Sigfun(x,f))
 using assms lam_replacement_Union lam_replacement_sing_fun lam_replacement_Pair
     tag_singleton_closed transM[of _ f(_)] lam_replacement_hcomp[of _ Union]
       lam_replacement_RepFun
  unfolding Sigfun_def
  by simp

lemma phrank_repl:
  assumes
```

$M(f)$
shows
$strong_replacement(M,\ \lambda x\ y.\ y = succ(f`x))$
using *assms lam_replacement_succ lam_replacement_apply*
lam_replacement_imp_strong_replacement lam_replacement_hcomp
by *auto*

lemma *lam_replacement_Powapply_rel*:
assumes $\forall A[M].\ separation(M,\ \lambda y.\ \exists x[M].\ x \in A \wedge y = \langle x,\ Pow_rel(M,x)\rangle)$
$M(f)$
shows
$lam_replacement(M,\ Powapply_rel(M,f))$
using *assms lam_replacement_Pow_rel lam_replacement_apply*[*THEN lam_replacement_hcomp*]
unfolding *Powapply_rel_def*
by *auto*

lemmas *Powapply_rel_replacement* = *lam_replacement_Powapply_rel*[*THEN*
lam_replacement_imp_strong_replacement]

lemma *surj_imp_inj_replacement2*:
$M(f) \Longrightarrow strong_replacement(M,\ \lambda x\ z.\ z = Sigfun(x,\ \lambda y.\ f\ \text{-``}\ \{y\}))$
using *lam_replacement_imp_strong_replacement lam_replacement_Sigfun*
lam_replacement_vimage_sing_fun
by *simp*

lemma *lam_replacement_Pi*: $M(y) \Longrightarrow lam_replacement(M,\ \lambda x.\ \bigcup xa{\in}y.\ \{\langle x,\ xa\rangle\})$
using *lam_replacement_constant lam_replacement_Sigfun*[*unfolded Sigfun_def*,*of* $\lambda x.\ y$]
by (*simp*)

lemma *Pi_replacement2*: $M(y) \Longrightarrow strong_replacement(M,\ \lambda x\ z.\ z = (\bigcup xa{\in}y.\ \{\langle x,\ xa\rangle\}))$
using *lam_replacement_Pi*[*THEN lam_replacement_imp_strong_replacement*, *of y*]
by *simp*

lemma *if_then_Inj_replacement*:
shows $M(A) \Longrightarrow strong_replacement(M,\ \lambda x\ y.\ y = \langle x,\ if\ x \in A\ then\ Inl(x)\ else\ Inr(x)\rangle)$
using *lam_replacement_if lam_replacement_Inl lam_replacement_Inr separation_in_constant*
unfolding *lam_replacement_def*
by *simp*

lemma *lam_if_then_replacement*:
$M(b) \Longrightarrow$
$M(a) \Longrightarrow M(f) \Longrightarrow strong_replacement(M,\ \lambda y\ ya.\ ya = \langle y,\ if\ y = a\ then\ b\ else\ f\ `\ y\rangle)$

using *lam_replacement_if lam_replacement_apply lam_replacement_constant separation_equal*
unfolding *lam_replacement_def*
by *simp*

lemma *if_then_replacement*:
$M(A) \Longrightarrow M(f) \Longrightarrow M(g) \Longrightarrow$ *strong_replacement*$(M, \lambda x\ y.\ y = \langle x,$ *if* $x \in A$ *then* f ‘ x *else* g ‘ $x\rangle)$
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*:
$M(f) \Longrightarrow$
$M(b) \Longrightarrow$ *strong_replacement*$(M, \lambda x\ y.\ y = \langle x,$ *if* $x \in$ *range*(f) *then converse*(f) ‘ x *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*:
$M(A) \Longrightarrow M(C) \Longrightarrow$ *strong_replacement*$(M, \lambda x\ y.\ y = \langle x,$ *if* $x =$ *Inl*(A) *then* C *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*:
$M(u) \Longrightarrow$
$M(f) \Longrightarrow$
strong_replacement
$(M,$
$\lambda z\ y.\ y = \langle z,$ *if* $z = u$ *then* f ‘ *0 else if* $z \in$ *range*(f) *then* f ‘ *succ*(*converse*(f) ‘ z) *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* $\lambda x.$ *converse*(f)‘x *succ*] *lam_replacement_hcomp*[*of* $\lambda x.$ *succ*(*converse*(f)‘x) λx . f‘x]
unfolding *lam_replacement_def*
by *simp*

lemma *Inl_replacement2*:
$M(A) \Longrightarrow$
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*:
$M(f) \Longrightarrow M(g) \Longrightarrow$ *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*

locale *M_Pi_replacement* = *M_Pi* + *M_replacement*

begin
lemma *curry_rel_exp* :
assumes $M(f)\ M(A)\ M(B)\ M(C)\ f \in A \times B \to C$
shows $curry(A,B,f) \in A \to^M (B \to^M C)$

using *assms transM*[*of* _ *A*] *lam_closed*[*OF apply_replacement2*]
lam_replacement_apply_const_id
lam_apply_replacement lam_replacement_iff_lam_closed
unfolding *curry_def*
by (*intro lam_type mem_function_space_rel_abs*[*THEN iffD2*],*auto*)

end — *M_Pi_replacement*

— 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*
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* _ ‹$M(A)$›],*rule_tac separation_conj*,*simp_all*)

lemma *separation_is_insnd_restrict_eq_dom* : $separation(M,\ \lambda p.\ (\exists\, r{\in}A.\ restrict(r,\ B) = fst(p) \wedge snd(p) = domain(r)))$
if $M(B)\ M(D)\ M(A)$ **for** $A\ B\ D$
using *that lam_replacement_fst lam_replacement_snd*
separation_eq lam_replacement_hcomp lam_replacement_domain
lam_replacement_hcomp[*OF* _ *lam_replacement_restrict'*]
by(*rule_tac separation_bex*[*OF* _ ‹$M(A)$›],*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*[*OF lam_replacement_Collect'*[*OF_ separation_is_insnd_restrict_eq_dom*]]
assms drSR_Y_equality 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* ‹$M(D)$›]
assms drSR_Y_equality separation_is_insnd_restrict_eq_dom separation_restrict_eq_dom_eq
by *simp*

lemma *lam_if_then_apply_replacement*: $M(f) \Longrightarrow M(v) \Longrightarrow M(u) \Longrightarrow$
lam_replacement(M, λx. *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*: $M(f) \Longrightarrow M(m) \Longrightarrow M(y) \Longrightarrow$
lam_replacement(M, λz . *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*: $M(A) \Longrightarrow M(f) \Longrightarrow$
lam_replacement(M, λx . *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*: $M(G) \Longrightarrow$ *lam_replacement*(M, λx. *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, λp. *domain*(*snd*(p)) = *fst*(p)) **if** $M(A)$ **for** A
using *separation_eq that*
lam_replacement_hcomp lam_replacement_fst lam_replacement_snd lam_replacement_domain
by *simp_all*
then

show *?thesis*
unfolding *dC_F_def*
using *assms lam_replacement_Collect'[of A λ d x . 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 λ d x . domain(x) = d]*
separation_eq[OF _ lam_replacement_domain _ lam_replacement_constant]
by *simp*

lemma *lam_replacement_Collect_ball_Pair*:
assumes $M(G)$ $M(Q)$
shows *lam_replacement*$(M, \lambda x . \{a \in G . \forall s{\in}x. \langle s, a\rangle \in Q\})$
proof -
have *separation*$(M, \lambda p. (\forall s{\in}fst(p). \langle s, snd(p)\rangle \in Q))$
using *separation_in assms lam_replacement_identity*
lam_replacement_constant lam_replacement_hcomp[OF lam_replacement_fst
lam_replacement_hcomp[OF lam_replacement_fst lam_replacement_snd]]
lam_replacement_snd lam_replacement_fst
lam_replacement_hcomp[OF lam_replacement_fst lam_replacement_fst]
by(*rule_tac separation_all,simp_all,rule_tac separation_in,simp_all,rule_tac lam_replacement_product[of snd λx . snd(fst(fst(x)))],simp_all*)
then
show *?thesis*
using *assms lam_replacement_Collect'*
by *simp_all*
qed

lemma *surj_imp_inj_replacement3*:
assumes $M(G)$ $M(Q)$ $M(x)$
shows *strong_replacement*$(M, \lambda y\ z.\ y \in \{a \in G . \forall s{\in}x. \langle s, a\rangle \in Q\} \wedge z = \{\langle x, y\rangle\})$
proof -
from *assms*
have *separation*$(M, \lambda z. \forall s{\in}x. \langle s, z\rangle \in Q)$
using *lam_replacement_swap lam_replacement_constant separation_in separation_ball*
by *simp*
with *assms*
show *?thesis*
using *separation_in_constant lam_replacement_sing_const_id[of x] separation_conj*
by(*rule_tac strong_replacement_separation,simp_all*)
qed

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

lemma *zermelo_separation*: $M(Q) \Longrightarrow M(f) \Longrightarrow$ *separation*$(M, \lambda X.\ Q \cup f\ ``\ X \subseteq X)$
using *lam_replacement_identity lam_replacement_Un*[*THEN* [*5*] *lam_replacement_hcomp2*]
lam_replacement_constant lam_replacement_Image[*THEN* [*5*] *lam_replacement_hcomp2*]
separation_subset
by *simp*

end — *M_replacement*

18.2 Some basic replacement lemmas

lemma (**in** *M_trans*) *strong_replacement_conj*:
assumes $\bigwedge A.\ M(A) \Longrightarrow$ *univalent*(M,A,P) *strong_replacement*(M,P)
separation$(M, \lambda x.\ \exists\, b[M].\ Q(x,b) \wedge P(x,b))$
shows *strong_replacement*$(M, \lambda x\ z.\ Q(x,z) \wedge P(x,z))$
proof -
{
fix A
assume $M(A)$
with ‹*separation*$(M, \lambda x.\ \exists\, b[M].\ Q(x,b) \wedge P(x,b))$›
have $M(\{x{\in}A.\ \exists\, b[M].\ Q(x,b) \wedge P(x,b)\})$
by *simp*
with ‹$M(_) \Longrightarrow$ *univalent*$(M,\{x{\in}A.\ \exists\, b[M].\ Q(x,b) \wedge P(x,b)\}, P)$› ‹*strong_replacement*$(M, P)$›
have $\exists\, Y[M].\ \forall\, b[M].\ b \in Y \longleftrightarrow (\exists\, x[M].\ x \in \{x{\in}A.\ \exists\, b[M].\ Q(x,b) \wedge P(x,b)\} \wedge P(x,\ b))$
unfolding *strong_replacement_def* **by** *blast*
then
obtain Y **where** $\forall\, b[M].\ b \in Y \longleftrightarrow (\exists\, x[M].\ x \in \{x{\in}A.\ \exists\, b[M].\ Q(x,b) \wedge P(x,b)\} \wedge P(x,\ b))\ M(Y)$
by *blast*
with ‹$M(A)$› ‹$M(A) \Longrightarrow$ *univalent*(M,A,P)›
have $\forall\, b[M].\ b \in Y \longleftrightarrow (\exists\, x[M].\ x \in A \wedge Q(x,\ b) \wedge P(x,\ b))$
unfolding *univalent_def* **by** *auto*
with ‹$M(Y)$›

have $\exists\, Y[M].\ \forall\, b[M].\ b \in Y \longleftrightarrow (\exists\, x[M].\ x \in A \wedge Q(x,\ b) \wedge P(x,\ b))$
by *auto*
}
then
show *?thesis*
unfolding *strong_replacement_def* **by** *simp*
qed

lemma *strong_replacement_iff_bounded_M*:
strong_replacement$(M,P) \longleftrightarrow$ *strong_replacement*$(M,\lambda\ x\ z\ .\ M(z) \wedge M(x) \wedge P(x,z))$
unfolding *strong_replacement_def* **by** *auto*

end

19 Basic relativization of cardinality

theory *Discipline_Cardinal*
imports
Discipline_Function
begin

relativize functional *cardinal cardinal_rel* **external**
relationalize *cardinal_rel is_cardinal*
synthesize *is_cardinal* **from_definition assuming** *nonempty*

notation *is_cardinal_fm* (‹*cardinal′*(_′) *is* _›)

abbreviation
cardinal_r :: $[i,i\Rightarrow o] \Rightarrow i$ (‹|_|⁻›) **where**
$|x|^M \equiv$ *cardinal_rel*(M,x)

abbreviation
cardinal_r_set :: $[i,i]\Rightarrow i$ (‹|_|⁻›) **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 — *M_trivial*

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 — *M_Perm*

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* (‹·_ < _·›)
arity_theorem intermediate for *lt_rel_fm*

lemma *arity_lt_rel_fm*[*arity*]: $a \in nat \Longrightarrow b \in nat \Longrightarrow arity(lt_rel_fm(a, b)) = succ(a) \cup succ(b)$
  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* (‹·*Card'*(_')·›)
arity_theorem for *is_Card_fm*

notation *Card_rel* (‹*Card–'*(_')›)

lemma (**in** *M_Perm*) *is_Card_iff*: $M(A) \Longrightarrow is_Card(M, A) \longleftrightarrow Card^M(A)$
  using *is_cardinal_iff*
  unfolding *is_Card_def Card_rel_def* **by** *simp*

abbreviation
  Card_r_set :: $[i,i] \Rightarrow o$ (‹*Card–'*(_')›) **where**
  $Card^M(i) \equiv Card_rel(\#\#M,i)$

relativize functional *InfCard InfCard_rel* **external**
relationalize *InfCard_rel is_InfCard*
synthesize *is_InfCard* **from_definition assuming** *nonempty*
notation *is_InfCard_fm* (‹·*InfCard'*(_')·›)
arity_theorem for *is_InfCard_fm*

notation *InfCard_rel* (‹*InfCard–'*(_')›)

abbreviation
  InfCard_r_set :: $[i,i] \Rightarrow o$ (‹*InfCard–'*(_')›) **where**
  $InfCard^M(i) \equiv InfCard_rel(\#\#M,i)$

19.1 Disicpline for ($\oplus$)

relativize functional *cadd cadd_rel* **external**

abbreviation
  cadd_r :: $[i,i \Rightarrow o,i] \Rightarrow i$ (‹_ $\oplus$– _› [*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 — *M_basic*

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 — *M_Perm*

19.2 Disicpline for ($\otimes$)

relativize functional *cmult cmult_rel* **external**

abbreviation
  cmult_r :: [*i*,*i*⇒*o*,*i*] ⇒ *i* (‹_ ⊗– _› [*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 — *M_Perm*

end

20 Relativization of the cumulative hierarchy

theory *Univ_Relative*
imports
ZF-Constructible.Rank
ZF.Univ
Discipline_Cardinal

begin

declare *arity_ordinal_fm*[*arity*]

context *M_trivial*
begin
declare *powerset_abs*[*simp*]

lemma *family_union_closed*: $[\![$*strong_replacement*$(M,\ \lambda x\ y.\ y = f(x));\ M(A);$ $\forall x{\in}A.\ M(f(x))]\!]$
$\Longrightarrow M(\bigcup x{\in}A.\ f(x))$
using *RepFun_closed* **..**

lemma *family_union_closed'*: $[\![$*strong_replacement*$(M,\ \lambda x\ y.\ x{\in}A \wedge y = f(x));$ $M(A);\ \forall x{\in}A.\ M(f(x))]\!]$
$\Longrightarrow M(\bigcup x{\in}A.\ f(x))$
using *RepFun_closed2*
by *simp*

end — *M_trivial*

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

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*:
$A \in nat \Longrightarrow$
$x \in nat \Longrightarrow$
$f \in nat \Longrightarrow$
$d \in nat \Longrightarrow$
$arity(is_HVfrom_fm(A,\ x,\ f,\ d)) = succ(A) \cup succ(d) \cup (succ(x) \cup succ(f))$
using *arity_is_HVfrom_fm' arity_is_Powapply_fm*
by(*simp*,*subst arity_Replace_fm*[*of is_Powapply_fm*(*succ*(*succ*(*succ*(*succ*(*f*)))),

0, *1*) *succ*(*succ*(*x*)) *1*])
(*simp_all*, *auto simp add*:*arity pred_Un_distrib*)

notation *HVfrom_rel* (‹*HVfrom*–'(_,_,_')›)

locale *M_HVfrom* = *M_eclose* +
assumes
Powapply_replacement:
$M(f) \Longrightarrow$ *strong_replacement*(M,$\lambda y\ z.\ z = Powapply^M(f,y)$)
begin

is_iff_rel for *HVfrom*
proof -
assume *assms*:$M(A)$ $M(x)$ $M(f)$ $M(res)$
then
have *is_Replace*(M,x,$\lambda y\ z.\ z = Powapply^M(f,y)$,$r$) $\longleftrightarrow$ $r = \{z\ .\ y{\in}x,\ z = Powapply^M(f,y)\}$
if $M(r)$ **for** r
using *that Powapply_rel_closed*
Replace_abs[*of x r* $\lambda y\ z.\ z = Powapply^M(f,y)$] *transM*[*of _ x*]
by *simp*
moreover
have *is_Replace*(M,x,*is_Powapply*(M,f),r) $\longleftrightarrow$
is_Replace(M,x,$\lambda y\ z.\ z = Powapply^M(f,y)$,$r$) **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) $\longleftrightarrow$ $r = \{z\ .\ y{\in}x,\ z = Powapply^M(f,y)\}$
if $M(r)$ **for** r
using *that assms*
by *simp*
moreover
have $M(\{z\ .\ y \in x,\ z = Powapply^M(f,y)\})$
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 $\{z\ .\ y \in x,\ z = Powapply^M(f,y)\} = \{y\ .\ w \in x,\ M(y) \wedge M(w) \wedge y = Powapply^M(f,w)\}$
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 $M(\{z\ .\ y \in x,\ z = Powapply^M(f,y)\})$
    using *assms strong_replacement_closed*[*OF Powapply_replacement*]
      Powapply_rel_closed transM[*of _ x*]
    by *simp*
  then
  have $M(A \cup \bigcup(\{z\ .\ y{\in}x,\ z = Powapply^M(f,y)\}))$
    using *assms*
    by *simp*
  moreover from *assms*
    — intermediate step for body of Replace
  have $\{z\ .\ y \in x,\ z = Powapply^M(f,y)\} = \{y\ .\ w \in x,\ M(y) \wedge M(w) \wedge y = Powapply^M(f,w)\}$
    by (*auto dest*:*transM*)
  ultimately
  show *?thesis*
    using *assms*
    unfolding *HVfrom_rel_def*
    by *simp*
qed

end — *M_HVfrom*

definition
  Vfrom_rel :: $[i{\Rightarrow}o,i,i] \Rightarrow i$ (‹*Vfrom*–′(_,_′)›) **where**
  $Vfrom^M(A,i) = transrec(i,\ HVfrom_rel(M,A))$

definition
  is_Vfrom :: $[i{\Rightarrow}o,i,i,i] \Rightarrow o$ **where**
  $is_Vfrom(M,A,i,z) \equiv is_transrec(M,is_HVfrom(M,A),i,z)$

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

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

relativize functional *Hrank Hrank_rel*
relationalize *Hrank_rel is_Hrank*
synthesize *is_Hrank* **from_definition assuming** *nonempty*

lemma *arity_is_Hrank_fm* : $x \in nat \Longrightarrow$
  $f \in nat \Longrightarrow$
  $d \in nat \Longrightarrow$
  $arity(is_Hrank_fm(x,\ f,\ d)) =$

```
    succ(d) ∪ succ(x) ∪ succ(f)
  unfolding is_Hrank_fm_def
  using  arity_fun_apply_fm arity_big_union_fm
    arity_fun_apply_fm arity_succ_fm arity_And arity_Exists
    arity_Replace_fm[of
      (·∃··succ(0) is 2· ∧ · succ(succ(succ(succ(f))))‘1 is 0···)
      succ(x) 0 4+ω f]
  by(simp_all add:Un_assoc pred_Un,simp add:ord_simp_union)

locale M_Vfrom = M_HVfrom +
  assumes
   trepl_HVfrom : ⟦ M(A);M(i) ⟧ ⟹ transrec_replacement(M,is_HVfrom(M,A),i)
    and
    Hrank_replacement : M(f) ⟹ strong_replacement(M,λx y . y = succ(f‘x))
    and
   is_Hrank_replacement : M(x) ⟹ wfrec_replacement(M,is_Hrank(M),rrank(x))
    and
    HVfrom_replacement : ⟦ M(i) ; M(A) ⟧ ⟹
                          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) ⟷ 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) ⟹ relation2(M,is_HVfrom(M,A),HVfrom_rel(M,A))
  using is_HVfrom_iff
  unfolding relation2_def
  by auto

lemma HVfrom_closed :
  M(A) ⟹ ∀ x[M]. ∀ g[M]. function(g) ⟶ 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 $Ord(i) \Longrightarrow M(i) \Longrightarrow \{x{\in}Vfrom(A,i).\ M(x)\} = transrec(i,HVfrom_rel(M,A))$
proof (*induct rule*:*trans_induct*)
have *eq*:$(\bigcup x{\in}i.\ \{x \in Pow(transrec(x,\ HVfrom_rel(M,\ A)))\ .\ M(x)\}) = \bigcup\{y\ .\ x \in i,\ y = Pow^M(transrec(x,\ HVfrom_rel(M,\ A)))\}$
if $Ord(i)$ $M(i)$ **for** i
using *assms Pow_rel_char*[*OF Vfrom_rel_closed*[*OF* ‹$M(A)$› *transM*[*of _ i*]]] *Ord_in_Ord′ that*
by *auto*
case (*step i*)
then
have *0*: $M(Pow^M(transrec(x,\ HVfrom_rel(M,\ A))))$ **if** $x{\in}i$ **for** x
using *assms that transM*[*of _ i*] *Ord_in_Ord*
transrec_closed[*of is_HVfrom*(M,A) *_ HVfrom_rel*(M,A)] *trepl_HVfrom relation2_HVfrom*
by *auto*
have $Vfrom(A,i) = A \cup (\bigcup y{\in}i.\ Pow((\lambda x{\in}i.\ Vfrom(A,\ x))\ `\ y))$
using *def_transrec*[*OF Vfrom_def*, *of A i*] **by** *simp*
then
have $Vfrom(A,i) = A \cup (\bigcup y{\in}i.\ Pow(Vfrom(A,\ y)))$
by *simp*
then
have $\{x{\in}Vfrom(A,i).\ M(x)\} = \{x{\in}A.\ M(x)\} \cup (\bigcup y{\in}i.\ \{x{\in}Pow(Vfrom(A,\ y)).\ M(x)\})$
by *auto*
with ‹$M(A)$›
have $\{x{\in}Vfrom(A,i).\ M(x)\} = A \cup (\bigcup y{\in}i.\ \{x{\in}Pow(Vfrom(A,\ y)).\ M(x)\})$
by (*auto intro*:*transM*)
also
have $... = A \cup (\bigcup y{\in}i.\ \{x{\in}Pow(\{z{\in}Vfrom(A,y).\ M(z)\}).\ M(x)\})$
proof -
have $\{x{\in}Pow(Vfrom(A,\ y)).\ M(x)\} = \{x{\in}Pow(\{z{\in}Vfrom(A,y).\ M(z)\}).\ M(x)\}$
if $y{\in}i$ **for** y **by** (*auto intro*:*transM*)
then
show *?thesis* **by** *simp*
qed
also from *step*
have $... = A \cup (\bigcup y{\in}i.\ \{x{\in}Pow(transrec(y,\ HVfrom_rel(M,\ A))).\ M(x)\})$
using *transM*[*of _ i*]
by *auto*
also
have $... = transrec(i,\ HVfrom_rel(M,\ A))$

using *0 step assms transM*[*of* _ *i*] *eq*
def_transrec[*of* λy. *transrec*(y, *HVfrom_rel*(M, A)) *HVfrom_rel*(M, A) *i*]
unfolding *Powapply_rel_def HVfrom_rel_def*
by *auto*
finally
show *?case* **.**
qed

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

20.1 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 $\{a\ .\ y \in x,\ M(a) \wedge M(y) \wedge a = succ(f\ `\ y)\} = \{a\ .\ y \in x,\ \ a = succ(f\ `\ y)\}$
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 $\{a\ .\ y \in x,\ M(a) \wedge M(y) \wedge a = succ(f\ `\ y)\} = \{a\ .\ y \in x,\ \ a = succ(f\ `\ y)\}$
    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* :
  $\forall x[M].\ \forall g[M].\ function(g) \longrightarrow M(Hrank(x,g))$
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* : $M(x) \Longrightarrow wf(rrank(x))$
  unfolding *rrank_def* **using** *wf_trancl*[*OF wf_Memrel*] **.**

lemma *trans_rrank* : $M(x) \Longrightarrow trans(rrank(x))$
  unfolding *rrank_def* **using** *trans_trancl* **.**

lemma *relation_rrank* : $M(x) \Longrightarrow relation(rrank(x))$
  unfolding *rrank_def* **using** *relation_trancl* **.**

lemma *rrank_in_M* : $M(x) \Longrightarrow M(rrank(x))$
  unfolding *rrank_def* **by** *simp*

end — *M_eclose*

lemma *Hrank_trancl*:*Hrank*(*y*, *restrict*(*f*,*Memrel*(*eclose*({*x*}))-``{*y*}))
= *Hrank*(*y*, *restrict*(*f*,(*Memrel*(*eclose*({*x*}))^+)-``{*y*}))
unfolding *Hrank_def*
using *restrict_trans_eq* **by** *simp*

lemma *rank_trancl*: *rank*(*x*) = *wfrec*(*rrank*(*x*), *x*, *Hrank*)
proof -
have *rank*(*x*) = *wfrec*(*Memrel*(*eclose*({*x*})), *x*, *Hrank*)
(**is** _ = *wfrec*(*?r*,_,_))
unfolding *rank_def transrec_def Hrank_def* **by** *simp*
also
have ... = *wftrec*(*?r*^+, *x*, λ*y f*. *Hrank*(*y*, *restrict*(*f*,*?r*-``{*y*})))
unfolding *wfrec_def* **..**
also
have ... = *wftrec*(*?r*^+, *x*, λ*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*] ⇒ *i* **where**
Vset'(*A*) ≡ *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∈*nat v*∈*nat env*∈*list*(*A*) *0*∈*A*
i < *length*(*env*) *v* < *length*(*env*)
shows
is_Vset(##*A*,*nth*(*i*, *env*),*nth*(*v*, *env*)) ⟷ *sats*(*A*,*?ivs_fm*(*i*,*v*),*env*)
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*: $[\![M(i); M(V); Ord(i)]\!] \Longrightarrow is_Vset(M,i,V) \longleftrightarrow V = \{x \in Vset(i).$
$M(x)\}$
  using *Vfrom_abs* **unfolding** *is_Vset_def* **by** *simp*

lemma *Vset_closed*: $[\![M(i); Ord(i)]\!] \Longrightarrow M(\{x \in Vset(i).\ M(x)\})$
  using *Vfrom_closed* **unfolding** *is_Vset_def* **by** *simp*

lemma *rank_closed*: $M(a) \Longrightarrow M(rank(a))$
  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 $\exists\, i[M].\ \exists\, V[M].\ ordinal(M,i) \wedge is_Vset(M,i,V) \wedge a \in V$
proof -
  let *?i*=*succ*(*rank*(*a*))
  from *assms*
  have $a \in \{x \in Vfrom(0,?i).\ M(x)\}$ (**is** $a \in ?V$)
    using *Vset_Ord_rank_iff* **by** *simp*
  moreover from *assms*
  have $M(?i)$
    using *rank_closed* **by** *simp*
  moreover
  note ‹$M(a)$›
  moreover from *calculation*
  have $M(?V)$
    using *Vfrom_closed* **by** *simp*
  moreover from *calculation*
  have $ordinal(M,?i) \wedge is_Vfrom(M,\ 0,\ ?i,\ ?V) \wedge a \in ?V$
    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

21 Relative, Choice-less Cardinal Numbers

theory *Cardinal_Relative*
imports
Lambda_Replacement
Univ_Relative
begin

The following command avoids that a commonly used one-letter variable be captured by the definition of the constructible universe L.

hide_const (**open**) L

We also return to the old notation for (+) to preserve the old Constructibility code.

no_notation *oadd* (**infixl** ‹+› *65*)
notation *sum* (**infixr** ‹+› *65*)

definition
Finite_rel :: $[i \Rightarrow o, i] => o$ **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**
$banach_functor(X,Y,f,g,W) \equiv X - g``(Y - f``W)$

lemma *banach_functor_subset*: $banach_functor(X,Y,f,g,W) \subseteq X$
unfolding *banach_functor_def* **by** *auto*

definition
is_banach_functor :: $[i \Rightarrow o, i,i,i,i,i,i] \Rightarrow o$ **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 $\forall\, W[M].\ M(banach_functor(X,Y,f,g,W))$
unfolding *banach_functor_def* **using** *assms image_closed*
by *simp*

context *M_trancl*

begin

lemma *iterates_banach_functor_closed*:
  assumes $n\in\omega$ $M(X)$ $M(Y)$ $M(f)$ $M(g)$
  shows $M(banach_functor(X,Y,f,g)\hat{}n(0))$
  using *assms banach_functor_closed*
  by (*induct*) *simp_all*

lemma *banach_repl_iter'*:
  assumes
    $\bigwedge A.\ M(A) \Longrightarrow separation(M,\ \lambda b.\ \exists x\in A.\ x \in \omega \wedge b = banach_functor(X,\ Y,\ f,\ g)\hat{}x\ (0))$
    $M(X)$ $M(Y)$ $M(f)$ $M(g)$
  shows
    $strong_replacement(M,\ \lambda x\ y.\ x\in nat \wedge y = banach_functor(X,\ Y,\ f,\ g)\hat{}x\ (0))$
  unfolding *strong_replacement_def*
proof *clarsimp*
  fix A
  let $?Z=\{b \in Pow^M(X)\ .\ (\exists x\in A.\ x \in \omega \wedge b = banach_functor(X,\ Y,\ f,\ g)\hat{}x\ (0))\}$
  assume $M(A)$
  moreover
  note *assms*
  moreover from *calculation*
  have $M(?Z)$ **by** *simp*
  moreover from *assms*(*2-*)
  have $b \in ?Z \longleftrightarrow (\exists x\in A.\ x \in \omega \wedge b = banach_functor(X,\ Y,\ f,\ g)\hat{}x\ (0))$ **for** b
    by (*auto*, *rename_tac x*, *induct_tac x*, (*auto simp*:*def_Pow_rel*)[*1*],
      rule_tac def_Pow_rel[*THEN iffD2*, *OF iterates_banach_functor_closed*[*of _ X Y f g*]])
      (*simp_all add*:*banach_functor_subset*)
  ultimately
  show $\exists Z[M].\ \forall b[M].\ b \in Z \longleftrightarrow (\exists x\in A.\ x \in \omega \wedge b = banach_functor(X,\ Y,\ f,\ g)\hat{}x\ (0))$
    by (*intro rexI*[*of _ ?Z*]) *simp*
qed

end — *M_trancl*

context *M_Perm*
begin

lemma *mem_Pow_rel*: $M(r) \Longrightarrow a \in Pow_rel(M,r) \Longrightarrow a \in Pow(r) \wedge M(a)$
  using *Pow_rel_char* **by** *simp*

lemma *mem_bij_abs*: $[\![M(f);M(A);M(B)]\!] \Longrightarrow\ f \in bij^M(A,B) \longleftrightarrow f\in bij(A,B)$
  using *bij_rel_char* **by** *simp*

lemma *mem_inj_abs*: $[\![M(f);M(A);M(B)]\!] \Longrightarrow\ f \in inj^M(A,B) \longleftrightarrow f\in inj(A,B)$

using *inj_rel_char* **by** *simp*

lemma *mem_surj_abs*: $[\![M(f);M(A);M(B)]\!] \Longrightarrow f \in surj^M(A,B) \longleftrightarrow f \in surj(A,B)$
using *surj_rel_char* **by** *simp*

end — *M_Perm*

locale *M_cardinals* = *M_ordertype* + *M_trancl* + *M_Perm* + *M_replacement*
+
assumes
radd_separation: $M(R) \Longrightarrow M(S) \Longrightarrow$
$separation(M, \lambda z.$
$(\exists x\ y.\ z = \langle Inl(x), Inr(y)\rangle) \vee$
$(\exists x'\ x.\ z = \langle Inl(x'), Inl(x)\rangle \wedge \langle x', x\rangle \in R) \vee$
$(\exists y'\ y.\ z = \langle Inr(y'), Inr(y)\rangle \wedge \langle y', y\rangle \in S))$
and
rmult_separation: $M(b) \Longrightarrow M(d) \Longrightarrow separation(M,$
$\lambda z.\ \exists x'\ y'\ x\ y.\ z = \langle\langle x', y'\rangle, x, y\rangle \wedge (\langle x', x\rangle \in b \vee x' = x \wedge \langle y', y\rangle \in d))$

begin

lemma *rvimage_separation*: $M(f) \Longrightarrow M(r) \Longrightarrow$
$separation(M, \lambda z.\ \exists x\ y.\ z = \langle x, y\rangle \wedge \langle f\ `\ x, f\ `\ y\rangle \in r)$
using *separation_Pair separation_in lam_replacement_product*
lam_replacement_fst lam_replacement_snd lam_replacement_constant
lam_replacement_apply2[*THEN* [*5*] *lam_replacement_hcomp2*]
by *simp*

lemma *radd_closed*[*intro*,*simp*]: $M(a) \Longrightarrow M(b) \Longrightarrow M(c) \Longrightarrow M(d) \Longrightarrow M(radd(a,b,c,d))$
using *radd_separation* **by** (*auto simp add*: *radd_def*)

lemma *rmult_closed*[*intro*,*simp*]: $M(a) \Longrightarrow M(b) \Longrightarrow M(c) \Longrightarrow M(d) \Longrightarrow M(rmult(a,b,c,d))$
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 $is_cardinal(M,A,\kappa) \longleftrightarrow \kappa = (\mu\ i.\ M(i) \wedge i \approx^M A)$
using *is_cardinal_iff assms*
unfolding *cardinal_rel_def* **by** *simp*

21.1 The Schroeder-Bernstein Theorem

See Davey and Priestly, page 106

context *M_cardinals*
begin

21.2 Banach's Decomposition Theorem

```
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 ∈ inj(Y,X) A≠0 ∀ a∈A. a ⊆ Y
  shows g``(⋂ A) = (⋂ a∈A. g``a)
proof (intro equalityI subsetI)
  fix x
  from assms
  obtain a where a∈A by blast
  moreover
  assume x ∈ (⋂ a∈A. g `` a)
  ultimately
  have x_in_im: x ∈ g``y if y∈A for y
    using that by auto
  have exists: ∃ z ∈ y. x = g`z if y∈A for y
  proof -
    note that
    moreover from this and x_in_im
    have x ∈ g``y by simp
    moreover from calculation
    have x ∈ g``y by simp
    moreover
    note assms
    ultimately
    show ?thesis
      using image_fun[OF inj_is_fun] by auto
  qed
  with ‹a∈A›
  obtain z where z ∈ a x = g`z by auto
  moreover
  have z ∈ y if y∈A for y
  proof -
    from that and exists
    obtain w where w ∈ y x = g`w by auto
    moreover from this ‹x = g`z› assms that ‹a∈A› ‹z∈a›
    have z = w unfolding inj_def by blast
    ultimately
    show ?thesis by simp
  qed
  moreover
  note assms
  moreover from calculation
  have z ∈ ⋂ A by auto
  moreover from calculation
  have z ∈ Y by blast
  ultimately
```

```
  show x ∈ g `` (⋂ A)
    using inj_is_fun[THEN funcI, of g] by fast
qed auto

lemma contin_banach_functor:
  assumes g ∈ inj(Y,X)
  shows contin(banach_functor(X,Y,f,g))
  unfolding contin_def
proof (intro allI impI)
  fix A
  assume directed(A)
  then
  have A ≠ 0
    unfolding directed_def ..
  have banach_functor(X, Y, f, g, ⋃ A) = X - g``(Y - f``(⋃ A))
    unfolding banach_functor_def ..
  also
  have ... = X - g``(Y - (⋃ a∈A. f``a))
    by auto
  also from ‹A≠0›
  have ... = X - g``(⋂ a∈A. Y-f``a)
    by auto
  also from ‹A≠0› and assms
  have ... = X - (⋂ a∈A. g``(Y-f``a))
    using inj_Inter[of g Y X {Y-f``a. a∈A} ] by fastforce
  also from ‹A≠0›
  have ... = (⋃ a∈A. X - g``(Y-f``a)) by simp
  also
  have ... = (⋃ a∈A. banach_functor(X, Y, f, g, a))
    unfolding banach_functor_def ..
  finally
  show banach_functor(X,Y,f,g,⋃ A) = (⋃ a∈A. banach_functor(X,Y,f,g,a)) .
qed

lemma lfp_banach_functor:
  assumes g∈inj(Y,X)
  shows lfp(X, banach_functor(X,Y,f,g)) =
      (⋃ n∈nat. 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∈inj(Y,X)
    and banach_repl_iter: M(X) ⟹ M(Y) ⟹ M(f) ⟹ M(g) ⟹
              strong_replacement(M, λx y. x∈nat ∧ y = banach_functor(X, Y, f,
g)^x (0))
  shows M(lfp(X, banach_functor(X,Y,f,g)))
proof -
  from assms
```

```
  have M(banach_functor(X,Y,f,g)^n (0)) if n∈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:
  assumes
    banach_repl_iter: M(f) ⟹ M(g) ⟹ M(X) ⟹ M(Y) ⟹
              strong_replacement(M, λx y. x∈nat ∧ y = banach_functor(X, Y, f,
g)^x (0))
  shows
    [| M(f); M(g); M(X); M(Y); f ∈ X->Y;  g ∈ inj(Y,X) |] ==>
      ∃XA[M]. ∃XB[M]. ∃YA[M]. ∃YB[M].
         (XA ∩ XB = 0) & (XA ∪ XB = X) &
         (YA ∩ YB = 0) & (YA ∪ 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] assms
  unfolding banach_functor_def by simp_all

lemma schroeder_bernstein_closed:
  assumes
    banach_repl_iter: M(X) ⟹ M(Y) ⟹ M(f) ⟹ M(g) ⟹
              strong_replacement(M, λx y. x∈nat ∧ y = banach_functor(X, Y, f,
g)^x (0))
  shows
  [| M(f); M(g); M(X); M(Y); f ∈ inj(X,Y);  g ∈ inj(Y,X) |] ==> ∃h[M]. h ∈
bij(X,Y)
  apply (insert banach_decomposition_rel [of f g X Y, OF banach_repl_iter])
  apply (simp add: inj_is_fun)
  apply (auto)
  apply (rule_tac x=restrict(f,XA) ∪ converse(restrict(g,YB)) in rexI)
   apply (auto intro!: restrict_bij bij_disjoint_Un intro: bij_converse_bij)
  done
```

The previous lemmas finish our original, direct relativization of the material involving the iterative proof (as appearing in *ZF.Cardinal*) of the Schröder-Bernstein theorem. Next, we formalize Zermelo's proof that replaces the recursive construction by a fixed point represented as an intersection [2, Exr. x4.27]. This allows to avoid at least one replacement assumption.

```
lemma dedekind_zermelo:
```

assumes
$A' \subseteq B$ $B \subseteq A$ $A \approx^M A'$
and *types*: $M(A')$ $M(B)$ $M(A)$
shows
$A \approx^M B$
proof -
from ‹$A \approx^M A'$› **and** *types*
obtain f **where** $f \in bij^M(A,A')$
unfolding *eqpoll_rel_def* **by** *blast*
let $?Q=B$ - $f``A$
from ‹$f \in _$› ‹$A' \subseteq B$› *types*
have $f``X \subseteq B$ **for** X
using *mem_bij_rel*[*THEN bij_is_fun*, *THEN Image_sub_codomain*, *of f A A'*] **by** *auto*
moreover
note ‹$B \subseteq A$›
moreover from *calculation*
have $f``X \subseteq A$ **for** X **by** *auto*
moreover
define $\mathcal{T}$ **where** $\mathcal{T} \equiv \{ X \in Pow^M(A) . ?Q \cup f `` X \subseteq X\}$
moreover from *calculation*
have $A \in \mathcal{T}$ **using** *def_Pow_rel* **by** (*auto simp*:*types*)
ultimately
have $\mathcal{T} \neq 0$ **unfolding** $\mathcal{T}$*_def* **by** *auto*
let $?T=\bigcap \mathcal{T}$
have $?T \subseteq A$
proof
fix x
assume $x \in ?T$
with ‹$\mathcal{T} \neq 0$›
obtain X **where** $x \in X$ $X \in \mathcal{T}$
by *blast*
moreover from *this*
have $X \in Pow^M(A)$
unfolding $\mathcal{T}$*_def* **by** *simp*
moreover from *calculation* **and** ‹$M(A)$›
have $M(x)$ **using** *Pow_rel_char* **by** (*auto dest*:*transM*)
ultimately
show $x \in A$ **using** *Pow_rel_char* **by** (*auto simp*:*types*)
qed
moreover from ‹$\mathcal{T} \neq 0$›
have $?Q \cup f``?T \subseteq ?T$
using *image_mono* **unfolding** $\mathcal{T}$*_def* **by** *blast*
moreover from ‹$f \in _$›
have $M(\mathcal{T})$
using *zermelo_separation* **unfolding** $\mathcal{T}$*_def* **by** (*auto simp*:*types dest*:*transM*)
moreover from *this*
have $M(?T)$ **by** *simp*
moreover

```
note ‹𝒯 ≠ 0›
ultimately
have ?T ∈ 𝒯
  unfolding 𝒯_def using def_Pow_rel
  by (auto simp:types)
have ?T ⊆ ?Q ∪ f``?T
proof (intro subsetI, rule ccontr)
  fix x
  from ‹_ ∪ f ``?T ⊆ ?T›
  have f `` (?T - {x}) ⊆ ?T f `` (?T - {x}) ⊆ f`` ?T by auto
  assume x ∈ ?T x ∉ ?Q ∪ f``?T
  then
  have ?T - {x} ∈ 𝒯
  proof -
    note ‹f `` (?T - {x}) ⊆ ?T›
    moreover from this and ‹x ∉ _ ∪ f``?T›
    have x ∉ f `` ?T by simp
    ultimately
    have f `` (?T - {x}) ⊆ ?T - {x} by auto
    moreover from ‹x ∉ ?Q ∪ _› ‹?Q ∪ _ ⊆ ?T›
    have ?Q ⊆ ?T - {x} by auto
    moreover
    note ‹?T ∈ _› ‹x ∈ ?T› ‹M(?T)› ‹?T ⊆ _›
    ultimately
    show ?thesis
      using def_Pow_rel[of ?T - {x} A] transM[of _ ?T]
      unfolding 𝒯_def
      by (auto simp:types)
  qed
  moreover
  note ‹f `` (?T - {x}) ⊆ f`` ?T›
  ultimately
  have ?T ⊆ ?T - {x} by auto
  with ‹x ∈ ?T›
  show False by auto
qed
with ‹?Q ∪ f``?T ⊆ ?T›
have ?T = ?Q ∪ f``?T by auto
from ‹⋀X. f``X ⊆ B›
have (?Q ∪ f``?T) ∪ (f``A - f``?T) ⊆ B by auto
with ‹?T = _›
have ?T ∪ (f``A - f``?T) ⊆ B by simp
with ‹𝒯 ≠ 0› ‹?T = _›
have B = ?T ∪ (f``A - f``?T)
proof (intro equalityI, intro subsetI)
  fix x
  assume x ∈ B
  with ‹𝒯 ≠ 0› ‹?T = _›
  show x ∈ ?T ∪ (f``A - f``?T)
```

```
    by (subst ‹?T = _›, clarify)
  qed
  moreover from ‹?T ⊆ A›
  have A = ?T ∪ (A - ?T) by auto
  moreover from ‹M(?T)› ‹f ∈ _›
  have M(id(?T) ∪ restrict(f,A-?T))
    using bij_rel_closed[THEN [2] transM] id_closed by (auto simp:types)
  moreover from ‹?T = _› ‹f ∈ _›
  have f``A - f``?T = f``(A - ?T)
    using bij_rel_char bij_is_inj[THEN inj_range_Diff, of f A]
    by (auto simp:types)
  from ‹f ∈ _› types
  have id(?T) ∪ restrict(f, A-?T) ∈ bij(?T ∪ (A - ?T),?T ∪ (f``A - f``?T))
    using id_bij mem_bij_rel[of _ A A', THEN bij_is_inj]
    by (rule_tac bij_disjoint_Un; clarsimp)
      (subst ‹f``A - f``?T =_ ›, auto intro:restrict_bij, subst ‹?T = _›, auto)
  moreover
  note types
  ultimately
  show ?thesis
    using bij_rel_char unfolding eqpoll_rel_def by fastforce
qed

lemma schroeder_bernstein_closed':
  assumes f ∈ inj^M(A,C)  g ∈ inj^M(C,A)
    and types:M(A) M(C)
  shows A ≈^M C
proof -
  from assms
  have f ∈ inj(A,C)  g ∈ inj(C,A) M(f) M(g)
    using inj_rel_char by auto
  moreover from this
  have g ∈ bij(C,range(g)) g O f ∈ bij(A,range(g O f))
    using inj_bij_range comp_inj by auto blast
  moreover from calculation
  have range(g O f) ⊆ range(g) range(g) ⊆ A
    using inj_is_fun[THEN range_fun_subset_codomain] by auto
  moreover from calculation
  obtain h where h ∈ bij^M(A, range(g))
    using dedekind_zermelo[of range(g O f) range(g) A]
      bij_rel_char def_eqpoll_rel
    by (auto simp:types)
  ultimately
  show ?thesis
    using bij_rel_char def_eqpoll_rel comp_bij[of h A range(g) converse(g) C]
      bij_converse_bij[of g C range(g)] by (auto simp:types)
qed
```

Relative equipollence is an equivalence relation

declare *mem_bij_abs*[*simp*] *mem_inj_abs*[*simp*]

lemma *bij_imp_eqpoll_rel*:
assumes $f \in bij(A,B)$ $M(f)$ $M(A)$ $M(B)$
shows $A \approx^M B$
using *assms* **by** (*auto simp add*:*def_eqpoll_rel*)

lemma *eqpoll_rel_refl*: $M(A) \Longrightarrow A \approx^M A$
using *bij_imp_eqpoll_rel*[*OF id_bij*, *OF id_closed*] **.**

lemma *eqpoll_rel_sym*: $X \approx^M Y \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow Y \approx^M X$
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*)

Relative le-pollence is a preorder

lemma *subset_imp_lepoll_rel*: $X \subseteq Y \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow X \lesssim^M Y$
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*: $X \approx^M Y ==> M(X) \Longrightarrow M(Y) \Longrightarrow X \lesssim^M Y$
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*: $[\![X \lesssim^M Y;\ Y \lesssim^M X;\ M(X)\ ;\ M(Y)]\!] \Longrightarrow X \approx^M Y$
unfolding *def_lepoll_rel* **using** *schroeder_bernstein_closed'*
by (*auto simp del*:*mem_inj_abs*)

lemma *eqpoll_relE*:
$[|\ X \approx^M Y;\ [|\ X \lesssim^M Y;\ Y \lesssim^M X\ |] ==> P\ ;\ \ M(X)\ ;\ M(Y)\ |] ==> P$
by (*blast intro*: *eqpoll_rel_imp_lepoll_rel eqpoll_rel_sym*)

lemma *eqpoll_rel_iff*: $M(X) \Longrightarrow M(Y) \Longrightarrow X \approx^M Y \longleftrightarrow X \lesssim^M Y\ \&\ Y \lesssim^M X$
by (*blast intro*: *eqpoll_relI elim*: *eqpoll_relE*)

lemma *lepoll_rel_0_is_0*: $A \lesssim^M 0 \Longrightarrow M(A) \Longrightarrow A = 0$
using *def_lepoll_rel*
by (*cases A=0*) (*auto simp add*: *inj_def*)

— $M(Y) \Longrightarrow 0 \lesssim^M Y$
lemmas *empty_lepoll_relI* = *empty_subsetI* [*THEN subset_imp_lepoll_rel*, *OF nonempty*]

lemma *lepoll_rel_0_iff*: $M(A) \Longrightarrow A \lesssim^M 0 \longleftrightarrow A=0$
by (*blast intro*: *lepoll_rel_0_is_0 lepoll_rel_refl*)

lemma *Un_lepoll_rel_Un*:
$[|\ A \lesssim^M B;\ C \lesssim^M D;\ B \cap D = 0;\ M(A);\ M(B);\ M(C);\ M(D)\ |] ==> A \cup C \lesssim^M B \cup D$
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*: $A \approx^M 0 \Longrightarrow M(A) \Longrightarrow A = 0$
using *eqpoll_rel_imp_lepoll_rel lepoll_rel_0_is_0 nonempty*
by *blast*

lemma *eqpoll_rel_0_iff*: $M(A) \Longrightarrow A \approx^M 0 \longleftrightarrow A=0$
by (*blast intro*: *eqpoll_rel_0_is_0 eqpoll_rel_refl*)

lemma *eqpoll_rel_disjoint_Un*:
$[|\ A \approx^M B;\ \ C \approx^M D;\ \ A \cap C = 0;\ \ B \cap D = 0;\ M(A);\ M(B);\ M(C)\ ;\ M(D)\ |]$
$==> A \cup C \approx^M B \cup D$
by (*auto intro*: *bij_disjoint_Un simp add*:*def_eqpoll_rel*)

21.3 lesspoll: contributions by Krzysztof Grabczewski

lemma *lesspoll_rel_not_refl*: $M(i) \Longrightarrow \sim (i \prec^M i)$
  by (*simp add*: *lesspoll_rel_def eqpoll_rel_refl*)

lemma *lesspoll_rel_irrefl*: $i \prec^M i$ ==> $M(i) \Longrightarrow P$
  by (*simp add*: *lesspoll_rel_def eqpoll_rel_refl*)

lemma *lesspoll_rel_imp_lepoll_rel*: $[\![A \prec^M B;\ M(A);\ M(B)]\!] \Longrightarrow A \lesssim^M B$
  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*: [| $A \lesssim^M B$; *well_ord*(B,r); $M(A)$; $M(B)$; $M(r)$ |]
==> $\exists\, s[M].$ *well_ord*(A,s)
  unfolding *def_lepoll_rel* **by** (*auto intro*:*well_ord_rvimage*)

lemma *lepoll_rel_iff_leqpoll_rel*: $[\![M(A);\ M(B)]\!] \Longrightarrow A \lesssim^M B \longleftrightarrow A \prec^M B \mid A \approx^M B$
  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*: $f \in inj^M(A,B) \Longrightarrow M(f) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow f \in A \rightarrow^M B$
  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 (*function_space_rel_char inj_rel_char*) 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: $y \in succ(m)$ $\bigwedge x.\ x{\in}A \Longrightarrow f\ `\ x \neq y\ M(y)$
    by (*auto simp add*: *def_surj_rel*)
  show *?thesis*

proof
from *types* **and** ‹$M(y)$›
show $M(\lambda z{\in}A.\ \mathit{if}\ f\ `\ z = m\ \mathit{then}\ y\ \mathit{else}\ f\ `\ z)$
using *transM*[*OF_* ‹$M(A)$›] *lam_if_then_apply_replacement2 lam_replacement_iff_lam_closed*
by (*auto*)
with *types y fi*
have $(\lambda z{\in}A.\ \mathit{if}\ f`z = m\ \mathit{then}\ y\ \mathit{else}\ f`z) \in A{\rightarrow}^M\ m$
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 $(\lambda z{\in}A.\ \mathit{if}\ f`z = m\ \mathit{then}\ y\ \mathit{else}\ f`z) \in \mathit{inj}^M(A,\ m)$
by (*simp add*: *def_inj_rel*) *blast*
qed
qed

— Variations on transitivity

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_lesspoll_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 $\exists\kappa[M].\ \mathit{is_cardinal}(M,X,\kappa) \wedge \mathit{is_cardinal}(M,Y,\kappa)$
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*: $X \approx^M Y \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow |X|^M = |Y|^M$
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** *is_cardinal*$(M,A,\kappa) \Longrightarrow M(A) \Longrightarrow M(\kappa) \Longrightarrow$
$M(r) \Longrightarrow \kappa \approx^M A$
proof (*subst is_cardinal_iff_Least*[*THEN iffD1*, *of A* κ])
assume $M(A)$ $M(\kappa)$ $M(r)$ *is_cardinal*(M,A,κ)
moreover from *assms* **and** *calculation*
obtain *f i* **where** $M(f)$ *Ord*(i) $M(i)$ $f \in$ *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*]

22 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*: $Ord(A) \Longrightarrow M(A) \Longrightarrow ||A|^M|^M = |A|^M$
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 ... $= |Y|^M$ **by** (*rule eq*)
also from *types*
have ... $\approx^M Y$ **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*); $M(X)$; $M(r)$; $M(Y)$; $M(s)$ |] ==> $|X|^M = |Y|^M \longleftrightarrow X \approx^M Y$
by (*blast intro*: *cardinal_rel_cong well_ord_cardinal_rel_eqE*)

lemma *Ord_cardinal_rel_le*: $Ord(i) \Longrightarrow M(i)$ ==> $|i|^M \leq i$
unfolding *cardinal_rel_def*
using *eqpoll_rel_refl Least_le* **by** *simp*

lemma *Card_rel_cardinal_rel_eq*: $Card^M(K)$ ==> $M(K) \Longrightarrow |K|^M = K$
apply (*unfold Card_rel_def*)
apply (*erule sym*)
done

lemma *Card_relI*: [| $Ord(i)$; !!j. $j<i \Longrightarrow M(j)$ ==> ~($j \approx^M i$); $M(i)$ |] ==> $Card^M(i)$
apply (*unfold Card_rel_def cardinal_rel_def*)
apply (*subst Least_equality*)
apply (*blast intro*: *eqpoll_rel_refl*)+
done

lemma *Card_rel_is_Ord*: $Card^M(i)$ ==> $M(i) \Longrightarrow Ord(i)$
apply (*unfold Card_rel_def cardinal_rel_def*)
apply (*erule ssubst*)
apply (*rule Ord_Least*)
done

lemma *Card_rel_cardinal_rel_le*: $Card^M(K)$ ==> $M(K) \Longrightarrow K \leq |K|^M$
apply (*simp* (*no_asm_simp*) *add*: *Card_rel_is_Ord Card_rel_cardinal_rel_eq*)
done

lemma *Ord_cardinal_rel* [*simp*,*intro*!]: $M(A) \Longrightarrow Ord(|A|^M)$
  apply (*unfold cardinal_rel_def*)
  apply (*rule Ord_Least*)
  done

lemma *Card_rel_iff_initial*: **assumes** *types*:$M(K)$
  shows $Card^M(K) \longleftrightarrow Ord(K)$ & $(\forall j[M].\ j<K \longrightarrow \sim (j \approx^M K))$
proof -
  { fix j
    assume K: $Card^M(K)\ M(j) \wedge j \approx^M K$
    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*: [| $Card^M(a)$; $i<a$; $M(a)$; $M(i)$ |] ==> $i \prec^M a$
  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*: $Card^M(0)$
  apply (*rule Ord_0* [*THEN Card_relI*])
   apply (*auto elim*!: *ltE*)
  done

lemma *Card_rel_Un*: [| $Card^M(K)$; $Card^M(L)$; $M(K)$; $M(L)$ |] ==> $Card^M(K \cup L)$
  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** $Card^M(|A|^M)$
  using *assms*
proof (*unfold cardinal_rel_def*)
  show $Card^M(\mu\ i.\ M(i) \wedge i \approx^M A)$
  proof (*cases* $\exists\, i[M].\ Ord\ (i) \wedge i \approx^M A$)
    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: Ord(i) i ≈^M A M(i) by blast
    show ?thesis
    proof (rule Card_relI [OF Ord_Least], rule notI)
      fix j
      assume j: j < (μ i. M(i) ∧ i ≈^M A) and M(j)
      assume j ≈^M (μ i. M(i) ∧ i ≈^M A)
      also have ... ≈^M A using i LeastI[of λi. M(i) ∧ i ≈^M A] by (auto)
      finally have j ≈^M A
        using Least_closed'[of λi. M(i) ∧ i ≈^M A] by (simp add: ‹M(j)› types)
      thus False
        using ‹M(j)› by (blast intro:less_LeastE [OF _ j])
    qed (auto intro:Least_closed)
  qed
qed

lemma cardinal_rel_eq_lemma:
  assumes i:|i|^M ≤ j and j: j ≤ i and types: M(i) M(j)
  shows |j|^M = |i|^M
proof (rule eqpoll_relI [THEN cardinal_rel_cong])
  show j ≲^M i by (rule le_imp_lepoll_rel [OF j]) (simp_all add:types)
next
  have Oi: Ord(i) using j by (rule le_Ord2)
  with types
  have i ≈^M |i|^M
    by (blast intro: Ord_cardinal_rel_eqpoll_rel eqpoll_rel_sym)
  also from types
  have ... ≲^M j
    by (blast intro: le_imp_lepoll_rel i)
  finally show i ≲^M j by (simp_all add:types)
qed (simp_all add:types)

lemma cardinal_rel_mono:
  assumes ij: i ≤ j and types:M(i) M(j) shows |i|^M ≤ |j|^M
  using Ord_cardinal_rel [OF ‹M(i)›] Ord_cardinal_rel [OF ‹M(j)›]
proof (cases rule: Ord_linear_le)
  case le then show ?thesis .
next
  case ge
  have i: Ord(i) using ij
    by (simp add: lt_Ord)
  have ci: |i|^M ≤ 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); $Card^M(K)$; $M(i)$; $M(K)$|]
==> $i < K$
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); $Card^M(K)$; $M(i)$; $M(K)$ |] ==> ($|i|^M < K$)
$\longleftrightarrow$ ($i < K$)
by (*blast intro*: *Card_rel_lt_imp_lt Ord_cardinal_rel_le* [*THEN lt_trans1*])

lemma *Card_rel_le_iff*: [| *Ord*(i); $Card^M(K)$; $M(i)$; $M(K)$ |] ==> ($K \leq |i|^M$)
$\longleftrightarrow$ ($K \leq i$)
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* ‹$M(A)$›] *Ord_cardinal_rel* [*OF* ‹$M(B)$›]
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); $M(A)$; $M(i)$ |] ==> $|A|^M$
$\leq i$
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*: [| $A \lesssim^M i$; *Ord*(i); $M(A)$; $M(i)$ |] ==> $|A|^M \approx^M A$
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*: [| $A \prec^M i$; *Ord*(i); $M(A)$; $M(i)$ |] ==> $|A|^M \approx^M A$
using *lepoll_rel_Ord_imp_eqpoll_rel*[*OF lesspoll_rel_imp_lepoll_rel*] **.**

lemma *lesspoll_cardinal_lt_rel*:
shows [| $A \prec^M i$; *Ord*(i); $M(i)$; $M(A)$ |] ==> $|A|^M < i$
proof -
assume *assms*:$A \prec^M i$ ‹*Ord*(i)› ‹$M(i)$› ‹$M(A)$›
then
have A:*Ord*($|A|^M$) $|A|^M \approx^M A$ $M(|A|^M)$
using *Ord_cardinal_rel lesspoll_rel_imp_eqpoll_rel*
by *simp_all*
with *assms*
have $|A|^M \prec^M i$
using *eq_lesspoll_rel_trans* **by** *auto*
consider $|A|^M \in i$ | $|A|^M = i$ | $i \in |A|^M$
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 ‹$|A|^M \prec^M i$› ‹$M(i)$›
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 $i \prec^M |A|^M$
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* ‹$|A|^M \prec^M i$› ‹$i \prec^M$ _ ›]
by *simp*
qed

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

lemma *cardinal_rel_subset_Ord*: [|A<=i; $Ord(i)$; $M(A)$; $M(i)$|] ==> $|A|^M \subseteq i$
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*:
[| $cons(u,A) \lesssim^M cons(v,B)$; $u \notin A$; $v \notin B$; $M(u)$; $M(A)$; $M(v)$; $M(B)$ |] ==> $A \lesssim^M B$
apply (*simp add*: *def_lepoll_rel*, *unfold inj_def*, *safe*)
apply (*rule_tac* $x = \lambda x \in A.$ *if* $f`x=v$ *then* $f`u$ *else* $f`x$ **in** *rexI*)
apply (*rule CollectI*)
— Proving it's in the function space $A \to B$
apply (*rule if_type* [*THEN lam_type*])
apply (*blast dest*: *apply_funtype*)
apply (*blast elim*!: *mem_irrefl dest*: *apply_funtype*)
— Proving it's injective
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*: [| $cons(u,A) \approx^M cons(v,B)$; $u \notin A$; $v \notin B$; $M(u)$; $M(A)$; $M(v)$; $M(B)$ |] ==> $A \approx^M B$
apply (*simp add*: *eqpoll_rel_iff*)
apply (*blast intro*: *cons_lepoll_rel_consD*)
done

lemma *succ_lepoll_rel_succD*: $succ(m) \lesssim^M succ(n) \Longrightarrow M(m) \Longrightarrow M(n)$ ==> $m \lesssim^M n$
apply (*unfold succ_def*)
apply (*erule cons_lepoll_rel_consD*)
apply (*rule mem_not_refl*)+
apply *assumption*+
done

lemma *nat_lepoll_rel_imp_le*:
$m \in nat$ ==> $n \in nat \Longrightarrow m \lesssim^M n \Longrightarrow M(m) \Longrightarrow M(n) \Longrightarrow m \leq n$
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** ‹$n \in nat$›
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*: [| $m \in nat$; $n \in nat$; $M(m)$; $M(n)$ |] ==> $m \approx^M n$ $\longleftrightarrow m = n$
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*: $n \in nat$ **and** *types*: $M(n)$ **shows** $Card^M(n)$
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 $(\mu\ i.\ M(i) \wedge i \approx^M n) = n$ **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*: [| $succ(n) \lesssim^M n$; $n \in nat$ |] ==> *P*
by (*rule nat_lepoll_rel_imp_le* [*THEN lt_irrefl*], *auto*)

lemma *nat_lepoll_rel_imp_ex_eqpoll_rel_n*:
[| $n \in nat$; $nat \lesssim^M X$; $M(n)$; $M(X)$|] ==> $\exists\, Y[M].\ Y \subseteq X$ & $n \approx^M Y$
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*: $M(i) \Longrightarrow i \lesssim^M succ(i)$
by (*blast intro*: *subset_imp_lepoll_rel*)

lemma *lepoll_rel_imp_lesspoll_rel_succ*:
  assumes A: $A \lesssim^M m$ **and** m: $m \in nat$
    and *types*: $M(A)$ $M(m)$
  shows $A \prec^M succ(m)$
proof -
  { assume $A \approx^M succ(m)$
    hence $succ(m) \approx^M A$ **by** (*rule eqpoll_rel_sym*) (*auto simp add:types*)
    also have ... $\lesssim^M m$ **by** (*rule A*)
    finally have $succ(m) \lesssim^M m$ **by** (*auto simp add:types*)
    hence *False* **by** (*rule succ_lepoll_rel_natE*) (*rule m*) **}**
 moreover have $A \lesssim^M succ(m)$ **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*:
  [| $A \prec^M succ(m)$; $m \in nat$; $M(A)$; $M(m)$ |] ==> $A \lesssim^M m$
proof -
  {
    assume $m \in nat$ $M(A)$ $M(m)$ $A \lesssim^M succ(m)$
      $\forall f{\in}inj^M(A,\ succ(m)).\ f \notin surj^M(A,\ succ(m))$
    moreover from *this*
    obtain f **where** $M(f)$ $f{\in}inj^M(A,succ(m))$
      using *def_lepoll_rel* **by** *auto*
    moreover from *calculation*
    have $f \notin surj^M(A,\ succ(m))$ **by** *simp*
    ultimately
    have $\exists f[M].\ f \in inj^M(A,\ m)$
      using *inj_rel_not_surj_rel_succ* **by** *auto*
  }
  from *this*
  show $[\![\ A \prec^M succ(m);\ m \in nat;\ M(A);\ M(m)\ ]\!] \Longrightarrow A \lesssim^M m$
    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*: $m \in nat \Longrightarrow M(A)$ ==> $A \prec^M succ(m) \longleftrightarrow A \lesssim^M m$
 by (*blast intro*!: *lepoll_rel_imp_lesspoll_rel_succ lesspoll_rel_succ_imp_lepoll_rel*)

lemma *lepoll_rel_succ_disj*: [| $A \lesssim^M succ(m)$; $m \in nat$; $M(A)$; $M(m)$|] ==> $A \lesssim^M m$ | $A \approx^M succ(m)$
  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*: [| $A \prec^M i$; $Ord(i)$; $M(A)$; $M(i)$ |] ==> $|A|^M < i$
  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*: $n<i$ $n \in nat$
    and *types*:$M(n)$ $M(i)$ **shows** $\sim i \lesssim^M n$
proof -
  { assume *i*: $i \lesssim^M n$
    have $succ(n) \lesssim^M i$ **using** *n*
      by (*elim ltE*, *blast intro*: *Ord_succ_subsetI* [*THEN subset_imp_lepoll_rel*] *types*)
    also have ... $\lesssim^M n$ **by** (*rule i*)
    finally have $succ(n) \lesssim^M n$ **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*: $n \in nat$
    and *types*: $M(i)$ $M(n)$
  shows $i \approx^M n \longleftrightarrow i=n$
  using *i nat_into_Ord* [*OF n*]
proof (*cases rule*: *Ord_linear_lt*)
  case *lt*
  hence $i \in nat$ **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 $\sim i \lesssim^M n$ **using** *n* **by** (*rule lt_not_lepoll_rel*) (*simp_all add*: *types*)
  hence $\sim i \approx^M n$ **using** *n* **by** (*blast intro*: *eqpoll_rel_imp_lepoll_rel types*)
  moreover have $i \neq n$ **using** ‹$n<i$› **by** *auto*
  ultimately show *?thesis* **by** *blast*
qed

lemma *Card_rel_nat*: $Card^M(nat)$
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 (μ *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*: $nat \leq i \Longrightarrow M(i) ==> nat \leq |i|^M$
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*: $n \in nat ==> n \prec^M nat$
by (*blast intro*: *Card_rel_nat ltI lt_Card_rel_imp_lesspoll_rel*)

lemma *cons_lepoll_rel_cong*:
[| $A \lesssim^M B$; $b \notin B$; $M(A)$; $M(B)$; $M(b)$; $M(a)$ |] ==> $cons(a,A) \lesssim^M cons(b,B)$
apply (*subst* (*asm*) *def_lepoll_rel*, *simp_all*, *subst def_lepoll_rel*, *simp_all*, *safe*)
apply (*rule_tac* $x = \lambda y \in cons\ (a,A)$. *if* $y=a$ *then* b *else* $f`y$ **in** *rexI*)
apply (*rule_tac* $d = \%z$. *if* $z \in B$ *then converse* (f) *`z 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*:
[| $A \approx^M B$; $a \notin A$; $b \notin B$; $M(A)$; $M(B)$; $M(a)$; $M(b)$ |] ==> $cons(a,A) \approx^M cons(b,B)$
by (*simp add*: *eqpoll_rel_iff cons_lepoll_rel_cong*)

lemma *cons_lepoll_rel_cons_iff*:
[| $a \notin A$; $b \notin B$; $M(a)$; $M(A)$; $M(b)$; $M(B)$ |] ==> $cons(a,A) \lesssim^M cons(b,B) \longleftrightarrow A \lesssim^M B$
by (*blast intro*: *cons_lepoll_rel_cong cons_lepoll_rel_consD*)

lemma *cons_eqpoll_rel_cons_iff*:
[| $a \notin A$; $b \notin B$; $M(a)$; $M(A)$; $M(b)$; $M(B)$ |] ==> $cons(a,A) \approx^M cons(b,B) \longleftrightarrow A \approx^M B$
by (*blast intro*: *cons_eqpoll_rel_cong cons_eqpoll_rel_consD*)

lemma *singleton_eqpoll_rel_1*: $M(a) \Longrightarrow \{a\} \approx^M 1$
apply (*unfold succ_def*)
apply (*blast intro*!: *eqpoll_rel_refl* [*THEN cons_eqpoll_rel_cong*])

done

lemma *cardinal_rel_singleton*: $M(a) \Longrightarrow |\{a\}|^M = 1$
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*: $A \neq 0 ==> M(A) \Longrightarrow 1 \lesssim^M A$
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*. $y \lesssim^M$ *cons* (*x*, *A*-{*x*}) **in** *subst*)
apply *auto*
proof -
fix *x*
assume *M*(*A*)
then
show $x \in A \Longrightarrow \{0\} \lesssim^M cons(x, A - \{x\})$
by (*auto intro*: *cons_lepoll_rel_cong transM*[*OF _* ‹*M*(*A*)›] *subset_imp_lepoll_rel*)
qed

lemma *succ_eqpoll_rel_cong*: $A \approx^M B \Longrightarrow M(A) \Longrightarrow M(B) ==> succ(A) \approx^M succ(B)$
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*:
[| $f \in bij^M(A,C)$; $g \in bij^M(B,D)$; *M*(*f*); *M*(*A*); *M*(*C*); *M*(*g*); *M*(*B*); *M*(*D*)|]
==> ($\lambda z \in A+B$. *case*(%*x*. *Inl*(*f*‘*x*), %*y*. *Inr*(*g*‘*y*), *z*)) $\in bij^M(A+B, C+D)$
proof -
assume *asm*:$f \in bij^M(A,C)$ $g \in bij^M(B,D)$ *M*(*f*) *M*(*A*) *M*(*C*) *M*(*g*) *M*(*B*) *M*(*D*)
then
have *M*($\lambda z \in A+B$. *case*(%*x*. *Inl*(*f*‘*x*), %*y*. *Inr*(*g*‘*y*), *z*))
using *transM*[*OF _* ‹*M*(*A*)›] *transM*[*OF _* ‹*M*(*B*)›]
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 ∈ bij^M(A,C) g ∈ bij^M(B,D) M(f)
    M(A) M(C) M(g) M(B) M(D)
  shows
    (λz∈A+B. case(λx. Inl(f`x), λy. Inr(g`y), z)) ∈ bij(A+B, C+D)
    M(λz∈A+B. case(λx. Inl(f`x), λy. Inr(g`y), z))
proof -
  from assms
  show M(λz∈A+B. case(λx. Inl(f`x), λy. Inr(g`y), z))
    using transM[OF _ ‹M(A)›] transM[OF _ ‹M(B)›]
    by (auto intro:case_replacement4[THEN lam_closed])
  with assms
  show (λz∈A+B. case(λx. Inl(f`x), λy. Inr(g`y), z)) ∈ bij(A+B, C+D)
    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_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 sum_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 prod_bij_rel':
  assumes f ∈ bij^M(A,C) g ∈ bij^M(B,D) M(f)
    M(A) M(C) M(g) M(B) M(D)
  shows
    (λ<x,y>∈A∗B. <f`x, g`y>) ∈ bij(A∗B, C∗D)
    M(λ<x,y>∈A∗B. <f`x, g`y>)
proof -
  from assms
  show M((λ<x,y>∈A∗B. <f`x, g`y>))
    using transM[OF _ ‹M(A)›] transM[OF _ ‹M(B)›]
      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 ∈ inj^M(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_funtype])
    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 $\exists\, T[M].\ well_ord(X \cup Y,\ T)$
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*: $M(A) \Longrightarrow M(B) \Longrightarrow A \cap B = 0 \Longrightarrow A \cup B \approx^M A + B$
apply (*simp add*: *def_eqpoll_rel*)
apply (*rule_tac* $x = \lambda a{\in}A \cup B.\ if\ a \in A\ then\ Inl\ (a)\ else\ Inr\ (a)$ **in** *rexI*)
apply (*rule_tac* $d = \%z.\ case\ (\%x.\ x,\ \%x.\ x,\ z)$ **in** *lam_bijective*)
apply *auto*
proof -
assume $M(A)\ M(B)$
then
show $M(\lambda x{\in}A \cup B.\ if\ x \in A\ then\ Inl(x)\ else\ Inr(x))$
using *transM*[*OF* _ ‹$M(A)$›] *transM*[*OF* _ ‹$M(B)$›] *if_then_Inj_replacement*
by (*rule_tac lam_closed*) *auto*
qed

lemma *eqpoll_rel_imp_Finite_rel_iff*: $A \approx^M B$ ==> $M(A) \Longrightarrow M(B) \Longrightarrow Finite_rel(M,A) \longleftrightarrow Finite_rel(M,B)$
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. *lesspoll_rel_succ_imp_lepoll_rel*.

lemma *Finite_abs*[*simp*]:
assumes $M(A)$
shows $Finite_rel(M,A) \longleftrightarrow Finite(A)$
unfolding *Finite_rel_def Finite_def*
proof (*simp*, *intro iffI*)
assume $\exists\, n{\in}nat.\ A \approx^M n$
then
obtain n **where** $A \approx^M n\ n{\in}nat$ **by** *blast*
with *assms*
show $\exists\, n{\in}nat.\ A \approx n$
unfolding *eqpoll_def* **using** *nat_into_M* **by** (*auto simp add*:*def_eqpoll_rel*)
next
fix n
assume $\exists\, n{\in}nat.\ A \approx n$
then
obtain n **where** $A \approx n\ n{\in}nat$ **by** *blast*
moreover from *this*
obtain f **where** $f \in bij(A,n)$ **unfolding** *eqpoll_def* **by** *auto*
moreover

```
  note assms
  moreover from calculation
  have converse(f) ∈ n→A  using bij_is_fun by simp
  moreover from calculation
  have M(converse(f)) using transM[of _ n→A] by simp
  moreover from calculation
  have M(f) using bij_is_fun
      fun_is_rel[of f A λ_. n, THEN converse_converse]
      converse_closed[of converse(f)] by simp
  ultimately
  show ∃n∈nat. A ≈^M n
    by (force dest:nat_into_M simp add:def_eqpoll_rel)
qed
```

— From the next result, the relative versions of *Finite_Fin_lemma* and *Fin_lemma* should follow

```
lemma lepoll_rel_nat_imp_Finite_rel:
  assumes A: A ≲^M n and n: n ∈ nat
    and types: M(A) M(n)
  shows Finite_rel(M,A)
proof -
  have A ≲^M n ⟹ Finite_rel(M,A) using n
  proof (induct n)
    case 0
    hence A = 0 by (rule lepoll_rel_0_is_0, simp_all add:types)
    thus ?case by simp
  next
    case (succ n)
   hence A ≲^M n ∨ A ≈^M succ(n) by (blast dest: lepoll_rel_succ_disj intro:types)
    thus ?case using succ by (auto simp add: Finite_rel_def types)
  qed
  thus ?thesis using A .
qed

lemma lesspoll_rel_nat_is_Finite_rel:
  A ≺^M nat ⟹ M(A) ⟹ Finite_rel(M,A)
  apply (unfold Finite_rel_def)
  apply (auto dest: ltD lesspoll_rel_cardinal_rel_lt
      lesspoll_rel_imp_eqpoll_rel [THEN eqpoll_rel_sym])
  done

lemma lepoll_rel_Finite_rel:
  assumes Y: Y ≲^M X and X: Finite_rel(M,X)
    and types:M(Y) M(X)
  shows Finite_rel(M,Y)
proof -
  obtain n where n: n ∈ nat X ≈^M n M(n) using X
    by (auto simp add: Finite_rel_def)
```

have $Y \lesssim^M X$ **by** (*rule Y*)
also have ... $\approx^M n$ **by** (*rule n*)
finally have $Y \lesssim^M n$ **by** (*simp_all add:types* ‹$M(n)$›)
thus *?thesis* **using** *n*
by (*simp add*: *lepoll_rel_nat_imp_Finite_rel types* ‹$M(n)$› *del:Finite_abs*)
qed

lemma *succ_lepoll_rel_imp_not_empty*: $succ(x) \lesssim^M y$ ==> $M(x) \Longrightarrow M(y) \Longrightarrow y \neq 0$
by (*fast dest*!: *lepoll_rel_0_is_0*)

lemma *eqpoll_rel_succ_imp_not_empty*: $x \approx^M succ(n)$ ==> $M(x) \Longrightarrow M(n) \Longrightarrow x \neq 0$
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 ‹$Finite(B)$› ‹$B \subseteq A$›
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 _* ‹$M(A)$›] **by** *simp*
qed
qed

lemma *Finite_Pow_abs*:
assumes $Finite(A)$ $M(A)$
shows $Pow(A) = Pow_rel(M,A)$
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*]: $Pow_rel(M,0) = \{0\}$
using *Finite_Pow_abs*[*of 0*] **by** *simp*

lemma *eqpoll_rel_imp_Finite*: $A \approx^M B \Longrightarrow Finite(A) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow Finite(B)$
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 $g\ O\ converse(f) \in bij(B,n)$
using *bij_converse_bij comp_bij* **by** *simp*
with ‹$n \in _$›
show $Finite(B)$
unfolding *Finite_def eqpoll_def* **by** *auto*
qed

lemma *eqpoll_rel_imp_Finite_iff*: $A \approx^M B \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow Finite(A) \longleftrightarrow Finite(B)$
using *eqpoll_rel_imp_Finite eqpoll_rel_sym* **by** *force*

end — *M_cardinals*

end

23 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 $jump_cardinal_body :: [i,i] \Rightarrow i$ **where**
$jump_cardinal_body(U,X) \equiv$
$\{z\ .\ r \in U,\ well_ord(X,\ r) \wedge z = ordertype(X,\ r)\}$

lemma *jump_cardinal_body_char* :

jump_cardinal_body$(U,X) = \{$*ordertype*$(X,\ r)\ .\ r \in \{r{\in}U\ .\ $*well_ord*$(X,\ r)\}\}$
unfolding *jump_cardinal_body_def*
by *auto*

— We internalize the relativized version of this particular instance.
definition *jump_cardinal_body′* **where**
jump_cardinal_body′$(x) \equiv$ *jump_cardinal_body*$(Pow(x{\times}x),x)$

lemma (**in** *M_cardinals*) *csquare_lam_closed*[*intro*,*simp*]: $M(K) \Longrightarrow M($*csquare_lam*$(K))$
using *csquare_lam_replacement* **unfolding** *csquare_lam_def*
by (*rule lam_closed*) (*auto dest*:*transM*)

locale *M_pre_cardinal_arith* = *M_cardinals* +
assumes
wfrec_pred_replacement:$M(A) \Longrightarrow M(r) \Longrightarrow$
wfrec_replacement$(M,\ \lambda x\ f\ z.\ z = f\ ``\ $*Order.pred*$(A,\ x,\ r),\ r)$

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 *rmult rmult_rel* **external**
relationalize *rmult_rel is_rmult*

lemma (**in** *M_trivial*) *rmultP_abs* [*absolut*]: $[\![\ M(r);\ M(s);\ M(z)\ ]\!] \Longrightarrow$ *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* _ ‹$M(A)$›]
unfolding *is_rvimage_def rvimage_def*
by *auto*

lemma *rmult_abs* [*absolut*]: $[\![\ M(A);\ M(r);\ M(B);\ M(s);\ M(z)\ ]\!] \Longrightarrow$
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*: *iffI*)

lemma *csquare_lam_body_abs*[*absolut*]: $M(x) \Longrightarrow M(z) \Longrightarrow$
is_csquare_lam_body$(M,x,z) \longleftrightarrow z = <fst(x) \cup snd(x), fst(x), snd(x)>$
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*$(K) = (\lambda z{\in}K{\times}K.\ <fst(z) \cup snd(z), fst(z), snd(z)>)$
proof -
have $(\lambda\langle x,y\rangle{\in}K \times K.\ \langle x \cup y, x, y\rangle)$‘$z =$
$(\lambda z{\in}K{\times}K.\ <fst(z) \cup snd(z), fst(z), snd(z)>)$‘$z$ **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*]: $M(K) \Longrightarrow M(csquare_rel(K))$
using *csquare_lam_replacement* **unfolding** *csquare_rel_def*
by (*intro rvimage_closed lam_closed*) (*auto dest*:*transM*)

lemma *csquare_rel_abs*[*absolut*]: $[\![\ M(K);\ M(cs)]\!] \Longrightarrow$
$is_csquare_rel(M,K,cs) \longleftrightarrow cs = csquare_rel(K)$
using *csquare_lam_closed*[*unfolded csquare_lam_eq_lam*]
unfolding *is_csquare_rel_def csquare_rel_def*
by (*simp add*:*absolut csquare_lam_eq_lam*[*unfolded csquare_lam_def*])

end — *M_pre_cardinal_arith*

23.1 Discipline for *csucc*

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
  csucc_r :: $[i,i\Rightarrow o] \Rightarrow i$ (‹'(_$^+$')–›) **where**
  $csucc_r(x,M) \equiv csucc_rel(M,x)$

abbreviation
  csucc_r_set :: $[i,i] \Rightarrow i$ (‹'(_$^+$')–›) **where**
  $csucc_r_set(x,M) \equiv csucc_rel(\#\#M,x)$

context *M_Perm*
begin

rel_closed for *csucc*
  using $Least_closed'[of\ \lambda\ L.\ M(L) \wedge Card^M(L) \wedge K < L]$
  unfolding *csucc_rel_def*
  by *simp*

is_iff_rel for *csucc*
  using $least_abs'[of\ \lambda\ L.\ M(L) \wedge Card^M(L) \wedge K < L\ res]$
    is_Card_iff
  unfolding *is_csucc_def csucc_rel_def*
  by (*simp add*:*absolut*)

end — *M_Perm*

notation *csucc_rel* (‹*csucc*–'(_')›)

context *M_cardinals*
begin

lemma *Card_rel_Union* [*simp*,*intro*,*TC*]:
  assumes *A*: $\bigwedge x.\ x\in A \Longrightarrow Card^M(x)$ **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 ‹M(j)›
  have jls: j ≺^M c
    by (simp add: lt_Card_rel_imp_lesspoll_rel types)
  { assume eqp: j ≈^M ⋃A
    have  c ≲^M ⋃A using c
      by (blast intro: subset_imp_lepoll_rel types)
    also from types ‹M(j)›
    have ... ≈^M j  by (rule_tac eqpoll_rel_sym [OF eqp]) (simp_all add:types)
    also have ... ≺^M c  by (rule jls)
    finally have c ≺^M c by (simp_all add:‹M(c)› ‹M(j)› types)
    with ‹M(c)›
    have False
      by (auto dest:lesspoll_rel_irrefl)
  } thus ¬ j ≈^M ⋃A by blast
qed (simp_all add:types)
```

We are not relativizing directly the results involving unions of (ordinal) indexed families (namely, *Card_UN* and *Card_OUN*, respectively) because of their higher order nature.

```
lemma in_Card_imp_lesspoll: [| Card^M(K); b ∈ K; M(K); M(b) |] ==> b ≺^M
K
  apply (unfold lesspoll_rel_def)
  apply (simp add: Card_rel_iff_initial)
  apply (fast intro!: le_imp_lepoll_rel ltI leI)
  done
```

23.2 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.

23.2.1 Cardinal addition is commutative

```
lemma sum_commute_eqpoll_rel: M(A) ⟹ M(B) ⟹ A+B ≈^M B+A
proof (simp add: def_eqpoll_rel, rule rexI)
  show (λz∈A+B. case(Inr,Inl,z)) ∈ bij(A+B, B+A)
    by (auto intro: lam_bijective [where d = case(Inr,Inl)])
  assume M(A) M(B)
  then
  show M(λz∈A + B. case(Inr, Inl, z))
    using case_replacement1
    by (rule_tac lam_closed) (auto dest:transM)
qed
```

lemma *cadd_rel_commute*: $M(i) \Longrightarrow M(j) \Longrightarrow i \oplus^M j = j \oplus^M i$
apply (*unfold cadd_rel_def*)
apply (*auto intro*: *sum_commute_eqpoll_rel* [*THEN cardinal_rel_cong*])
done

23.2.2 Cardinal addition is associative

lemma *sum_assoc_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (A+B)+C \approx^M A+(B+C)$
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*: *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 $(i \oplus^M j) \oplus^M k = i \oplus^M (j \oplus^M k)$
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 ... $\approx^M i + (j + k)$
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*)

23.2.3 0 is the identity for addition

lemma *case_id_eq*: $x \in sum(A,B) \Longrightarrow case(\lambda z\ .\ z,\ \lambda z.\ z\ ,x) = snd(x)$
unfolding *case_def cond_def* **by** (*auto simp*:*Inl_def Inr_def*)

lemma *lam_case_id*: $(\lambda z \in 0 + A.\ case(\lambda x.\ x,\ \lambda y.\ y,\ z)) = (\lambda z \in 0 + A\ .\ snd(z))$
using *case_id_eq* **by** *simp*

lemma *sum_0_eqpoll_rel*: $M(A) \Longrightarrow 0+A \approx^M A$
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*]: $Card^M(K) \Longrightarrow M(K) \Longrightarrow 0 \oplus^M K = K$
apply (*simp add*: *cadd_rel_def*)
apply (*simp add*: *sum_0_eqpoll_rel* [*THEN cardinal_rel_cong*] *Card_rel_cardinal_rel_eq*)
done

23.2.4 Addition by another cardinal

lemma *sum_lepoll_rel_self*: $M(A) \Longrightarrow M(B) \Longrightarrow A \lesssim^M A+B$
proof (*simp add*: *def_lepoll_rel*, *rule rexI*)
show ($\lambda x{\in}A$. *Inl* (*x*)) $\in$ *inj*(*A*, *A* + *B*)
by (*simp add*: *inj_def*)
assume $M(A)$ $M(B)$
then
show $M(\lambda x{\in}A.\ Inl(x))$
using *Inl_replacement1 transM*[*OF _* ‹$M(A)$›]
by (*rule_tac lam_closed*) (*auto simp add*: *Inl_def*)
qed

lemma *cadd_rel_le_self*:
assumes *K*: $Card^M(K)$ **and** *L*: *Ord*(*L*) **and**
types:$M(K)$ $M(L)$
shows $K \le (K \oplus^M L)$
proof (*simp add*:*types cadd_rel_def*)
have $K \le |K|^M$
by (*rule Card_rel_cardinal_rel_le* [*OF K*]) (*simp add*:*types*)
moreover have $|K|^M \le |K + L|^M$ **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 $K \le |K + L|^M$
by (*blast intro*: *le_trans*)
qed

23.2.5 Monotonicity of addition

lemma *sum_lepoll_rel_mono*:
[| $A \lesssim^M C$; $B \lesssim^M D$; $M(A)$; $M(B)$; $M(C)$; $M(D)$ |] ==> $A + B \lesssim^M C + D$
apply (*simp add*: *def_lepoll_rel*)
apply (*elim rexE*)
apply (*rule_tac x* = $\lambda z{\in}A+B$. *case* (%*w*. *Inl*(*f*‘*w*), %*y*. *Inr*(*fa*‘*y*), *z*) **in** *rexI*)
apply (*rule_tac d* = *case* (%*w*. *Inl*(*converse*(*f*) ‘*w*), %*y*. *Inr*(*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*:
[| $K' \leq K$; $L' \leq L$;$M(K')$;$M(K)$;$M(L')$;$M(L)$ |] ==> $(K' \oplus^M L') \leq (K \oplus^M L)$
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

23.2.6 Addition of finite cardinals is "ordinary" addition

lemma *sum_succ_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow succ(A)+B \approx^M succ(A+B)$
apply (*simp add*:*def_eqpoll_rel*)
apply (*rule rexI*)
apply (*rule_tac c = %z. if z=Inl (A) then A+B else z*
and *d = %z. if z=A+B then Inl (A) 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*: $m \in nat$ **and** [*simp*]: $n \in nat$ **shows**$m \oplus^M n = m +_\omega n$
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

23.3 Cardinal multiplication

23.3.1 Cardinal multiplication is commutative

lemma *prod_commute_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow A*B \approx^M B*A$
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*: $M(i) \Longrightarrow M(j) \Longrightarrow i \otimes^M j = j \otimes^M i$
apply (*unfold cmult_rel_def*)
apply (*rule prod_commute_eqpoll_rel* [*THEN cardinal_rel_cong*], *simp_all*)
done

23.3.2 Cardinal multiplication is associative

lemma *prod_assoc_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (A*B)*C \approx^M A*(B*C)$
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*: *assms 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 ... $\approx^M i * (j * k)$
by (*rule prod_assoc_eqpoll_rel*, *simp_all add*:*types*)
also have ... $\approx^M i * |j * k|^M$
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*)

23.3.3 Cardinal multiplication distributes over addition

lemma *sum_prod_distrib_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (A+B)*C \approx^M (A*C)+(B*C)$
  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*: *assms 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 ... $\approx^M i * k + j * k$
    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*)

23.3.4 Multiplication by 0 yields 0

lemma *prod_0_eqpoll_rel*: $M(A) \Longrightarrow 0*A \approx^M 0$
  apply (*simp add*: *def_eqpoll_rel*)
  apply (*rule rexI*)
  apply (*rule lam_bijective*, *auto*)
  done

lemma *cmult_rel_0* [*simp*]: $M(i) \Longrightarrow 0 \otimes^M i = 0$
  by (*simp add*: *cmult_rel_def prod_0_eqpoll_rel* [*THEN cardinal_rel_cong*])

23.3.5 1 is the identity for multiplication

lemma *prod_singleton_eqpoll_rel*: $M(x) \Longrightarrow M(A) \Longrightarrow \{x\}*A \approx^M A$
  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*]: $Card^M(K) \Longrightarrow M(K) \Longrightarrow 1 \otimes^M K = K$
  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

23.4 Some inequalities for multiplication

lemma *prod_square_lepoll_rel*: $M(A) \Longrightarrow A \lesssim^M A*A$
apply (*simp add*:*def_lepoll_rel inj_def*)
apply (*rule_tac x* = $\lambda x{\in}A$. <*x*,*x*> **in** *rexI*, *simp*)
apply(*rule_tac lam_closed*, *auto intro*:*id_replacement dest*:*transM*)
done

lemma *cmult_rel_square_le*: $Card^M(K) \Longrightarrow M(K) \Longrightarrow K \leq K \otimes^M K$
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

23.4.1 Multiplication by a non-zero cardinal

lemma *prod_lepoll_rel_self*: $b \in B \Longrightarrow M(b) \Longrightarrow M(B) \Longrightarrow M(A) \Longrightarrow A \lesssim^M$
$A*B$
apply (*simp add*: *def_lepoll_rel inj_def*)
apply (*rule_tac x* = $\lambda x{\in}A$. <*x*,*b*> **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_rel_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

23.4.2 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* = *lam* <*w*,*y*>:*A*∗*B*. <*f*‘*w*, *fa*‘*y*> **in** *rexI*)
apply (*rule_tac d* = %<*w*,*y*>. <*converse* (*f*) ‘*w*, *converse* (*fa*) ‘*y*>
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

23.5 Multiplication of finite cardinals is "ordinary" multiplication

lemma *prod_succ_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow succ(A)*B \approx^M B + A*B$
apply (*simp add*: *def_eqpoll_rel*)
apply (*rule rexI*)
apply (*rule_tac c* = λp. *if fst*(*p*)=*A then Inl* (*snd*(*p*)) *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*: $Card^M(n) \Longrightarrow M(n) \Longrightarrow 2 \otimes^M n = n \oplus^M n$
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*

23.6 Infinite Cardinals are Limit Ordinals

context *M_pre_cardinal_arith*
begin

lemma *nat_cons_lepoll_rel*: $nat \lesssim^M A \Longrightarrow M(A) \Longrightarrow M(u)$ ==> $cons(u,A) \lesssim^M A$
apply (*simp add*: *def_lepoll_rel*)
apply (*erule rexE*)
apply (*rule_tac x* =
$\lambda z \in cons\ (u,A)$.
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*(λ*z*∈*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*: $nat \lesssim^M A$ ==> $M(A) \Longrightarrow M(u) \Longrightarrow cons(u,A) \approx^M$

A
apply (*erule nat_cons_lepoll_rel* [*THEN eqpoll_relI*], *assumption*+)
apply (*rule subset_consI* [*THEN subset_imp_lepoll_rel*], *simp_all*)
done

lemma *nat_succ_eqpoll_rel*: $nat \subseteq A$ ==> $M(A) \Longrightarrow succ(A) \approx^M A$
apply (*unfold succ_def*)
apply (*erule subset_imp_lepoll_rel* [*THEN nat_cons_eqpoll_rel*], *simp_all*)
done

lemma *InfCard_rel_nat*: $InfCard^M(nat)$
apply (*simp add*: *InfCard_rel_def*)
apply (*blast intro*: *Card_rel_nat Card_rel_is_Ord*)
done

lemma *InfCard_rel_is_Card_rel*: $M(K) \Longrightarrow InfCard^M(K) \Longrightarrow Card^M(K)$
apply (*simp add*: *InfCard_rel_def*)
done

lemma *InfCard_rel_Un*:
[| $InfCard^M(K)$; $Card^M(L)$; $M(K)$; $M(L)$ |] ==> $InfCard^M(K \cup L)$
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*: $InfCard^M(K)$ ==> $M(K) \Longrightarrow Limit(K)$
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*]:
assumes *wellordered*(M,A,r) $M(A)$ $M(r)$ $M(i)$
shows *otype*(M,A,r,i) $\longleftrightarrow$ i = *ordertype*(A,r)
proof -
have *eq*:$j = k$ **if**

```
  g ∈ ⟨A, r⟩ ≅ ⟨j, Memrel(j)⟩ Ord(j)
  h ∈ ⟨A, r⟩ ≅ ⟨k, Memrel(k)⟩ Ord(k)
for j k g h
proof -
  from that
  have h O converse(g) ∈ ⟨j, Memrel(j)⟩ ≅ ⟨k, Memrel(k)⟩
    using ord_iso_sym ord_iso_trans by blast
  with ‹Ord(j)› ‹Ord(k)›
  show j=k
    using Ord_iso_implies_eq[of j k h O converse(g)]
    by simp
qed
have otypeE: ∃ h . h ∈ ⟨A, r⟩ ≅ ⟨k, Memrel(k)⟩ if otype(M, A, r, k) M(k) for k
proof -
  note that assms
  moreover from this
  obtain h where omap(M, A, r, h) M(h)
    using that omap_exists[of A r] by auto
  moreover from calculation
  have h ∈ ⟨A, r⟩ ≅ ⟨k, Memrel(k)⟩
    using that omap_ord_iso obase_equals by simp
  then
  show ?thesis ..
qed
show ?thesis
proof(intro iffI)
  note assms
  moreover
  assume otype(M, A, r, i)
  moreover from calculation
  obtain f j where M(f) Ord(j) f ∈ ⟨A, r⟩ ≅ ⟨j, Memrel(j)⟩
    using ordertype_exists[of A r] by auto
  ultimately
  show i = ordertype(A, r)
  using Ord_otype[OF _ well_ord_is_trans_on] eq[of f j _ i] ordertypes_are_absolute
      otypeE[of i]
    by auto
next
  note assms
  moreover
  assume i = ordertype(A, r)
  moreover from calculation
  obtain j where otype(M,A,r,j) M(j)
    using otype_exists by auto
  ultimately
  show otype(M, A, r, i)
   using Ord_ordertype eq[of _ i _ j] Ord_otype[OF _ well_ord_is_trans_on]
      otypeE[of j] ordertype_ord_iso[of A r]
    by auto
```

qed
qed

lemma (**in** *M_ordertype*) *ordertype_closed*[*intro*,*simp*]: ⟦ *wellordered*(*M*,*A*,*r*);*M*(*A*);*M*(*r*)⟧ ⟹ *M*(*ordertype*(*A*,*r*))
using *ordertype_exists ordertypes_are_absolute* **by** *blast*

— This apparent duplication of definitions is needed because in `ZF-Constructible` pairs are in their absolute version and this breaks the synthesis of formulas.

relationalize *transitive_rel is_transitive* **external**
synthesize *is_transitive* **from_definition assuming** *nonempty*
arity_theorem for *is_transitive_fm*

lemma (**in** *M_trivial*) *is_transitive_iff_transitive_rel*:
M(*A*)⟹ *M*(*r*) ⟹ *transitive_rel*(*M*, *A*, *r*) ⟷ *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*
arity_theorem for *is_linear_fm*
reldb_add relational *transitive_on transitive_rel*
reldb_add relational *linear_on linear_rel*
reldb_add functional *transitive_on transitive_rel*
reldb_add functional *linear_on linear_rel*

lemma (**in** *M_trivial*) *is_linear_iff_linear_rel*:
M(*A*)⟹ *M*(*r*) ⟹ *is_linear*(*M*,*A*, *r*) ⟷ *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*
arity_theorem for *is_wellfounded_on_fm*

lemma (**in** *M_trivial*) *is_wellfounded_on_iff_wellfounded_on*:
M(*A*)⟹ *M*(*r*) ⟹ *is_wellfounded_on*(*M*,*A*, *r*) ⟷ *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*
is_well_ord(*M*,*A*,*r*) ==
is_transitive(*M*,*A*,*r*) ∧ *is_linear*(*M*,*A*,*r*) ∧ *is_wellfounded_on*(*M*,*A*,*r*)

lemma (**in** *M_trivial*) *is_well_ord_iff_wellordered*:
M(*A*)⟹ *M*(*r*) ⟹ *is_well_ord*(*M*,*A*, *r*) ⟷ *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*

synthesize *is_well_ord* **from_definition assuming** *nonempty*
arity_theorem for *is_well_ord_fm*
reldb_add relational *well_ord is_well_ord*
reldb_add functional *well_ord well_ord*

— 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*

— Two keywords denote origin and destination, respectively

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 *trans_on_iff_trans*: $trans[A](r) \longleftrightarrow trans(r \cap A \times A)$
  unfolding *trans_on_def trans_def* **by** *auto*

lemma *trans_on_subset*: $trans[A](r) \Longrightarrow B \subseteq A \Longrightarrow trans[B](r)$
  unfolding *trans_on_def*
  by *auto*

lemma *relation_Int*: $relation(r \cap B \times B)$
  unfolding *relation_def*
  by *auto*

Discipline for *ordermap*

relativize functional *ordermap ordermap_rel* **external**
relationalize *ordermap_rel is_ordermap*

lemma *wfrec_on_pred_eq*:
assumes $r \in Pow(A \times A)$
shows $wfrec[A](r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r)) = wfrec(r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r))$
proof -
from ‹$r \in Pow(A \times A)$›
have $r \cap A \times A = r$ **by** *auto*
then
show *?thesis*
unfolding *wfrec_on_def* **by** *simp*
qed

context *M_pre_cardinal_arith*
begin

lemma *wfrec_on_pred_closed*:
assumes $wf[A](r)$ $trans[A](r)$ $r \in Pow^M(A \times A)$ $M(A)$ $x \in A$
shows $M(wfrec(r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r)))$
proof -
note *assms*
moreover from *this*
have $M(r)$ $M(x)$
using $transM[of\ _\ Pow^M(A \times A)]$ $transM[of\ _\ A]$
by *simp_all*
moreover from *calculation*
have $wfrec[A](r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r)) = wfrec(r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r))$
using *wfrec_on_pred_eq Pow_rel_char*
by *simp*
moreover from *calculation*
have $M(wfrec(r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r)))$
using *wfrec_pred_replacement wf_on_imp_wf trans_on_imp_trans subset_Sigma_imp_relation Pow_rel_char*
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 $wf[A](r)$ $trans[A](r)$ $r \in Pow^M(A \times A)$ $M(A)$ $x \in A$
shows $M(wfrec[A](r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r)))$
using *assms wfrec_on_pred_closed wfrec_on_pred_eq Pow_rel_char*
by *simp*

lemma *ordermap_rel_closed*[*intro*,*simp*]:
  assumes $wf[A](r)$ $trans[A](r)$ $r \in Pow^M(A\times A)$ $M(A)$
  shows $M(ordermap_rel(M,\ A,\ r))$
proof -
  from *assms*
  have $r \cap A\times A = r$ $r\in Pow(A\times A)$ $M(r)$
    using *Pow_rel_char*
    by *auto*
  with *assms*
  have $wf(r)$ $trans(r)$ $relation(r)$
    unfolding *wf_on_def*
    using *trans_on_iff_trans relation_def* **by** *auto*
  with ‹$M(A)$› ‹$M(r)$›
  have *1*:
    $(\exists\, y[M].\ pair(M,\ x,\ y,\ z) \wedge is_wfrec(M,\ \lambda x\ f\ z.\ z = f\ ``\ Order.pred(A,\ x,\ r),\ r,\ x,\ y))$
      $\longleftrightarrow$
    $z = \langle x,wfrec(r,x,\lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r))\rangle$ **if** $M(x)$ $M(z)$ **for** x z
    using *that trans_wfrec_abs*[*of r*,**where**
      $H=\lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r)$ **and**
      $MH=\lambda x\ f\ z\ .\ z= f\ ``\ Order.pred(A,\ x,\ r)$,*simplified*]
      wfrec_pred_replacement
    unfolding *relation2_def*
    by *auto*
  with ‹$M(A)$› ‹$M(r)$›
  have $strong_replacement(M,\lambda x\ z.\ z = \langle x,wfrec(r,x,\lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r))\rangle)$
    using *strong_replacement_cong*[*of M*,*OF 1*,*THEN iffD1*,*OF _ _*
      wfrec_pred_replacement[*unfolded wfrec_replacement_def*]]
    by *simp*
  with ‹$M(A)$› ‹$M(r)$› ‹$r \in Pow(A\times A)$› *assms*
  show *?thesis*
    unfolding *ordermap_rel_def*
    using *lam_cong*[*OF refl wfrec_on_pred_eq*] *wfrec_on_pred_closed lam_closed*
    by *auto*
qed

lemma *is_ordermap_iff*:
  assumes $r \in Pow^M(A\times A)$ $wf[A](r)$ $trans[A](r)$
    $M(A)$ $M(res)$
  shows $is_ordermap(M,\ A,\ r,\ res) \longleftrightarrow res = ordermap_rel(M,\ A,\ r)$
proof -
  from ‹$r \in _$› ‹$M(A)$›
  have $r \cap A\times A = r$ $M(r)$ $r \in Pow(A\times A)$ $\bigwedge x\ y\ .\ \langle x,y\rangle\in r \Longrightarrow x\in A$
    using *Pow_rel_char*
    by *auto*
  with *assms*
  have $wf(r)$ $trans(r)$ $relation(r)$

unfolding *wf_on_def*
using *trans_on_iff_trans relation_def* **by** *auto*
with ‹$r \in Pow(A \times A)$›
show *?thesis*
using *ordermap_rel_closed*[*of r A*] *assms wfrec_on_pred_closed wfrec_pred_replacement*
unfolding *is_ordermap_def ordermap_rel_def*
apply (*rule_tac lambda_abs2*,*simp_all add*:*Relation1_def*,*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*

Discipline for *ordertype*

relativize functional *ordertype ordertype_rel* **external**
relationalize *ordertype_rel is_ordertype*

definition *is_order_body*
where $is_order_body(M,p,z) \equiv \exists X[M].\exists r[M].\exists A[M].\ is_fst(M,p,X) \wedge is_snd(M,p,r) \wedge$
$cartprod(M,X,X,A) \wedge subset(M,r,A) \wedge M(z) \wedge M(r) \wedge is_well_ord(M,X,r) \wedge is_ordertype(M,X,r,z)$

context *M_pre_cardinal_arith*
begin

lemma *is_ordertype_iff*:
assumes $r \in Pow^M(A \times A)$ $well_ord(A,r)$
shows $M(A) \Longrightarrow M(res) \Longrightarrow is_ordertype(M,\ A,\ r,\ res) \longleftrightarrow res = ordertype_rel(M,\ A,\ r)$
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 well_ord_def part_ord_def tot_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 ordertypes_are_absolute*[*of X r*]
unfolding *ordertype_def ordertype_rel_def ordermap_rel_def ordermap_def*
by *simp*

lemma (**in** *M_pre_cardinal_arith*) *is_order_body_abs* :
$M(Xr) \Longrightarrow M(z) \Longrightarrow is_order_body(M,\ Xr,\ z) \longleftrightarrow$
$snd(Xr) \in Pow^M(fst(Xr) \times fst(Xr)) \wedge well_ord(fst(Xr),\ snd(Xr)) \wedge z = ordertype(fst(Xr),\ snd(Xr))$
using *is_ordertype_iff ordertype_rel_abs well_ord_is_linear Pow_rel_char*

is_well_ord_iff_wellordered snd_abs fst_abs
unfolding *is_order_body_def*
by *simp*

lemma *well_ord_restr*: $well_ord(X, r) \Longrightarrow well_ord(X, r \cap X{\times}X)$
proof -
have $r \cap X{\times}X \cap X{\times}X = r \cap X{\times}X$ **by** *auto*
moreover
assume $well_ord(X, r)$
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 $well_ord(X,r)$
shows $ordertype(X, r) = ordertype(X, r \cap X{\times}X)$
using *ordermap_restr_eq assms* **unfolding** *ordertype_def*
by *simp*

lemma *ordertype_rel_closed*[*intro*,*simp*]:
assumes $well_ord(A,r)$ $r \in Pow^M(A{\times}A)$ $M(A)$
shows $M(ordertype_rel(M,A,r))$
using *assms Pow_rel_char*
unfolding *well_ord_def tot_ord_def part_ord_def ordertype_rel_def*
by *simp*

end — *M_pre_cardinal_arith*

relativize functional *jump_cardinal_body jump_cardinal_body_rel*
relationalize *jump_cardinal_body_rel is_jump_cardinal_body_rel*

relativize functional *jump_cardinal_body′ jump_cardinal_body′_rel*
relationalize *jump_cardinal_body′_rel is_jump_cardinal_body′_rel*

Notice that this is not quite the same as *jump_cardinal*: observe $Pow(X \times X)$.

definition
$jump_cardinal' :: i{\Rightarrow}i$ **where**
$jump_cardinal'(K) \equiv$
$\bigcup X{\in}Pow(K).\ \{z.\ r \in Pow(X{*}X),\ z = jump_cardinal_body(Pow(X{*}X),X)\}$

relativize functional *jump_cardinal jump_cardinal_rel* **external**
relativize functional *trans_on transitivie_rel* **external**
relationalize *jump_cardinal_rel is_jump_cardinal*

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) $f \in$ *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) $\in$ *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

context *M_pre_cardinal_arith*
begin

lemma *def_jump_cardinal_body*:
  $M(X) \Longrightarrow M(U) \Longrightarrow$
  jump_cardinal_body(U,X) = $\{z$. $r \in U$, $M(z) \wedge$ *well_ord*(X, r) $\wedge$ z = *ordertype*(X, r)$\}$
  unfolding *jump_cardinal_body_def*
  using *ordertype_closed*
  by(*rule_tac Replace_cong*,*auto dest*:*transM*)

lemma *jump_cardinal_body_abs*:
 shows $M(A) \Longrightarrow$ *jump_cardinal_body_rel*(M,$Pow^M(A\times A)$,A) = *jump_cardinal_body*($Pow^M(A\times A)$,A)
  unfolding *jump_cardinal_body_rel_def*
  using *def_jump_cardinal_body ordertype_rel_abs transM*[*of* _ $Pow^M(A\times A)$]
  by *simp*

end — *M_pre_cardinal_arith*

locale *M_cardinal_arith* = *M_pre_cardinal_arith* +
  assumes
    is_ordertype_replacement :
    strong_replacement(M,λ x z . $\exists$ y[M]. *is_order_body*(M,x,y) $\wedge$ z=$\langle x,y\rangle$)
    and
     pow_rel_separation : $\forall$ A[M]. *separation*(M, λy. $\exists$ x[M]. $x \in A \wedge y = \langle x, Pow^M(x)\rangle$)
    and
    separation_is_well_ord : *separation*(M, λx . *is_well_ord*(M,*fst*(x),*snd*(x)))
begin

lemmas *Pow_rel_replacement* = *lam_replacement_Pow_rel*[*OF pow_rel_separation*]

lemma *ordtype_replacement* :
strong_replacement(M , $\lambda x\, z$. (*snd*(x) $\in$ Pow^M(*fst*(x)$\times$*fst*(x)) $\wedge$ *well_ord*(*fst*(x),*snd*(x)))
$\wedge$
z =$\langle x$, *ordertype*(*fst*(x),*snd*(x))$\rangle$)
using *strong_replacement_cong*[*THEN iffD1*,*OF _ is_ordertype_replacement*]
is_order_body_abs
by *simp*

lemma *separation_well_ord* : *separation*(M, λx . *well_ord*(*fst*(x),*snd*(x)))
using *separation_cong*[*THEN iffD1*] *separation_is_well_ord is_well_ord_iff_wellordered*
well_ord_abs
by *simp*

lemma *jump_cardinal_body_lam_replacement* :
shows *lam_replacement*(M, λX .*jump_cardinal_body*(Pow^M(X $\times$ X),X)) **and**
$M(X) \Longrightarrow M$(*jump_cardinal_body*(Pow^M(X $\times$ X), X))
proof -
define *WO* **where** *WO* $\equiv$ λX . {$r\in Pow^M(X\times X)$. *well_ord*(X,r)}
define *lam_jump_cardinal_body* **where**
lam_jump_cardinal_body $\equiv$ λX . $\lambda r\in$ *WO*(X) . *ordertype*(X,r)
have *lam_jump_cardinal_body*(X) = {$\langle r$,*ordertype*(X,r)$\rangle$. r $\in$ *WO*(X)} **for** X
unfolding *lam_jump_cardinal_body_def WO_def lam_def*
by *simp*
then
have *jump_cardinal_body*(Pow^M(X $\times$ X),X) = {*snd*(p) . p $\in$ *lam_jump_cardinal_body*(X)}
if $M(X)$ **for** X
unfolding *jump_cardinal_body_def WO_def lam_def*
by *force*
moreover
have *lam_replacement*(M, λx. {y . r $\in$ Pow^M(x $\times$ x), (r $\in$ Pow^M(x $\times$ x) $\wedge$
well_ord(x, r)) $\wedge$ y = *ordertype*(x, r)})
(**is** *lam_replacement*(M,*?q*))
using *Pow_rel_replacement*[*THEN* [*2*] *lam_replacement_hcomp*]
lam_replacement_CartProd lam_replacement_identity lam_replacement_fst
lam_replacement_snd
fst_closed[*OF transM*] *snd_closed*[*OF transM*]
separation_well_ord separation_in separation_conj transM[*of _* Pow^M(_$\times$_)]
strong_lam_replacement_imp_lam_replacement_RepFun[*OF ordtype_replacement*,
where g=λX. Pow^M(X $\times$ X)]
by *force*
moreover
have M(*?q*(x)) **if** $M(x)$ **for** x
using *that Pow_rel_replacement*[*THEN* [*2*] *lam_replacement_hcomp*]
lam_replacement_CartProd lam_replacement_identity
ordertype_closed transM[*of _* Pow^M(_$\times$_)]
fst_closed[*OF transM*] *snd_closed*[*OF transM*]
by(*rule_tac strong_lam_replacement_imp_strong_replacement*[*OF ordtype_replacement*,*THEN*

strong_replacement_closed],
auto)
moreover
have *?q*(*x*)={*snd*(*p*) . *p* ∈ (λ*r*∈*WO*(*x*). *ordertype*(*x*, *r*))} **if** *M*(*x*) **for** *x*
using *that*
unfolding *lam_def WO_def*
by *force*
moreover from *calculation*
have *?q*(*X*) = *jump_cardinal_body*(Pow^M(*X* × *X*), *X*) **if** *M*(*X*) **for** *X*
using *that*
unfolding *lam_jump_cardinal_body_def*
by *simp*
moreover from *calculation*
have *M*(*jump_cardinal_body*(Pow^M(*X* × *X*), *X*)) **if** *M*(*X*) **for** *X*
using *that* **by** *simp*
moreover from *calculation*
have *1*:*lam_replacement*(*M*, λ*x* . {*snd*(*p*) . *p* ∈ *lam_jump_cardinal_body*(*x*)})
using *lam_replacement_cong*
by *auto*
ultimately
show *lam_replacement*(*M*, λ*X* .*jump_cardinal_body*(Pow^M(*X* × *X*),*X*)) **and**
M(*X*) ⟹ *M*(*jump_cardinal_body*(Pow^M(*X* × *X*), *X*))
using *lam_replacement_imp_strong_replacement*[*OF lam_replacement_snd*]
lam_replacement_cong[*OF 1*,*unfolded lam_jump_cardinal_body_def*]
by *simp_all*
qed

lemmas *jump_cardinal_body_closed* = *jump_cardinal_body_lam_replacement*(*2*)

lemma *jump_cardinal_closed*:
assumes *M*(*K*)
shows *M*({*jump_cardinal_body*(Pow^M(*X* × *X*),*X*) . *X* ∈ Pow^M(*K*)})
using *assms jump_cardinal_body_lam_replacement lam_replacement_imp_strong_replacement*
transM[*of* _ Pow^M(*K*)] *RepFun_closed*
by *auto*

end — *M_cardinal_arith*

context *M_pre_cardinal_arith*
begin

lemma *ordermap_eqpoll_pred*:
[| *well_ord*(*A*,*r*); *x* ∈ *A* ; *M*(*A*);*M*(*r*);*M*(*x*)|] ==> *ordermap*(*A*,*r*)‘*x* $\approx^M$ *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 $|ordermap(K \times K,\ csquare_rel(K))\ `\ \langle x,y\rangle|^M \leq |succ(succ(x \cup y))|^M \otimes^M |succ(succ(x \cup y))|^M$
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 $ordermap(K \times K,\ csquare_rel(K))\ `\ \langle x,y\rangle \lesssim^M ordermap(K \times K,\ csquare_rel(K))\ `\ \langle ?z,?z\rangle$
by (*blast intro*: *ordermap_z_lt leI le_imp_lepoll_rel K x y types*)
also have ... $\approx^M$ $Order.pred(K \times K,\ \langle ?z,?z\rangle,\ csquare_rel(K))$
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 ... $\lesssim^M succ(?z) \times succ(?z)$ **using** zK
by (*rule_tac pred_csquare_subset* [*THEN subset_imp_lepoll_rel*]) (*simp_all add*:*types*)
also have ... $\approx^M |succ(?z)|^M \times |succ(?z)|^M$ **using** oz
by (*blast intro*: *prod_eqpoll_rel_cong Ord_cardinal_rel_eqpoll_rel eqpoll_rel_sym types*)
finally show $ordermap(K \times K,\ csquare_rel(K))\ `\ \langle x,y\rangle \lesssim^M |succ(?z)|^M \times |succ(?z)|^M$
by (*simp_all add*:*types Mom*)
from *Mom*
show $M(ordermap(K \times K,\ csquare_rel(K))\ `\ \langle x,\ y\rangle)$ **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*: $\bigwedge y.\ y{\in}K \Longrightarrow InfCard^M(y) \Longrightarrow M(y) \Longrightarrow y \otimes^M y = y$
— Note the weakened hypothesis *eq*
and *types*: $M(K)$
shows $ordertype(K{*}K,\ csquare_rel(K)) \leq K$
proof -
have *CK*: $Card^M(K)$ **using** *IK* **by** (*rule_tac InfCard_rel_is_Card_rel*) (*simp_all add*:*types*)
hence *OK*: $Ord(K)$ **by** (*rule Card_rel_is_Ord*) (*simp_all add*:*types*)
moreover have $Ord(ordertype(K \times K,\ csquare_rel(K)))$ **using** *OK*
by (*rule well_ord_csquare* [*THEN Ord_ordertype*])
ultimately show *?thesis*
proof (*rule all_lt_imp_le*)
fix *i*
assume $i{:}i < ordertype(K \times K,\ csquare_rel(K))$
hence *Oi*: $Ord(i)$ **by** (*elim ltE*)
obtain *x y* **where** *x*: $x \in K$ **and** *y*: $y \in K$
and *ieq*: $i = ordermap(K \times K,\ csquare_rel(K))$ ‘ $\langle x,y\rangle$
using *i* **by** (*auto simp add*: *ordertype_unfold elim*: *ltE*)
hence *xy*: $Ord(x)$ $Ord(y)$ $x < K$ $y < K$ **using** *OK*
by (*blast intro*: *Ord_in_Ord ltI*)+
hence *ou*: $Ord(x \cup y)$
by (*simp*)
from *OK types*
have $M(ordertype(K \times K,\ csquare_rel(K)))$
using *well_ord_csquare* **by** *fastforce*
with *i x y types*
have *types′*: $M(K)$ $M(i)$ $M(x)$ $M(y)$
using *types* **by** (*auto dest*:*transM ltD*)
show $i < K$
proof (*rule Card_rel_lt_imp_lt* [*OF _ Oi CK*])
have $|i|^M \leq |succ(succ(x \cup y))|^M \otimes^M |succ(succ(x \cup y))|^M$ **using** *IK xy*
by (*auto simp add*: *ieq types intro*: *InfCard_rel_is_Limit* [*THEN ordermap_csquare_le*] *types′*)
moreover have $|succ(succ(x \cup y))|^M \otimes^M |succ(succ(x \cup y))|^M < K$
proof (*cases rule*: *Ord_linear2* [*OF ou Ord_nat*])
assume $x \cup y < nat$
hence $|succ(succ(x \cup y))|^M \otimes^M |succ(succ(x \cup y))|^M \in nat$
by (*simp add*: *lt_def nat_cmult_rel_eq_mult nat_succI nat_into_Card_rel* [*THEN Card_rel_cardinal_rel_eq*] *types′*)
also have ... $\subseteq K$ **using** *IK*
by (*simp add*: *InfCard_rel_def le_imp_subset types*)
finally show $|succ(succ(x \cup y))|^M \otimes^M |succ(succ(x \cup y))|^M < K$
by (*simp add*: *ltI OK*)
next

assume *natxy*: $nat \leq x \cup y$
hence *seq*: $|succ(succ(x \cup y))|^M = |x \cup y|^M$ **using** *xy*
by (*simp add*: *le_imp_subset nat_succ_eqpoll_rel* [*THEN cardinal_rel_cong*] *le_succ_iff types'*)
also have ... $< K$ **using** *xy*
by (*simp add*: *Un_least_lt Ord_cardinal_rel_le* [*THEN lt_trans1*] *types'*)
finally have $|succ(succ(x \cup y))|^M < K$ **.**
moreover have $InfCard^M(|succ(succ(x \cup y))|^M)$ **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 $K \otimes^M K = K$
proof -
have *OK*: $Ord(K)$ **using** *IK* **by** (*simp add*: *Card_rel_is_Ord InfCard_rel_is_Card_rel types*)
from *OK assms*
show $K \otimes^M K = K$
proof (*induct rule*: *trans_induct*)
case (*step i*)
note *types* = ‹$M(K)$› ‹$M(i)$›
show $i \otimes^M i = i$
proof (*rule le_anti_sym*)
from *step types*
have *Mot*:$M(ordertype(i \times i, csquare_rel(i)))$ $M(ordermap(i \times i, csquare_rel(i)))$
using *well_ord_csquare Limit_is_Ord* **by** *simp_all*
then
have $|i \times i|^M = |ordertype(i \times i, csquare_rel(i))|^M$
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 $i \otimes^M i \leq ordertype(i \times i, csquare_rel(i))$
by (*simp add*: *step.hyps cmult_rel_def Ord_cardinal_rel_le well_ord_csquare* [*THEN Ord_ordertype*] *types*)
moreover
have $ordertype(i \times i, csquare_rel(i)) \leq i$ **using** *step*
by (*rule_tac ordertype_csquare_le_M*) (*simp add*: *types*)
ultimately show $i \otimes^M i \leq i$ **by** (*rule le_trans*)
next
show $i \leq i \otimes^M i$ **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 ... $\approx^M A$
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 $||A|^M \times |A|^M|^M = |A|^M$ **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*: [| $Card^M(i)$; ~ *Finite_rel*(*M*,*i*) ; *M*(*i*) |] ==> $InfCard^M(i)$
by (*simp add*: *InfCard_rel_def Card_rel_is_Ord* [*THEN nat_le_infinite_Ord*])

23.6.1 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*: [| $InfCard^M(K)$; $InfCard^M(L)$; $M(K)$; $M(L)$ |] ==> $K \otimes^M L = K \cup L$
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*: $InfCard^M(K) \Longrightarrow M(K) \Longrightarrow K \oplus^M K = K$
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*: [| $InfCard^M(K)$; $L \leq K$; $M(K)$; $M(L)$|] ==> $K \oplus^M L = K$
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*: [| $InfCard^M(K)$; $InfCard^M(L)$; $M(K)$; $M(L)$ |] ==> $K \oplus^M L = K \cup L$
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_pre_cardinal_arith*

23.7 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 *def_jump_cardinal_rel_aux*:
$X \in Pow^M(K) \Longrightarrow M(K) \Longrightarrow$
$\{z\ .\ r \in Pow^M(X \times X),\ M(z) \wedge well_ord(X,\ r) \wedge z = ordertype(X,\ r)\} =$
$\{z\ .\ r \in Pow^M(K \times K),\ M(z) \wedge well_ord(X,\ r) \wedge z = ordertype(X,\ r)\}$
proof(*rule*,*auto simp*:*Pow_rel_char dest*:*transM*)
let $?L=\{z\ .\ r \in Pow^M(X \times X),\ M(z) \wedge well_ord(X,\ r) \wedge z = ordertype(X,\ r)\}$
let $?R=\{z\ .\ r \in Pow^M(K \times K),\ M(z) \wedge well_ord(X,\ r) \wedge z = ordertype(X,\ r)\}$
show $ordertype(X,\ r) \in \{y\ .\ x \in \{x \in Pow(X \times X)\ .\ M(x)\},\ M(y) \wedge well_ord(X,\ x) \wedge y = ordertype(X,\ x)\}$
if $M(K)$ $M(r)$ $r \subseteq K \times K$ $X \subseteq K$ $M(X)$ $well_ord(X,r)$ **for** r
proof -
from *that*
have $ordertype(X,r) = ordertype(X,r \cap X \times X)$ $(r \cap X \times X) \subseteq X \times X$ $M(r \cap X \times X)$ $well_ord(X,r \cap X \times X)$ $wellordered(M,X,r \cap X \times X)$
using *well_ord_restr ordertype_restr_eq* **by** *auto*
moreover from *this*
have $ordertype(X,r \cap X \times X) \in ?L$
using *that Pow_rel_char*
ReplaceI[*of* $\lambda\ z\ r\ .\ M(z) \wedge well_ord(X,\ r) \wedge z = ordertype(X,\ r)$ $ordertype(X,r \cap X \times X)$]
by *auto*
ultimately
show *?thesis*
using *Pow_rel_char* **by** *auto*
qed
qed

lemma *def_jump_cardinal_rel_aux2*:
assumes $X \in Pow^M(K)$ $M(K)$
shows $jump_cardinal_body(Pow^M(K \times K),X) = jump_cardinal_body(Pow^M(X \times X),X)$
proof -
from *assms*
have $jump_cardinal_body(Pow^M(K \times K),X) = \{z\ .\ r \in Pow^M(K \times K),\ M(z) \wedge well_ord(X,\ r) \wedge z = ordertype(X,\ r)\}$
using *def_jump_cardinal_body transM*[*of* _ $Pow^M(K)$]
by *simp*
also from *assms*
have ... $= \{z\ .\ r \in Pow^M(X \times X),\ M(z) \wedge well_ord(X,\ r) \wedge z = ordertype(X,\ r)\}$
using *def_jump_cardinal_rel_aux transM*[*of* _ $Pow^M(K)$]
by *simp*
also from *assms*
have ... $= jump_cardinal_body(Pow^M(X \times X),X)$

using *def_jump_cardinal_body transM*[*of* _ $Pow^M(K)$] **by** *auto*
finally
show *?thesis* **.**
qed

lemma *def_jump_cardinal_rel*:
assumes $M(K)$
shows *jump_cardinal_rel*(M,K) =
$(\bigcup\{$*jump_cardinal_body*$(Pow^M(K*K),X)$. $X\in Pow^M(K)\})$
proof -
have *jump_cardinal_body*$(Pow^M(X \times X),X) = \{z$. $r \in Pow^M(X \times X)$, $M(z)$ $\wedge$ *well_ord*(X, r) $\wedge$ z=*ordertype*$(X, r)\}$
if $M(X)$ **for** X
using *that ordertype_closed transM*[*of* _ $Pow^M(X\times X)$]
unfolding *jump_cardinal_body_def*
by(*intro equalityI subsetI*,*auto*)
then
have $M(\{z$. $r \in Pow^M(X \times X)$, $M(z)$ $\wedge$ *well_ord*(X, r) $\wedge$ z = *ordertype*$(X, r)\})$
if $M(Y)$ $X \in Pow^M(Y)$ **for** Y X
using *jump_cardinal_body_closed*[*of X*] *that transM*[*of* _ $Pow^M(Y)$]
ordertype_closed transM[*of* _ $Pow^M(X\times X)$]
unfolding *jump_cardinal_body_def*
by *auto*
moreover from ‹$M(K)$›
have $R \in Pow^M(X \times X) \Longrightarrow X \in Pow^M(K) \Longrightarrow R \in Pow^M(K \times K)$ **for** X R
using *mem_Pow_rel_abs transM*[*OF* _ *Pow_rel_closed*, *of R* $X\times X$]
transM[*OF* _ *Pow_rel_closed*, *of X K*] **by** *auto*
ultimately
show *?thesis*
using *assms is_ordertype_iff*
ordertype_rel_abs transM[*of* _ $Pow^M(K)$] *transM*[*of* _ $Pow^M(K\times K)$]
def_jump_cardinal_rel_aux jump_cardinal_body_closed ordertype_closed
unfolding *jump_cardinal_rel_def jump_cardinal_body_def*
apply(*intro equalityI subsetI*,*auto dest*:*transM*[*of* _ $Pow^M(K)$],*rename_tac y r*)
by(*rule_tac x*=$\{z$. $xa \in Pow^M(K \times K)$, $M(z)$ $\wedge$ *well_ord*(y, xa) $\wedge$ z = *ordertype*$(y, xa)\}$ **in** *bexI*,*auto*)
qed

lemma *Ord_jump_cardinal_rel*: $M(K) \Longrightarrow$ *Ord*(*jump_cardinal_rel*(M,K))
apply (*unfold def_jump_cardinal_rel jump_cardinal_body_def*)
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 ∈ Pow^M(K) r ∈ Pow^M(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) ∈ Pow^M(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 jump_cardinal_body_def)
  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 *def_jump_cardinal_rel'*:
assumes $M(K)$
shows *jump_cardinal_rel*(M,K) =
$(\bigcup X{\in}Pow^M(K).\ \{z.\ r \in Pow^M(X{\times}X),\ well_ord(X,r) \wedge z = ordertype(X,r)\})$
proof -
from *assms*
have *jump_cardinal_rel*$(M, K) = (\bigcup X{\in}Pow^M(K).\ jump_cardinal_body(Pow^M(K \times K), X))$
using *def_jump_cardinal_rel*
by *simp*
also from ‹$M(K)$›
have ... $= (\bigcup X{\in}Pow^M(K).\ jump_cardinal_body(Pow^M(X \times X), X))$
using *def_jump_cardinal_rel_aux2*
by *simp*
finally
show *?thesis*
unfolding *jump_cardinal_body_def* **by** *simp*
qed

rel_closed for *jump_cardinal*
proof -
fix K
assume $M(K)$
moreover from *this*
have *jump_cardinal_rel*(M, K) =
$\bigcup\{jump_cardinal_body(Pow^M(X \times X), X)\ .\ X{\in}Pow^M(K)\}$
using *def_jump_cardinal_rel'*
unfolding *jump_cardinal_body_def*[*symmetric*]
by *simp*
moreover from *calculation*
have $M((\bigcup\{jump_cardinal_body(Pow^M(X \times X), X)\ .\ X{\in}Pow^M(K)\}))$
using *def_jump_cardinal_rel_aux jump_cardinal_closed*
by *simp*
ultimately
show $M(jump_cardinal_rel(M,K))$
by *simp*
qed

lemma *Card_rel_jump_cardinal_rel_lemma*:
[| $well_ord(X,r)$; $r \subseteq K * K$; $X \subseteq K$;
$f \in bij(ordertype(X,r), jump_cardinal_rel(M,K))$;
$M(X)$; $M(r)$; $M(K)$; $M(f)$ |]
==> $jump_cardinal_rel(M,K) \in jump_cardinal_rel(M,K)$
apply (*subgoal_tac f O ordermap* $(X,r) \in bij\ (X, jump_cardinal_rel\ (M,K))$)
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*: $M(K) \Longrightarrow$ *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

23.8 Basic Properties of Successor Cardinals

lemma *csucc_rel_basic*: *Ord*(K) ==> $M(K) \Longrightarrow$ *Card_rel*(M,*csucc_rel*(M,K)) & $K <$ *csucc_rel*(M,K)
apply (*unfold csucc_rel_def*)
apply (*rule LeastI*[*of* $\lambda i.\ M(i) \wedge$ *Card_rel*(M,i) $\wedge\ K < i$, *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) ==> $M(K) \Longrightarrow 0 <$ *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) $\leq L$
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) $\longleftrightarrow |i|^M \leq K$
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*:
[| $Card_rel(M,K')$; $Card_rel(M,K)$; $M(K')$; $M(K)$ |] ==> $K' < csucc_rel(M,K)$
$\longleftrightarrow K' \leq K$
by (*simp add*: *lt_csucc_rel_iff Card_rel_cardinal_rel_eq Card_rel_is_Ord*)

lemma *InfCard_rel_csucc_rel*: $InfCard_rel(M,K) \Longrightarrow M(K)$ ==> $InfCard_rel(M,csucc_rel(M,K))$
by (*simp add*: *InfCard_rel_def Card_rel_csucc_rel Card_rel_is_Ord*
lt_csucc_rel [*THEN leI*, *THEN* [*2*] *le_trans*])

23.8.1 Theorems by Krzysztof Grabczewski, proofs by lcp

lemma *nat_sum_eqpoll_rel_sum*:
assumes *m*: $m \in nat$ **and** *n*: $n \in nat$ **shows** $m + n \approx^M m +_\omega n$
proof -
have $m + n \approx^M |m+n|^M$ **using** *m n*
by (*blast intro*: *nat_implies_well_ord well_ord_radd well_ord_cardinal_rel_eqpoll_rel eqpoll_rel_sym*)
also have ... $= m +_\omega n$ **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*: [| $Ord(i)$; $i \subseteq nat$ |] ==> $Card^M(i)$
by (*blast dest*: *Ord_subset_natD intro*: *Card_rel_nat nat_into_Card_rel*)

end — *M_cardinal_arith_jump*

end
theory *Aleph_Relative*
imports
CardinalArith_Relative
begin

definition
HAleph :: $[i,i] \Rightarrow i$ **where**
$HAleph(i,r) \equiv if(\neg(Ord(i)),i,if(i=0,\ nat,\ if(\neg Limit(i) \wedge i\neq 0,$
$csucc(r`(\bigcup\ i\ )),$
$\bigcup j\in i.\ r`j)))$

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 intermediate for *is_HAleph_fm*

lemma *arity_is_HAleph_fm_aux*:
  assumes
    $i \in nat$ $r \in nat$
    — NOTE: assumptions are **not** used, but if omitted, next lemma fails!
  shows
    $arity(Replace_fm(8 +_\omega i, \cdot 10 +_\omega r`0 \text{ is } 1\cdot, 3)) = 9 +_\omega i \cup pred(pred(11 +_\omega r))$
  using $arity_Replace_fm[of \cdot (10+_\omega r)`0 \text{ is } 1\cdot\ 8+_\omega i\ 3\ (11+_\omega r) \cup 1 \cup 2]$
    ord_simp_union
  by (*auto simp*:*arity*)

lemma *arity_is_HAleph_fm*[*arity*]:
  assumes
    $i \in nat$ $r \in nat$ $l \in nat$
  shows
    $arity(is_HAleph_fm(i, r, l)) = succ(i) \cup succ(l) \cup succ(r)$
  using *assms pred_Un arity_is_HAleph_fm_aux arity_is_HAleph_fm'*
  by *auto*

definition
  $Aleph' :: i => i$ **where**
  $Aleph'(a) == transrec(a,\lambda i\ r.\ HAleph(i,r))$

relativize functional *Aleph' Aleph_rel*
relationalize *Aleph_rel is_Aleph*

The extra assumptions $a < length(env)$ and $c < length(env)$ in this schematic goal (and the following results on synthesis that depend on it) are imposed by *is_transrec_iff_sats*.

schematic_goal *sats_is_Aleph_fm_auto*:
  $a \in nat \Longrightarrow c \in nat \Longrightarrow env \in list(A) \Longrightarrow$
  $a < length(env) \Longrightarrow c < length(env) \Longrightarrow 0 \in A \Longrightarrow$
  $is_Aleph(\#\#A, nth(a, env), nth(c, env)) \longleftrightarrow A, env \models ?fm(a, c)$
  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 $?env' = Cons(a0, Cons(a1, Cons(a2, Cons(a3, Cons(a4, Cons(a5, Cons(a6, Cons(a7, env))))))))$
  show $nth(2, ?env') = a2$
    $nth(1, ?env') = a1$
    $nth(0, ?env') = a0$
    $nth(c, env) = nth(c, env)$
    by *simp_all*

qed *simp_all*

synthesize_notc *is_Aleph* **from_schematic**

notation *is_Aleph_fm* (‹·ℵ′(_′) *is* _·›)

lemma *is_Aleph_fm_type* [*TC*]: $a \in nat \Longrightarrow c \in nat \Longrightarrow is_Aleph_fm(a, c) \in formula$
  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* ‹$M(a)$›] *assms*
  by *auto*

lemma *HAleph_eq_Aleph_recursive*:
  $Ord(i) \Longrightarrow HAleph(i,r) = (if\ i = 0\ then\ nat$
    $else\ if\ \exists j.\ i = succ(j)\ then\ csucc(r\ `\ (THE\ j.\ i = succ(j)))\ else\ \bigcup j{<}i.\ r\ `\ j)$
proof -
  assume $Ord(i)$
  moreover from *this*
  have $i = succ(j) \Longrightarrow (\bigcup succ(j)) = j$ **for** j
    using *Ord_Union_succ_eq* **by** *simp*
  moreover from ‹$Ord(i)$›
  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*: $Ord(a) \Longrightarrow Aleph'(a) = Aleph(a)$
  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 :: $[i,i\Rightarrow o] \Rightarrow i$ (‹ℵ_–›) **where**
  $Aleph_r(a,M) \equiv Aleph_rel(M,a)$

abbreviation
  Aleph_r_set :: $[i,i] \Rightarrow i$ (‹ℵ_–›) **where**
  $Aleph_r_set(a,M) \equiv Aleph_rel(\#\#M,a)$

lemma *Aleph_rel_def′*: $Aleph_rel(M,a) \equiv transrec(a,\ \lambda i\ r.\ HAleph_rel(M,\ i,\ r))$
  unfolding *Aleph_rel_def* **.**

lemma *succ_mem_Limit*: $Limit(j) \Longrightarrow i \in j \Longrightarrow succ(i) \in j$
  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: $M(a) \Longrightarrow transrec_replacement(M,is_HAleph(M),a)$

begin

lemma *aux_ex_Replace_funapply*:
  assumes $M(a)$ $M(f)$
  shows $\exists x[M].\ is_Replace(M,\ a,\ \lambda j\ y.\ f\ `\ j = y,\ x)$
proof -
  have $\{f`j\ .\ j\in a\} = \{y\ .\ j\in a\ ,\ f\ `\ j=y\}$
    $\{y\ .\ j\in a\ ,\ f\ `\ j=y\} = \{y\ .\ j\in a\ ,\ y = f\ `\ j\}$
    by *auto*
  moreover
  note *assms*
  moreover from *calculation*
  have $x \in a \Longrightarrow y = f\ `x \Longrightarrow M(y)$ **for** $x\ y$
    using *transM*[*OF _* ‹$M(a)$›] **by** *auto*
  moreover from *assms*
  have $M(\{f`j\ .\ j\in a\})$
    using *transM*[*OF _* ‹$M(a)$›] *RepFun_closed*[*OF apply_replacement*] **by** *simp*
  ultimately
  have *2*:$is_Replace(M,\ a,\ \lambda j\ y.\ y = f\ `\ j,\ \{f`j\ .\ j\in a\})$
    using *Replace_abs*[*of _ _* $\lambda j\ y.\ y = f\ `\ j$,*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 is_HAleph(M,0,f,res) ⟷ res = 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) Ord(x) M(res)
  shows is_HAleph(M,succ(x),f,res) ⟷ res = 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) Limit(x) M(res)
  shows is_HAleph(M,x,f,res) ⟷ res = (⋃{y . i∈x ,M(i) ∧ M(y) ∧ y = f‘i})
proof -
  from assms
  have univalent(M, x, λj y. y = f ‘ j )
    (⋀xa y. xa ∈ x ⟹ f ‘ xa = y ⟹ M(y))
    {y . x ∈ x, f ‘ x = y} = {y . i∈x ,M(i) ∧ M(y) ∧ y = f‘i}
    using univalent_triv[of M x λj .f ‘ j] transM[OF _ ‹M(x)›]
    by auto
  moreover
  from this
  have univalent(M, x, λj y. f ‘ j = y )
     by (rule_tac univalent_cong[of x x M  λj y. y = f ‘ j  λj y. f ‘ j=y,THEN
iffD1], auto)
  moreover
  from this
  have univalent(M, x, λj y. M(j) ∧ M(y) ∧ 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(res)
  shows is_HAleph(M, a, f, res) ⟷ res = HAleph_rel(M, a, f)
proof(cases Ord(a))
  case True
  note Ord_cases[OF ‹Ord(a)›]
  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 ¬Limit(a) a≠0 ⋀ x . Ord(x) ⟹ a≠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(HAleph_rel(M,a,f))
  unfolding HAleph_rel_def
  using assms apply_replacement
  by simp

lemma Aleph_rel_closed[intro, simp]:
```

assumes $Ord(a)$ $M(a)$
shows $M(Aleph_rel(M,a))$
proof -
have $relation2(M,\ is_HAleph(M),\ HAleph_rel(M))$
unfolding *relation2_def* **using** *is_HAleph_iff assms* **by** *simp*
moreover
have $\forall x[M].\ \forall g[M].\ function(g) \longrightarrow M(HAleph_rel(M,\ x,\ g))$
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*: ${\aleph_0}^M = nat$
using *def_transrec* [*OF Aleph_rel_def'*,*of _ 0*]
unfolding *HAleph_rel_def* **by** *simp*

lemma *Aleph_rel_succ*: $Ord(\alpha) \Longrightarrow M(\alpha) \Longrightarrow {\aleph_{succ(\alpha)}}^M = ({\aleph_\alpha}^{M+})^M$
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 _* ‹$M(\alpha)$›]
from ‹$M(\alpha)$›
have ${\aleph_\alpha}^M = HAleph_rel(M,\ \alpha,\ \lambda x{\in}\alpha.\ {\aleph_x}^M)$
using *def_transrec* [*OF Aleph_rel_def'*,*of M* α] **by** *simp*
also
have ... $= \bigcup\{a\ .\ j \in \alpha,\ M(a) \wedge a = {\aleph_j}^M\}$
unfolding *HAleph_rel_def*
using *assms zero_not_Limit Limit_is_Ord trans* **by** *auto*
also
have ... $= \bigcup\{{\aleph_j}^M\ .\ j \in \alpha\}$
using *Aleph_rel_closed*[*OF _ trans*] *Ord_in_Ord Limit_is_Ord*[*OF* ‹$Limit(\alpha)$›]
by *auto*
finally
show *?thesis* **.**
qed

lemma *is_Aleph_iff*:
assumes $Ord(a)$ $M(a)$ $M(res)$
shows $is_Aleph(M,\ a,\ res) \longleftrightarrow res = {\aleph_a}^M$

```
proof -
  have relation2(M, is_HAleph(M), HAleph_rel(M))
    unfolding relation2_def using is_HAleph_iff assms by simp
  moreover
  have ∀x[M]. ∀g[M]. function(g) ⟶ M(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_separation: Ord(x) ⟹ M(x) ⟹ separation(M, λy. ∃z∈x. y = ℵ_z^M)
begin

lemma Aleph_rel_cont: Limit(l) ⟹ M(l) ⟹ ℵ_l^M = (⋃i<l. ℵ_i^M)
  using Limit_is_Ord Aleph_rel_limit
  by (simp add:OUnion_def)

lemma Ord_Aleph_rel:
  assumes Ord(a)
  shows M(a) ⟹ Ord(ℵ_a^M)
  using ‹Ord(a)›
proof(induct a rule:trans_induct3)
  case 0
  show ?case using Aleph_rel_zero by simp
next
  case (succ x)
  with ‹Ord(x)›
  have M(x) Ord(ℵ_x^M) by simp_all
  with ‹Ord(x)›
  have Ord(csucc_rel(M,ℵ_x^M))
    using Card_rel_is_Ord Card_rel_csucc_rel
    by simp
  with ‹Ord(x)› ‹M(x)›
  show ?case using Aleph_rel_succ by simp
next
  case (limit x)
  note trans=transM[OF _ ‹M(x)›]
  from limit
  have ℵ_x^M = (⋃i∈x. ℵ_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 Aleph_rel_increasing:
  assumes a < b and types: M(a) M(b)
  shows ℵ_a^M < ℵ_b^M
proof -
  { fix x
    from assms
    have Ord(b)
      by (blast intro: lt_Ord2)
    moreover
    assume M(x)
    moreover
    note ‹M(b)›
    ultimately
    have x < b ⟹ ℵ_x^M < ℵ_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 _ ℵ_b^M csucc^M(ℵ_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 ℵ_x^M < (⋃j<l. ℵ_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 ‹x<l› ‹Limit(l)›
        have Ord(x)
          using Limit_is_Ord Ord_in_Ord
          by (auto dest!:ltD)
        with ‹M(x)›
        show ℵ_x^M < ℵ_succ(x)^M
          using Card_rel_csucc_rel Ord_Aleph_rel lt_csucc_rel
            ltD[THEN [2] Ord_in_Ord] succ_in_MI[OF ‹M(x)›]
            Aleph_rel_succ[of x]
          by (simp)
      next
        from ‹M(l)› ‹Limit(l)›
        show Ord(⋃j<l. ℵ_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

lemma Card_rel_Aleph_rel [simp, intro]:
  assumes Ord(a) and types: M(a) shows Card^M(ℵ_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 Ord(x)
    using Limit_is_Ord by simp
  from this
  have {y . z ∈ x, M(y) ∧ M(z) ∧ y = ℵ_z^M} = {y . z ∈ x, M(y) ∧ M(z) ∧ Ord(z)
∧ y = ℵ_z^M}
    using Ord_in_Ord by simp
  moreover from ‹Ord(x)›
  have {y . z ∈ x, M(y) ∧ M(z) ∧ Ord(z) ∧ y = ℵ_z^M} = {y . z ∈ x, M(z) ∧ y =
ℵ_z^M}
    using Ord_in_Ord by blast
  moreover from ‹Ord(x)› ‹M(x)›
  have {y . z ∈ x, M(z) ∧ y = ℵ_z^M} ⊆ ℵ_x^M
    using Aleph_rel_increasing
    by (auto dest:ltD transM intro:ltI)
  with calculation
  have {y . z ∈ x, M(z) ∧ y = ℵ_z^M} = {y ∈ ℵ_x^M . (∃ z ∈ x. y = ℵ_z^M)}
    by (blast dest:transM)
  with calculation
  have {y . z ∈ x, M(y) ∧ M(z) ∧ y = ℵ_z^M} = {y ∈ ℵ_x^M . (∃ z ∈ x. y = ℵ_z^M)}
    by simp
```

moreover from ‹$Ord(x)$› ‹$M(x)$›
have $M(\{y \in \aleph_x{}^M \,.\, (\exists\, z \in x.\ y = \aleph_z{}^M)\})$
using *aleph_rel_separation*
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

lemmas *nat_subset_Aleph_rel_1* =
Ord_lt_subset[*OF Ord_Aleph_rel*[*of 1*] *Aleph_rel_increasing*[*of 0 1*,*simplified*],*simplified*]

end — *M_aleph*

end

24 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* +
assumes
lam_replacement_minimum:*lam_replacement*(M, λp. *minimum*(*fst*(*p*),*snd*(*p*)))
begin

lemma *lam_replacement_minimum_vimage*:
$M(f) \Longrightarrow M(r) \Longrightarrow$ *lam_replacement*(M, λx. *minimum*(r, f -`` $\{x\}$))
using *lam_replacement_minimum lam_replacement_vimage_sing_fun lam_replacement_constant*
lam_replacement_identity lam_replacement_hcomp2[*of _ _ minimum*]
by *simp*

lemmas *surj_imp_inj_replacement4* = *lam_replacement_minimum_vimage*[*unfolded*
lam_replacement_def]

lemmas *surj_imp_inj_replacement* =
surj_imp_inj_replacement1 surj_imp_inj_replacement2 surj_imp_inj_replacement4
lam_replacement_vimage_sing_fun[*THEN lam_replacement_imp_strong_replacement*]

lemma *well_ord_surj_imp_lepoll_rel*:
  assumes *well_ord*(A,r) $h \in$ *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* ‹$h\in$_›]]
  from *assms*
  have ($\lambda b\in B$. *minimum*$(r, \{a\in A.\ h`a=b\}) \in$ *inj*(B,A) (**is** *?f*$\in$_)
    using *well_ord_surj_imp_inj_inverse assms*(*1*,*2*) **by** *simp*
  with *assms*
  have M(*?f*`b) **if** $b\in B$ **for** b
    using *apply_type*[*OF inj_is_fun*[*OF* ‹*?f*$\in$_›]] *that transM*[*OF* _ ‹$M(A)$›] **by** *simp*
  with *assms*
  have M(*?f*)
    using *lam_closed surj_imp_inj_replacement4 eq* **by** *auto*
  with ‹*?f*$\in$_› *assms*
  have *?f* $\in$ *inj*$^M(B,A)$
    using *mem_inj_abs* **by** *simp*
  with ‹M(*?f*)›
  show *?thesis* **unfolding** *lepoll_rel_def* **by** *auto*
qed

lemma *surj_imp_well_ord_M*:
  assumes *wos*: *well_ord*(A,r) $h \in$ *surj*(A,B)
    and
    types: $M(A)$ $M(r)$ $M(h)$ $M(B)$
  shows $\exists\, s[M].$ *well_ord*(B,s)
  using *assms lepoll_rel_well_ord*
    well_ord_surj_imp_lepoll_rel **by** *fast*

lemma *choice_ax_well_ord*: $M(S) \Longrightarrow \exists\, r[M].$ *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: $M(r) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ y.\ y = \langle x,$ *minimum*$(r, B(x))\rangle$)
begin

lemma *AC_M*:
assumes $a \in A\ \bigwedge x.\ x \in A \Longrightarrow \exists\, y.\ y \in B(x)$
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 ‹*well_ord*(_,r)› ‹$x{\in}A$›
have *well_ord*$(B(x),r)$ **using** *well_ord_subset UN_upper* **by** *simp*
with *assms* ‹$x{\in}A$› ‹$M(r)$›
show *?thesis* **using** *Pi_assumptions* **by** *blast*
qed
with *assms* **and** ‹$M(r)$›
have $M($*?f*$)$
using *Pi_assumptions minimum_replacement lam_closed*
by *simp*
moreover from *assms* **and** *calculation*
have *?f* $\in Pi^M(A,B)$
using *lam_type*[*OF minimum_in*, *OF* ‹*well_ord*$(\bigcup x{\in}A.\ B(x),r)$›, *of A B*]
Pi_rel_char **by** *auto*
ultimately
show *?thesis* **by** *blast*
qed

lemma *AC_Pi_rel*: **assumes** $\bigwedge x.\ x \in A \Longrightarrow \exists\, y.\ y \in B(x)$
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** $a \in A$ **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

24.1 Strengthened Forms of Existing Theorems on Cardinals

lemma *cardinal_rel_eqpoll_rel*: $M(A) \Longrightarrow |A|^M \approx^M A$
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*: $|X|^M = |Y|^M ==> M(X) \Longrightarrow M(Y) \Longrightarrow X \approx^M Y$
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*: $M(X) \Longrightarrow M(Y) \Longrightarrow |X|^M = |Y|^M \longleftrightarrow X \approx^M Y$
by (*blast intro*: *cardinal_rel_cong cardinal_rel_eqE*)

lemma *cardinal_rel_disjoint_Un*:
[| $|A|^M$=$|B|^M$; $|C|^M$=$|D|^M$; $A \cap C = 0$; $B \cap D = 0$; $M(A)$; $M(B)$; $M(C)$; $M(D)$|]
==> $|A \cup C|^M = |B \cup D|^M$
by (*simp add*: *cardinal_rel_eqpoll_rel_iff eqpoll_rel_disjoint_Un*)

lemma *lepoll_rel_imp_cardinal_rel_le*: $A \lesssim^M B ==> M(A) \Longrightarrow M(B) \Longrightarrow |A|^M \leq |B|^M$

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*: $[\![M(i);\ M(j);\ M(k)]\!] \Longrightarrow (i \oplus^M j) \oplus^M k = i \oplus^M (j \oplus^M k)$
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*: $[\![M(i);\ M(j);\ M(k)]\!] \Longrightarrow (i \otimes^M j) \otimes^M k = i \otimes^M (j \otimes^M k)$
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*: $[\![M(i);\ M(j);\ M(k)]\!] \Longrightarrow (i \oplus^M j) \otimes^M k = (i \otimes^M k) \oplus^M (j \otimes^M k)$
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*: $InfCard^M(|A|^M) \Longrightarrow M(A) \Longrightarrow A \times A \approx^M A$
apply (*rule choice_ax_well_ord* [*THEN rexE*]) **prefer** *2*
apply (*erule well_ord_InfCard_rel_square_eq*, *assumption*, *simp_all*)
done

24.2 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 ... $\lesssim^M |B|^M$
by (*rule le_imp_subset* [*THEN subset_imp_lepoll_rel*]) (*rule assms*, *simp_all add*:*types*)
also have ... $\approx^M B$
by (*rule cardinal_rel_eqpoll_rel*, *simp_all add*:*types*)
finally show *?thesis* **by** (*simp_all add*:*types*)

qed

lemma *le_Card_rel_iff*: $Card^M(K)$ ==> $M(K) \Longrightarrow M(A) \Longrightarrow |A|^M \leq K \longleftrightarrow A \lesssim^M K$
  apply (*erule Card_rel_cardinal_rel_eq* [*THEN subst*], *assumption*, *rule iffI*,
    erule Card_rel_le_imp_lepoll_rel, *assumption*+)
  apply (*erule lepoll_rel_imp_cardinal_rel_le*, *assumption*+)
  done

lemma *cardinal_rel_0_iff_0* [*simp*]: $M(A) \Longrightarrow |A|^M = 0 \longleftrightarrow A = 0$
  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*: $Ord(i)$ **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 ... $\approx^M A$
    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 ... $\approx^M |A|^M$
    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_lepoll_rel*: $i \leq |A|^M$ ==> $M(i) \Longrightarrow M(A) \Longrightarrow i \lesssim^M A$
  by (*blast intro*: *lt_Ord Card_rel_le_imp_lepoll_rel Ord_cardinal_rel_le le_trans*)

24.3 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*: $f \in surj^M(X,Y)$ **and**
    types: $M(f)$ $M(X)$ $M(Y)$
  shows $\exists\, g[M].\ g \in inj^M(Y,X)$
proof -
  from *types*
  interpret *M_Pi_assumptions_choice _ Y* $\lambda y.\ f\text{-``}\{y\}$

by *unfold_locales* (*auto intro*:*surj_imp_inj_replacement dest*:*transM*)
from *f AC_Pi_rel*
obtain *z* **where** *z*: $z \in Pi^M(Y, \lambda y.\ f\text{-``}\{y\})$
— 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 $z \in inj^M(Y, X)\ M(z)$
using *z 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*: $f \in surj^M(X,Y)$ **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 surj_rel_implies_inj_rel*]
obtain *g* **where** $g \in inj^M(Y,X)$
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 :: $i \Rightarrow o$ (‹$\mathcal{V}$›) **where**
$\mathcal{V}(x) \equiv True$

lemma *separation_absolute*: *separation*($\mathcal{V}$, *P*)
unfolding *separation_def*
by (*rule rallI*, *rule_tac x*={*x*∈_ . *P*(*x*)} **in** *rexI*) *auto*

lemma *univalent_absolute*:
assumes *univalent*($\mathcal{V}$, *A*, *P*) *P*(*x*, *b*) $x \in A$
shows $P(x, y) \Longrightarrow y = b$
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 *Least_closed*
    X_witness_in_M: $w \in X(x) \Longrightarrow M(x)$
    and
    lam_m_replacement:$M(f) \Longrightarrow$ *strong_replacement*(M,
      $\lambda x\ y.\ y = \langle x,\ \mu\ i.\ x \in X(i),\ f\ `\ (\mu\ i.\ x \in X(i))\ `\ x\rangle$)
    and
    inj_replacement:
    $M(x) \Longrightarrow$ *strong_replacement*(M, $\lambda y\ z.\ y \in inj^M(X(x),\ K) \wedge z = \{\langle x,\ y\rangle\}$)
    strong_replacement(M, $\lambda x\ y.\ y = inj^M(X(x),\ K)$)
    strong_replacement(M,
      $\lambda x\ z.\ z = Sigfun(x,\ \lambda i.\ inj^M(X(i),\ K))$)
    $M(r) \Longrightarrow$ *strong_replacement*(M,
      $\lambda x\ y.\ y = \langle x,\ minimum(r,\ inj^M(X(x),\ K))\rangle$)

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: $InfCard^M(K)$
  shows $(\bigwedge i.\ i{\in}K \Longrightarrow |X(i)|^M \le K) \Longrightarrow |\bigcup i{\in}K.\ X(i)|^M \le K$
proof (*simp add*: *K InfCard_rel_is_Card_rel le_Card_rel_iff Pi_assumptions*)
  have $M(f) \Longrightarrow M(\lambda x{\in}(\bigcup x{\in}K.\ X(x)).\ \langle \mu\ i.\ x \in X(i),\ f\ `\ (\mu\ i.\ x \in X(i))\ `\ x\rangle)$
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*]: $Ord(K)$ **by** (*blast intro*: *InfCard_rel_is_Card_rel Card_rel_is_Ord K types*)
  interpret *pii*:*M_Pi_assumptions_choice _ K* $\lambda i.\ inj^M(X(i),\ K)$
    using *inj_replacement Pi_assumptions transM*[*of _ K*]
    by *unfold_locales* (*simp_all del*:*mem_inj_abs*)
  assume *asm*:$\bigwedge i.\ i{\in}K \Longrightarrow X(i) \lesssim^M K$
  then
  have $\bigwedge i.\ i{\in}K \Longrightarrow M(inj^M(X(i),\ K))$
    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 *abs*:$inj_rel(\mathcal{V},x,y) = inj(x,y)$ **for** $x\ y$
    using *V.N.inj_rel_char* **by** *simp*
  from *asm*
  have $\bigwedge i.\ i{\in}K \Longrightarrow \exists f[M].\ f \in inj^M(X(i),\ K)$
    by (*simp add*: *types def_lepoll_rel*)
  then
  obtain f **where** $f \in (\prod i{\in}K.\ inj^M(X(i),\ K))\ M(f)$
    using *pii.AC_Pi_rel pii.Pi_rel_char* **by** *auto*
  with *abs*
  have f:$f \in (\prod i{\in}K.\ inj(X(i),\ K))$
    using *Pi_weaken_type*[*OF _ V.inj_rel_transfer*, *of f K X* $\lambda_.\ K$]
      Pi_assumptions **by** *simp*
  { fix z
    assume z: $z \in (\bigcup i{\in}K.\ X(i))$
    then obtain i **where** i: $i \in K\ Ord(i)\ z \in X(i)$
      by (*blast intro*: *Ord_in_Ord* [*of K*])
    hence $(\mu\ i.\ z \in X(i)) \le i$ **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 (⋃i∈K. X(i)) ≲^M K × K
  proof (simp add:types def_lepoll_rel)
    show ∃f[M]. f ∈ inj(⋃x∈K. X(x), K × K)
      apply (rule rexI)
       apply (rule_tac c = λz. ⟨μ i. z ∈ X(i), f ‘ (μ i. z ∈ X(i)) ‘ z⟩
          and d = λ⟨i,j⟩. converse (f‘i) ‘ j in lam_injective)
        apply (force intro: f inj_is_fun mems apply_type Perm.left_inverse)+
      apply (simp add:types ‹M(f)›)
      done
  qed
  also have ... ≈^M K
    by (simp add: K InfCard_rel_square_eq InfCard_rel_is_Card_rel
        Card_rel_cardinal_rel_eq types)
  finally have (⋃i∈K. X(i)) ≲^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
```

25 Relativization of Finite Functions

```
theory FiniteFun_Relative
  imports
    Lambda_Replacement
begin

lemma FiniteFunI :
  assumes f∈Fin(A×B) function(f)
  shows f ∈ A -||> B
  using assms
proof(induct)
  case 0
  then show ?case using emptyI by simp
next
  case (cons p f)
  moreover
  from assms this
  have fst(p)∈A snd(p)∈B function(f)
    using snd_type[OF ‹p∈_›] function_subset
    by auto
  moreover
  from ‹function(cons(p,f))› ‹p∉f› ‹p∈_›
  have fst(p)∉domain(f)
    unfolding function_def
```

```
    by force
  ultimately
  show ?case
    using consI[of fst(p) _ snd(p)]
    by auto
qed
```

25.1 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] ⇒ i (‹_<–› [100,1]100) where
  B^<α ≡ ⋃n∈α. (n→B)

lemma seqspaceI[intro]: n∈α ⟹ f:n→B ⟹ f∈B^<α
  unfolding seqspace_def by blast

lemma seqspaceD[dest]: f∈B^<α ⟹ ∃n∈α. f:n→B
  unfolding seqspace_def by blast

locale M_pre_seqspace =  M_trancl + M_replacement + M_Pi
begin

lemma function_space_subset_Pow_rel:
  assumes n∈ω M(B)
  shows n→B ⊆ Pow^M(⋃(ω→^M B))
proof -
  {
    fix f p
    assume f ⊆ n × B p ∈ f
    with assms
    obtain x y where p =⟨x,y⟩ x∈n y∈B by auto
    with assms
    have p ∈ (λ_∈ω. y)
      using Ord_trans[of _ _ ω] lam_constant_eq_cartprod by simp
    moreover
    note assms and ‹y∈B›
    moreover from this
    have M(λ_∈ω. y) using lam_constant_eq_cartprod by (auto dest:transM)
    moreover from calculation
    have (λ_∈ω. y) : ω →^M B
      using mem_function_space_rel_abs[of ω B, THEN iffD2]
      by simp
    ultimately
    have ∃B∈ω →^M B. p ∈ B
      by (rule_tac x=λ_∈ω. y in bexI)
  }
  with assms
```

show *?thesis*
by (*auto dest*:*transM intro*!:*mem_Pow_rel_abs*[*THEN iffD2*])
(*unfold Pi_def*, *auto*)
qed

lemma *seqspace_subset_Pow_rel*:
assumes $M(B)$
shows $B^{<\omega} \subseteq Pow^M(\bigcup(\omega \to^M B))$
using *assms function_space_subset_Pow_rel* **unfolding** *seqspace_def*
by *auto*

lemma *seqspace_imp_M*:
assumes $x \in B^{<\omega}$ $M(B)$
shows $M(x)$
using *assms seqspace_subset_Pow_rel*
by (*auto dest*:*transM*)

lemma *seqspace_eq_Collect*:
assumes $M(B)$
shows $B^{<\omega} = \{z \in Pow^M(\bigcup(\omega \to^M B)).\ \exists x[M].\ \exists n[M].\ n \in \omega \wedge z \in x \wedge x = n \to^M B\}$
using *assms seqspace_subset_Pow_rel nat_into_M seqspace_imp_M*
transM[*OF _ finite_funspace_closed*, *of _ _ B*] *function_space_rel_char*
by (*intro equalityI*) (*auto dest*:*transM dest*!:*seqspaceD*)

end — *M_pre_seqspace*

locale *M_seqspace* = *M_pre_seqspace* +
assumes
seqspace_separation: $M(B) \Longrightarrow separation(M, \lambda z.\ \exists x[M].\ \exists n[M].\ n \in \omega \wedge z \in x \wedge x = n \to^M B)$
begin

lemma *seqspace_closed*:
$M(B) \Longrightarrow M(B^{<\omega})$
using *seqspace_eq_Collect* **using** *seqspace_separation*
by *simp*

end — *M_seqspace*

25.2 Representation of finite functions

A function $f \in A \to_{fin} B$ can be represented by a function $g \in |f| \to A \times B$. It is clear that f can be represented by any $g' = g \cdot \pi$, where π is a permutation $\pi \in dom(g) \to dom(g)$. We use this representation of $A \to_{fin} B$ to prove that our model is closed under $_ \to_{fin} _$.

A function $g \in n \to A \times B$ that is functional in the first components.

definition *cons_like* :: $i \Rightarrow o$ **where**

cons_like(f) $\equiv \forall\ i{\in}domain(f)\ .\ \forall j{\in}i\ .\ fst(f`i) \neq fst(f`j)$

definition *FiniteFun_iso* :: $[i,i,i,i,i] \Rightarrow o$ **where**
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 $g \in n \to A \times B$ we obtain a finite function in A -||> B.

definition *to_FiniteFun* :: $i \Rightarrow i$ **where**
to_FiniteFun(f) $\equiv \{f`i.\ i{\in}domain(f)\}$

definition *FiniteFun_Repr* :: $[i,i] \Rightarrow i$ **where**
FiniteFun_Repr(A,B) $\equiv \{f \in (A{\times}B)^{<\omega}\ .\ cons_like(f)\ \}$

locale *M_FiniteFun* = *M_seqspace* +
assumes
separation_is_function : *separation*(M, *is_function*(M))
begin

lemma *cons_like_separation* : *separation*(M,$\lambda f.$ *cons_like*(f))
unfolding *cons_like_def*
using *lam_replacement_identity lam_replacement_domain lam_replacement_snd*
lam_replacement_hcomp[*OF _ lam_replacement_snd*]
lam_replacement_hcomp[*OF _ lam_replacement_fst*]
separation_eq lam_replacement_apply2[*THEN* [5] *lam_replacement_hcomp2*]
separation_neg
by(*rule_tac separation_all*,*auto*,*rule_tac separation_all*,*auto*)

lemma *supset_separation*: *separation*(M, $\lambda\ x.\ \exists\, a.\ \exists\, b.\ x = \langle a,b\rangle \wedge b \subseteq a$)
using *separation_Pair separation_subset lam_replacement_fst lam_replacement_snd*
by *simp*

lemma *to_finiteFun_replacement*: *strong_replacement*(M, $\lambda x\ y.\ y = range(x)$)
using *lam_replacement_range lam_replacement_imp_strong_replacement*
by *simp*

lemma *fun_range_eq*: $f{\in}A{\to}B \Longrightarrow \{f`i\ .\ i{\in}domain(f)\ \} = range(f)$
using *ZF_Library.range_eq_image*[*of f*] *domain_of_fun image_fun func.apply_rangeI*
by *simp*

lemma *FiniteFun_fst_type*:
assumes $h{\in}A$-||>B $p{\in}h$
shows $fst(p){\in}domain(h)$
using *assms*
by(*induct h*, *auto*)

lemma *FinFun_closed*:
$M(A) \Longrightarrow M(B) \Longrightarrow M(\bigcup\{n{\to}A{\times}B\ .\ n{\in}\omega\})$
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 $g \in n \to A \times B$, with $n = |f|$.

```
lemma FiniteFun_iso_intro1:
  assumes f ∈ (A -||> B)
  shows ∃ n∈ω . ∃ g∈n→A×B. FiniteFun_iso(A,B,n,g,f) ∧ 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∈ω g∈n→A×B FiniteFun_iso(A,B,n,g,h) cons_like(g) by auto
  let ?G=λ i ∈ succ(n) . if i=n then <a,b> else g`i
  from HI ‹a∈_› ‹b∈_›
  have G: ?G ∈ succ(n)→A×B
    by (auto intro:lam_type)
  have FiniteFun_iso(A,B,succ(n),?G,cons(<a,b>,h))
    unfolding FiniteFun_iso_def
  proof(intro conjI)
    {
      fix i
      assume i∈succ(n)
      then consider i=n | i∈n∧i≠n by auto
      then have ?G ` i ∈ cons(<a,b>,h)
        using HI
        by(cases,simp;auto simp add:HI FiniteFun_iso_def)
    }
    then show ∀ i∈succ(n). ?G ` i ∈ cons(⟨a, b⟩, h) ..
  next
    { fix ab'
```

```
      assume ab' ∈ cons(<a,b>,h)
      then
      consider  ab' = <a,b> | ab' ∈ h using cons_iff by auto
      then
      have ∃ i ∈ succ(n) . ?G`i = ab' unfolding FiniteFun_iso_def
      proof(cases,simp)
        case 2
        with HI obtain i
          where i∈n g`i=ab' unfolding FiniteFun_iso_def by auto
        with HI show ?thesis using  ltI[OF ‹i∈_›] by auto
      qed
    }
    then
    show ∀ ab∈cons(⟨a, b⟩, h). ∃ i∈succ(n). ?G`i = ab  ..
  qed
  with HI G
 have 1: ?G∈succ(n)→A×B FiniteFun_iso(A,B,succ(n),?G,cons(<a,b>,h)) succ(n)∈ω
by simp_all
  have cons_like(?G)
  proof -
    from ‹?G∈_› ‹g∈_›
    have domain(g) = n using domain_of_fun by simp
    {
      fix i j
      assume i∈domain(?G) j∈i
      with ‹n∈_›
      have j∈n using Ord_trans[of j _ n] by auto
      from ‹i∈_› consider (a) i=n ∧ i∉n | (b) i∈n by auto
      then
      have  fst(?G`i) ≠ fst(?G`j)
      proof(cases)
        case a
        with ‹j∈n› HI
        have ?G`i=<a,b> ?G`j=g`j g`j∈h
          unfolding FiniteFun_iso_def by auto
        with ‹a∉_› ‹h∈_›
        show ?thesis using  FiniteFun_fst_type by auto
      next
        case b
        with ‹i∈n› ‹j∈i› ‹j∈n› HI ‹domain(g) = n›
        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 $f \in A$ -||> B are equal.

```
lemma FiniteFun_isoD :
  assumes n∈ω g∈n→A×B f∈A-||>B FiniteFun_iso(A,B,n,g,f)
  shows to_FiniteFun(g) = f
proof
  show to_FiniteFun(g) ⊆ f
  proof
    fix ab
    assume ab∈to_FiniteFun(g)
    moreover
    note assms
    moreover from calculation
    obtain i where i∈n g‘i=ab ab∈A×B
      unfolding to_FiniteFun_def using domain_of_fun by auto
    ultimately
    show ab∈f unfolding FiniteFun_iso_def by auto
  qed
next
  show f ⊆ to_FiniteFun(g)
  proof
    fix ab
    assume ab∈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 $g \in n \to A \times B$ 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∈x <fst(?g′‘i),b>∈?h i<x fst(g‘x) = fst(?g′‘i)
       unfolding FiniteFun_iso_def using ltI by auto
     with ‹cons_like(g)› ‹domain(g) = _›
     have False
       unfolding cons_like_def by auto
   }
   then show ?thesis ..
 qed
 with HI assms ‹g‘x∈_›
 have cons(g‘x,?h) ∈ A-||>B (is ?h′ ∈_) using consI by auto
 have FiniteFun_iso(A,B,succ(x),g,?h′)
   unfolding FiniteFun_iso_def
 proof
   { fix i
     assume i∈succ(x)
     with ‹x∈_› consider (a) i=x| (b) i∈x∧i≠x by auto
     then have g‘i∈ ?h′
     proof(cases,simp)
       case b
       with ‹FiniteFun_iso(_,_,_,?g′,?h)›
       show ?thesis unfolding FiniteFun_iso_def by simp
     qed
   }
   then show ∀ i∈succ(x). g ‘ i ∈ cons(g ‘ x, ?h) ..
 next
   {
     fix ab
```

```
      assume ab∈?h′
      then consider ab=g‘x | ab ∈ ?h using cons_iff by auto
      then
      have ∃ i ∈ succ(x) . g‘i = ab unfolding FiniteFun_iso_def
      proof(cases,simp)
        case 2
        with HI obtain i
          where 2:i∈x ?g′‘i=ab  unfolding FiniteFun_iso_def by auto
        with ‹x∈_›
        have i≠x i∈succ(x) using  ltI[OF ‹i∈_›] by auto
        with 2 HI show ?thesis by auto
      qed
    } then show ∀ ab∈cons(g ‘ x, ?h). ∃ i∈succ(x). g ‘ i = ab ..
  qed
  with ‹?h′∈_›
 show to_FiniteFun(g) ∈ A -||>B ∧ FiniteFun_iso(A,B,succ(x),g,to_FiniteFun(g))
    using to_FiniteFun_succ_eq[OF ‹x∈_› ‹g∈_›,symmetric] by auto
qed

lemma FiniteFun_iso_intro2:
  assumes n∈ω f∈n→A×B cons_like(f)
  shows ∃  g ∈ (A -||> B) . FiniteFun_iso(A,B,n,f,g)
  using assms FiniteFun_iso_intro_to by blast

lemma FiniteFun_eq_range_Repr :
  shows {range(h) . h ∈ FiniteFun_Repr(A,B) } = {to_FiniteFun(h) . h ∈ Finite-
Fun_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-||>B = {to_FiniteFun(h) . h ∈ FiniteFun_Repr(A,B) }
    (is ?Y=?X)
proof
  {
    fix f
    assume f∈A-||>B
    then obtain n g where
      1: n∈ω g∈n→A×B FiniteFun_iso(A,B,n,g,f) cons_like(g)
      using FiniteFun_iso_intro1 by blast
    with ‹f∈_›
   have cons_like(g) f=to_FiniteFun(g) domain(g) = n g∈FiniteFun_Repr(A,B)
      using FiniteFun_isoD domain_of_fun
      unfolding FiniteFun_Repr_def
      by auto
    with 1 have f∈?X
      by auto
```

```
  } then show ?Y⊆?X ..
next
  {
    fix f
    assume f∈?X
    then obtain g where
      A:g∈FiniteFun_Repr(A,B) f=to_FiniteFun(g) cons_like(g)
      using RepFun_iff unfolding FiniteFun_Repr_def by auto
    then obtain n where n∈ω g∈n→A×B domain(g) = n
      unfolding FiniteFun_Repr_def using domain_of_fun by force
    with A
    have f∈?Y
      using FiniteFun_iso_intro_to by simp
  } then show ?X⊆?Y ..
qed

lemma FiniteFun_Repr_closed :
  assumes M(A) M(B)
  shows M(FiniteFun_Repr(A,B))
  unfolding FiniteFun_Repr_def
  using assms cartprod_closed
    seqspace_closed separation_closed cons_like_separation
  by simp

lemma to_FiniteFun_closed:
  assumes M(A) f∈A
  shows M(range(f))
  using assms transM[of _ A] by simp

lemma To_FiniteFun_Repr_closed :
  assumes M(A) M(B)
  shows M({range(h) . h ∈ 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 -||> B)
  using assms To_FiniteFun_Repr_closed FiniteFun_eq_to_FiniteFun_Repr
    FiniteFun_eq_range_Repr
  by simp

end — M_FiniteFun

end
```

26 Library of basic *ZF* results

theory *ZF_Library_Relative*
  imports
    Aleph_Relative — must be before *Transitive_Models.Cardinal_AC_Relative*!
    Cardinal_AC_Relative
    FiniteFun_Relative
begin

locale *M_Pi_assumption_repl* = *M_Pi_replacement* +
  fixes *A f*
  assumes *A_in_M*: $M(A)$ **and**
  f_repl : *lam_replacement*(M,f) **and**
  f_closed : $\forall x[M].\ M(f(x))$
begin

sublocale *M_Pi_assumptions M A f*
    using *Pi_replacement Pi_separation A_in_M f_repl f_closed*
      lam_replacement_Sigfun[*THEN lam_replacement_imp_strong_replacement*]
    by *unfold_locales* (*simp_all add*:*transM*[*of _ A*])

end — *M_Pi_assumption_repl*

no_notation *sum* (**infixr** ‹+› *65*)
notation *oadd* (**infixl** ‹+› *65*)

lemma (**in** *M_cardinal_arith_jump*) *csucc_rel_cardinal_rel*:
  assumes $Ord(\kappa)$ $M(\kappa)$
  shows $(|\kappa|^{M+})^M = (\kappa^+)^M$
proof (*intro le_anti_sym*) — we show both inequalities
  from *assms*
  have *hips*:$M((\kappa^+)^M)$ $Ord((\kappa^+)^M)$ $\kappa < (\kappa^+)^M$
    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** $Card^M(L)$ $|\kappa|^M < L$ $M(L)$ **for** L
    using *that Card_rel_is_Ord leI Card_rel_le_iff*[*of* κ L]
    by (*rule_tac ccontr*, *auto dest*:*not_lt_imp_le*) (*fast dest*: *le_imp_not_lt*)
  with *hips*
  show $(\kappa^+)^M \leq (|\kappa|^{M+})^M$
    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 $(\nu^+)^M \leq |\kappa|^M$
      using *le_Ord2*[*THEN Card_rel_csucc_rel*, *of* κ ν]
        Card_rel_lt_csucc_rel_iff[*of* $(\nu^+)^M$ $|\kappa|^M$, *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* κ ν] *Card_rel_is_Ord*
      le_Ord2
      by (*rule_tac* j=$(\nu^+)^M$ **in** *lt_trans2*) *simp_all*
    with ‹$\kappa < \nu$›
    show *False* **using** *le_imp_not_lt leI* **by** *blast*
  qed
qed

lemma (**in** *M_cardinal_AC*) *cardinal_rel_succ_not_0*: $|A|^M = succ(n) \Longrightarrow$ $M(A) \Longrightarrow M(n) \Longrightarrow A \neq 0$
  by *auto*

reldb_add functional *Finite Finite* — wrongly done? Finite is absolute
relativize functional *Finite_to_one Finite_to_one_rel* **external**
— don't have *is_Finite* yet

notation *Finite_to_one_rel* (‹*Finite'_to'_one–'*(_,_')›)

abbreviation
Finite_to_one_r_set :: [*i*,*i*,*i*] ⇒ *i* (‹*Finite'_to'_one–'*(_,_')›) **where**
Finite_to_one^M(*X*,*Y*) ≡ *Finite_to_one_rel*(##*M*,*X*,*Y*)

locale *M_ZF_library* = *M_aleph* + *M_FiniteFun*
begin

lemma *Finite_Collect_imp*: *Finite*({*x*∈*X* . *Q*(*x*)}) ⟹ *Finite*({*x*∈*X* . *M*(*x*) ∧ *Q*(*x*)})
(**is** *Finite*(*?A*) ⟹ *Finite*(*?B*))
using *subset_Finite*[*of ?B ?A*] **by** *auto*

lemma *Finite_to_one_relI*[*intro*]:
assumes *f*:*X*→^{M}*Y* ⋀*y*. *y*∈*Y* ⟹ *Finite*({*x*∈*X* . *f*'*x* = *y*})
and *types*:*M*(*f*) *M*(*X*) *M*(*Y*)
shows *f* ∈ *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*→^{M}*Y* ⋀*y*. *y*∈*Y* ⟹ *Finite*({*x*∈*X* . *M*(*x*) ∧ *f*'*x* = *y*})
and *types*:*M*(*f*) *M*(*X*) *M*(*Y*)
shows *f* ∈ *Finite_to_one*^{M}(*X*,*Y*)
using *assms* **unfolding** *Finite_to_one_rel_def*
by (*simp*)

lemma *Finite_to_one_relD*[*dest*]:
f ∈ *Finite_to_one*^{M}(*X*,*Y*) ⟹*f*:*X*→^{M}*Y*
f ∈ *Finite_to_one*^{M}(*X*,*Y*) ⟹ *y*∈*Y* ⟹ *M*(*Y*) ⟹ *Finite*({*x*∈*X* . *M*(*x*) ∧ *f*'*x* = *y*})
unfolding *Finite_to_one_rel_def* **by** *simp_all*

lemma *Diff_bij_rel*:
assumes ∀ *A*∈*F*. *X* ⊆ *A*
and *types*: *M*(*F*) *M*(*X*) **shows** (λ*A*∈*F*. *A*-*X*) ∈ *bij*^{M}(*F*, {*A*-*X*. *A*∈*F*})
proof -
from *assms*
have *A* - *X* = *C* - *X* ⟹ *A* = *C* **if** *A*∈*F C*∈*F* **for** *A C*
using *that subset_Diff_Un*[*of X A*] *subset_Diff_Un*[*of X C*]
by *simp*
moreover
note *assms*
moreover from *this*
have *M*({*A*-*X* . *A*∈*F*}) (λ*A*∈*F*. *A*-*X*) ∈ *F* → {*A*-*X*. *A*∈*F*} (**is** *?H* ∈ _) *M*(λ*A*∈*F*. *A*-*X*)
using *lam_type lam_replacement_Diff*[*THEN* [*5*] *lam_replacement_hcomp2*] *lam_replacement_constant*

lam_replacement_identity transM[*of_ F*] *lam_replacement_imp_strong_replacement RepFun_closed*
lam_replacement_imp_lam_closed
by (*simp*,*rule_tac lam_type*,*auto*)
ultimately
show *?thesis*
using *assms def_inj_rel def_surj_rel function_space_rel_char*
unfolding *bij_rel_def*
by *auto*
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*: $f \in Pi^M(A,B) \Longrightarrow f \in Pi(A,\ B)$
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

lemmas *rel_apply_in_range* = *apply_in_codomain_Ord*[*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*: $[\![f{:}A{\to}^M B;\ f \in inj^M(A,B);\ M(A);\ M(B)]\!] \Longrightarrow f \in inj^M(A,f``A)$
  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** ‹$b{\in}B$›
  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 $f \in A{\to}^M B$ $g \in C{\to}^M D$ $A \cap C = 0$
    and *types*:$M(A)$ $M(B)$ $M(C)$ $M(D)$
  shows $f \cup g \in (A \cup C){\to}^M (B \cup D)$
  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 $f \in A{\to}^M B$ $g \in C{\to}^M D$ $restrict(f, A \cap C) = restrict(g, A \cap C)$
    and *types*:$M(A)$ $M(B)$ $M(C)$ $M(D)$
  shows $f \cup g \in (A \cup C){\to}^M (B \cup D)$
  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*]: $A \lesssim^M B \Longrightarrow \exists f[M].\ f \in inj^M(A, B)$
  unfolding *lepoll_rel_def* **.**

— Should the assumptions be on f or on A and B? Should BOTH be intro rules?
lemma *lepoll_relI*[*intro*]: $f \in inj^M(A,\ B) \Longrightarrow M(f) \Longrightarrow A \lesssim^M B$
unfolding *lepoll_rel_def* **by** *blast*

lemma *eqpollD*[*dest*]: $A \approx^M B \Longrightarrow \exists f[M].\ f \in bij^M(A,\ B)$
unfolding *eqpoll_rel_def* **.**

— Same as *lepoll_relI*
lemma *bij_rel_imp_eqpoll_rel*[*intro*]: $f \in bij^M(A,B) \Longrightarrow M(f) \Longrightarrow A \approx^M B$
unfolding *eqpoll_rel_def* **by** *blast*

lemma *restrict_bij_rel*:— Unused
assumes $f \in inj^M(A,B)$ $C \subseteq A$
and *types*:$M(A)$ $M(B)$ $M(C)$
shows $restrict(f,C) \in bij^M(C,\ f``C)$
using *assms restrict_bij inj_rel_char bij_rel_char* **by** *auto*

lemma *range_of_subset_eqpoll_rel*:
assumes $f \in inj^M(X,Y)$ $S \subseteq X$
and *types*:$M(X)$ $M(Y)$ $M(S)$
shows $S \approx^M f``S$
using *assms restrict_bij bij_rel_char*
trans_inj_rel_closed[*OF* ‹$f \in inj^M(X,Y)$›]
unfolding *eqpoll_rel_def*
by (*rule_tac x=restrict*(f,S) **in** *rexI*) *auto*

lemmas *inj_rel_is_fun* = *inj_is_fun*[*OF mem_inj_rel*]

lemma *inj_rel_bij_rel_range*: $f \in inj^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow f \in bij^M(A,range(f))$
using *bij_rel_char inj_rel_char inj_bij_range* **by** *force*

lemma *bij_rel_is_inj_rel*: $f \in bij^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow f \in inj^M(A,B)$
unfolding *bij_rel_def* **by** *simp*

lemma *inj_rel_weaken_type*: [| $f \in inj^M(A,B)$; $B \subseteq D$; $M(A)$; $M(B)$; $M(D)$ |] ==> $f \in inj^M(A,D)$
using *inj_rel_char inj_rel_is_fun inj_weaken_type* **by** *auto*

lemma *bij_rel_converse_bij_rel* [*TC*]: $f \in bij^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B)$ ==> $converse(f)$: $bij^M(B,A)$
using *bij_rel_char* **by** *force*

lemma *bij_rel_is_fun_rel*: $f \in bij^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow f \in A \rightarrow^M B$
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*:
$g \in bij^M(A,B) \Longrightarrow f \in bij^M(B,C) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (f\ O\ g) \in bij^M(A,C)$
using *bij_rel_char comp_bij* **by** *force*

lemma *inj_rel_converse_fun*: $f \in inj^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow converse(f) \in range(f) \rightarrow^M A$
proof -
assume $f \in inj^M(A,B)\ M(A)\ M(B)$
then
have $M(f)\ M(converse(f))\ M(range(f))\ f \in inj(A,B)$
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 $f \in A \rightarrow^M B\ \ g \in B \rightarrow^M A\ \ f\ O\ g = id(B)\ g\ O\ f = id(A)\ M(A)\ M(B)$
shows $f \in bij^M(A,B)$
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*

26.1 Discipline for *cexp*

relativize functional *cexp cexp_rel* **external**
relationalize *cexp_rel is_cexp*

context *M_ZF_library*
begin

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* (‹·_$^{\uparrow}$– *is* _·›)
arity_theorem for *is_cexp_fm*

abbreviation
cexp_r :: $[i,i,i \Rightarrow o] \Rightarrow i$ (‹_$^{\uparrow}$–·–›) **where**

$cexp_r(x,y,M) \equiv cexp_rel(M,x,y)$

abbreviation
$cexp_r_set :: [i,i,i] \Rightarrow i$ (‹_$^{\uparrow}$–›–›) **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
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_leqpoll_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* _ ‹$M(\kappa)$›]
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 ≲^M κ
    using eq_lepoll_rel_trans by simp
next
  from ‹Ord(κ)›
  have κ < (κ⁺)^M Card^M((κ⁺)^M) M((κ⁺)^M)
      using Card_rel_csucc_rel lt_csucc_rel_iff types eq_csucc_rel_ord[OF _
‹M(κ)›]
    by simp_all
  then
  have κ ≺^M (κ⁺)^M
    using lt_Card_rel_imp_lesspoll_rel[OF _ ‹κ <_›] types by simp
  moreover
  assume d ≲^M κ
  ultimately
  have d ≲^M (κ⁺)^M
    using Card_rel_csucc_rel types lesspoll_succ_rel lepoll_rel_trans ‹Ord(κ)›
    by simp
  moreover
  from ‹d ≲^M κ› ‹Ord(κ)›
  have (κ⁺)^M ≲^M κ if d ≈^M (κ⁺)^M
    using eqpoll_rel_sym[OF that] types eq_lepoll_rel_trans[OF _ ‹d≲^Mκ›]
    by simp
  moreover from calculation ‹κ ≺^M (κ⁺)^M›
  have False if d ≈^M (κ⁺)^M
    using lesspoll_rel_irrefl[OF _ ‹M((κ⁺)^M)›] lesspoll_rel_trans1 types that
    by auto
  ultimately
  show d ≺^M (κ⁺)^M
    unfolding lesspoll_rel_def by auto
qed

lemma Infinite_imp_nats_lepoll:
  assumes Infinite(X) n ∈ ω
  shows n ≲ X
  using ‹n ∈ ω›
proof (induct)
  case 0
  then
  show ?case using empty_lepollI by simp
next
  case (succ x)
  show ?case
  proof -
    from ‹Infinite(X)› and ‹x ∈ ω›
    have ¬ (x ≈ X)
      using eqpoll_sym unfolding Finite_def by auto
```

```
    with ‹x ≲ X›
    obtain f where f ∈ inj(x,X) f ∉ surj(x,X)
      unfolding bij_def eqpoll_def by auto
    moreover from this
    obtain b where b ∈ X ∀ a∈x. f‘a ≠ b
      using inj_is_fun unfolding surj_def by auto
    ultimately
    have f ∈ inj(x,X-{b})
      unfolding inj_def by (auto intro:Pi_type)
    then
    have cons(⟨x, b⟩, f) ∈ inj(succ(x), cons(b, X - {b}))
      using inj_extend[of f x X-{b} x b] unfolding succ_def
      by (auto dest:mem_irrefl)
    moreover from ‹b∈X›
    have cons(b, X - {b}) = X by auto
    ultimately
    show succ(x) ≲ X by auto
  qed
qed

lemma nepoll_imp_nepoll_rel :
  assumes ¬ x ≈ X M(x) M(X)
  shows ¬ (x ≈^M X)
  using assms unfolding eqpoll_def eqpoll_rel_def by simp

lemma Infinite_imp_nats_lepoll_rel:
  assumes Infinite(X) n ∈ ω
    and types: M(X)
  shows n ≲^M X
  using ‹n ∈ ω›
proof (induct)
  case 0
  then
  show ?case using empty_lepoll_relI types by simp
next
  case (succ x)
  show ?case
  proof -
    from ‹Infinite(X)› and ‹x ∈ ω›
    have ¬ (x ≈ X) M(x) M(succ(x))
      using eqpoll_sym unfolding Finite_def by auto
    then
    have ¬ (x ≈^M X)
      using nepoll_imp_nepoll_rel types by simp
    with ‹x ≲^M X›
    obtain f where f ∈ inj^M(x,X) f ∉ surj^M(x,X) M(f)
      unfolding bij_rel_def eqpoll_rel_def by auto
    with ‹M(X)› ‹M(x)›
    have f∉surj(x,X) f∈inj(x,X)
```

```
      using surj_rel_char by simp_all
    moreover
    from this
    obtain b where b ∈ X ∀ a∈x. f‘a ≠ b
      using inj_is_fun unfolding surj_def by auto
    moreover
    from this calculation ‹M(x)›
    have f ∈ inj(x,X-{b}) M(<x,b>)
      unfolding inj_def using transM[OF _ ‹M(X)›]
      by (auto intro:Pi_type)
    moreover
    from this
    have cons(⟨x, b⟩, f) ∈ inj(succ(x), cons(b, X - {b})) (is ?g∈_)
      using inj_extend[of f x X-{b} x b] unfolding succ_def
      by (auto dest:mem_irrefl)
    moreover
    note ‹M(<x,b>)› ‹M(f)› ‹b∈X› ‹M(X)› ‹M(succ(x))›
    moreover from this
    have M(?g) 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: InfCard^M(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 ⟹ surj^M(α,κ) = 0
    and types:M(κ)
  shows Card^M(κ)
proof -
  {
    define min where min≡μ x. ∃f[M]. f ∈ bij^M(x,κ)
```

```
    moreover
    note ‹Ord(κ)› ‹M(κ)›
    moreover
    assume |κ|^M < κ
    moreover from calculation
    have ∃f. f ∈ bij^M(min,κ)
      using LeastI[of λi. 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_trans1[of min μ i. M(i) ∧ (∃f[M]. f ∈ bij^M(i, κ)) κ]
        Least_le[of λi. 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 ⟹ surj^M(min,κ) = 0›
    ultimately
    have False
      unfolding bij_rel_def by simp
  }
  with assms
  show ?thesis
    using Ord_cardinal_rel_le[of κ] not_lt_imp_le[of |κ|^M κ] 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 (‹mono'_map–'(_,_,_,_')›)

abbreviation
  mono_map_r_set :: [i,i,i,i,i]⇒i  (‹mono'_map–'(_,_,_,_')›) where
  mono_map^M(a,r,b,s) ≡ mono_map_rel(##M,a,r,b,s)

context M_ZF_library
begin

lemma mono_map_rel_char:
  assumes M(a) M(b)
  shows mono_map^M(a,r,b,s) = {f∈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
    $f \in mono_map^M(A,r,B,s)$ $B \subseteq C$
    and *types*:$M(A)$ $M(B)$ $M(C)$
  shows
    $f \in mono_map^M(A,r,C,s)$
  using *assms mono_map_mono mono_map_rel_char* **by** *auto*

lemma *nats_le_InfCard_rel*:
  assumes $n \in \omega$ $InfCard^M(\kappa)$
  shows $n \leq \kappa$
  using *assms Ord_is_Transset*
    le_trans[*of* n ω κ, *OF le_subset_iff*[*THEN iffD2*]]
  unfolding *InfCard_rel_def Transset_def* **by** *simp*

lemma *nat_into_InfCard_rel*:
  assumes $n \in \omega$ $InfCard^M(\kappa)$
  shows $n \in \kappa$
  using *assms le_imp_subset*[*of* ω κ]
  unfolding *InfCard_rel_def* **by** *auto*

lemma *Finite_lesspoll_rel_nat*:
  assumes $Finite(x)$ $M(x)$
  shows $x \prec^M nat$
proof -
  note *assms*
  moreover from *this*
  obtain n **where** $n \in \omega$ $M(n)$ $x \approx n$
    unfolding *Finite_def* **by** *auto*
  moreover from *calculation*
  obtain f **where** $f \in bij(x,n)$ f: x-||>n
    using *Finite_Fin*[*THEN fun_FiniteFunI*, *OF _ subset_refl*] *bij_is_fun*
    unfolding *eqpoll_def* **by** *auto*
  ultimately
  have $x \approx^M n$ **unfolding** *eqpoll_rel_def* **by** (*auto dest*:*transM*)
  with *assms* **and** ‹$M(n)$›
  have $n \approx^M x$ **using** *eqpoll_rel_sym* **by** *simp*
  moreover
  note ‹$n \in \omega$› ‹$M(n)$›
  ultimately
  show *?thesis*
    using *assms eq_lesspoll_rel_trans*[*OF* ‹$x \approx^M n$› *n_lesspoll_rel_nat*]
    by *simp*
qed

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 ∈ ω M(n) A ≈ n
    unfolding Finite_def by auto
  moreover from calculation
  obtain f where f ∈ bij(A,n) f: A-||>n
    using Finite_Fin[THEN fun_FiniteFunI, OF _ subset_refl] bij_is_fun
    unfolding eqpoll_def by auto
  ultimately
  have A ≈^M n unfolding eqpoll_rel_def by (auto dest:transM)
  with assms and ‹M(n)›
  have n ≈^M A using eqpoll_rel_sym by simp
  moreover
  note ‹n∈ω› ‹M(n)›
  ultimately
  show ?thesis
    using assms Least_le[of λi. M(i) ∧ i ≈^M A n]
      lt_trans1[of _ n ω, 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 Finite_cardinal_rel_in_nat
  note assms
  moreover from this
  obtain n where n ∈ ω M(n) A ≈ 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 ∈ bij(A,n) f: A-||>n
    using Finite_Fin[THEN fun_FiniteFunI, OF _ subset_refl] bij_is_fun
    unfolding eqpoll_def by auto
  ultimately
  have A ≈^M n unfolding eqpoll_rel_def by (auto dest:transM)
  with assms and ‹M(n)› ‹n∈ω›
  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 ‹|A| = n›
  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 $succ(|A\text{-}\{a\}|^M) = |A|^M$
proof -
from *assms*
have *inM*: $M(A\text{-}\{a\})$ $M(a)$ $M(A)$ **by** (*auto dest*:*transM*)
with ‹*Finite*(A)›
have $succ(|A\text{-}\{a\}|^M) = succ(|A\text{-}\{a\}|)$
using *Diff_subset*[*THEN subset_Finite*,
THEN Finite_cardinal_rel_eq_cardinal, *of A* $\{a\}$] **by** *simp*
also from *assms*
have $\ldots = |A|$
using *Finite_imp_succ_cardinal_Diff* **by** *simp*
also from *assms*
have $\ldots = |A|^M$ **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* α=*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 $f \in A \rightarrow^M B$ $\forall x{\in}A.\ d(f\ `\ x) = x$ $M(A)$ $M(B)$
  shows $f \in inj^M(A,\ B)$
  using *assms*
  by (*simp* (*no_asm_simp*) *add*: *def_inj_rel*,*auto intro*: *subst_context* [*THEN box_equals*])

lemma *lam_injective_rel*:
  assumes $\bigwedge x.\ x \in A \Longrightarrow c(x) \in B$
    $\bigwedge x.\ x \in A \Longrightarrow d(c(x)) = x$
    $\forall x[M].\ M(c(x))$ *lam_replacement*(M,c)
    $M(A)$ $M(B)$
  shows $(\lambda x{\in}A.\ c(x)) \in inj^M(A,\ B)$
  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 $f \in A \rightarrow^M B$ $\bigwedge y.\ y \in B \Longrightarrow d(y) \in A$ $\bigwedge y.\ y \in B \Longrightarrow f\ `\ d(y) = y$
    $M(A)$ $M(B)$
  shows $f \in surj^M(A,\ B)$
  using *assms*
  by (*simp add*: *def_surj_rel*, *blast*)

lemma *lam_surjective_rel*:
  assumes $\bigwedge x.\ x \in A \Longrightarrow c(x) \in B$
    $\bigwedge y.\ y \in B \Longrightarrow d(y) \in A$
    $\bigwedge y.\ y \in B \Longrightarrow c(d(y)) = y$
    $\forall x[M].\ M(c(x))$ *lam_replacement*(M,c)
    $M(A)$ $M(B)$
  shows $(\lambda x{\in}A.\ c(x)) \in surj^M(A,\ B)$
  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 $\bigwedge x.\ x \in A \Longrightarrow c(x) \in B$
    $\bigwedge y.\ y \in B \Longrightarrow d(y) \in A$
    $\bigwedge x.\ x \in A \Longrightarrow d(c(x)) = x$
    $\bigwedge y.\ y \in B \Longrightarrow c(d(y)) = y$
    $\forall x[M].\ M(c(x))$ *lam_replacement*(M,c)

$M(A)$ $M(B)$
shows $(\lambda x{\in}A.\ c(x)) \in bij^M(A,\ B)$
using *assms*
unfolding *bij_rel_def*
by (*blast intro*!: *lam_injective_rel lam_surjective_rel*)

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
$A \to^M B \approx^M A' \to^M B'$
proof -
from *assms*(*1*)[*THEN eqpoll_rel_sym*] *assms*(*2*) *assms lam_type*
obtain f g **where** $f \in bij^M(A',A)$ $g \in bij^M(B,B')$
by *blast*
with *assms*
have $converse(g) : bij^M(B',\ B)$ $converse(f)$: $bij^M(A,\ A')$
using *bij_converse_bij* **by** *auto*
let $?H{=}\lambda\ h \in A \to^M B\ .\ g\ O\ h\ O\ f$
let $?I{=}\lambda\ h \in A' \to^M B'\ .\ converse(g)\ O\ h\ O\ converse(f)$
have $go{:}g\ O\ F\ O\ f : A' \to^M B'$ **if** F: $A \to^M B$ **for** F
proof -
note *assms* ‹$f{\in}_$› ‹$g{\in}_$› *that*
moreover from *this*
have $g\ O\ F\ O\ f : A' \to B'$
using *bij_rel_is_fun*[*OF* ‹$g{\in}_$›] *bij_rel_is_fun*[*OF* ‹$f{\in}_$›] *comp_fun*
mem_function_space_rel[*OF* ‹$F{\in}_$›]
by *blast*
ultimately
show $g\ O\ F\ O\ f : A' \to^M B'$
using *comp_closed function_space_rel_char bij_rel_char*
by *auto*
qed
have $og{:}converse(g)\ O\ F\ O\ converse(f) : A \to^M B$ **if** F: $A' \to^M B'$ **for** F
proof -
note *assms that* ‹$converse(f) \in _$› ‹$converse(g) \in _$›
moreover from *this*
have $converse(g)\ O\ F\ O\ converse(f) : A \to B$
using *bij_rel_is_fun*[*OF* ‹$converse(g){\in}_$›] *bij_rel_is_fun*[*OF* ‹$converse(f){\in}_$›]
comp_fun
mem_function_space_rel[*OF* ‹$F{\in}_$›]
by *blast*
ultimately
show $converse(g)\ O\ F\ O\ converse(f) : A \to^M B$ (**is** $?G{\in}_$)
using *comp_closed function_space_rel_char bij_rel_char*
by *auto*
qed
with *go*
have $tc{:}?H \in (A \to^M B) \to (A' \to^M B')$ $?I \in (A' \to^M B') \to (A \to^M B)$

```
  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 (⋀x. 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 ... = (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[OF ‹g∈_›]]]
      right_comp_inverse[OF bij_is_surj[OF mem_bij_rel[OF ‹f∈_›]]] 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 ?I O ?H = id(A →^M B) .
  next
    from assms tc ‹M(?H)› ‹M(?I)›
    show ?H ∈ (A→^M B) →^M (A'→^M B') M(?H)
      ?I ∈ (A'→^M B') →^M (A→^M B)
      using mem_function_space_rel_abs by auto
  qed
qed

lemma curry_eqpoll_rel:
  fixes ν1 ν2 κ
  assumes M(ν1) M(ν2) M(κ)
  shows ν1 →^M (ν2 →^M κ) ≈^M ν1 × ν2 →^M κ
  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=λx. λw∈ν1 × ν2. x ‘ fst(w) ‘ snd(w)
  fix f z
  assume f : ν1 →^M (ν2 →^M κ)
  moreover
  note assms
  moreover from calculation
  have M(ν2 →^M κ)
    using function_space_rel_closed by simp
  moreover from calculation
  have M(f) f : ν1 → (ν2 →^M κ)
    using function_space_rel_char by (auto dest:transM)
  moreover from calculation
  have x ∈ ν1 ⟹ f‘x : ν2 → κ for x
    by (auto dest:transM intro!:mem_function_space_rel_abs[THEN iffD1])
  moreover from this
  show (λa∈ν1. λb∈ν2. ?cur(f) ‘ ⟨a, b⟩) = 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 ν2]
    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*]:$d(A) \equiv (\lambda x{\in}X.\ bool_of_o(x{\in}A))$
    — the witnessing map for the thesis:
  assumes $M(X)$
  shows $Pow^M(X) \approx^M X \rightarrow^M 2$
proof -
  from *assms*
  interpret *M_Pi_assumption_repl M X* λx . *2*
    by *unfold_locales* (*simp_all add*:*lam_replacement_constant*)
  have $lam_replacement(M, \lambda x.\ bool_of_o(x{\in}A))$ **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* λ_.*1* λ_.*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 $A{\in}Pow^M(X)$
    moreover
    note ‹$M(X)$›
    moreover from *calculation*
    have $M(A)$ **by** (*auto dest*:*transM*)
    moreover
    note ‹_ $\Longrightarrow$ $lam_replacement(M, \lambda x.\ bool_of_o(x{\in}A))$›
    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* $\lambda x.\ bool_of_o(x{\in}A)$ λ_. *2*, *simplified*])
        auto
    from ‹$A{\in}Pow^M(X)$› ‹$M(X)$›
    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 $d(\{y \in X\ .\ f`y = 1\}) = f$
      using *apply_type*[*OF* ‹f: $X{\rightarrow}2$›] **by** (*force intro*:*fun_extension*)
    from ‹$M(X)$› ‹$M(f)$›

```
    show {ya ∈ X . f ‘ ya = 1} ∈ Pow^M(X)
      using Pow_rel_char separation_equal_apply by auto
  next
    from assms ‹lam_replacement(M, λx. d(x))›
      ‹⋀A. _ ⟹ lam_replacement(M, λx. bool_of_o(x∈A))›
    show M(λx∈Pow^M(X). d(x)) lam_replacement(M, λx. d(x))
      ∀x[M]. M(d(x))
      using lam_replacement_iff_lam_closed[THEN iffD1] by auto
  qed (auto simp:‹M(X)›)
qed

lemma Pow_rel_bottom: M(B) ⟹ 0 ∈ Pow^M(B)
  using Pow_rel_char by simp

lemma cantor_surj_rel:
  assumes M(f) M(A)
  shows f ∉ surj^M(A,Pow^M(A))
proof
  assume f ∈ surj^M(A,Pow^M(A))
  with assms
  have f ∈ surj(A,Pow^M(A)) using surj_rel_char by simp
  moreover
  note assms
  moreover from this
  have M({x ∈ A . x ∈ f ‘ x}) {x ∈ A . x ∉ f ‘ x} = A - {x ∈ A . x ∈ f ‘ x}
    using lam_replacement_apply[THEN [4] separation_in, of λx. x]
      lam_replacement_identity lam_replacement_constant by auto
  with ‹M(A)›
  have {x∈A . x ∉ f‘x} ∈ Pow^M(A)
    by (intro mem_Pow_rel_abs[THEN iffD2]) auto
  ultimately
  obtain d where d∈A f‘d = {x∈A . x ∉ f‘x}
    unfolding surj_def by blast
  show False
  proof (cases d ∈ f‘d)
    case True
    note ‹d ∈ f‘d›
    also
    note ‹f‘d = {x∈A . x ∉ f‘x}›
    finally
    have d ∉ f‘d using ‹d∈A› by simp
    then
    show False using ‹d ∈ f ‘ d› by simp
  next
    case False
    with ‹d∈A›
    have d ∈ {x∈A . x ∉ f‘x} by simp
    also from ‹f‘d = ...›
    have ... = f‘d by simp
```

finally
show *False* **using** ‹$d \notin f`d$› **by** *simp*
qed
qed

lemma *cantor_inj_rel*: $M(f) \Longrightarrow M(A) \Longrightarrow f \notin inj^M(Pow^M(A),A)$
using *inj_rel_imp_surj_rel*[*OF _ Pow_rel_bottom*, *of f A A*]
cantor_surj_rel[*of* $\lambda x{\in}A$. *if* $x \in$ *range*(*f*) *then converse*(*f*) ‘ *x 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

27 Lambda-replacements required for cardinal inequalities

theory *Replacement_Lepoll*
imports
ZF_Library_Relative
begin

definition
lepoll_assumptions1 :: $[i{\Rightarrow}o,i,[i,i]{\Rightarrow}i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions1$(M,A,F,S,fa,K,x,f,r) \equiv \forall x{\in}S.$ *strong_replacement*$(M, \lambda y\ z.\ y \in F(A, x) \wedge z = \{\langle x, y\rangle\})$

definition
lepoll_assumptions2 :: $[i{\Rightarrow}o,i,[i,i]{\Rightarrow}i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions2$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M, \lambda x\ z.\ z = Sigfun(x, F(A)))$

definition
lepoll_assumptions3 :: $[i{\Rightarrow}o,i,[i,i]{\Rightarrow}i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions3$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M, \lambda x\ y.\ y = F(A, x))$

definition
lepoll_assumptions4 :: $[i{\Rightarrow}o,i,[i,i]{\Rightarrow}i,i,i,i,i,i,i] \Rightarrow o$ **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 :: $[i{\Rightarrow}o,i,[i,i]{\Rightarrow}i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions5$(M,A,F,S,fa,K,x,f,r) \equiv$

strong_replacement(M, $\lambda x\ y.\ y = \langle x,\ \mu\ i.\ x \in F(A,\ i),\ f\ `\ (\mu\ i.\ x \in F(A,\ i))\ `\ x\rangle$)

definition
lepoll_assumptions6 :: $[i \Rightarrow o, i, [i,i] \Rightarrow i, i, i, i, i, i, i] \Rightarrow o$ **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 :: $[i \Rightarrow o, i, [i,i] \Rightarrow i, i, i, i, i, i, i] \Rightarrow o$ **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 :: $[i \Rightarrow o, i, [i,i] \Rightarrow i, i, i, i, i, i, i] \Rightarrow o$ **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 :: $[i \Rightarrow o, i, [i,i] \Rightarrow i, i, i, i, i, i, i] \Rightarrow o$ **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 :: $[i \Rightarrow o, i, [i,i] \Rightarrow i, i, i, i, i, i, i] \Rightarrow o$ **where**
lepoll_assumptions10(M,A,F,S,fa,K,x,f,r) $\equiv$ *strong_replacement*
(M, $\lambda x\ z.\ z = Sigfun(x,\ \lambda k.\ if\ k \in range(f)\ then\ F(A,\ converse(f)\ `\ k)\ else\ 0)$)

definition
lepoll_assumptions11 :: $[i \Rightarrow o, i, [i,i] \Rightarrow i, i, i, i, i, i, i] \Rightarrow o$ **where**
lepoll_assumptions11(M,A,F,S,fa,K,x,f,r) $\equiv$ *strong_replacement*(M, $\lambda x\ y.\ y = (if\ x \in range(f)\ then\ F(A,\ converse(f)\ `\ x)\ else\ 0)$)

definition
lepoll_assumptions12 :: $[i \Rightarrow o, i, [i,i] \Rightarrow i, i, i, i, i, i, i] \Rightarrow o$ **where**
lepoll_assumptions12(M,A,F,S,fa,K,x,f,r) $\equiv$ *strong_replacement*(M, $\lambda y\ z.\ y \in F(A,\ converse(f)\ `\ x) \wedge z = \{\langle x,\ y\rangle\}$)

definition
lepoll_assumptions13 :: $[i \Rightarrow o, i, [i,i] \Rightarrow i, i, i, i, i, i, i] \Rightarrow o$ **where**
lepoll_assumptions13(M,A,F,S,fa,K,x,f,r) $\equiv$ *strong_replacement*
(M, $\lambda x\ y.\ y = \langle x,\ minimum(r,\ if\ x \in range(f)\ then\ F(A,converse(f)\ `\ x)\ else\ 0)\rangle$)

definition
lepoll_assumptions14 :: $[i \Rightarrow o, i, [i,i] \Rightarrow i, i, i, i, i, i, i] \Rightarrow o$ **where**
lepoll_assumptions14(M,A,F,S,fa,K,x,f,r) $\equiv$ *strong_replacement*
(M, $\lambda x\ y.\ y = \langle x,\ \mu\ i.\ x \in (if\ i \in range(f)\ then\ F(A,\ converse(f)\ `\ i)\ else\ 0)$,

fa ‘ $(\mu\ i.\ x \in (\mathit{if}\ i \in \mathit{range}(f)\ \mathit{then}\ F(A,\ \mathit{converse}(f)$ ‘ $i)\ \mathit{else}\ 0))$ ‘ $x\rangle)$

definition
lepoll_assumptions15 :: $[i{\Rightarrow}o,i,[i,i]{\Rightarrow}i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions15$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*
$(M,\ \lambda y\ z.\ y \in \mathit{inj}^M(\mathit{if}\ x \in \mathit{range}(f)\ \mathit{then}\ F(A,\ \mathit{converse}(f)$ ‘ $x)\ \mathit{else}\ 0,K) \wedge z = \{\langle x,\ y\rangle\})$

definition
lepoll_assumptions16 :: $[i{\Rightarrow}o,i,[i,i]{\Rightarrow}i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions16$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M,\ \lambda x\ y.\ y = \mathit{inj}^M(\mathit{if}\ x \in \mathit{range}(f)\ \mathit{then}\ F(A,\ \mathit{converse}(f)$ ‘ $x)\ \mathit{else}\ 0,K))$

definition
lepoll_assumptions17 :: $[i{\Rightarrow}o,i,[i,i]{\Rightarrow}i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions17$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*
$(M,\ \lambda x\ z.\ z = \mathit{Sigfun}(x,\ \lambda i.\ \mathit{inj}^M(\mathit{if}\ i \in \mathit{range}(f)\ \mathit{then}\ F(A,\ \mathit{converse}(f)$ ‘ $i)\ \mathit{else}\ 0,K)))$

definition
lepoll_assumptions18 :: $[i{\Rightarrow}o,i,[i,i]{\Rightarrow}i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions18$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*
$(M,\ \lambda x\ y.\ y = \langle x,\ \mathit{minimum}(r,\ \mathit{inj}^M(\mathit{if}\ x \in \mathit{range}(f)\ \mathit{then}\ F(A,\ \mathit{converse}(f)$ ‘ $x)\ \mathit{else}\ 0,K))\rangle)$

lemmas *lepoll_assumptions_defs*[*simp*] = *lepoll_assumptions1_def*
lepoll_assumptions2_def lepoll_assumptions3_def lepoll_assumptions4_def
lepoll_assumptions5_def lepoll_assumptions6_def lepoll_assumptions7_def
lepoll_assumptions8_def lepoll_assumptions9_def lepoll_assumptions10_def
lepoll_assumptions11_def lepoll_assumptions12_def lepoll_assumptions13_def
lepoll_assumptions14_def lepoll_assumptions15_def lepoll_assumptions16_def
lepoll_assumptions17_def lepoll_assumptions18_def

definition *if_range_F* **where**
[*simp*]: *if_range_F*$(H,f,i) \equiv \mathit{if}\ i \in \mathit{range}(f)\ \mathit{then}\ H(\mathit{converse}(f)$ ‘ $i)\ \mathit{else}\ 0$

definition *if_range_F_else_F* **where**
if_range_F_else_F$(H,b,f,i) \equiv \mathit{if}\ b{=}0\ \mathit{then}\ \mathit{if_range_F}(H,f,i)\ \mathit{else}\ H(i)$

lemma (**in** *M_basic*) *lam_Least_assumption_general*:
assumes
separations:
$\forall A'[M].\ \mathit{separation}(M,\ \lambda y.\ \exists x{\in}A'.\ y = \langle x,\ \mu\ i.\ x \in \mathit{if_range_F_else_F}(F(A),b,f,i)\rangle)$
and
mem_F_bound:$\bigwedge x\ c.\ x{\in}F(A,c) \Longrightarrow c \in \mathit{range}(f) \cup U(A)$
and
types:$M(A)\ M(b)\ M(f)\ M(U(A))$
shows *lam_replacement*$(M,\lambda x\ .\ \mu\ i.\ x \in \mathit{if_range_F_else_F}(F(A),b,f,i))$

```
proof -
  have ∀x∈X. (μ i. x ∈ if_range_F_else_F(F(A),b,f,i)) ∈
    Pow^M(⋃(X ∪ range(f) ∪ U(A))) if M(X) for X
  proof
    fix x
    assume x∈X
    moreover
    note ‹M(X)›
    moreover from calculation
    have M(x) by (auto dest:transM)
    moreover
    note assms
    ultimately
    show (μ i. x ∈ if_range_F_else_F(F(A),b,f,i)) ∈
        Pow^M(⋃(X ∪ range(f) ∪ U(A)))
    proof (rule_tac Least_in_Pow_rel_Union, cases b=0, simp_all)
      case True
      fix c
      assume asm:x ∈ if_range_F_else_F(F(A), 0, f, c)
      with mem_F_bound
      show c∈X ∨ c ∈ range(f) ∨ c ∈ U(A)
       unfolding if_range_F_else_F_def if_range_F_def by (cases c∈range(f))
auto
    next
      case False
      fix c
      assume x ∈ if_range_F_else_F(F(A), b, f, c)
      with False mem_F_bound[of x c]
      show c∈X ∨ c ∈ range(f) ∨ c∈U(A)
        unfolding if_range_F_else_F_def if_range_F_def by auto
    qed
  qed
  with assms
  show ?thesis
   using bounded_lam_replacement[of λx.(μ i. x ∈ if_range_F_else_F(F(A),b,f,i))
        λX. Pow^M(⋃(X ∪ range(f) ∪ U(A)))] by simp
qed

lemma (in M_basic) lam_Least_assumption_ifM_b0:
  fixes F
  defines F ≡ λ_ x. if M(x) then x else 0
  assumes
    separations:
  ∀A′[M]. separation(M, λy. ∃x∈A′. y = ⟨x, μ i. x ∈ if_range_F_else_F(F(A),0,f,i)⟩)
    and
    types:M(A) M(f)
  shows lam_replacement(M,λx . μ i. x ∈ if_range_F_else_F(F(A),0,f,i))
    (is lam_replacement(M,λx . Least(?P(x))))
proof -
```

```
  {
    fix x X
    assume M(X) x∈X (μ i. ?P(x,i)) ≠ 0
    moreover from this
    obtain m where Ord(m) ?P(x,m)
      using Least_0[of ?P(_)] by auto
    moreover
    note assms
    moreover
     have ?P(x,i) ⟷ (M(converse(f) ‘ i) ∧ i ∈ range(f) ∧ x ∈ converse(f) ‘ i)
for i
      unfolding F_def if_range_F_else_F_def if_range_F_def by auto
    ultimately
    have (μ i. ?P(x,i)) ∈ range (f)
      unfolding F_def if_range_F_else_F_def if_range_F_def
      by (rule_tac LeastI2) auto
  }
  with assms
  show ?thesis
    by (rule_tac bounded_lam_replacement[of _ λX. range(f) ∪ {0}]) auto
qed

lemma (in M_replacement) lam_Least_assumption_ifM_bnot0:
  fixes F
  defines F ≡ λ_ x. if M(x) then x else 0
  assumes
   lam_replacement_minimum:lam_replacement(M, λp. minimum(fst(p),snd(p)))
    and
    separations:
  ∀ A′[M]. separation(M, λy. ∃ x∈A′. y = ⟨x, μ i. x ∈ if_range_F_else_F(F(A),b,f,i)⟩)
    separation(M,Ord)
    and
    types:M(A) M(f)
    and
    b≠0
  shows lam_replacement(M,λx . μ i. x ∈ if_range_F_else_F(F(A),b,f,i))
    (is lam_replacement(M,λx . Least(?P(x))))
proof -
  have M(x) ⟹(μ i. (M(i) ⟶ x ∈ i) ∧ M(i)) = (if Ord(x) then succ(x) 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, λx. if Ord(x) then succ(x) else 0)
    using lam_replacement_if[OF _ _ separations(2)] lam_replacement_identity
      lam_replacement_constant lam_replacement_hcomp lam_replacement_succ
    by simp
  moreover
```

```
  note types ‹b≠0›
  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) lam_Least_assumption_drSR_Y:
  fixes F r′ D
  defines F ≡ drSR_Y(r′,D)
 assumes ∀ A′[M]. separation(M, λy. ∃ x∈A′. y = ⟨x, μ i. x ∈ if_range_F_else_F(F(A),b,f,i)⟩)
    M(A) M(b) M(f) M(r′)
    and
   lam_replacement_minimum:lam_replacement(M, λp. minimum(fst(p),snd(p)))
  shows lam_replacement(M,λx . μ i. x ∈ if_range_F_else_F(F(A),b,f,i))
proof -
  from assms(2-)
  have [simp]: M(X) ⟹ M(X ∪ range(f) ∪ {domain(x) . x ∈ A})
    M(r′) ⟹ M(X) ⟹ M({restrict(x,r′) . x ∈ A})
    for X r′
   using lam_replacement_domain[THEN lam_replacement_imp_strong_replacement,
        THEN RepFun_closed, of A]
    lam_replacement_restrict′[THEN lam_replacement_imp_strong_replacement,
        THEN RepFun_closed, of r′ A] by (auto dest:transM)
  have ∀ x∈X. (μ i. x ∈ if_range_F_else_F(F(A),b,f,i)) ∈
     Pow^M(⋃(X ∪ range(f) ∪ {domain(x). x∈A} ∪ {restrict(x,r′). x∈A} ∪ do-
main(A) ∪ range(A) ∪ ⋃A)) if M(X) for X
  proof
    fix x
    assume x∈X
    moreover
    note ‹M(X)›
    moreover from calculation
    have M(x) by (auto dest:transM)
    moreover
    note assms(2-)
    ultimately
    show (μ i. x ∈ if_range_F_else_F(F(A),b,f,i)) ∈
          Pow^M(⋃(X ∪ range(f) ∪ {domain(x). x∈A} ∪ {restrict(x,r′). x∈A} ∪
domain(A) ∪ range(A) ∪ ⋃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 ∈ (if c ∈ range(f) then F(A, converse(f) ‘ c) else 0)
      then
    show c∈X ∨ c∈range(f) ∨ (∃ x∈A. c = domain(x)) ∨ (∃ x∈A. c = restrict(x,r′))
∨ c ∈ domain(A) ∨ c ∈ range(A) ∨ (∃ x∈A. c∈x) by auto
```

next
case *False*
fix c
assume $x \in F(A,\ c)$
then
show $c{\in}X \vee c{\in}range(f) \vee (\exists\, x{\in}A.\ c = domain(x)) \vee (\exists\, x{\in}A.\ c = restrict(x,r')) \vee c \in domain(A) \vee c \in range(A) \vee (\exists\, x{\in}A.\ c{\in}x)$
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* $\lambda x.(\mu\ i.\ x \in if_range_F_else_F(F(A),b,f,i))$ $\lambda X.\ Pow^M(\bigcup(X \cup range(f) \cup \{domain(x).\ x{\in}A\} \cup \{restrict(x,r').\ x{\in}A\} \cup domain(A) \cup range(A) \cup \bigcup A))$] **by** *simp*
qed

locale *M_replacement_lepoll* = *M_replacement* + *M_inj* +
fixes F
assumes
F_type[*simp*]: $M(A) \Longrightarrow \forall\, x[M].\ M(F(A,x))$
and
lam_lepoll_assumption_F:$M(A) \Longrightarrow lam_replacement(M,F(A))$
and
— Here b is a Boolean.
lam_Least_assumption:$M(A) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow$
$lam_replacement(M,\lambda x\ .\ \mu\ i.\ x \in if_range_F_else_F(F(A),b,f,i))$
and
F_args_closed: $M(A) \Longrightarrow M(x) \Longrightarrow x \in F(A,i) \Longrightarrow M(i)$
and
lam_replacement_inj_rel:$lam_replacement(M,\ \lambda p.\ inj^M(fst(p),snd(p)))$
and
lam_replacement_minimum:$lam_replacement(M,\ \lambda p.\ minimum(fst(p),snd(p)))$
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 ∀x[M]. M(μ i. x ∈ if_range_F_else_F(F(A),b,f,i))
proof -
  have x ∈ (if i ∈ range(f) then F(A, converse(f) ‘ i) else 0) ⟹ M(i) for x i
  proof (cases i∈range(f))
    case True
    with ‹M(f)›
    show ?thesis by (auto dest:transM)
  next
    case False
    moreover
    assume x ∈ (if i ∈ range(f) then F(A, converse(f) ‘ i) 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*]
assms lam_Least_assumption[**where** *b=1,OF* ‹*M*(*A*)› _ ‹*M*(*f*)›]
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* ‹*M*(*f*)›]]
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* ‹$M(f)$›]]
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* ‹$M(f)$›]]
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* _ _ (‘)], *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

28 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: $M(\kappa) \Longrightarrow separation(M, \lambda x\ .\ x \prec^M \kappa)$
begin

declare *eqpoll_rel_refl* [*simp*]

28.1 Miscellaneous

lemma *cardinal_rel_RepFun_apply_le*:
  assumes $S \in A \to B$ $M(S)$ $M(A)$ $M(B)$
  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* λ *x*. *S`x*] *apply_type*[*OF* ‹*S*∈_›] *transM*[*OF* _ ‹*M*(*B*)›]
    by *auto*
  moreover from *assms this*
  have $(\lambda x \in A.\ S`x) \in surj_rel(M,A, \{S`a\ .\ a \in A\})$
    using *mem_surj_abs lam_funtype*[*of A* λ *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 ‹*M*(*X*)› *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*: $A \subseteq B \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow |A|^M \leq |B|^M$
using *subset_imp_lepoll_rel*[*THEN lepoll_rel_imp_cardinal_rel_le*] **.**

lemma *lt_cardinal_rel_imp_not_subset*: $|A|^M < |B|^M \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow \neg\ B \subseteq A$
using *subset_imp_le_cardinal_rel le_imp_not_lt* **by** *blast*

lemma *cardinal_rel_lt_csucc_rel_iff*:
Card_rel(*M*,*K*) $\Longrightarrow M(K) \Longrightarrow M(K') \Longrightarrow |K'|^M < (K^+)^M \longleftrightarrow |K'|^M \leq K$
by (*simp add*: *Card_rel_lt_csucc_rel_iff*)

end — *M_library*

locale *M_cardinal_UN_nat* = *M_cardinal_UN _ ω X* **for** *X*
begin
lemma *cardinal_rel_UN_le_nat*:
assumes $\bigwedge i.\ i{\in}\omega \Longrightarrow |X(i)|^M \leq \omega$
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 λk. if k∈range*(*f*) *then X*(*converse*(*f*)‘*k*) *else 0* **for** *J K f* +
assumes
f_inj: $f \in$ *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*) ≡ *if k∈range*(*f*) *then X*(*converse*(*f*)‘*k*) *else 0*
assumes $InfCard^M(K)$ $\bigwedge i.\ i{\in}J \Longrightarrow |X(i)|^M \leq K$
shows $|\bigcup i{\in}J.\ X(i)|^M \leq K$
proof -
have $M(K)$ $M(J)$ $\bigwedge w\ x.\ w \in X(x) \Longrightarrow M(x)$
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 = ‹M(f)› ‹M(K)› ‹M(J)› ‹⋀w x. w ∈ X(x) ⟹ M(x)›
  have i∈J ⟹ f`i ∈ K for i
    using inj_rel_is_fun[OF f_inj] apply_type
      function_space_rel_char by (auto simp add:inM)
  have (⋃i∈J. X(i)) ⊆ (⋃i∈K. Y(i))
  proof (standard, elim UN_E)
    fix x i
    assume i∈J x∈X(i)
    with ‹i∈J ⟹ f`i ∈ K›
    have x ∈ Y(f`i) f`i ∈ 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 ∈ (⋃i∈K. Y(i)) by auto
  qed
  then
  have |⋃i∈J. X(i)|^M ≤ |⋃i∈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 ‹⋀i. i∈J ⟹ f`i ∈ K› inM
  moreover from this
  have k∈range(f) ⟹ converse(f)`k ∈ J for k
    using inj_rel_converse_fun[OF f_inj]
      range_fun_subset_codomain function_space_rel_char by simp
  ultimately
  show |⋃i∈J. X(i)|^M ≤ 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 _ λ_. X +
  j:M_cardinal_UN _ J for J
begin

lemma lepoll_rel_imp_cardinal_rel_UN_le:
  notes [dest] = InfCard_is_Card Card_is_Ord
  assumes InfCard^M(K) J ≲^M K ⋀i. i∈J ⟹ |X(i)|^M ≤ K
    M(K)
  shows |⋃i∈J. X(i)|^M ≤ K
proof -
  from ‹J ≲^M K›
  obtain f where f ∈ inj_rel(M,J,K) M(f) by blast
```

```
  moreover
  let ?Y=λk. if k∈range(f) then X(converse(f)‘k) else 0
  note ‹M(K)›
  moreover from calculation
  have k ∈ range(f) ⟹ converse(f)‘k ∈ J for k
    using mem_inj_rel[THEN inj_converse_fun, THEN apply_type]
      j.Pi_assumptions by blast
  moreover from ‹M(f)›
  have w ∈ ?Y(x) ⟹ M(x) for w x
    by (cases x∈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, λy z. 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,κ)
    and types:M(κ) M(X)
  shows κ < |X|^M ⟷ (κ⁺)^M ≤ |X|^M
  using assms cardinal_rel_lt_csucc_rel_iff[of κ X] Card_rel_csucc_rel[of κ]
    not_le_iff_lt[of (κ⁺)^M |X|^M] not_le_iff_lt[of |X|^M κ]
  by blast

lemma lepoll_rel_imp_subset_bij_rel:
  assumes M(X) M(Y)
  shows X ≲^M Y ⟷ (∃ Z[M]. Z ⊆ Y ∧ Z ≈^M X)
proof
  assume X ≲^M Y
  then
  obtain j where j ∈ inj_rel(M,X,Y)
    by blast
  with assms
  have range(j) ⊆ Y j ∈ bij_rel(M,X,range(j)) M(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* ‹$M(j)$›
show $\exists Z[M].\ Z \subseteq Y \wedge Z \approx^M X$
using *eqpoll_rel_sym*[*OF* ‹$X \approx^M range(j)$›]
by *auto*
next
assume $\exists Z[M].\ Z \subseteq Y \wedge Z \approx^M X$
then
obtain $Z\ f$ **where** $f \in bij_rel(M,Z,X)\ Z \subseteq Y\ M(Z)\ M(f)$
unfolding *eqpoll_rel_def* **by** *blast*
with *assms*
have $converse(f) \in inj_rel(M,X,Y)\ M(converse(f))$
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*:
$Card_rel(M,\kappa) \Longrightarrow M(\kappa) \Longrightarrow M(X) \Longrightarrow |X|^M = \kappa \longleftrightarrow X \approx^M \kappa$
using *Card_rel_cardinal_rel_eq*[*of* κ] *cardinal_rel_eqpoll_rel_iff*[*of* X κ] **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 $InfCard_rel(M,\kappa)\ A \lesssim^M \kappa\ B \lesssim^M \kappa\ M(A)\ M(B)\ M(\kappa)$
shows $A \cup B \lesssim^M \kappa$
proof -
from *assms*
have $A \cup B \lesssim^M sum(A,B)$
using *Un_lepoll_rel_sum* **by** *simp*
moreover
note *assms*
moreover from *this*

have $|sum(A,B)|^M \leq \kappa \oplus^M \kappa$
using *sum_lepoll_rel_mono*[*of A* κ *B* κ] *lepoll_rel_imp_cardinal_rel_le*
unfolding *cadd_rel_def* **by** *auto*
ultimately
show *?thesis*
using *InfCard_rel_cdouble_eq Card_rel_cardinal_rel_eq*
InfCard_rel_is_Card_rel Card_rel_le_imp_lepoll_rel[*of sum*(*A*,*B*) κ]
lepoll_rel_trans[*of* $A \cup B$]
by *auto*
qed

lemma *cardinal_rel_Un_le*:
assumes *InfCard_rel*(M,κ) $|A|^M \leq \kappa$ $|B|^M \leq \kappa$ $M(\kappa)$ $M(A)$ $M(B)$
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'*: $M(i) \Longrightarrow Finite(|i|^M) \longleftrightarrow Finite(i)$
using *eqpoll_rel_imp_Finite_iff*[*OF cardinal_rel_eqpoll_rel*]
by *auto*

lemma *cardinal_rel_subset_of_Card_rel*:
assumes *Card_rel*(M,γ) $a \subseteq \gamma$ $M(a)$ $M(\gamma)$
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,γ) $\Longrightarrow |X|^M < \gamma \longleftrightarrow \neg\, |X|^M \geq \gamma$
using *assms not_le_iff_lt Card_rel_is_Ord Ord_cardinal_rel*
by *auto*

end — *M_library*

28.2 Countable and uncountable sets

relativize functional *countable countable_rel* **external**
relationalize *countable_rel is_countable*

notation *countable_rel* (‹*countable*–'(_')›)

abbreviation

countable_r_set :: $[i,i]\Rightarrow o$ (‹*countable*–'(_')›) **where**
$countable^M(i) \equiv countable_rel(\#\#M,i)$

context *M_library*
begin

lemma *countableI*[*intro*]: $X \lesssim^M \omega \Longrightarrow countable_rel(M,X)$
unfolding *countable_rel_def* **by** *simp*

lemma *countableD*[*dest*]: $countable_rel(M,X) \Longrightarrow X \lesssim^M \omega$
unfolding *countable_rel_def* **by** *simp*

lemma *countable_rel_iff_cardinal_rel_le_nat*: $M(X) \Longrightarrow countable_rel(M,X) \longleftrightarrow |X|^M \leq \omega$
using *le_Card_rel_iff*[*of* ω *X*] *Card_rel_nat*
unfolding *countable_rel_def* **by** *simp*

lemma *lepoll_rel_countable_rel*: $X \lesssim^M Y \Longrightarrow countable_rel(M,Y) \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow countable_rel(M,X)$
using *lepoll_rel_trans*[*of X Y*] **by** *blast*

— Next lemma can be proved without using AC
lemma *surj_rel_countable_rel*:
$countable_rel(M,X) \Longrightarrow f \in surj_rel(M,X,Y) \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow M(f) \Longrightarrow countable_rel(M,Y)$
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*: $Finite_rel(M,X) \Longrightarrow M(X) \Longrightarrow countable_rel(M,X)$
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 $countable_rel(M,J)$ $\bigwedge i.\ i\in J \Longrightarrow countable_rel(M,X(i))$
shows $countable_rel(M,\bigcup i\in J.\ X(i))$
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
cardinal_lib_assms1:

$M(A) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow$
$separation(M, \lambda y. \exists x{\in}A. y = \langle x, \mu\ i. x \in if_range_F_else_F(\lambda x. if\ M(x)$
$then\ x\ else\ 0,b,f,i)\rangle)$
and
cardinal_lib_assms2:
$M(A') \Longrightarrow M(G) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow$
$separation(M, \lambda y. \exists x{\in}A'. y = \langle x, \mu\ i. x \in if_range_F_else_F(\lambda a. if\ M(a)$
$then\ G`a\ else\ 0,b,f,i)\rangle)$
and
cardinal_lib_assms3:
$M(A') \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow M(F) \Longrightarrow$
$separation(M, \lambda y. \exists x{\in}A'. y = \langle x, \mu\ i. x \in if_range_F_else_F(\lambda a. if\ M(a)$
$then\ F\text{-}``\{a\}\ else\ 0,b,f,i)\rangle)$
and
cardinal_rel_separation :
$separation(M, \lambda\langle x,y\rangle. cardinal_rel(M,x) = y)$
and
separation_cardinal_rel_lt :
$M(\gamma) \Longrightarrow separation(M, \lambda Z\ .\ cardinal_rel(M,Z) < \gamma)$

begin

lemma *cdlt_assms*: $M(x) \Longrightarrow M(Q) \Longrightarrow separation(M, \lambda a\ .\ \forall s{\in}x. \langle s, a\rangle \in Q)$
using *separation_in*[*OF _ lam_replacement_product*] *separation_ball*
lam_replacement_fst lam_replacement_snd lam_replacement_constant
by *simp_all*

lemma *countable_rel_union_countable_rel*:
assumes $\bigwedge x.\ x \in C \Longrightarrow countable_rel(M,x)\ countable_rel(M,C)\ M(C)$
shows $countable_rel(M,\bigcup C)$
proof -
have $x \in (if\ M(i)\ then\ i\ else\ 0) \Longrightarrow M(i)$ **for** $x\ i$
by (*cases* $M(i)$) *auto*
then
interpret $M_replacement_lepoll\ M\ \lambda_\ x.\ if\ M(x)\ then\ x\ else\ 0$
using *lam_replacement_if*[*OF lam_replacement_identity*
lam_replacement_constant[*OF nonempty*], **where** *b=M*] *lam_replacement_inj_rel*
lam_replacement_minimum
proof(*unfold_locales,auto simp add*: *separation_def*)
fix *b f*
assume $M(b)\ M(f)$
show $lam_replacement(M, \lambda x. \mu\ i. x \in if_range_F_else_F(\lambda x. if\ M(x)\ then$
$x\ else\ 0, b, f, i))$
proof (*cases b=0*)
case *True*
with ‹$M(f)$›
show *?thesis*
using *cardinal_lib_assms1*
by (*simp_all*; *rule_tac lam_Least_assumption_ifM_b0*)+

```
    next
      case False
      with ‹M(f)› ‹M(b)›
      show ?thesis
        using cardinal_lib_assms1 separation_Ord lam_replacement_minimum
        by (rule_tac lam_Least_assumption_ifM_bnot0) auto
    qed
  qed
  note ‹M(C)›
  moreover
  have  w ∈ (if M(x) then x else 0) ⟹ M(x) for w x
    by (cases M(x)) auto
  ultimately
  interpret M_cardinal_UN_lepoll _ λc. if M(c) then c else 0 C
    using lepoll_assumptions lam_replacement_minimum
    by unfold_locales auto
  have (if M(i) then i else 0) = i if i∈C for i
    using transM[OF _ ‹M(C)›] 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⇒o,i]⇒o where
  uncountable_rel(M,X) ≡ ¬ countable_rel(M,X)

context M_library
begin

lemma uncountable_rel_iff_nat_lt_cardinal_rel:
  M(X) ⟹ uncountable_rel(M,X) ⟷ ω < |X|^M
  using countable_rel_iff_cardinal_rel_le_nat not_le_iff_lt by simp

lemma uncountable_rel_not_empty: uncountable_rel(M,X) ⟹ X ≠ 0
  using empty_lepoll_relI by auto

lemma uncountable_rel_imp_Infinite: uncountable_rel(M,X) ⟹ M(X) ⟹ In-
finite(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 ¬ (Y ⊆ X)
```

using *assms lepoll_rel_trans subset_imp_lepoll_rel*[*of Y X*]
by *blast*

28.3 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*: $M(k) \Longrightarrow Card_rel(M,k) \Longrightarrow k<\aleph_1{}^M \Longrightarrow k \leq \omega$
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 $Ord(\alpha)$
and *types*:$M(\alpha)$ $M(d)$
shows $d \prec^M \aleph_{succ(\alpha)}{}^M \longleftrightarrow d \lesssim^M \aleph_\alpha{}^M$
using *assms Aleph_rel_succ Card_rel_is_Ord Ord_Aleph_rel lesspoll_rel_csucc_rel*
by *simp*

lemma *cardinal_rel_Aleph_rel* [*simp*]: $Ord(\alpha) \Longrightarrow M(\alpha) \Longrightarrow |\aleph_\alpha{}^M|^M = \aleph_\alpha{}^M$
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 $a < b \Longrightarrow \aleph_a{}^M \prec^M \aleph_b{}^M$
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 $uncountable_rel(M,X) \longleftrightarrow (\exists S[M].\ S \subseteq X \wedge S \approx^M \aleph_1{}^M)$
proof
assume $uncountable_rel(M,X)$
with ‹$M(X)$›
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 ‹$M(X)$›
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* ω] *Card_rel_cardinal_rel_eq*
by *auto*
next
note *Aleph_rel_lesspoll_rel_increasing*[*of 1 0,simplified*]
assume $\exists S[M].\ S \subseteq X \wedge S \approx^M \aleph_1{}^M$
moreover
have *eq*:$\aleph_1{}^M = (\omega^+)^M$ **by** *auto*
moreover from *calculation* ‹$M(X)$›
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 ‹$M(X)$›
show *uncountable_rel*(M,X)
using
lesspoll_rel_trans1[*OF lepoll_rel_trans*[*OF A* _] ‹$\omega \prec^M (\omega^+)^M$›]
lesspoll_rel_not_refl
by *auto*
qed

lemma *UN_if_zero*: $M(K) \Longrightarrow (\bigcup x{\in}K.$ *if* $M(x)$ *then* G ‘ x *else* $0) = (\bigcup x{\in}K.\ G$ ‘ $x)$
using *transM*[*of* _ K] **by** *auto*

lemma *mem_F_bound1*:
fixes F G
defines $F \equiv \lambda_\ x.$ *if* $M(x)$ *then* G‘x *else* 0
shows $x{\in}F(A,c) \Longrightarrow c \in (range(f) \cup domain(G)\ )$
using *apply_0* **unfolding** *F_def*
by (*cases* $M(c)$, *auto simp*:*F_def drSR_Y_def dC_F_def*)

end — *M_library*

context *M_cardinal_library*
begin

lemma *lt_Aleph_rel_imp_cardinal_rel_UN_le_nat*: $function(G) \Longrightarrow domain(G) \lesssim^M \omega \Longrightarrow$
$\forall n{\in}domain(G).\ |G‘n|^M{<}\aleph_1{}^M \Longrightarrow M(G) \Longrightarrow |\bigcup n{\in}domain(G).\ G‘n|^M{\leq}\omega$
proof -
assume $M(G)$
moreover from *this*
have $x \in$ (*if* $M(i)$ *then* G ‘ i *else* 0) $\Longrightarrow M(i)$ **for** x i
by (*cases* $M(i)$) *auto*
moreover
have *separation*(M, M) **unfolding** *separation_def* **by** *auto*
ultimately
interpret *M_replacement_lepoll* M $\lambda_\ x.$ *if* $M(x)$ *then* G‘x *else* 0

```
    using lam_replacement_inj_rel cardinal_lib_assms2 mem_F_bound1[of _ _
G]
      lam_if_then_replacement_apply lam_replacement_minimum
    by (unfold_locales, simp_all)
      (rule lam_Least_assumption_general[where U=λ_. domain(G)], auto)
  note ‹M(G)›
  moreover
  have  w ∈ (if M(x) then G ‘ x else 0) ⟹ M(x) for w x
    by (cases M(x)) auto
  ultimately
  interpret M_cardinal_UN_lepoll _  λn. if M(n) then G‘n else 0 domain(G)
   using lepoll_assumptions1[where S=domain(G),unfolded lepoll_assumptions1_def]
      cardinal_lib_assms2 lepoll_assumptions
    by (unfold_locales, auto)
  assume function(G)
  let ?N=domain(G) and ?R=⋃n∈domain(G). G‘n
  assume ?N ≲^M ω
  assume Eq1: ∀ n∈?N. |G‘n|^M<ℵ1^M
  {
    fix n
    assume n∈?N
    with Eq1 ‹M(G)›
    have |G‘n|^M ≤ ω
      using le_Aleph_rel1_nat[of |G ‘ n|^M] lepoll_rel_imp_cardinal_rel_UN_le
        UN_if_zero[of domain(G) G]
      by (auto dest:transM)
  }
  then
  have n∈?N ⟹ |G‘n|^M ≤ ω for n .
  moreover
  note ‹?N ≲^M ω› ‹M(G)›
  moreover
  have (if M(i) then G ‘ i else 0) ⊆ G ‘ i for i
    by auto
  with ‹M(G)›
  have |if M(i) then G ‘ i else 0|^M ≤ |G ‘ i|^M for i
  proof(cases M(i))
    case True
    with ‹M(G)› show ?thesis using Ord_cardinal_rel[OF apply_closed]
      by simp
  next
    case False
    then
    have i∉domain(G)
      using transM[OF _ domain_closed[OF ‹M(G)›]] 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* ω]
UN_if_zero[*of domain*(*G*) *G*]
le_trans[*of* |*if M*(_) *then G* ‘ _ *else 0*|^M |*G* ‘ _|^M ω]
by *auto blast*
qed

lemma *Aleph_rel1_eq_cardinal_rel_vimage*: $f{:}\aleph_1{}^M \to^M \omega \Longrightarrow \exists\, n{\in}\omega.\ |f\text{-}``\{n\}|^M = \aleph_1{}^M$
proof -
assume $f{:}\aleph_1{}^M \to^M \omega$
then
have *function*(*f*) *domain*(*f*) = $\aleph_1{}^M$ *range*(*f*)$\subseteq\omega$ $f{\in}\aleph_1{}^M{\to}\omega$ *M*(*f*)
using *mem_function_space_rel*[*OF* ‹*f*∈_›] *domain_of_fun fun_is_function range_fun_subset_codomain*
function_space_rel_char
by *auto*
let *?G*=$\lambda n{\in}$*range*(*f*). *f*-``{*n*}
from ‹$f{:}\aleph_1{}^M{\to}\omega$›
have *range*(*f*) $\subseteq \omega$ *domain*(*?G*) = *range*(*f*)
using *range_fun_subset_codomain*
by *simp_all*
from ‹$f{:}\aleph_1{}^M{\to}\omega$› ‹*M*(*f*)› ‹*range*(*f*) $\subseteq \omega$›
have *M*(*f*-``{*n*}) **if** $n \in$ *range*(*f*) **for** *n*
using *that transM*[*of* _ ω] **by** *auto*
with ‹*M*(*f*)› ‹*range*(*f*) $\subseteq \omega$›
have *domain*(*?G*) $\lesssim^M \omega$ *M*(*?G*)
using *subset_imp_lepoll_rel lam_closed*[*of* λx . *f*-``{*x*}] *cardinal_lib_assms4*
by *simp_all*
have *function*(*?G*) **by** (*simp add*:*function_lam*)
from ‹$f{:}\aleph_1{}^M{\to}\omega$›
have $n{\in}\omega \Longrightarrow$ *f*-``{*n*} $\subseteq \aleph_1{}^M$ **for** *n*
using *Pi_vimage_subset* **by** *simp*
with ‹*range*(*f*) $\subseteq \omega$›
have $\aleph_1{}^M = (\bigcup n{\in}$*range*(*f*). *f*-``{*n*})
proof (*intro equalityI*, *intro subsetI*)
fix *x*
assume $x \in \aleph_1{}^M$
with ‹$f{:}\aleph_1{}^M{\to}\omega$› ‹*function*(*f*)› ‹*domain*(*f*) = $\aleph_1{}^M$›
have $x \in$ *f*-``{*f*‘*x*} *f*‘*x* $\in$ *range*(*f*)
using *function_apply_Pair vimage_iff apply_rangeI* **by** *simp_all*
then
show $x \in (\bigcup n{\in}$*range*(*f*). *f*-``{*n*}) **by** *auto*
qed *auto*
{
assume $\forall\, n{\in}$*range*(*f*). |*f*-``{*n*}|$^M < \aleph_1{}^M$
then
have $\forall\, n{\in}$*domain*(*?G*). |*?G*‘*n*|$^M < \aleph_1{}^M$

```
      using zero_lt_Aleph_rel1 by (auto)
    with ‹function(?G)› ‹domain(?G) ≲^M ω› ‹M(?G)›
    have |⋃n∈domain(?G). ?G`n|^M≤ω
      using lt_Aleph_rel_imp_cardinal_rel_UN_le_nat[of ?G]
      by auto
    then
    have |⋃n∈range(f). f-``{n}|^M≤ω by simp
    with ‹ℵ1^M = _›
    have |ℵ1^M|^M ≤ ω by auto
    then
    have ℵ1^M ≤ ω
      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 ‹range(f)⊆ω› ‹M(f)›
  obtain n where n∈ω ¬(|f -`` {n}|^M < ℵ1^M) M(f -`` {n})
    using nat_into_M by auto
  moreover from this
  have ℵ1^M ≤ |f-``{n}|^M
    using not_lt_iff_le Card_rel_is_Ord by simp
  moreover
  note ‹n∈ω ⟹ f-``{n} ⊆ ℵ1^M›
  ultimately
  show ?thesis
    using subset_imp_le_cardinal_rel[THEN le_anti_sym, of _ ℵ1^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 ≈^M (ℵ1^M) f ∈ Z →^M ω M(Z)
  shows ∃n∈ω. |f-``{n}|^M = ℵ1^M
proof -
  have M(1) M(ω) by simp_all
  then
  have M(ℵ1^M) by simp
  with assms ‹M(1)›
  obtain g where A:g∈bij_rel(M,ℵ1^M,Z) M(g)
    using eqpoll_rel_sym unfolding eqpoll_rel_def by blast
  with ‹f : Z →^M ω› assms
  have M(f) converse(g) ∈ bij_rel(M,Z, ℵ1^M) f∈Z→ω g∈ ℵ1^M→Z
   using bij_rel_is_fun_rel bij_rel_converse_bij_rel bij_rel_char function_space_rel_char
    by simp_all
```

with ‹$g\in bij_rel(M,\aleph_1{}^M,Z)$› ‹$M(g)$›
have $f\ O\ g : \aleph_1{}^M \to^M \omega\ M(converse(g))$
using *comp_fun*[*OF* _ ‹$f\in Z\to_$›,*of g*] *function_space_rel_char*
by *simp_all*
then
obtain n **where** $n\in\omega\ |(f\ O\ g)\text{-``}\{n\}|^M = \aleph_1{}^M$
using *Aleph_rel1_eq_cardinal_rel_vimage*
by *auto*
with ‹$M(f)$› ‹$M(converse(g))$›
have $M(converse(g)\ \text{``}\ (f\text{-``}\ \{n\}))\ f\text{-``}\ \{n\} \subseteq Z$
using *image_comp converse_comp Pi_iff*[*THEN iffD1*,*OF* ‹$f\in Z\to\omega$›] *vimage_subset*
unfolding *vimage_def*
using *transM*[*OF* ‹$n\in\omega$›]
by *auto*
from ‹$n\in\omega$› ‹$|(f\ O\ g)\text{-``}\{n\}|^M = \aleph_1{}^M$›
have $\aleph_1{}^M = |converse(g)\ \text{``}\ (f\text{-``}\{n\})|^M$
using *image_comp converse_comp* **unfolding** *vimage_def*
by *auto*
also from ‹$converse(g) \in bij_rel(M,Z,\ \aleph_1{}^M)$› ‹$f\colon Z\to^M \omega$› ‹$M(Z)$› ‹$M(f)$› ‹$M(\aleph_1{}^M)$›
‹$M(converse(g)\ \text{``}\ (f\text{-``}\ \{n\}))$›
have $\ldots = |f\text{-``}\{n\}|^M$
using *range_of_subset_eqpoll_rel*[*of* $converse(g)\ Z$ _ $f\text{-``}\{n\}$,
OF bij_rel_is_inj_rel[*OF* ‹$converse(g)\in_$›] ‹$f\text{-``}\ \{n\} \subseteq Z$›]
cardinal_rel_cong vimage_closed[*OF singleton_closed*[*OF transM*[*OF* ‹$n\in\omega$›]],*of f*]
by *auto*
finally
show *?thesis* **using** ‹$n\in_$› **by** *auto*
qed

end — *M_cardinal_library*

28.4 Applications of transfinite recursive constructions

locale *M_cardinal_library_extra* = *M_cardinal_library* +
assumes
replacement_trans_apply_image:
$M(f) \Longrightarrow M(\beta) \Longrightarrow Ord(\beta) \Longrightarrow$
$strong_replacement(M,\ \lambda x\ y.\ x\in\beta \wedge y = \langle x,\ transrec(x,\ \lambda a\ g.\ f\ `\ (g\ \text{``}\ a))\rangle)$
begin

lemma *rec_constr_type_rel*:
assumes $f\colon Pow_rel(M,G)\to^M G\ Ord(\alpha)\ M(G)$
shows $M(\alpha) \Longrightarrow rec_constr(f,\alpha) \in G$
using *assms*(*2*)
proof(*induct rule*:*trans_induct*)
case (*step* β)

```
  with assms
  have {rec_constr(f, x) . x ∈ β} = {y . x ∈ β, y=rec_constr(f, x)} (is _ = ?Y)
    M(f)
    using transM[OF _ ‹M(β)›] function_space_rel_char  Ord_in_Ord
    by auto
  moreover from assms this step ‹M(β)› ‹Ord(β)›
  have M({y . x ∈ β, y=<x,rec_constr(f, x)>}) (is M(?Z))
     using strong_replacement_closed[OF replacement_trans_apply_image,of f β
β,OF _ _ _ _
        univalent_conjI2[where P=λx _ . x∈β,OF univalent_triv]]
      transM[OF _ ‹M(β)›] transM[OF step(2) ‹M(G)›] Ord_in_Ord
    unfolding rec_constr_def
    by auto
  moreover from assms this step ‹M(β)› ‹Ord(β)›
  have ?Y = {snd(y) . y∈?Z}
  proof(intro equalityI, auto)
    fix u
    assume u∈β
    with assms this step ‹M(β)› ‹Ord(β)›
   have <u,rec_constr(f,u)> ∈ ?Z  rec_constr(f, u) = snd(<u,rec_constr(f,u)>)
      by auto
    then
    show ∃x∈{y . x ∈ β, y = ⟨x, rec_constr(f, x)⟩}. rec_constr(f, u) = snd(x)
      using bexI[of _ u] by force
  qed
  moreover from ‹M(?Z)› ‹?Y = _›
  have M(?Y)
    using
    RepFun_closed[OF lam_replacement_imp_strong_replacement[OF lam_replacement_snd]
‹M(?Z)›]
      fst_snd_closed[THEN conjunct2] transM[OF _ ‹M(?Z)›]
    by simp
  moreover from assms step
  have {rec_constr(f, x) . x ∈ β} ∈ Pow(G) (is ?X∈_)
    using transM[OF _ ‹M(β)›] function_space_rel_char
    by auto
  moreover from assms calculation step
  have M(?X)
    by simp
  moreover from calculation ‹M(G)›
  have ?X∈Pow_rel(M,G)
    using Pow_rel_char by simp
  ultimately
  have f`?X ∈ 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:*Pow_rel*(M,G)$\rightarrow^M$ G *Ord*(α) $M(G)$ $M(\alpha)$
shows M(*rec_constr*(f,α))
using *transM*[*OF rec_constr_type_rel* ‹$M(G)$›] *assms* **by** *auto*

lemma *lambda_rec_constr_closed* :
assumes *Ord*(γ) $M(\gamma)$ $M(f)$ f:*Pow_rel*(M,G)$\rightarrow^M$ G $M(G)$
shows $M(\lambda\alpha\in\gamma$. *rec_constr*(f,α))
using *lam_closed2*[*OF replacement_trans_apply_image*,*unfolded rec_constr_def*[*symmetric*],*of f* γ]
rec_constr_type_rel[*OF* ‹$f\in$_› *Ord_in_Ord*[*of* γ]] *transM*[*OF* _ ‹$M(G)$›] *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
$\bigwedge Z.\ |Z|^M < \gamma \Longrightarrow Z \subseteq G \Longrightarrow M(Z) \Longrightarrow \exists\, a\in G.\ \forall\, s\in Z.\ <s,a>\in Q\ \ b\in G$ *Card_rel*(M,γ)
$M(G)$ $M(Q)$ $M(\gamma)$
shows
$\exists\, S[M].\ S : \gamma \rightarrow^M G \wedge (\forall\, \alpha \in \gamma.\ \forall\, \beta \in \gamma.\ \ \alpha<\beta \longrightarrow <S`\alpha,S`\beta>\in Q)$
proof -
from *assms*
have $M(x) \Longrightarrow M(\{a \in G\ .\ \forall\, s\in x.\ \langle s,\ a\rangle \in Q\})$ **for** x
using *separation_orthogonal* **by** *simp*
let *?cdlt*γ=$\{Z\in$*Pow_rel*(M,G) . $|Z|^M<\gamma\}$ — "cardinal_rel less than γ"
and *?inQ*=$\lambda Y.\{a\in G.\ \forall\, s\in Y.\ <s,a>\in Q\}$
from ‹$M(G)$› ‹*Card_rel*(M,γ)› ‹$M(\gamma)$›
have M(*?cdlt*γ) *Ord*(γ)
using *separation_cardinal_rel_lt*[*OF* ‹$M(\gamma)$›] *Card_rel_is_Ord*
by *simp_all*
from *assms*
have H:$\exists\, a.\ a \in$ *?inQ*(Y) **if** $Y\in$*?cdlt*γ **for** Y
proof -
{
fix Y
assume $Y\in$*?cdlt*γ
then
have A: $Y\in$*Pow_rel*(M,G) $|Y|^M<\gamma$ **by** *simp_all*
then
have $Y\subseteq G$ $M(Y)$ **using** *Pow_rel_char*[*OF* ‹$M(G)$›] **by** *simp_all*
with A

```
      obtain a where a∈G ∀s∈Y. <s,a>∈Q
        using assms(1) by force
      with ‹M(G)›
      have ∃a. a ∈ ?inQ(Y) by auto
    }
    then show ?thesis using that by simp
  qed
  then
  have ∃f[M]. f ∈ Pi_rel(M,?cdltγ,?inQ) ∧ f ∈ Pi(?cdltγ,?inQ)
  proof -
    from ‹⋀x. M(x) ⟹ M({a ∈ G . ∀s∈x. ⟨s, a⟩ ∈ Q})› ‹M(G)›
     have x ∈ {Z ∈ Pow^M(G) . |Z|^M < γ} ⟹ M({a ∈ G . ∀s∈x. ⟨s, a⟩ ∈ Q})
for x
      by (auto dest:transM)
      with‹M(G)› ‹⋀x. M(x) ⟹ M({a ∈ G . ∀s∈x. ⟨s, a⟩ ∈ Q})› ‹M(Q)›
‹M(?cdltγ)›
    interpret M_Pi_assumptions_choice M ?cdltγ ?inQ
      using cdlt_assms[where Q=Q] surj_imp_inj_replacement3
     lam_replacement_Collect_ball_Pair[THEN lam_replacement_imp_strong_replacement]
       lam_replacement_minimum[THEN [5] lam_replacement_hcomp2,OF
         lam_replacement_constant lam_replacement_Collect_ball_Pair,unfolded
lam_replacement_def]
       lam_replacement_Sigfun[OF lam_replacement_Collect_ball_Pair,THEN
         lam_replacement_imp_strong_replacement]
      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 ∈ Pi_rel(M,?cdltγ,?inQ) f ∈ Pi(?cdltγ,?inQ) and
M(f)
    by auto
  moreover
  define Cb where Cb ≡ λ_∈Pow_rel(M,G)-?cdltγ. b
  moreover from ‹b∈G› ‹M(?cdltγ)› ‹M(G)›
  have Cb ∈ Pow_rel(M,G)-?cdltγ → G M(Cb)
    using lam_closed[of λ_.b Pow_rel(M,G)-?cdltγ] tag_replacement transM[OF
‹b∈G›]
    unfolding Cb_def by auto
  moreover
  note ‹Card_rel(M,γ)›
  ultimately
  have f ∪ Cb : (∏x∈Pow_rel(M,G). ?inQ(x) ∪ G) using
      fun_Pi_disjoint_Un[ of f ?cdltγ  ?inQ Cb Pow_rel(M,G)-?cdltγ λ_.G]
        Diff_partition[of {Z∈Pow_rel(M,G). |Z|^M<γ} Pow_rel(M,G), OF Col-
lect_subset]
    by auto
  moreover
  have ?inQ(x) ∪ G = G for x by auto
  moreover from calculation
```

have $f \cup Cb : Pow_rel(M,G) \to G$
  using *function_space_rel_char* **by** *simp*
ultimately
have $f \cup Cb : Pow_rel(M,G) \to^M G$
  using *function_space_rel_char* ‹$M(f)$› ‹$M(Cb)$› *Pow_rel_closed* ‹$M(G)$›
  by *auto*
define S **where** $S \equiv \lambda\alpha\in\gamma.\ rec_constr(f \cup Cb, \alpha)$
from ‹$f \cup Cb$: $Pow_rel(M,G) \to^M G$› ‹$Card_rel(M,\gamma)$› ‹$M(\gamma)$› ‹$M(G)$›
have $S : \gamma \to G\ M(f \cup Cb)$
  unfolding *S_def*
  using *Ord_in_Ord*[*OF Card_rel_is_Ord*] *rec_constr_type_rel lam_type transM*[*OF _* ‹$M(\gamma)$›]
    function_space_rel_char
  by *auto*
moreover from ‹$f \cup Cb \in _ \to^M G$› ‹$Card_rel(M,\gamma)$› ‹$M(\gamma)$› ‹$M(G)$› ‹$M(f \cup Cb)$› ‹$Ord(\gamma)$›
have $M(S)$
  unfolding *S_def*
  using *lambda_rec_constr_closed*
  by *simp*
moreover
have $\forall\alpha\in\gamma.\ \forall\beta\in\gamma.\ \alpha < \beta \longrightarrow \langle S`\alpha,\ S`\beta\rangle \in Q$
proof (*intro ballI impI*)
  fix $\alpha\ \beta$
  assume $\beta\in\gamma$
  with ‹$Card_rel(M,\gamma)$› ‹$M(S)$› ‹$M(\gamma)$›
  have $\beta\subseteq\gamma\ M(S``\beta)\ M(\beta)\ \{S`x\ .\ x \in \beta\} = \{restrict(S,\beta)`x\ .\ x \in \beta\}$
    using *transM*[*OF* ‹$\beta\in\gamma$› ‹$M(\gamma)$›] *image_closed Card_rel_is_Ord OrdmemD*
    by *auto*
  with ‹$\beta\in_$› ‹$Card_rel(M,\gamma)$› ‹$M(\gamma)$›
  have $\{rec_constr(f \cup Cb, x)\ .\ x\in\beta\} = \{S`x\ .\ x \in \beta\}$
    using *Ord_trans*[*OF _ _ Card_rel_is_Ord, of _ β γ*]
    unfolding *S_def*
    by *auto*
  moreover from ‹$\beta\in\gamma$› ‹$S : \gamma \to G$› ‹$Card_rel(M,\gamma)$› ‹$M(\gamma)$› ‹$M(S``\beta)$›
  have $\{S`x\ .\ x \in \beta\} \subseteq G\ M(\{S`x\ .\ x \in \beta\})$
    using *Ord_trans*[*OF _ _ Card_rel_is_Ord, of _ β γ*]
      apply_type[*of S γ λ_. G*]
    by(*auto,simp add:image_fun_subset*[*OF* ‹$S\in_$› ‹$\beta\subseteq_$›])
moreover from ‹$Card_rel(M,\gamma)$› ‹$\beta\in\gamma$› ‹$S\in_$› ‹$\beta\subseteq\gamma$› ‹$M(S)$› ‹$M(\beta)$› ‹$M(G)$› ‹$M(\gamma)$›
  have $|\{S`x\ .\ x \in \beta\}|^M < \gamma$
    using
      ‹$\{S`x\ .\ x\in\beta\} = \{restrict(S,\beta)`x\ .\ x\in\beta\}$›[*symmetric*]
      cardinal_rel_RepFun_apply_le[*of* $restrict(S,\beta)\ \beta\ G$,
        OF restrict_type2[*of S γ λ_.G β*] *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 β γ, OF _ _ _ _ ltI*]

```
      Card_rel_is_Ord
    by auto
  moreover
  have ∀x∈β. <S`x, f ` {S`x . x ∈ β}> ∈ Q
  proof -
    from calculation and f_type
    have f ` {S`x . x ∈ β} ∈ {a∈G. ∀x∈β. <S`x,a>∈Q}
      using apply_type[of f ?cdltγ ?inQ {S`x . x ∈ β}]
        Pow_rel_char[OF ‹M(G)›]
      by simp
    then
    show ?thesis by simp
  qed
  moreover
  assume α∈γ α < β
  moreover from this
  have α∈β using ltD by simp
  moreover
  note ‹β∈γ› ‹Cb ∈ Pow_rel(M,G)-?cdltγ → G›
  ultimately
  show ⟨S`α, S`β⟩∈Q
    using fun_disjoint_apply1[of {S`x . x ∈ β} Cb f]
      domain_of_fun[of Cb] ltD[of α β]
    by (subst (2) S_def, auto) (subst rec_constr_unfold, auto)
qed
moreover
note ‹M(G)› ‹M(γ)›
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) ⟹ Infinite(Z) ⟷ ω ≲^M Z
proof
  define Distinct where Distinct = {<x,y> ∈ Z×Z . x≠y}
  have Distinct = {xy ∈ Z×Z . ∃a b. xy = ⟨a, b⟩ ∧ a ≠ b} (is _=?A)
    unfolding Distinct_def by auto
  moreover
  assume Infinite(Z) M(Z)
  moreover from calculation
  have M(Distinct)
    using separation_dist by simp
  from ‹Infinite(Z)› ‹M(Z)›
  obtain b where b∈Z
    using Infinite_not_empty by auto
  {
    fix Y
    assume |Y|^M < ω M(Y)
```

```
    then
    have Finite(Y)
      using Finite_cardinal_rel_iff' ltD nat_into_Finite by auto
    with ‹Infinite(Z)›
    have Z ≠ Y by auto
  }
  moreover
 have (⋀W. M(W) ⟹ |W|^M < ω ⟹ W ⊆ Z ⟹ ∃a∈Z. ∀s∈W. <s,a>∈Distinct)
  proof -
    fix W
    assume M(W) |W|^M < ω W ⊆ Z
    moreover from this
    have Finite_rel(M,W)
      using
        cardinal_rel_closed[OF ‹M(W)›] Card_rel_nat
        lt_Card_rel_imp_lesspoll_rel[of ω,simplified,OF _ ‹|W|^M < ω›]
        lesspoll_rel_nat_is_Finite_rel[of W]
       eqpoll_rel_imp_lepoll_rel eqpoll_rel_sym[OF cardinal_rel_eqpoll_rel,of W]
        lesspoll_rel_trans1[of W |W|^M ω] by auto
    moreover from calculation
    have ¬Z⊆W
      using equalityI ‹Infinite(Z)› by auto
    moreover from calculation
    show ∃a∈Z. ∀s∈W. <s,a>∈Distinct
      unfolding Distinct_def by auto
  qed
  moreover from ‹b∈Z› ‹M(Z)› ‹M(Distinct)› this
  obtain S where S : ω →^M Z M(S) ∀α∈ω. ∀β∈ω. α < β ⟶ <S‘α,S‘β> ∈
Distinct
    using bounded_cardinal_rel_selection[OF _ ‹b∈Z› Card_rel_nat,of Distinct]
    by blast
  moreover from this
  have α ∈ ω ⟹ β ∈ ω ⟹ α≠β ⟹ S‘α ≠ S‘β for α β
    unfolding Distinct_def
    by (rule_tac lt_neq_symmetry[of ω λα β. S‘α ≠ S‘β])
      auto
  moreover from this ‹S∈_› ‹M(Z)›
  have S∈inj(ω,Z) using function_space_rel_char unfolding inj_def by auto
  ultimately
  show ω ≲^M Z
    unfolding lepoll_rel_def using inj_rel_char ‹M(Z)› by auto
next
  assume ω ≲^M Z M(Z)
  then
  show Infinite(Z) using lepoll_rel_nat_imp_Infinite by simp
qed

lemma Infinite_InfCard_rel_cardinal_rel: Infinite(Z) ⟹ M(Z) ⟹ 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 $F \equiv \lambda_\ x.\ if\ M(x)\ then\ A\text{-``}\{x\}\ else\ 0$
shows $x{\in}F(A,c) \Longrightarrow c \in (range(f) \cup range(A))$
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 $F \in Finite_to_one_rel(M,Z,Y) \cap surj_rel(M,Z,Y)\ Infinite(Z)\ M(Z)$
$M(Y)$
shows $|Y|^M = |Z|^M$
proof -
have *sep_true*: *separation*(M, M) **unfolding** *separation_def* **by** *auto*
note ‹$M(Z)$› ‹$M(Y)$›
moreover from *this assms*
have $M(F)\ F \in Z \to Y$
unfolding *Finite_to_one_rel_def*
using *function_space_rel_char* **by** *simp_all*
moreover from *this*
have $x \in (if\ M(i)\ then\ F\text{-``}\ \{i\}\ else\ 0) \Longrightarrow M(i)$ **for** *x i*
by (*cases* $M(i)$) *auto*
moreover from *calculation*
interpret *M_replacement_lepoll M* $\lambda_\ x.\ if\ M(x)\ then\ F\text{-``}\{x\}\ else\ 0$
using *lam_replacement_inj_rel mem_F_bound2 cardinal_lib_assms3*
lam_replacement_vimage_sing_fun lam_replacement_minimum
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** $U{=}\lambda_.\ range(F)$], *auto*)
have $w \in (if\ M(y)\ then\ F\text{-``}\{y\}\ else\ 0) \Longrightarrow M(y)$ **for** *w y*
by (*cases* $M(y)$) *auto*
moreover from ‹$F{\in}_{\cap}_$›
have *0*:$Finite(F\text{-``}\{y\})$ **if** $y{\in}Y$ **for** *y*
unfolding *Finite_to_one_rel_def*
using *vimage_fun_sing* ‹$F{\in}Z{\to}Y$› *transM*[*OF that* ‹$M(Y)$›] *transM*[*OF _*
‹$M(Z)$›] *that* **by** *simp*
ultimately
interpret *M_cardinal_UN_lepoll _* $\lambda y.\ if\ M(y)\ then\ F\text{-``}\{y\}\ else\ 0\ Y$
using *cardinal_lib_assms3 lepoll_assumptions*
by *unfold_locales* (*auto dest*:*transM simp del*:*mem_inj_abs*)
from ‹$F{\in}Z{\to}Y$›
have $Z = (\bigcup y{\in}Y.\ \{x{\in}Z\ .\ F`x = y\})$
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(λa∈X. a ∪ b)
      using lam_closed[of λ x . x∪b,OF _ ‹M(X)›] Un_closed[OF transM[OF _
‹M(X)›] ‹M(b)›]
        tag_union_replacement[OF ‹M(b)›]
      by simp
    moreover from this
    have {a ∪ b . a ∈ X} = (λx∈X. x ∪ b) ‘‘ X
      using image_lam by simp
    with calculation
    show M({a ∪ b . a ∈ X}) by auto
    moreover from calculation
    show (λa∈X. a ∪ b) ∈ X →^M {a ∪ b . a ∈ X}
```

using *function_space_rel_char* **by** (*auto intro*:*lam_funtype*)
ultimately
show $(\lambda a{\in}X.\ a \cup b) \in surj^M(X,\{a \cup b\ .\ a \in X\})\ M(\{a \cup b\ .\ a \in X\})$
using *surj_rel_char function_space_rel_char*
unfolding *surj_def* **by** *auto*
next
qed (*simp add*:‹$M(X)$›)
moreover from *assms this*
show *?thesis*
using *Finite_to_one_rel_surj_rel_imp_cardinal_rel_eq* **by** *simp*
qed

end — *M_cardinal_library_extra*

context *M_library*
begin

28.5 Results on relative cardinal exponentiation

lemma *cexp_rel_eqpoll_rel_cong*:
assumes
$A \approx^M A'\ B \approx^M B'\ M(A)\ M(A')\ M(B)\ M(B')$
shows
$A^{\uparrow B,M} = A'^{\uparrow B',M}$
unfolding *cexp_rel_def* **using** *cardinal_rel_eqpoll_rel_iff*
function_space_rel_eqpoll_rel_cong assms
by *simp*

lemma *cexp_rel_cexp_rel_cmult*:
assumes $M(\kappa)\ M(\nu 1)\ M(\nu 2)$
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 cexp_rel_eqpoll_rel_cong*) (*simp_all add*:*assms cexp_rel_def*)
also from *assms*
have $\ldots = \kappa^{\uparrow \nu 2 \times \nu 1,M}$
unfolding *cexp_rel_def* **using** *curry_eqpoll_rel*[*THEN cardinal_rel_cong*] **by** *blast*
also
have $\ldots = \kappa^{\uparrow \nu 2 \otimes^M \nu 1,M}$
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*: $M(X) \Longrightarrow |Pow_rel(M,X)|^M = 2^{\uparrow X,M}$ — 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) $M(\nu)$
shows $\nu < 2^{\uparrow\nu,M}$
using *assms Card_rel_is_Ord Card_rel_cexp_rel*
proof (*intro not_le_iff_lt*[*THEN iffD1*] *notI*)
assume $2^{\uparrow\nu,M} \leq \nu$
with *assms*
have $|Pow_rel(M,\nu)|^M \leq \nu$
using *cardinal_rel_Pow_rel*[*of* ν] **by** *simp*
with *assms*
have $Pow_rel(M,\nu) \lesssim^M \nu$
using *cardinal_rel_eqpoll_rel_iff Card_rel_le_imp_lepoll_rel Card_rel_cardinal_rel_eq*
by *auto*
then
obtain g **where** $g \in inj_rel(M,Pow_rel(M,\nu), \nu)$
by *blast*
moreover
note ‹$M(\nu)$›
moreover from *calculation*
have $M(g)$ **by** (*auto dest*:*transM*)
ultimately
show *False*
using *cantor_inj_rel* **by** *simp*
qed *simp*

lemma *countable_iff_lesspoll_rel_Aleph_rel_one*:
notes *iff_trans*[*trans*]
assumes $M(C)$
shows $countable^M(C) \longleftrightarrow C \prec^M \aleph_1{}^M$
using *assms lesspoll_rel_csucc_rel*[*of* ω C] *Aleph_rel_succ Aleph_rel_zero*
unfolding *countable_rel_def* **by** *simp*

lemma *countable_iff_le_rel_Aleph_rel_one*:
notes *iff_trans*[*trans*]
assumes $M(C)$
shows $countable^M(C) \longleftrightarrow |C|^M \prec^M \aleph_1{}^M$
proof -
from *assms*
have $countable^M(C) \longleftrightarrow C \prec^M \aleph_1{}^M$
using *countable_iff_lesspoll_rel_Aleph_rel_one*
by *simp*
also from *assms*
have $\ldots \longleftrightarrow |C|^M \prec^M \aleph_1{}^M$

using *cardinal_rel_eqpoll_rel*[*THEN eqpoll_rel_sym*, *THEN eq_lesspoll_rel_trans*]
by (*auto intro*:*cardinal_rel_eqpoll_rel*[*THEN eq_lesspoll_rel_trans*])
finally
show *?thesis* **.**
qed

end — *M_library*

lemma (**in** *M_cardinal_library*) *countable_fun_imp_countable_image*:
assumes $f{:}C \to^M B$ $countable^M(C)$ $\bigwedge c.\ c{\in}C \Longrightarrow countable^M(f`c)$
$M(C)$ $M(B)$
shows $countable^M(\bigcup(f``C))$
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

29 The Delta System Lemma, Relativized

theory *Delta_System_Relative*
imports
Cardinal_Library_Relative
begin

relativize functional *delta_system delta_system_rel* **external**

locale *M_delta* = *M_cardinal_library_extra* +
assumes
countable_lepoll_assms:
$M(G) \Longrightarrow M(A) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow separation(M,\ \lambda y.\ \exists x{\in}A.$
$y = \langle x,\ \mu\ i.\ x \in if_range_F_else_F(\lambda x.\ \{xa \in G\ .\ x \in xa\},$
$b,\ f,\ i)\rangle)$
begin

lemma *disjoint_separation*: $M(c) \Longrightarrow separation(M,\ \lambda\ x.\ \exists\, a.\ \exists\, b.\ x{=}\langle a{,}b\rangle \wedge a \cap$
$b = c)$
using *separation_Pair separation_eq lam_replacement_constant lam_replacement_Int*
by *simp*

lemma (**in** *M_trans*) *mem_F_bound6*:
fixes *F G*
defines $F \equiv \lambda_\ x.\ Collect(G,\ (\in)(x))$
shows $x{\in}F(G{,}c) \Longrightarrow c \in (range(f) \cup \bigcup G)$

using *apply_0* **unfolding** *F_def*
by (*cases* $M(c)$, *auto simp*:*F_def*)

lemma *delta_system_Aleph_rel1*:
assumes $\forall A{\in}F.\ Finite(A)\ F \approx^M \aleph_1{}^M\ M(F)$
shows $\exists D[M].\ D \subseteq F \wedge delta_system(D) \wedge D \approx^M \aleph_1{}^M$
proof -
have $M(G) \Longrightarrow M(p) \Longrightarrow M(\{A{\in}G\ .\ p \in A\})$ **for** $G\ p$
proof -
assume $M(G)\ M(p)$
have $\{A{\in}G\ .\ p \in A\} = G \cap (Memrel(\{p\} \cup G)\ ``\ \{p\})$
unfolding *Memrel_def* **by** *auto*
with ‹$M(G)$› ‹$M(p)$›
show *?thesis* **by** *simp*
qed
from ‹$\forall A{\in}F.\ Finite(A)$› ‹$M(F)$›
have $M(\lambda A{\in}F.\ |A|^M)$
using *cardinal_rel_separation Finite_cardinal_rel_in_nat*[*OF _ transM*[*of _ F*]]
separation_imp_lam_closed[*of _ cardinal_rel*(M) ω]
by *simp*

Since all members are finite,

with ‹$\forall A{\in}F.\ Finite(A)$› ‹$M(F)$›
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* -`` $\{n\} = \{\ A{\in}F\ .\ |A|^M = n\ \}$ **for** n
using *vimage_lam* **by** *auto*
moreover
note ‹$F \approx^M \aleph_1{}^M$› ‹$M(F)$›
moreover from *calculation*

there are uncountably many have the same cardinal:

obtain n **where** $n{\in}\omega\ |?cards\ \text{-``}\ \{n\}|^M = \aleph_1{}^M$
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** ‹$M(?cards)$›
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 $A{\in}G \Longrightarrow |A|^M = n$ **and** $G \approx^M \aleph_1{}^M$ **and** $M(G)$ **for** A
using *cardinal_rel_Card_rel_eqpoll_rel_iff* **by** *auto*

with ‹$n\in\omega$›

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 $\exists\, D[M].\ D \subseteq G \wedge delta_system(D) \wedge D \approx^M \aleph_1{}^M$
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 ‹$G \approx^M \aleph_1{}^M$› ‹$M(G)$›
  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 $\forall\, a\in G.\ Finite(a)$
  proof
    fix a
    assume $a \in G$
    moreover
    note ‹$M(G)$› ‹$n\in\omega$›
    moreover from *calculation*
    have $M(a)$ **by** (*auto dest*: *transM*)
    moreover from *succ* **and** ‹$a\in G$›
    have $|a|^M = succ(n)$ **by** *simp*
    ultimately
    show $Finite(a)$
      using *Finite_cardinal_rel_iff′ nat_into_Finite*[*of* $succ(n)$]
      by *fastforce*
  qed
  show $\exists\, D[M].\ D \subseteq G \wedge delta_system(D) \wedge D \approx^M \aleph_1{}^M$
  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** $\{A\in G\ .\ p \in A\} \approx^M \aleph_1{}^M\ M(p)$ **by** *blast*
    moreover
  note *1*=‹$M(G)$› ‹$M(G) \Longrightarrow M(p) \Longrightarrow M(\{A\in G\ .\ p\in A\})$› *singleton_closed*[*OF* ‹$M(p)$›]
    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 $M(converse(\lambda x\in\{x \in G\ .\ p \in x\}.\ x - \{p\}))$ (**is** $M(converse(?h))$)
      using *converse_closed*[*of ?h*] *lam_closed*[*OF diff_Pair_replacement*]
        Diff_closed[*OF transM*[*OF _ 1*(*2*)]]

by *auto*
moreover from *calculation*
have $\{A\text{-}\{p\}\ .\ A\in\{X\in G.\ p\in X\}\} \approx^M \aleph_1{}^M$ (**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 bij_rel_imp_eqpoll_rel, *of* _ _ *?F*]]
unfolding *eqpoll_rel_def*
by (*auto simp del*:*mem_bij_abs*)

Now using the hypothesis of the successor case,

moreover from ‹$\bigwedge A.\ A\in G \Longrightarrow |A|^M{=}succ(n)$› ‹$\forall\, a\in G.\ Finite(a)$›
and *this* ‹$M(G)$›
have $p\in A \Longrightarrow A\in G \Longrightarrow |A\ \text{-}\ \{p\}|^M = n$ **for** A
using *Finite_imp_succ_cardinal_rel_Diff*[*of* _ *p*] **by** (*force dest*: *transM*)
moreover
have $\forall\, a\in ?F.\ Finite(a)$
proof (*clarsimp*)
fix A
assume $p\in A$ $A\in G$
with ‹$\bigwedge A.\ p \in A \Longrightarrow A \in G \Longrightarrow |A\ \text{-}\ \{p\}|^M = n$› **and** ‹$n\in\omega$› ‹$M(G)$›
have $Finite(|A\ \text{-}\ \{p\}|^M)$
using *nat_into_Finite* **by** *simp*
moreover from ‹$p\in A$› ‹$A\in G$› ‹$M(G)$›
have $M(A\ \text{-}\ \{p\})$ **by** (*auto dest*: *transM*)
ultimately
show $Finite(A\ \text{-}\ \{p\})$
using *Finite_cardinal_rel_iff'* **by** *simp*
qed
moreover

we may apply the inductive hypothesis to the new family $\{A\ \text{-}\ \{p\}\ .\ A \in \{X \in G\ .\ p \in X\}\}$:

note ‹$(\bigwedge A.\ A \in ?F \Longrightarrow |A|^M = n) \Longrightarrow ?F \approx^M \aleph_1{}^M \Longrightarrow M(?F) \Longrightarrow$
$\exists\, D[M].\ D \subseteq ?F \wedge delta_system(D) \wedge D \approx^M \aleph_1{}^M$›
moreover
note *1*=‹$M(G)$› ‹$M(G) \Longrightarrow M(p) \Longrightarrow M(\{A\in G\ .\ p \in A\})$› *singleton_closed*[*OF* ‹$M(p)$›]
moreover from *this*
have $M(\{x\ \text{-}\ \{p\}\ .\ x \in \{x \in G\ .\ p \in x\}\})$
using *RepFun_closed*[*OF lam_replacement_Diff'*[*THEN*
lam_replacement_imp_strong_replacement]]
Diff_closed[*OF transM*[*OF* _ *1*(*2*)]] **by** *auto*
ultimately
obtain D **where** $D\subseteq\{A\text{-}\{p\}\ .\ A\in\{X\in G.\ p\in X\}\}$ $delta_system(D)$ $D \approx^M \aleph_1{}^M$ $M(D)$
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 *delta_system*($\{B \cup \{p\}\ .\ B{\in}D\}$) (**is** *delta_system*(*?D*))
by *fastforce*
moreover from ‹$M(D)$› ‹$M(p)$›
have $M(?D)$
using *RepFun_closed un_Pair_replacement transM*[*of _ D*] **by** *auto*
moreover from ‹$D \approx^M \aleph_1{}^M$› ‹$M(D)$›
have *Infinite*(D) $|D|^M = \aleph_1{}^M$
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* $\aleph_1{}^M$] ‹$M(D)$› ‹$D \approx^M \aleph_1{}^M$›
by *auto*
moreover from *this* ‹$M(?D)$› ‹$M(D)$› ‹$M(p)$›
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 ‹$D \subseteq \{A\text{-}\{p\}\ .\ A{\in}\{X{\in}G.\ p{\in}X\}\}$›
have $?D \subseteq G$
proof -
{
fix A
assume $A{\in}G$ $p{\in}A$
moreover from *this*
have $A = A - \{p\} \cup \{p\}$
by *blast*
ultimately
have $A\text{-}\{p\} \cup \{p\} \in G$
by *auto*
}
with ‹$D \subseteq \{A\text{-}\{p\}\ .\ A{\in}\{X{\in}G.\ p{\in}X\}\}$›
show *?thesis*
by *blast*
qed
moreover
note ‹$M(?D)$›
ultimately
show $\exists D[M].\ D \subseteq G \wedge$ *delta_system*(D) $\wedge\ D \approx^M \aleph_1{}^M$ **by** *auto*
next
case *False*
note ‹$\neg\ (\exists p[M].\ \{A \in G\ .\ p \in A\} \approx^M \aleph_1{}^M)$› — the other case
‹$M(G)$› ‹$\bigwedge p.\ M(G) \Longrightarrow M(p) \Longrightarrow M(\{A{\in}G\ .\ p \in A\})$›
moreover from ‹$G \approx^M \aleph_1{}^M$› **and** *this*
have $M(p) \Longrightarrow \{A \in G\ .\ p \in A\} \lesssim^M \aleph_1{}^M$ (**is** $_ \Longrightarrow ?G(p) \lesssim^M _$) **for** p
by (*auto intro*!:*lepoll_rel_eq_trans*[*OF subset_imp_lepoll_rel*] *dest*:*transM*)
moreover from *calculation*

have $M(p) \Longrightarrow ?G(p) \prec^M \aleph_1{}^M$ **for** p
using ‹$M(G) \Longrightarrow M(p) \Longrightarrow M(\{A{\in}G\ .\ p \in A\})$›
unfolding *lesspoll_rel_def* **by** *simp*
moreover from *calculation*
have $M(p) \Longrightarrow ?G(p) \lesssim^M \omega$ **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 $M(S) \Longrightarrow i \in S \Longrightarrow M(\{x \in G\ .\ i \in x\})$ **for** $i\ S$
by (*auto dest*: *transM*)
moreover
have $M(S) \Longrightarrow$ *countable_rel*$(M,S) \Longrightarrow$ *countable_rel*$(M,\{A \in G\ .\ S \cap A \neq 0\})$ **for** S
proof -
from ‹$M(G)$›
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 G* $\lambda x\ y.\ x{\in}y$] *mem_F_bound6*[*of _ G*]
lam_replacement_minimum separation_in lam_replacement_fst lam_replacement_snd
by *unfold_locales*
(*auto dest*:*transM intro*:*lam_Least_assumption_general*[*of _ _ _ _ Union*])
fix S
assume $M(S)$
with ‹$M(G)$› ‹$\bigwedge i.\ M(S) \Longrightarrow i \in S \Longrightarrow M(\{x \in G\ .\ i \in x\})$›
interpret *M_cardinal_UN_lepoll _ ?G S*
using *lepoll_assumptions*
by *unfold_locales* (*auto dest*:*transM*)
assume *countable_rel*(M,S)
with ‹$M(S)$› *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*] *lepoll_rel_imp_cardinal_rel_UN_le*,
THEN [*4*] *le_Card_rel_iff*[*THEN iffD1*], *of* ω] *j.UN_closed*
unfolding *countable_rel_def* **by** (*auto dest*: *transM*)
qed
define *Disjoint* **where** *Disjoint* $= \{<A,B> \in G{\times}G\ .\ B \cap A = 0\}$
have *Disjoint* $= \{x \in G{\times}G\ .\ \exists\ a\ b.\ x{=}<a,b> \wedge a{\cap}b{=}0\}$
unfolding *Disjoint_def* **by** *force*
with ‹$M(G)$›
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 $\exists\, A{\in}G.\ \forall\, S{\in}X.\ <S,A>{\in}$*Disjoint* **if** $|X|^M < \aleph_1{}^M$ $X \subseteq G$ $M(X)$ **for** X
proof -

note ‹$n\in\omega$› ‹$M(G)$›
moreover from *this* **and** ‹$\bigwedge A.\ A\in G \Longrightarrow |A|^M = succ(n)$›
have $|A|^M = succ(n)$ $M(A)$ **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 $A\in G \Longrightarrow Finite(A)$ **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 ‹$X\subseteq G$› ‹$M(X)$›
have $A\in X \Longrightarrow countable_rel(M,A)$ **for** A
using *Finite_imp_countable_rel* **by** (*auto dest*: *transM*)
moreover from ‹$M(X)$›
have $M(\bigcup X)$ **by** *simp*
moreover
note ‹$|X|^M < \aleph_1{}^M$› ‹$M(X)$›
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 ‹$M(\bigcup X)$› ‹$M(_) \Longrightarrow countable_rel(M,_) \Longrightarrow countable_rel(M,\{A \in G\ .\ _\ \cap A \neq 0\})$›
have $countable_rel(M,\{A \in G\ .\ (\bigcup X) \cap A \neq 0\})$ **by** *simp*
with ‹$G \approx^M \aleph_1{}^M$› ‹$M(G)$›
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 ‹$X\subseteq G$›
show $\exists A\in G.\ \forall S\in X.\ \langle S,A\rangle\in Disjoint$ **unfolding** *Disjoint_def*
using *subsetD* **by** *simp*
qed
moreover from ‹$G \approx^M \aleph_1{}^M$› ‹$M(G)$›
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** $S:\aleph_1{}^M\rightarrow^M G$ $\alpha\in\aleph_1{}^M \Longrightarrow \beta\in\aleph_1{}^M \Longrightarrow \alpha<\beta \Longrightarrow \langle S`\alpha,\ S`\beta\rangle$

$\in$*Disjoint*
for α β
using *bounded_cardinal_rel_selection*[*of* $\aleph_1{}^M$ *G Disjoint*] ‹*M*(*Disjoint*)›
by *force*
moreover from *this* ‹$n\in\omega$› ‹*M*(*G*)›
have *inM*:*M*(*S*) *M*(*n*) $\bigwedge x.\ x \in \aleph_1{}^M \Longrightarrow S\text{`}x \in G$ $\bigwedge x.\ x \in \aleph_1{}^M \Longrightarrow M(x)$
using *function_space_rel_char* **by** (*auto dest*: *transM*)
ultimately
have $\alpha \in \aleph_1{}^M \Longrightarrow \beta \in \aleph_1{}^M \Longrightarrow \alpha\neq\beta \Longrightarrow S\text{`}\alpha \cap S\text{`}\beta = 0$ **for** α β
unfolding *Disjoint_def*
using *lt_neq_symmetry*[*of* $\aleph_1{}^M$ $\lambda\alpha\ \beta.\ S\text{`}\alpha \cap S\text{`}\beta = 0$] *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** ‹$\bigwedge A.\ A\in G \Longrightarrow |A|^M = succ(n)$› *inM*
‹$S : \aleph_1{}^M \rightarrow^M G$› ‹*M*(*G*)›
have $S \in inj_rel(M,\aleph_1{}^M,\ G)$
using *def_inj_rel*[*OF Aleph_rel_closed* ‹*M*(*G*)›, *of 1*]
proof (*clarsimp*)
fix *w x*
from *inM*
have $a \in \aleph_1{}^M \Longrightarrow b \in \aleph_1{}^M \Longrightarrow a \neq b \Longrightarrow S\text{`}a \neq S\text{`}b$ **for** *a b*
using ‹$\bigwedge A.\ A\in G \Longrightarrow |A|^M = succ(n)$›[*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* _ _ _ _ *apply_closed*] **by** *auto*
moreover
assume $w \in \aleph_1{}^M$ $x \in \aleph_1{}^M$ $S\text{`}w = S\text{`}x$
ultimately
show $w = x$ **by** *blast*
qed
moreover from *this* ‹*M*(*G*)›
have $range(S) \approx^M \aleph_1{}^M$
using *inj_rel_bij_rel_range eqpoll_rel_sym* **unfolding** *eqpoll_rel_def*
by (*blast dest*: *transM*)
moreover
note ‹*M*(*G*)›
moreover from *calculation*
have $range(S) \subseteq G$
using *inj_rel_is_fun range_fun_subset_codomain*
by (*fastforce dest*: *transM*)
moreover
note ‹*M*(*S*)›
ultimately
show $\exists D[M].\ D \subseteq G \wedge delta_system(D) \wedge D \approx^M \aleph_1{}^M$
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 ‹G ⊆ F›
  show ?thesis by blast
qed

lemma delta_system_uncountable_rel:
  assumes ∀ A∈F. Finite(A) uncountable_rel(M,F) M(F)
  shows ∃ D[M]. D ⊆ F ∧ delta_system(D) ∧ D ≈^M ℵ1^M
proof -
  from assms
  obtain S where S ⊆ F S ≈^M ℵ1^M M(S)
    using uncountable_rel_iff_subset_eqpoll_rel_Aleph_rel1[of F] by auto
  moreover from ‹∀ A∈F. Finite(A)› and this
  have ∀ A∈S. Finite(A) by auto
  ultimately
  show ?thesis using delta_system_Aleph_rel1[of S]
    by (auto dest:transM)
qed

end — M_delta

end
```

30 Relative DC

```
theory Pointed_DC_Relative
  imports
    Cardinal_Library_Relative

begin

consts dc_witness :: i ⇒ i ⇒ i ⇒ i ⇒ i ⇒ i
primrec
  wit0   : dc_witness(0,A,a,s,R) = a
  witrec : dc_witness(succ(n),A,a,s,R) = s‘{x∈A. ⟨dc_witness(n,A,a,s,R),x⟩∈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) e < length(env)
  shows
      na ∈ ω ⟹
```

$A \in \omega \Longrightarrow$
$a \in \omega \Longrightarrow$
$s \in \omega \Longrightarrow$
$R \in \omega \Longrightarrow$
$e \in \omega \Longrightarrow$
$env \in list(Aa) \Longrightarrow$
$0 \in Aa \Longrightarrow$
$is_dc_witness(\#\#Aa,\ nth(na,\ env),\ nth(A,\ env),\ nth(a,\ env),\ nth(s,\ env),$
$nth(R,\ env),\ nth(e,\ env)) \longleftrightarrow$
$Aa,\ env \models ?fm(nat,\ A,\ a,\ s,\ R,\ e)$
unfolding *is_dc_witness_def is_recursor_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* :: $[i,i,i,i,i] \Rightarrow o$ **where**
$dcwit_body(A,a,g,R) \equiv \lambda p.\ snd(p) = dc_witness(fst(p),\ A,\ a,\ g,\ R)$

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 $M(n)$ $M(A)$ $M(a)$ $M(s)$ $M(R)$ $n \in nat$
shows $M(dc_witness(n,A,a,s,R))$
using ‹$n \in nat$›
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 $M((\{?b\}\times A)\cap R)$ (**is** $M(?X)$) **by** *auto*
moreover
have $\{x\in A.\ \langle ?b,x\rangle\in R\} = \{snd(y)\ .\ y\in ?X\}$ (**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 ‹$M(s)$› *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 ‹$n\in nat$›

proof(*induct n*)
  case *0*
  then show *?case* **using** ‹*a*∈*A*› **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:
    $M(A) \Longrightarrow M(a) \Longrightarrow M(g) \Longrightarrow M(R) \Longrightarrow separation(M, is_dcwit_body(M, A, a, g, R))$
    and
    dcwit_replacement:$Ord(na) \Longrightarrow$
    $M(na) \Longrightarrow$
    $M(A) \Longrightarrow$
    $M(a) \Longrightarrow$
    $M(s) \Longrightarrow$
    $M(R) \Longrightarrow$
    transrec_replacement
     $(M, \lambda n\ f\ ntc.$
        is_nat_case
         $(M, a,$
          $\lambda m\ bmfm.$
            $\exists fm[M].\ \exists cp[M].$
              $is_apply(M, f, m, fm) \wedge$
              $is_Collect(M, A, \lambda x.\ \exists fmx[M].\ (M(x) \wedge fmx \in R) \wedge pair(M, fm, x, fmx),\ cp) \wedge$
              $is_apply(M, s, cp, bmfm),$
          $n, ntc), na)$
begin

lemma *is_dc_witness_iff*:
  assumes $Ord(na)\ M(na)\ M(A)\ M(a)\ M(s)\ M(R)\ M(res)$
  shows $is_dc_witness(M, na, A, a, s, R, res) \longleftrightarrow res = dc_witness_rel(M, na, A, a, s, R)$
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 $M(fm) \Longrightarrow M(\{x \in A\ .\ M(x) \wedge \langle fm,\ x\rangle \in R\})$ **for** *fm*
using *first_section_closed* **by** (*auto dest*:*transM*)
moreover from *calculation*
have $M(x) \Longrightarrow M(z) \Longrightarrow M(mesa) \Longrightarrow$
$(\exists\, ya[M].\ pair(M,\ x,\ ya,\ z) \wedge$
$is_wfrec(M,\ \lambda n\ f.\ is_nat_case(M,\ a,\ \lambda m\ bmfm.\ \exists\, fm[M].\ is_apply(M,\ f,\ m,$
$fm) \wedge$
$is_apply(M,\ s,\ \{x \in A\ .\ \langle fm,\ x\rangle \in R\},\ bmfm),\ n),\ mesa,\ x,\ ya))$
$\longleftrightarrow$
$(\exists\, y[M].\ pair(M,\ x,\ y,\ z) \wedge$
$is_wfrec(M,\ \lambda n\ f.\ is_nat_case(M,\ a,$
$\lambda m\ bmfm.$
$\exists\, fm[M].\ \exists\, cp[M].\ is_apply(M,\ f,\ m,\ fm) \wedge$
$is_Collect(M,\ A,\ \lambda x.\ M(x) \wedge \langle fm,\ x\rangle \in R,\ cp) \wedge\ \ is_apply(M,\ s,\ cp,$
$bmfm),n),$
$mesa,\ x,\ y))$ **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*:
$fst(x) \in \omega \Longrightarrow M(A) \Longrightarrow M(a) \Longrightarrow M(g) \Longrightarrow M(R) \Longrightarrow M(x) \Longrightarrow$
$is_dcwit_body(M,A,a,g,R,x) \longleftrightarrow dcwit_body(A,a,g,R,x)$
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 *Lambda_dc_witness_closed*:
assumes $g \in Pow^M(A)\text{-}\{0\} \rightarrow A\ \ a{\in}A\ \forall\, y{\in}A.\ \{x \in A\ .\ \langle y,\ x\rangle \in R\} \neq 0$
$M(g)\ M(A)\ M(a)\ M(R)$
shows $M(\lambda n{\in}nat.\ dc_witness(n,A,a,g,R))$
proof -
from *assms*
have $(\lambda n{\in}nat.\ dc_witness(n,A,a,g,R)) = \{p \in \omega \times A\ .\ is_dcwit_body(M,A,a,g,R,p)\}$
using *witness_into_A*[*of a A g R*]
Pow_rel_char apply_type[*of g* $\{x \in Pow(A)\ .\ M(x)\}\text{-}\{0\}$ $\lambda_.A$]
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 $a \in A$
$\forall X[M].\ X \neq 0 \wedge X \subseteq A \longrightarrow s`X \in X$
$\forall y \in A.\ \{x \in A.\ \langle y,x \rangle \in R\} \neq 0$ $n \in nat$
$M(a)$ $M(A)$ $M(s)$ $M(R)$ $M(n)$
shows $\langle dc_witness(n,\ A,\ a,\ s,\ R), dc_witness(succ(n),\ A,\ a,\ s,\ R) \rangle \in R$
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 $dc_witness(n,\ A,\ a,\ s,\ R) \in A$ (**is** $?x \in A$)
using *witness_into_A*[*of* _ _ s R n] **by** *simp*
moreover from *assms*
have $M(\{x \in A\ .\ \langle dc_witness(n,\ A,\ a,\ s,\ R),\ x \rangle \in R\})$
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 $(\lambda n \in nat.\ dc_witness(n,\ A,\ a,\ s,\ R)) \in nat \rightarrow A$ (**is** $?f \in _ \rightarrow _$)
proof -
have $?f \in nat \rightarrow \{dc_witness(n,\ A,\ a,\ s,\ R).\ n \in nat\}$ (**is** $_ \in _ \rightarrow ?B$)
using *lam_funtype assms* **by** *simp*
then
have $?B \subseteq A$
using *witness_into_A assms* **by** *auto*
with ‹$?f \in _$›
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 $\exists f \in nat \rightarrow A.\ \forall n \in nat.\ f`n = dc_witness(n,A,a,s,R)$

using *assms bexI*[*of* _ $\lambda n{\in}nat.$ *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 $\bigwedge x.\ x \in A \Longrightarrow (\exists\, y.\ y \in x)\ M(A)$
shows $\exists f \in A \to^M \bigcup(A).\ \forall x \in A.\ f`x \in x$
proof -
from ‹$M(A)$›
interpret *mpiA*:*M_Pi_assumption_repl* _ *A* λx . *x*
by *unfold_locales* (*simp_all add*:*lam_replacement_identity*)
from ‹$M(A)$›
interpret *mpi2*:*M_Pi_assumption_repl* _ $\bigcup A$ λx . *x*
by *unfold_locales* (*simp_all add*:*lam_replacement_identity*)
from ‹$M(A)$›
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 *assms*
show *?thesis*
using *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*,*rename_tac f*,*rule_tac*
x=*f* **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*: $0 \notin A \Longrightarrow M(A) \Longrightarrow \exists f \in A \to^M \bigcup(A).\ \forall x \in A.\ f`x \in x$
by (*rule AC_M_func*) (*simp_all add*: *non_empty_family*)

lemma *AC_M_func_Pow_rel*:
assumes $M(C)$
shows $\exists f \in (Pow^M(C)\text{-}\{0\}) \to^M C.\ \forall x \in Pow^M(C)\text{-}\{0\}.\ f`x \in x$
proof -
have $0{\notin}Pow^M(C)\text{-}\{0\}$ **by** *simp*
with *assms*
obtain *f* **where** $f \in (Pow^M(C)\text{-}\{0\}) \to^M \bigcup(Pow^M(C)\text{-}\{0\})\ \forall x \in Pow^M(C)\text{-}\{0\}.$
$f`x \in x$
using *AC_M_func0*[*OF* ‹$0{\notin}$_›] **by** *auto*

```
  moreover
  have x⊆C if x ∈ Pow^M(C) - {0} for x
    using that Pow_rel_char assms
    by auto
  moreover
  have ⋃(Pow^M(C) - {0}) ⊆ 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 ∀x∈A. ∃y∈A. ⟨x,y⟩∈ R M(A) M(R)
  shows ∀a∈A. ∃f ∈ nat→^M A. f‘0 = a ∧ (∀n ∈ nat. ⟨f‘n,f‘succ(n)⟩∈R)
proof -
  have 0:∀y∈A. {x ∈ A . ⟨y, x⟩ ∈ R} ≠ 0
    using assms by auto
  from ‹M(A)›
  obtain g where 1: g ∈ Pow^M(A)-{0} → A ∀X[M]. X ≠ 0 ∧ X ⊆ A ⟶ g ‘ X
∈ X
    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:∀X[M]. X ≠ 0 ∧ X ⊆ A ⟶ g ‘ X ∈ A by auto
  {
    fix a
    assume a∈A
    let ?f =λn∈nat. dc_witness(n,A,a,g,R)
    note ‹a∈A›
    moreover from this
    have f0: ?f‘0 = a by simp
    moreover
    note ‹a∈A› ‹M(g)› ‹M(A)› ‹M(R)›
    moreover from calculation
    have ⟨?f ‘ n, ?f ‘ succ(n)⟩ ∈ R if n∈nat for n
      using witness_related[OF ‹a∈A› _ 0, of g n] 1 that
      by (auto dest:transM)
    ultimately
    have ∃f[M]. f∈nat → A ∧ f ‘ 0 = a ∧ (∀n∈nat. ⟨f ‘ n, f ‘ succ(n)⟩ ∈ R)
      using 0 1 ‹a∈_›
      by (rule_tac x=(λn∈ω. dc_witness(n, A, a, g, R)) in rexI)
        (simp_all, rule_tac witness_funtype,
          auto intro!: Lambda_dc_witness_closed dest:transM)
  }
  with ‹M(A)›
  show ?thesis using function_space_rel_char by auto
```

qed

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

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

corollary *DC_on_A_x_nat* :
  assumes $(\forall x{\in}A{\times}nat.\ \exists y{\in}A.\ \langle x,\langle y,succ(snd(x))\rangle\rangle \in R)$ $a{\in}A$ $M(A)$ $M(R)$
  shows $\exists f \in nat{\rightarrow}^M A.\ f`0 = a \wedge (\forall n \in nat.\ \langle\langle f`n,n\rangle,\langle f`succ(n),succ(n)\rangle\rangle{\in}R)$ (**is** $\exists x{\in}_.\ ?P(x)$)
proof -
  let $?R'{=}\{\langle a,b\rangle{\in}R.\ snd(b) = succ(snd(a))\}$
  from *assms*(*1*)
  have $\forall x{\in}A{\times}nat.\ \exists y{\in}A{\times}nat.\ \langle x,y\rangle \in ?R'$
    using *aux_DC_on_AxNat2* **by** *simp*
  moreover
  note ‹$a{\in}_$› ‹$M(A)$›
  moreover from *this*
  have $M(A \times \omega)$ **by** *simp*
  have $lam_replacement(M,\ \lambda x.\ succ(snd(fst(x))))$
    using *lam_replacement_fst lam_replacement_snd lam_replacement_hcomp*
      lam_replacement_hcomp[*of* _ $\lambda x.\ succ(snd(x))$]
      lam_replacement_succ **by** *simp*
  with ‹$M(R)$›
  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**
    $F{:}f{\in}nat{\rightarrow}^M\ A{\times}\omega\ f`0 = \langle a,0\rangle\ \ \forall n{\in}nat.\ \langle f`n,\ f`succ(n)\rangle \in ?R'$
    using *pointed_DC*[*of* $A{\times}nat$ $?R'$] **by** *blast*
  with ‹$M(A)$›
  have $M(f)$ **using** *function_space_rel_char* **by** *simp*
  then
  have $M(\lambda x{\in}nat.\ fst(f`x))$ (**is** $M(?f)$)
    using *lam_replacement_fst lam_replacement_hcomp*
      lam_replacement_constant lam_replacement_identity
      lam_replacement_apply
    by (*rule_tac lam_replacement_imp_lam_closed*, *auto*)
  with F ‹$M(A)$›
  have $?f{\in}nat{\rightarrow}^M\ A\ \ ?f`0 = a$ **using** *function_space_rel_char* **by** *auto*
  have $1{:}n{\in}\ nat \Longrightarrow f`n{=}\ \langle ?f`n,\ n\rangle$ **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 Pair_fst_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_replacement_fst lam_replacement_snd lam_replacement_apply[of
S]
      lam_replacement_product[of λx. fst(fst(x)) λx. fst(snd(x)),
        THEN [2] separation_in, of λx. S ‘ snd(snd(x))]
      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

31 Cohen forcing notions

theory *Partial_Functions_Relative*
  imports
    Cardinal_Library_Relative
begin

In this theory we introduce bounded partial functions and its relative version; for historical reasons the relative version is based on a proper definition of partial functions.

We note that finite partial functions are easier and are used to prove some lemmas about finite sets in the theory *Transitive_Models.ZF_Library_Relative*.

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 $?f=\lambda x\in domain(r)\ .\ \langle x,r`x\rangle$
  have $\langle x,\ r\ `\ x\rangle \in r$ **if** $\langle x,\ y\rangle \in r$ **for** $x\ y$
  proof -
    have $x\in domain(r)$ **using** *that* **by** *auto*
    with *assms*
    show *?thesis* **using** *function_apply_Pair* **by** *auto*
  qed
  then
  have $?f \in inj(domain(r),\ r)$
    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 *calculation*
  have $r`fst(x) = snd(x)$ **if** $x\in r$ **for** x
    using *that apply_equality*
    by *auto*

ultimately
have *?f∈inj(r,d)*
by(*rule_tac d= λu . ⟨u, r‘u⟩* **in** *lam_injective,auto*)
then
show *?thesis*
unfolding *lepoll_def*
by *auto*
qed

lemma *function_eqpoll* :
assumes *r:d→J*
shows *r ≈ 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* : *Fn(κ,I,J) = {f ∈ Pow(I×J) . function(f) ∧ f ≺ κ}* (**is** *?L=?R*)
proof (*intro equalityI subsetI*)
fix *x*
assume *x ∈ ?R*
moreover from *this*
have *domain(x) ∈ Pow(I) domain(x) ≲ x x≺ κ*
using *domain_function_lepoll*
by *auto*
ultimately
show *x ∈ ?L*
unfolding *Fn_def*
using *lesspoll_trans1 Pi_iff*
by (*auto,rule_tac rev_bexI*[*of domain(x) → J*],*auto*)
next
fix *x*
assume *x ∈ ?L*
then
obtain *d* **where** *x:d→J d ∈ Pow(I) d≺κ*
unfolding *Fn_def*
by *auto*
moreover from *this*
have *x≺κ*
using *function_lepoll*[*THEN lesspoll_trans1*] **by** *auto*
moreover from *calculation*
have *x ∈ Pow(I×J) function(x)*
using *Pi_iff* **by** *auto*
ultimately
show *x ∈ ?R* **by** *simp*
qed

lemma *zero_in_Fn*:
assumes *0 < κ*

```
  shows 0 ∈ Fn(κ, I, J)
  using lt_Card_imp_lesspoll assms zero_lesspoll
  unfolding Fn_def
  by (simp,rule_tac x=0→J in bexI,simp)
    (rule ReplaceI[of _ 0],simp_all)

lemma Fn_nat_eq_FiniteFun: Fn(nat,I,J) = I -||> J
proof (intro equalityI subsetI)
  fix x
  assume x ∈ I -||> J
  then
  show x ∈ Fn(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→J d≺nat d⊆I
      unfolding Fn_def by auto
    moreover from this
    have Finite(d)
      using lesspoll_nat_is_Finite by simp
    ultimately
    have h : d -||> J
      using fun_FiniteFunI Finite_into_Fin by blast
    note ‹h:d→J›
    moreover from this
    have domain(cons(⟨a, b⟩, h)) = cons(a,d) (is domain(?h) = ?d)
      and domain(h) = d
      using domain_of_fun by simp_all
    moreover
    note consI
    moreover from calculation
    have cons(⟨a, b⟩, h) ∈ cons(a,d) → J
      using fun_extend3 by simp
    moreover from ‹Finite(d)›
    have Finite(cons(a,d)) by simp
    moreover from this
    have cons(a,d) ≺ nat using Finite_imp_lesspoll_nat by simp
    ultimately
    show ?case
      unfolding Fn_def
      by (simp,rule_tac x=?d→J in bexI)
        (force dest:app_fun)+
  qed
```

next
  fix x
  assume $x \in Fn(nat,I,J)$
  then
  obtain d **where** $x{:}d{\to}J\ d \in Pow(I)\ d{\prec}nat$
    unfolding *Fn_def*
    by *auto*
  moreover from *this*
  have $Finite(d)$
    using *lesspoll_nat_is_Finite* **by** *simp*
  moreover from *calculation*
  have $d \in Fin(I)$
    using *Finite_into_Fin*[*of d*] *Fin_mono* **by** *auto*
  ultimately
  show $x \in I$ -||> J **using** *fun_FiniteFunI FiniteFun_mono* **by** *blast*
qed

lemma *Fn_nat_subset_Pow*: $Fn(\kappa,I,J) \subseteq Pow(I{\times}J)$
  using *Fn_char* **by** *auto*

lemma *FnI*:
  assumes $p : d \to J\ d \subseteq I\ d \prec \kappa$
  shows $p \in Fn(\kappa,I,J)$
  using *assms*
  unfolding *Fn_def* **by** *auto*

lemma *FnD*[*dest*]:
  assumes $p \in Fn(\kappa,I,J)$
  shows $\exists\, d.\ p : d \to J \wedge d \subseteq I \wedge d \prec \kappa$
  using *assms*
  unfolding *Fn_def* **by** *auto*

lemma *Fn_is_function*: $p \in Fn(\kappa,I,J) \Longrightarrow function(p)$
  unfolding *Fn_def* **using** *fun_is_function* **by** *auto*

lemma *Fn_csucc*:
  assumes $Ord(\kappa)$
  shows $Fn(csucc(\kappa),I,J) = \bigcup\{y\ .\ d \in Pow(I),\ y{=}(d{\to}J) \wedge d{\lesssim}\kappa\}$
  using *assms*
  unfolding *Fn_def* **using** *lesspoll_csucc* **by** (*simp*)

definition
  $FnleR :: i \Rightarrow i \Rightarrow o$ (**infixl** ‹$\supseteq$› *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

Fnlerel :: $i \Rightarrow i$ **where**
Fnlerel$(A) \equiv$ *Rrel*$(\lambda x\ y.\ x \supseteq y, A)$

definition
Fnle :: $[i,i,i] \Rightarrow i$ **where**
Fnle$(\kappa,I,J) \equiv$ *Fnlerel*$(Fn(\kappa,I,J))$

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** *Fnlerel_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** *Fnlerel_def Fnle_def FnleR_def Rrel_def*
by *auto*

definition *PFun_Space_Rel* :: $[i,i\Rightarrow o,\ i] \Rightarrow i$ (‹_⇀_›)
where $A \rightharpoonup^M B \equiv \{f \in Pow(A\times B)\ .\ M(f) \wedge function(f)\}$

lemma (**in** *M_library*) *PFun_Space_subset_Powrel* :
assumes $M(A)$ $M(B)$
shows $A \rightharpoonup^M B = \{f \in Pow^M(A\times B)\ .\ function(f)\}$
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 $M(A \rightharpoonup^M B)$
using *assms PFun_Space_subset_Powrel separation_is_function*
by *auto*

lemma *pfun_is_function* :
$f \in A \rightharpoonup^M B \Longrightarrow function(f)$
unfolding *PFun_Space_Rel_def* **by** *simp*

lemma *pfun_range* :
$f \in A \rightharpoonup^M B \Longrightarrow range(f) \subseteq B$
unfolding *PFun_Space_Rel_def* **by** *auto*

lemma *pfun_domain* :
$f \in A \rightharpoonup^M B \Longrightarrow domain(f) \subseteq A$
unfolding *PFun_Space_Rel_def* **by** *auto*

lemma *Un_filter_fun_space_closed*:
assumes $G \subseteq I \rightarrow J \bigwedge f\ g\ .\ f\in G \Longrightarrow g\in G \Longrightarrow \exists\, d\in I\rightarrow J\ .\ d \supseteq f \wedge d \supseteq g$

```
  shows ⋃G ∈ Pow(I×J) function(⋃G)
proof -
  from assms
  show ⋃G ∈ Pow(I×J)
    using Union_Pow_iff
    unfolding Pi_def
    by auto
next
  show function(⋃G)
    unfolding function_def
  proof(auto)
    fix B B' x y y'
    assume B ∈ G ⟨x, y⟩ ∈ B B' ∈ G ⟨x, y'⟩ ∈ B'
    moreover from assms this
    have B ∈ I → J B' ∈ I → J
      by auto
    moreover from calculation assms(2)[of B B']
    obtain d where d ⊇ B  d ⊇ B' d∈I → J  ⟨x, y⟩ ∈ d ⟨x, y'⟩ ∈ d
      using subsetD[OF ‹G⊆_›]
      by auto
    then
    show y=y'
      using fun_is_function[OF ‹d∈_›]
      unfolding function_def
      by force
  qed
qed

lemma Un_filter_is_fun :
  assumes G ⊆ I → J ⋀ f g . f∈G ⟹ g∈G ⟹ ∃ d∈I→ J . d⊇f ∧ d⊇g G≠0
  shows ⋃G ∈ I → J
  using assms Un_filter_fun_space_closed Pi_iff
proof(simp_all)
  show I⊆domain(⋃G)
  proof -
    from ‹G≠0›
    obtain f where f⊆⋃G f∈G
      by auto
    with ‹G⊆_›
    have f∈I→J by auto
    then
    show ?thesis
      using subset_trans[OF _ domain_mono[OF ‹f⊆⋃G›],of I]
      unfolding Pi_def by auto
  qed
qed

context M_Pi
begin
```

lemma *mem_function_space_relD*:
  assumes $f \in$ *function_space_rel*(M,A,y) $M(A)$ $M(y)$
  shows $f \in A \to y$ **and** $M(f)$
  using *assms function_space_rel_char* **by** *simp_all*

lemma *pfunI* :
  assumes $C \subseteq A$ $f \in C \to^M B$ $M(C)$ $M(B)$
  shows $f \in A \rightharpoonup^M B$
proof -
  from *assms*
  have $f \in C \to B$ $M(f)$
    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 $M(I)$ $M(J)$
  shows $0 \in I \rightharpoonup^M J$
  using *pfunI*[*of 0*] *nonempty mem_function_space_rel_abs assms*
  by *simp*

lemma *pfun_subsetI* :
  assumes $f \in A \rightharpoonup^M B$ $g \subseteq f$ $M(g)$
  shows $g \in A \rightharpoonup^M B$
  using *assms function_subset*
  unfolding *PFun_Space_Rel_def*
  by *auto*

lemma *pfun_Un_filter_closed*:
  assumes $G \subseteq I \rightharpoonup^M J$ $\bigwedge f\ g\ .\ f \in G \Longrightarrow g \in G \Longrightarrow \exists\, d \in I \rightharpoonup^M J\ .\ d \supseteq f \wedge d \supseteq g$
  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 *PFun_Space_Rel_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 calculation assms
    obtain d where d ∈ I ⇀^M J function(d) ⟨x, y⟩ ∈ d  ⟨x, y'⟩ ∈ 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 ⊆I⇀^M J ⋀ f g . f∈G ⟹ g∈G ⟹ ∃ d∈G . d⊇f ∧ d⊇g
  shows ⋃ G ∈ Pow(I×J) function(⋃ G)
proof -
  from assms
  have ⋀ f g . f∈G ⟹ g∈G ⟹ ∃ d∈I⇀^M J . d⊇f ∧ d⊇g
    using subsetD[OF assms(1),THEN [2] bexI]
    by force
  then
  show ⋃ G ∈ Pow(I×J)  function(⋃ G)
    using assms pfun_Un_filter_closed
    unfolding PFun_Space_Rel_def
    by auto
qed

lemma pfun_Un_filter_closed':
  assumes G ⊆I⇀^M J ⋀ f g . f∈G ⟹ g∈G ⟹ ∃ d∈G . d⊇f ∧ d⊇g M(G)
  shows ⋃ G ∈ I⇀^M J
  using assms pfun_Un_filter_closed''
  unfolding PFun_Space_Rel_def
  by auto

lemma pfunD :
  assumes f ∈ A⇀^M B
  shows ∃ C[M]. C⊆A ∧ f ∈ C→B
proof -
  note assms
  moreover from this
  have f∈Pow(A×B) function(f) M(f)
    unfolding PFun_Space_Rel_def
    by simp_all
  moreover from this
  have domain(f) ⊆ A f ∈ domain(f) → 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 $f \in A \rightharpoonup^M B$
  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 $\{\langle x,b\rangle\} \in A \rightharpoonup^M B$
  using *assms transM*[*of x A*] *transM*[*of b B*]
  unfolding *PFun_Space_Rel_def function_def*
  by *auto*

lemma *pfun_unionI* :
  assumes $f \in A \rightharpoonup^M B$ $g \in A \rightharpoonup^M B$ $domain(f) \cap domain(g) = 0$
  shows $f \cup g \in A \rightharpoonup^M B$
  using *assms*
  unfolding *PFun_Space_Rel_def function_def*
  by *blast*

lemma (**in** *M_library*) *pfun_restrict_eq_imp_compat*:
  assumes $f \in I \rightharpoonup^M J$ $g \in I \rightharpoonup^M J$ $M(J)$
    $restrict(f, domain(f) \cap domain(g)) = restrict(g, domain(f) \cap domain(g))$
  shows $f \cup g \in I \rightharpoonup^M J$
proof -
  note *assms*
  moreover from *this*
  obtain C D **where** $f : C \to J$ $C \subseteq I$ $D \subseteq I$ $M(C)$ $M(D)$ $g : D \to J$
    using *pfunD*[*of f*] *pfunD*[*of g*] **by** *force*
  moreover from *calculation*
  have $f \cup g \in C \cup D \to J$
    using *restrict_eq_imp_Un_into_Pi'*[*OF* ‹$f \in C \to _$› ‹$g \in D \to _$›]
    by *auto*
  ultimately
  show *?thesis*
   using *pfunI*[*of* $C \cup D$ _ $f \cup g$] *Un_subset_iff pfunD_closed function_space_rel_char*
    by *auto*
qed

lemma *FiniteFun_pfunI* :
  assumes $f \in A$-||> B $M(A)$ $M(B)$
  shows $f \in A \rightharpoonup^M B$
  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 $M(a)$ $M(b)$ $M(h)$ $domain(h) \subseteq A$
    using *transM*[*OF _* ‹$M(A)$›] *transM*[*OF _* ‹$M(B)$›]
      FinD
      FiniteFun_domain_Fin
      pfunD_closed
    by *simp_all*
  moreover from *calculation*
  have $\{a\}\cup domain(h)\subseteq A$ $M(\{a\}\cup domain(h))$ $M(cons(<a,b>,h))$ $domain(cons(<a,b>,h))$
$= \{a\}\cup domain(h)$
    by *auto*
  moreover from *calculation*
  have $cons(<a,b>,h) \in \{a\}\cup domain(h) \to B$
    using *FiniteFun_is_fun*[*OF FiniteFun.consI*, *of a A b B h*]
    by *auto*
  ultimately
  show $cons(<a,b>,h) \in A \rightharpoonup^M B$
    using *assms mem_function_space_rel_abs pfunI*
    by *simp_all*
qed

lemma *PFun_FiniteFunI* :
  assumes $f \in A \rightharpoonup^M B$ $Finite(f)$
  shows $f \in A\text{-}||> B$
proof -
  from *assms*
  have $f\in Fin(A\times B)$ $function(f)$
    using *Finite_Fin Pow_iff*
    unfolding *PFun_Space_Rel_def*
    by *auto*
  then
  show *?thesis*
    using *FiniteFunI* **by** *simp*
qed

end — *M_Pi*

definition
  Fn_rel :: $[i\Rightarrow o,i,i,i] \Rightarrow i$ (‹*Fn*–′(_,_,_′)›) **where**
  $Fn_rel(M,\kappa,I,J) \equiv \{f \in I\rightharpoonup^M J \,.\, f \prec^M \kappa\}$
— Eventually we can define *Fn_rel* as the relativization of *Fn*

context *M_library*
begin

lemma *Fn_rel_subset_PFun_rel* : $Fn^M(\kappa,I,J) \subseteq I \rightharpoonup^M J$
  unfolding *Fn_rel_def* **by** *auto*

lemma *Fn_relI*[*intro*]:
  assumes $f : d \to J$ $d \subseteq I$ $f \prec^M \kappa$ $M(d)$ $M(J)$ $M(f)$
  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 $\exists\, C[M].\ C \subseteq I \wedge p : C \to J \wedge p \prec^M \kappa$
  using *assms pfunD*
  unfolding *Fn_rel_def*
  by *simp*

lemma *Fn_rel_is_function*:
  assumes $p \in Fn_rel(M,\kappa,I,J)$
  shows $function(p)$ $M(p)$ $p \prec^M \kappa$ $p \in I \rightharpoonup^M J$
  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 $\kappa \prec^M \kappa'$ | $\kappa \approx^M \kappa'$
    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(κ) M(κ)
  shows Fn_rel(M,(κ⁺)^M,I,J) = {p∈ I⇀^M J . p ≲^M κ}   (is ?L=?R)
  using assms
proof(intro equalityI)
  show ?L ⊆ ?R
  proof(intro subsetI)
    fix p
    assume p∈?L
    then
    have p ≺^M csucc_rel(M,κ) M(p) p∈ I⇀^M J
      using Fn_rel_is_function by simp_all
    then
    show p∈?R
      using  assms lesspoll_rel_csucc_rel by simp
  qed
next
  show ?R⊆?L
  proof(intro subsetI)
    fix p
    assume p∈?R
    then
    have  p∈ I⇀^M J p ≲^M κ
      using assms lesspoll_rel_csucc_rel by simp_all
    then
    show p∈?L
      using  assms lesspoll_rel_csucc_rel pfunD_closed
      unfolding Fn_rel_def
      by simp
  qed
qed

lemma Finite_imp_lesspoll_nat:
  assumes Finite(A)
  shows A ≺ nat
  using assms subset_imp_lepoll[OF naturals_subset_nat] eq_lepoll_trans
    n_lesspoll_nat eq_lesspoll_trans
  unfolding Finite_def lesspoll_def by auto

lemma FinD_Finite :
  assumes a∈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 ≺^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 ω
    using Finite_lesspoll_rel_nat pfunD_closed[of p] n_lesspoll_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 ≺^M J  p ≺^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

lemma Fn_rel_singletonI:
  assumes x ∈ I j ∈ J 1 ≺^M κ M(κ) M(I) M(J)
  shows {⟨x,j⟩} ∈ Fn^M(κ,I,J)
  using pfun_singletonI transM[of x] transM[of j] assms
    eq_lesspoll_rel_trans[OF singleton_eqpoll_rel_1]
```

unfolding *Fn_rel_def*
by *auto*

end — *M_library*

definition
Fnle_rel :: $[i \Rightarrow o,i,i,i] \Rightarrow i$ (‹*Fnle*–′(_,_,_′)›) **where**
$Fnle_rel(M,\kappa,I,J) \equiv Fnlerel(Fn^M(\kappa,I,J))$

abbreviation
Fn_r_set :: $[i,i,i,i] \Rightarrow i$ (‹*Fn*–′(_,_,_′)›) **where**
$Fn_r_set(M) \equiv Fn_rel(\#\#M)$

abbreviation
Fnle_r_set :: $[i,i,i,i] \Rightarrow i$ (‹*Fnle*–′(_,_,_′)›) **where**
$Fnle_r_set(M) \equiv Fnle_rel(\#\#M)$

context *M_library*
begin

lemma *Fnle_relI*[*intro*]:
assumes $p \in Fn_rel(M,\kappa,I,J)$ $q \in Fn_rel(M,\kappa,I,J)$ $p \supseteq q$
shows $\langle p,\ q\rangle \in Fnle_rel(M,\kappa,I,J)$
using *assms* **unfolding** *Fnlerel_def Fnle_rel_def FnleR_def Rrel_def*
by *auto*

lemma *Fnle_relD*[*dest*]:
assumes $\langle p,\ q\rangle \in Fnle_rel(M,\kappa,I,J)$
shows $p \in Fn_rel(M,\kappa,I,J)$ $q \in Fn_rel(M,\kappa,I,J)$ $p \supseteq q$
using *assms* **unfolding** *Fnlerel_def Fnle_rel_def FnleR_def Rrel_def*
by *auto*

lemma *Fn_rel_closed*[*intro*,*simp*]:
assumes $M(\kappa)$ $M(I)$ $M(J)$
shows $M(Fn^M(\kappa,I,J))$
using *assms separation_cardinal_rel_lesspoll_rel PFun_Space_closed*
unfolding *Fn_rel_def*
by *auto*

lemma *Fn_rel_subset_Pow*:
assumes $M(\kappa)$ $M(I)$ $M(J)$
shows $Fn^M(\kappa,I,J) \subseteq Pow(I \times J)$
unfolding *Fn_rel_def PFun_Space_Rel_def*
by *auto*

lemma *Fnle_rel_closed*[*intro*,*simp*]:
assumes $M(\kappa)$ $M(I)$ $M(J)$
shows $M(Fnle^M(\kappa,I,J))$

unfolding *Fnle_rel_def Fnlerel_def Rrel_def FnleR_def*
using *assms supset_separation Fn_rel_closed*
by *auto*

lemma *zero_in_Fn_rel*:
assumes $0<\kappa$ $M(\kappa)$ $M(I)$ $M(J)$
shows $0 \in Fn^M(\kappa, I, J)$
unfolding *Fn_rel_def*
using *zero_in_PFun_rel zero_lesspoll_rel assms*
by *simp*

lemma *zero_top_Fn_rel*:
assumes $p \in Fn^M(\kappa, I, J)$ $0<\kappa$ $M(\kappa)$ $M(I)$ $M(J)$
shows $\langle p, 0\rangle \in Fnle^M(\kappa, I, J)$
using *assms zero_in_Fn_rel* **unfolding** *preorder_on_def refl_def trans_on_def*
by *auto*

lemma *preorder_on_Fnle_rel*:
assumes $M(\kappa)$ $M(I)$ $M(J)$
shows $preorder_on(Fn^M(\kappa, I, J), Fnle^M(\kappa, I, J))$
unfolding *preorder_on_def refl_def trans_on_def*
by *blast*

end — *M_library*

context *M_cardinal_library*
begin

lemma *lesspoll_nat_imp_lesspoll_rel*:
assumes $A \prec \omega$ $M(A)$
shows $A \prec^M \omega$
proof -
note *assms*
moreover from *this*
obtain f n **where** $f \in bij^M(A,n)$ $n \in \omega$ $A \approx^M n$
using *lesspoll_nat_is_Finite Finite_rel_def*[*of M A*]
by *auto*
moreover from *calculation*
have $A \lesssim^M \omega$
using *lesspoll_nat_is_Finite Infinite_imp_nats_lepoll_rel*[*of* ω n]
nat_not_Finite eq_lepoll_rel_trans[*of A n* ω]
by *auto*
moreover from *calculation*
have $\neg$ $g \in bij^M(A,\omega)$ **for** g
using *mem_bij_rel* **unfolding** *lesspoll_def* **by** *auto*
ultimately
show *?thesis*
unfolding *lesspoll_rel_def*
by *auto*

```
qed

lemma Finite_imp_lesspoll_rel_nat:
  assumes Finite(A) M(A)
  shows A ≺^M ω
  using Finite_imp_lesspoll_nat assms lesspoll_nat_imp_lesspoll_rel
  by auto

end — M_cardinal_library

context M_cardinal_library_extra
begin

lemma InfCard_rel_lesspoll_rel_Un:
  includes Ord_dests
  assumes InfCard_rel(M,κ) A ≺^M κ B ≺^M κ
    and types: M(κ) M(A) M(B)
  shows A ∪ B ≺^M κ
proof -
  from assms
  have |A|^M < κ |B|^M < κ
    using lesspoll_rel_cardinal_rel_lt InfCard_rel_is_Card_rel
    by auto
  show ?thesis
  proof (cases Finite(A) ∧ Finite(B))
    case True
    with assms
    show ?thesis
      using lesspoll_rel_trans2[OF _ le_imp_lepoll_rel, of _ nat κ]
        Finite_imp_lesspoll_rel_nat[OF Finite_Un]
      unfolding InfCard_rel_def
      by simp
  next
    case False
    with types
    have InfCard_rel(M,max(|A|^M,|B|^M))
      using Infinite_InfCard_rel_cardinal_rel InfCard_rel_is_Card_rel
        le_trans[of nat] not_le_iff_lt[THEN iffD1, THEN leI, of |A|^M |B|^M]
      unfolding max_def InfCard_rel_def
      by auto
    with ‹M(A)› ‹M(B)›
    have |A ∪ B|^M ≤ max(|A|^M,|B|^M)
      using cardinal_rel_Un_le[of max(|A|^M,|B|^M) A B]
        not_le_iff_lt[THEN iffD1, THEN leI, of |A|^M |B|^M ]
      unfolding max_def
      by simp
    moreover from ‹|A|^M < κ› ‹|B|^M < κ›
    have max(|A|^M,|B|^M) < κ
      unfolding max_def
```

```
      by simp
    ultimately
    have |A ∪ B|^M <  κ
      using lt_trans1
      by blast
    moreover
    note ‹InfCard_rel(M,κ)›
    moreover from calculation types
    have |A ∪ B|^M ≺^M κ
      using lt_Card_rel_imp_lesspoll_rel InfCard_rel_is_Card_rel
      by simp
    ultimately
    show ?thesis
      using cardinal_rel_eqpoll_rel eq_lesspoll_rel_trans[of A ∪ B |A ∪ B|^M κ]
        eqpoll_rel_sym types
      by simp
  qed
qed

lemma Fn_rel_unionI:
  assumes p ∈ Fn^M(κ,I,J) q∈Fn^M(κ,I,J) InfCard^M(κ)
    M(κ) M(I) M(J) domain(p) ∩ domain(q) = 0
  shows p∪q ∈ Fn^M(κ,I,J)
proof -
  note assms
  moreover from calculation
  have p ≺^M κ q ≺^M κ M(p) M(q)
    using Fn_rel_is_function by simp_all
  moreover from calculation
  have p∪q ≺^M κ
    using eqpoll_rel_sym cardinal_rel_eqpoll_rel InfCard_rel_lesspoll_rel_Un
    by simp_all
  ultimately
  show ?thesis
    unfolding Fn_rel_def
    using pfun_unionI cardinal_rel_eqpoll_rel eq_lesspoll_rel_trans[OF _ ‹p∪q
≺^M _›]
    by auto
qed

lemma restrict_eq_imp_compat_rel:
  assumes p ∈ Fn^M(κ, I, J) q ∈ Fn^M(κ, I, J) InfCard^M(κ) M(J) M(κ)
    restrict(p, domain(p) ∩ domain(q)) = restrict(q, domain(p) ∩ domain(q))
  shows p ∪ q ∈ Fn^M(κ, I, J)
proof -
  note assms
  moreover from calculation
  have p ≺^M κ q ≺^M κ M(p) M(q)
    using Fn_rel_is_function by simp_all
```

moreover from *calculation*
have $p \cup q \prec^M \kappa$
using *InfCard_rel_lesspoll_rel_Un cardinal_rel_eqpoll_rel*[*THEN eqpoll_rel_sym*]
by *auto*
ultimately
show *?thesis*
unfolding *Fn_rel_def*
using *pfun_restrict_eq_imp_compat cardinal_rel_eqpoll_rel*
eq_lesspoll_rel_trans[*OF* _ ‹$p \cup q \prec^M$ _›]
by *auto*
qed

lemma *InfCard_rel_imp_n_lesspoll_rel* :
assumes $InfCard^M(\kappa)$ $M(\kappa)$ $n \in \omega$
shows $n \prec^M \kappa$
proof -
note *assms*
moreover from *this*
have $n \prec^M \omega$
using *n_lesspoll_rel_nat* **by** *simp*
ultimately
show *?thesis*
using *lesspoll_rel_trans2 Infinite_iff_lepoll_rel_nat InfCard_rel_imp_Infinite nat_into_M*
by *simp*
qed

lemma *cons_in_Fn_rel*:
assumes $x \notin domain(p)$ $p \in Fn^M(\kappa,I,J)$ $x \in I$ $j \in J$ $InfCard^M(\kappa)$
$M(\kappa)$ $M(I)$ $M(J)$
shows $cons(\langle x,j\rangle, p) \in Fn^M(\kappa,I,J)$
using *assms cons_eq Fn_rel_unionI*[*OF Fn_rel_singletonI*[*of x I j J*] ‹$p \in$_›]
InfCard_rel_imp_n_lesspoll_rel
by *auto*

lemma *dense_dom_dense*:
assumes $x \in I$ $InfCard^M(\kappa)$ $M(I)$ $M(J)$ $z \in J$ $M(\kappa)$ $p \in Fn^M(\kappa, I, J)$
shows $\exists\, d \in \{ p \in Fn^M(\kappa, I, J) . x \in domain(p) \}. \langle d,p\rangle \in Fnle^M(\kappa, I, J)$
proof (*cases* $x \in domain(p)$)
case *True*
with ‹$x \in I$› ‹$p \in Fn^M(\kappa, I, J)$›
show *?thesis* **by** *auto*
next
case *False*
note ‹$p \in Fn^M(\kappa, I, J)$›
moreover from *this* **and** *False* **and** *assms*
have $cons(\langle x,z\rangle, p) \in Fn^M(\kappa, I, J)$ $M(x)$
using *cons_in_Fn_rel* **by** (*auto dest*:*transM*)
ultimately

```
  show ?thesis using Fn_relD by blast
qed

lemma (in M_cardinal_library) domain_lepoll_rel :
  assumes function(r) M(r)
  shows domain(r) ≲^M r
proof -
  let ?f=λx∈domain(r) . ⟨x,r‘x⟩
  have ⟨x, r ‘ x⟩ ∈ r if ⟨x, y⟩ ∈ r for x y
  proof -
    have x∈domain(r) using that by auto
    with assms
    show ?thesis using function_apply_Pair by auto
  qed
  then
  have ?f ∈ inj(domain(r), r)
    by(rule_tac lam_injective,auto)
  moreover note assms
  moreover from calculation
  have M(?f)
    using lam_replacement_constant[of r] lam_replacement_identity assms
    lam_replacement_apply lam_replacement_Pair[THEN [5] lam_replacement_hcomp2]
    by(rule_tac lam_replacement_imp_lam_closed,auto dest:transM[of _ r])
  ultimately
  have ?f ∈ inj^M(domain(r),r) using inj_rel_char
    by auto
  with ‹M(?f)›
  show ?thesis
    using lepoll_relI by simp
qed

lemma dense_surj_dense:
  assumes x ∈ J InfCard^M(κ) M(I) M(J) M(κ) p∈Fn^M(κ, I, J) κ ≲^M I
  shows ∃ d∈{ p ∈ Fn^M(κ, I, J) . x ∈ range(p) }. ⟨d,p⟩ ∈ Fnle^M(κ, I, J)
proof -
  from ‹p ∈ _› ‹M(J)› ‹x∈_› lesspoll_rel_def
  have domain(p) ⊆ I M(p) p ≺^M κ M(x) function(p)
   using pfun_domain[OF Fn_rel_is_function(4)] Fn_rel_is_function transM[of
x J]
    by simp_all
  moreover from calculation assms
  have 1:domain(p) ≺^M κ M(domain(p)) domain(p) ≺^M I
   using domain_lepoll_rel lesspoll_rel_trans1[of domain(p) p κ] lesspoll_rel_trans2
    by auto
  with calculation ‹κ ≲^M I› ‹M(I)› ‹M(κ)›
  have domain(p) ≠ I
  proof -
    {assume domain(p) = I
      with 1
```

```
      have domain(p) ≺^M domain(p)
        by auto
      with ‹M(domain(p))›
      have False
        using lesspoll_rel_irrefl[of domain(p)] by simp
    }
    then show ?thesis by auto
  qed
  ultimately
  obtain α where α ∉ domain(p) α∈I
    by force
  moreover note assms
  moreover from calculation
  have cons(⟨α,x⟩, p) ∈ Fn^M(κ, I, J)
    using InfCard_rel_Aleph_rel cons_in_Fn_rel[of α p κ I J x]
    by auto
  ultimately
  show ?thesis
    using Fnle_relI by blast
qed

end — M_cardinal_library_extra

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] Y.N. Moschovakis, "Notes on Set Theory", Springer Texts in Electrical Engineering, Springer-Verlag (1994).

[3] 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/.