

# The Constructible Universe and the Relative Consistency of the Axiom of Choice

Lawrence C Paulson

December 17, 2025

## Abstract

Gödel’s proof of the relative consistency of the axiom of choice [1] is one of the most important results in the foundations of mathematics. It bears on Hilbert’s first problem, namely the continuum hypothesis, and indeed Gödel also proved the relative consistency of the continuum hypothesis. Just as important, Gödel’s proof introduced the *inner model* method of proving relative consistency, and it introduced the concept of *constructible set*. Kunen [2] gives an excellent description of this body of work.

This Isabelle/ZF formalization demonstrates Gödel’s claim that his proof can be undertaken without using metamathematical arguments, for example arguments based on the general syntactic structure of a formula. Isabelle’s automation replaces the metamathematics, although it does not eliminate the requirement at least to state many tedious results that would otherwise be unnecessary.

This formalization [4] is by far the deepest result in set theory proved in any automated theorem prover. It rests on a previous formal development of the reflection theorem [3].

## Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>First-Order Formulas and the Definition of the Class L</b> | <b>10</b> |
| 1.1      | Internalized formulas of FOL . . . . .                        | 10        |
| 1.2      | Dividing line between primitive and derived connectives . .   | 12        |
| 1.2.1    | Derived rules to help build up formulas . . . . .             | 12        |
| 1.3      | Arity of a Formula: Maximum Free de Bruijn Index . . . . .    | 13        |
| 1.4      | Renaming Some de Bruijn Variables . . . . .                   | 14        |
| 1.5      | Renaming all but the First de Bruijn Variable . . . . .       | 16        |
| 1.6      | Definable Powerset . . . . .                                  | 16        |
| 1.7      | Internalized Formulas for the Ordinals . . . . .              | 18        |
| 1.7.1    | The subset relation . . . . .                                 | 18        |

|          |   |           |
|----------|---|-----------|
| 1.7.2    | Transitive sets . . . . .   | 18        |
| 1.7.3    | Ordinals . . . . .  | 19        |
| 1.8      | Constant Lset: Levels of the Constructible Universe . . . . .                       | 19        |
| 1.8.1    | Transitivity . . . . .  | 20        |
| 1.8.2    | Monotonicity . . . . .  | 20        |
| 1.8.3    | 0, successor and limit equations for Lset . . . . .                                 | 20        |
| 1.8.4    | Lset applied to Limit ordinals . . . . .  | 21        |
| 1.8.5    | Basic closure properties . . . . .  | 21        |
| 1.9      | Constructible Ordinals: Kunen's VI 1.9 (b) . . . . .                                | 21        |
| 1.9.1    | Unions . . . . .  | 22        |
| 1.9.2    | Finite sets and ordered pairs . . . . .   | 22        |
| 1.9.3    | For L to satisfy the Powerset axiom . . . . .                                       | 24        |
| 1.10     | Eliminating <i>arity</i> from the Definition of <i>Lset</i> . . . . .               | 24        |
| <b>2</b> | <b>Relativization and Absoluteness</b>  | <b>25</b> |
| 2.1      | Relativized versions of standard set-theoretic concepts . . . . .                   | 25        |
| 2.2      | The relativized ZF axioms . . . . .   | 31        |
| 2.3      | A trivial consistency proof for $V_\omega$ . . . . .                                | 32        |
| 2.4      | Lemmas Needed to Reduce Some Set Constructions to Instances of Separation . . . . . | 34        |
| 2.5      | Introducing a Transitive Class Model . . . . .                                      | 35        |
| 2.5.1    | Trivial Absoluteness Proofs: Empty Set, Pairs, etc. . . . .                         | 35        |
| 2.5.2    | Absoluteness for Unions and Intersections . . . . .                                 | 37        |
| 2.5.3    | Absoluteness for Separation and Replacement . . . . .                               | 38        |
| 2.5.4    | The Operator <i>is_Replace</i> . . . . .  | 38        |
| 2.5.5    | Absoluteness for <i>Lambda</i> . . . . .  | 39        |
| 2.5.6    | Relativization of Powerset . . . . .  | 40        |
| 2.5.7    | Absoluteness for the Natural Numbers . . . . .                                      | 40        |
| 2.6      | Absoluteness for Ordinals . . . . .   | 41        |
| 2.7      | Some instances of separation and strong replacement . . . . .                       | 42        |
| 2.7.1    | converse of a relation . . . . .  | 43        |
| 2.7.2    | image, preimage, domain, range . . . . .  | 44        |
| 2.7.3    | Domain, range and field . . . . .   | 44        |
| 2.7.4    | Relations, functions and application . . . . .                                      | 45        |
| 2.7.5    | Composition of relations . . . . .  | 45        |
| 2.7.6    | Some Facts About Separation Axioms . . . . .  | 46        |
| 2.7.7    | Functions and function space . . . . .  | 47        |
| 2.8      | Relativization and Absoluteness for Boolean Operators . . . . .                     | 48        |
| 2.9      | Relativization and Absoluteness for List Operators . . . . .                        | 49        |
| 2.9.1    | <i>quasilist</i> : For Case-Splitting with <i>list_case'</i> . . . . .              | 51        |
| 2.9.2    | <i>list_case'</i> , the Modified Version of <i>list_case</i> . . . . .              | 51        |
| 2.9.3    | The Modified Operators <i>hd'</i> and <i>tl'</i> . . . . .                          | 51        |
| <b>3</b> | <b>Relativized Wellorderings</b>  | <b>52</b> |

|          |   |           |
|----------|---|-----------|
| 3.1      | Wellorderings . . . . .   | 52        |
| 3.1.1    | Trivial absoluteness proofs . . . . .   | 53        |
| 3.1.2    | Well-founded relations . . . . .  | 54        |
| 3.1.3    | Kunen's lemma IV 3.14, page 123 . . . . .                                     | 55        |
| 3.2      | Relativized versions of order-isomorphisms and order types . . . . .          | 55        |
| 3.3      | Main results of Kunen, Chapter 1 section 6 . . . . .                          | 56        |
| <b>4</b> | <b>Relativized Well-Founded Recursion</b>                                     | <b>56</b> |
| 4.1      | General Lemmas . . . . .  | 56        |
| 4.2      | Reworking of the Recursion Theory Within $M$ . . . . .                        | 57        |
| 4.3      | Relativization of the ZF Predicate <i>is_recfun</i> . . . . .                 | 59        |
| <b>5</b> | <b>Absoluteness of Well-Founded Recursion</b>                                 | <b>60</b> |
| 5.1      | Transitive closure without fixedpoints . . . . .                              | 61        |
| 5.2      | $M$ is closed under well-founded recursion . . . . .                          | 64        |
| 5.3      | Absoluteness without assuming transitivity . . . . .                          | 64        |
| <b>6</b> | <b>Absoluteness Properties for Recursive Datatypes</b>                        | <b>65</b> |
| 6.1      | The lfp of a continuous function can be expressed as a union . . . . .        | 65        |
| 6.1.1    | Some Standard Datatype Constructions Preserve Continuity . . . . .            | 66        |
| 6.2      | Absoluteness for "Iterates" . . . . .   | 66        |
| 6.3      | lists without univ . . . . .  | 67        |
| 6.4      | formulas without univ . . . . .   | 67        |
| 6.5      | $M$ Contains the List and Formula Datatypes . . . . .                         | 68        |
| 6.5.1    | Towards Absoluteness of <i>formula_rec</i> . . . . .                          | 70        |
| 6.5.2    | Absoluteness of the List Construction . . . . .                               | 72        |
| 6.5.3    | Absoluteness of Formulas . . . . .  | 72        |
| 6.6      | Absoluteness for $\varepsilon$ -Closure: the <i>eclose</i> Operator . . . . . | 73        |
| 6.7      | Absoluteness for <i>transrec</i> . . . . .                                    | 74        |
| 6.8      | Absoluteness for the List Operator <i>length</i> . . . . .                    | 75        |
| 6.9      | Absoluteness for the List Operator <i>nth</i> . . . . .                       | 75        |
| 6.10     | Relativization and Absoluteness for the <i>formula</i> Constructors . . . . . | 76        |
| 6.11     | Absoluteness for <i>formula_rec</i> . . . . .                                 | 77        |
| 6.11.1   | Absoluteness for the Formula Operator <i>depth</i> . . . . .                  | 77        |
| 6.11.2   | <i>is_formula_case</i> : relativization of <i>formula_case</i> . . . . .      | 78        |
| 6.11.3   | Absoluteness for <i>formula_rec</i> : Final Results . . . . .                 | 78        |
| <b>7</b> | <b>Closed Unbounded Classes and Normal Functions</b>                          | <b>80</b> |
| 7.1      | Closed and Unbounded (c.u.) Classes of Ordinals . . . . .                     | 80        |
| 7.1.1    | Simple facts about c.u. classes . . . . .                                     | 81        |
| 7.1.2    | The intersection of any set-indexed family of c.u. classes is c.u. . . . .    | 81        |
| 7.2      | Normal Functions . . . . .  | 83        |

|           |   |           |
|-----------|---|-----------|
| 7.2.1     | Immediate properties of the definitions . . . . .           | 83        |
| 7.2.2     | The class of fixedpoints is closed and unbounded . .        | 84        |
| 7.2.3     | Function <i>normalize</i> . . . . .                         | 85        |
| 7.3       | The Alephs . . . . .  | 86        |
| <b>8</b>  | <b>The Reflection Theorem</b>                               | <b>86</b> |
| 8.1       | Basic Definitions . . . . .                                 | 86        |
| 8.2       | Easy Cases of the Reflection Theorem . . . . .              | 87        |
| 8.3       | Reflection for Existential Quantifiers . . . . .            | 88        |
| 8.4       | Packaging the Quantifier Reflection Rules . . . . .         | 89        |
| 8.5       | Simple Examples of Reflection . . . . .                     | 90        |
| <b>9</b>  | <b>The meta-existential quantifier</b>                      | <b>92</b> |
| <b>10</b> | <b>The ZF Axioms (Except Separation) in L</b>               | <b>92</b> |
| 10.1      | For L to satisfy Replacement . . . . .                      | 93        |
| 10.2      | Instantiating the locale <i>M_trivial</i> . . . . .         | 93        |
| 10.3      | Instantiation of the locale <i>reflection</i> . . . . .     | 94        |
| 10.4      | Internalized Formulas for some Set-Theoretic Concepts . . . | 96        |
| 10.4.1    | Some numbers to help write de Bruijn indices . . .          | 96        |
| 10.4.2    | The Empty Set, Internalized . . . . .                       | 96        |
| 10.4.3    | Unordered Pairs, Internalized . . . . .                     | 97        |
| 10.4.4    | Ordered pairs, Internalized . . . . .                       | 98        |
| 10.4.5    | Binary Unions, Internalized . . . . .                       | 98        |
| 10.4.6    | Set “Cons,” Internalized . . . . .                          | 99        |
| 10.4.7    | Successor Function, Internalized . . . . .                  | 100       |
| 10.4.8    | The Number 1, Internalized . . . . .                        | 100       |
| 10.4.9    | Big Union, Internalized . . . . .                           | 101       |
| 10.4.10   | Variants of Satisfaction Definitions for Ordinals, etc.     | 101       |
| 10.4.11   | Membership Relation, Internalized . . . . .                 | 102       |
| 10.4.12   | Predecessor Set, Internalized . . . . .                     | 103       |
| 10.4.13   | Domain of a Relation, Internalized . . . . .                | 103       |
| 10.4.14   | Range of a Relation, Internalized . . . . .                 | 104       |
| 10.4.15   | Field of a Relation, Internalized . . . . .                 | 105       |
| 10.4.16   | Image under a Relation, Internalized . . . . .              | 105       |
| 10.4.17   | Pre-Image under a Relation, Internalized . . . . .          | 106       |
| 10.4.18   | Function Application, Internalized . . . . .                | 106       |
| 10.4.19   | The Concept of Relation, Internalized . . . . .             | 107       |
| 10.4.20   | The Concept of Function, Internalized . . . . .             | 108       |
| 10.4.21   | Typed Functions, Internalized . . . . .                     | 108       |
| 10.4.22   | Composition of Relations, Internalized . . . . .            | 109       |
| 10.4.23   | Injections, Internalized . . . . .                          | 110       |
| 10.4.24   | Surjections, Internalized . . . . .                         | 111       |
| 10.4.25   | Bijections, Internalized . . . . .                          | 111       |

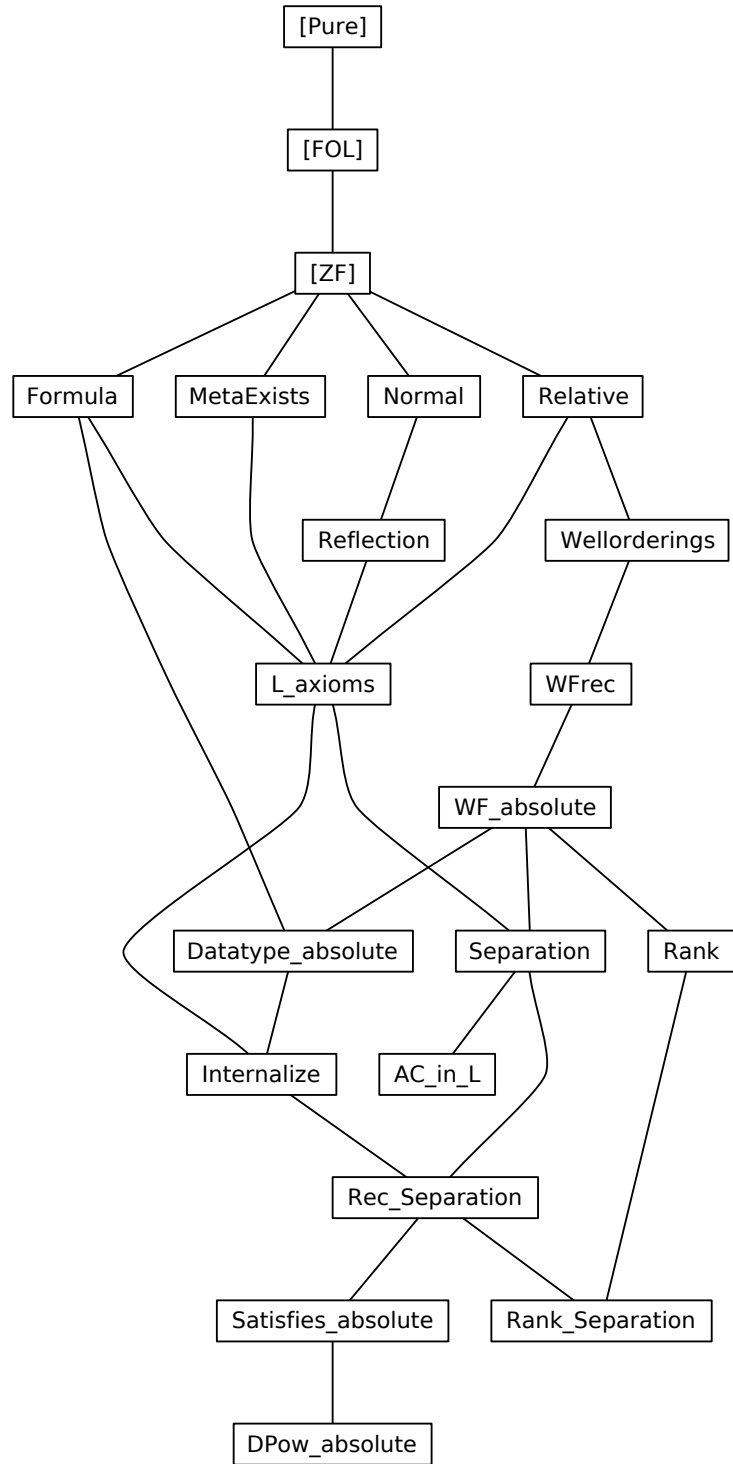
|           |   |            |
|-----------|---|------------|
| 10.4.26   | Restriction of a Relation, Internalized . . . . .                   | 112        |
| 10.4.27   | Order-Isomorphisms, Internalized . . . . .                          | 113        |
| 10.4.28   | Limit Ordinals, Internalized . . . . .                              | 113        |
| 10.4.29   | Finite Ordinals: The Predicate “Is A Natural Num-<br>ber” . . . . . | 114        |
| 10.4.30   | Omega: The Set of Natural Numbers . . . . .                         | 115        |
| <b>11</b> | <b>Early Instances of Separation and Strong Replacement</b>         | <b>116</b> |
| 11.1      | Separation for Intersection . . . . .                               | 117        |
| 11.2      | Separation for Set Difference . . . . .                             | 117        |
| 11.3      | Separation for Cartesian Product . . . . .                          | 117        |
| 11.4      | Separation for Image . . . . .                                      | 117        |
| 11.5      | Separation for Converse . . . . .                                   | 118        |
| 11.6      | Separation for Restriction . . . . .                                | 118        |
| 11.7      | Separation for Composition . . . . .                                | 118        |
| 11.8      | Separation for Predecessors in an Order . . . . .                   | 118        |
| 11.9      | Separation for the Membership Relation . . . . .                    | 119        |
| 11.10     | Replacement for FunSpace . . . . .                                  | 119        |
| 11.11     | Separation for a Theorem about <i>is_recfun</i> . . . . .           | 119        |
| 11.12     | Instantiating the locale <i>M_basic</i> . . . . .                   | 120        |
| 11.13     | Internalized Forms of Data Structuring Operators . . . . .          | 120        |
| 11.13.1   | The Formula <i>is_Inl</i> , Internalized . . . . .                  | 120        |
| 11.13.2   | The Formula <i>is_Inr</i> , Internalized . . . . .                  | 121        |
| 11.13.3   | The Formula <i>is_Nil</i> , Internalized . . . . .                  | 121        |
| 11.13.4   | The Formula <i>is_Cons</i> , Internalized . . . . .                 | 122        |
| 11.13.5   | The Formula <i>is_quasilist</i> , Internalized . . . . .            | 122        |
| 11.14     | Absoluteness for the Function <i>nth</i> . . . . .                  | 123        |
| 11.14.1   | The Formula <i>is_hd</i> , Internalized . . . . .                   | 123        |
| 11.14.2   | The Formula <i>is_tl</i> , Internalized . . . . .                   | 123        |
| 11.14.3   | The Operator <i>is_bool_of_o</i> . . . . .                          | 124        |
| 11.15     | More Internalizations . . . . .                                     | 125        |
| 11.15.1   | The Operator <i>is_lambda</i> . . . . .                             | 125        |
| 11.15.2   | The Operator <i>is_Member</i> , Internalized . . . . .              | 126        |
| 11.15.3   | The Operator <i>is_Equal</i> , Internalized . . . . .               | 126        |
| 11.15.4   | The Operator <i>is_Nand</i> , Internalized . . . . .                | 127        |
| 11.15.5   | The Operator <i>is_Forall</i> , Internalized . . . . .              | 127        |
| 11.15.6   | The Operator <i>is_and</i> , Internalized . . . . .                 | 128        |
| 11.15.7   | The Operator <i>is_or</i> , Internalized . . . . .                  | 128        |
| 11.15.8   | The Operator <i>is_not</i> , Internalized . . . . .                 | 129        |
| 11.16     | Well-Founded Recursion! . . . . .                                   | 130        |
| 11.16.1   | The Operator <i>M_is_recfun</i> . . . . .                           | 130        |
| 11.16.2   | The Operator <i>is_wfrec</i> . . . . .                              | 131        |
| 11.17     | For Datatypes . . . . .   | 132        |
| 11.17.1   | Binary Products, Internalized . . . . .                             | 132        |

|           |   |            |
|-----------|---|------------|
| 11.17.2   | Binary Sums, Internalized . . . . .                                 | 133        |
| 11.17.3   | The Operator <i>quasinat</i> . . . . .                              | 134        |
| 11.17.4   | The Operator <i>is_nat_case</i> . . . . .                           | 134        |
| 11.18     | The Operator <i>iterates_MH</i> , Needed for Iteration . . . . .    | 135        |
| 11.18.1   | The Operator <i>is_iterates</i> . . . . .                           | 136        |
| 11.18.2   | The Formula <i>is_eclose_n</i> , Internalized . . . . .             | 138        |
| 11.18.3   | Membership in <i>eclose(A)</i> . . . . .                            | 138        |
| 11.18.4   | The Predicate “Is <i>eclose(A)</i> ” . . . . .                      | 139        |
| 11.18.5   | The List Functor, Internalized . . . . .                            | 139        |
| 11.18.6   | The Formula <i>is_list_N</i> , Internalized . . . . .               | 140        |
| 11.18.7   | The Predicate “Is A List” . . . . .                                 | 141        |
| 11.18.8   | The Predicate “Is <i>list(A)</i> ” . . . . .                        | 141        |
| 11.18.9   | The Formula Functor, Internalized . . . . .                         | 142        |
| 11.18.10  | The Formula <i>is_formula_N</i> , Internalized . . . . .            | 142        |
| 11.18.11  | The Predicate “Is A Formula” . . . . .                              | 143        |
| 11.18.12  | The Predicate “Is <i>formula</i> ” . . . . .                        | 144        |
| 11.18.13  | The Operator <i>is_transrec</i> . . . . .                           | 144        |
| <b>12</b> | <b>Separation for Facts About Recursion</b>                         | <b>145</b> |
| 12.1      | The Locale <i>M_trancl</i> . . . . .                                | 145        |
| 12.1.1    | Separation for Reflexive/Transitive Closure . . . . .               | 145        |
| 12.1.2    | Reflexive/Transitive Closure, Internalized . . . . .                | 146        |
| 12.1.3    | Transitive Closure of a Relation, Internalized . . . . .            | 147        |
| 12.1.4    | Separation for the Proof of <i>wellfounded_on_trancl</i> . . . . .  | 148        |
| 12.1.5    | Instantiating the locale <i>M_trancl</i> . . . . .                  | 148        |
| 12.2      | <i>L</i> is Closed Under the Operator <i>list</i> . . . . .         | 148        |
| 12.2.1    | Instances of Replacement for Lists . . . . .                        | 148        |
| 12.3      | <i>L</i> is Closed Under the Operator <i>formula</i> . . . . .      | 149        |
| 12.3.1    | Instances of Replacement for Formulas . . . . .                     | 149        |
| 12.3.2    | The Formula <i>is_nth</i> , Internalized . . . . .                  | 150        |
| 12.3.3    | An Instance of Replacement for <i>nth</i> . . . . .                 | 150        |
| 12.3.4    | Instantiating the locale <i>M_datatypes</i> . . . . .               | 151        |
| 12.4      | <i>L</i> is Closed Under the Operator <i>eclose</i> . . . . .       | 151        |
| 12.4.1    | Instances of Replacement for <i>eclose</i> . . . . .                | 151        |
| 12.4.2    | Instantiating the locale <i>M_eclose</i> . . . . .                  | 152        |
| <b>13</b> | <b>Absoluteness for the Satisfies Relation on Formulas</b>          | <b>152</b> |
| 13.1      | More Internalization . . . . .                                      | 152        |
| 13.1.1    | The Formula <i>is_depth</i> , Internalized . . . . .                | 152        |
| 13.1.2    | The Operator <i>is_formula_case</i> . . . . .                       | 153        |
| 13.2      | Absoluteness for the Function <i>satisfies</i> . . . . .            | 155        |
| 13.3      | Internalizations Needed to Instantiate <i>M_satisfies</i> . . . . . | 161        |
| 13.3.1    | The Operator <i>is_depth_apply</i> , Internalized . . . . .         | 161        |
| 13.3.2    | The Operator <i>satisfies_is_a</i> , Internalized . . . . .         | 162        |

|           |  |            |
|-----------|--|------------|
| 13.3.3    | The Operator <i>satisfies_is_b</i> , Internalized . . . . .      | 162        |
| 13.3.4    | The Operator <i>satisfies_is_c</i> , Internalized . . . . .      | 163        |
| 13.3.5    | The Operator <i>satisfies_is_d</i> , Internalized . . . . .      | 164        |
| 13.3.6    | The Operator <i>satisfies_MH</i> , Internalized . . . . .        | 165        |
| 13.4      | Lemmas for Instantiating the Locale <i>M_satisfies</i> . . . . . | 166        |
| 13.4.1    | The <i>Member</i> Case . . . . .                                 | 166        |
| 13.4.2    | The <i>Equal</i> Case . . . . .                                  | 166        |
| 13.4.3    | The <i>Nand</i> Case . . . . .                                   | 167        |
| 13.4.4    | The <i>Forall</i> Case . . . . .                                 | 167        |
| 13.4.5    | The <i>transrec_replacement</i> Case . . . . .                   | 168        |
| 13.4.6    | The Lambda Replacement Case . . . . .                            | 168        |
| 13.5      | Instantiating <i>M_satisfies</i> . . . . .                       | 169        |
| <b>14</b> | <b>Absoluteness for the Definable Powerset Function</b>          | <b>169</b> |
| 14.1      | Preliminary Internalizations . . . . .                           | 169        |
| 14.1.1    | The Operator <i>is_formula_rec</i> . . . . .                     | 169        |
| 14.1.2    | The Operator <i>is_satisfies</i> . . . . .                       | 170        |
| 14.2      | Relativization of the Operator <i>DPow'</i> . . . . .            | 171        |
| 14.2.1    | The Operator <i>is_DPow_sats</i> , Internalized . . . . .        | 171        |
| 14.3      | A Locale for Relativizing the Operator <i>DPow'</i> . . . . .    | 172        |
| 14.4      | Instantiating the Locale <i>M_DPow</i> . . . . .                 | 173        |
| 14.4.1    | The Instance of Separation . . . . .                             | 173        |
| 14.4.2    | The Instance of Replacement . . . . .                            | 173        |
| 14.4.3    | Actually Instantiating the Locale . . . . .                      | 174        |
| 14.4.4    | The Operator <i>is_Collect</i> . . . . .                         | 174        |
| 14.4.5    | The Operator <i>is_Replace</i> . . . . .                         | 175        |
| 14.4.6    | The Operator <i>is_DPow'</i> , Internalized . . . . .            | 176        |
| 14.5      | A Locale for Relativizing the Operator <i>Lset</i> . . . . .     | 177        |
| 14.6      | Instantiating the Locale <i>M_Lset</i> . . . . .                 | 178        |
| 14.6.1    | The First Instance of Replacement . . . . .                      | 178        |
| 14.6.2    | The Second Instance of Replacement . . . . .                     | 178        |
| 14.6.3    | Actually Instantiating <i>M_Lset</i> . . . . .                   | 179        |
| 14.7      | The Notion of Constructible Set . . . . .                        | 179        |
| <b>15</b> | <b>The Axiom of Choice Holds in L!</b>                           | <b>179</b> |
| 15.1      | Extending a Wellordering over a List – Lexicographic Power       | 179        |
| 15.1.1    | Type checking . . . . .  | 180        |
| 15.1.2    | Linearity . . . . .  | 180        |
| 15.1.3    | Well-foundedness . . . . .                                       | 180        |
| 15.2      | An Injection from Formulas into the Natural Numbers . . .        | 181        |
| 15.3      | Defining the Wellordering on <i>DPow(A)</i> . . . . .            | 182        |
| 15.4      | Limit Construction for Well-Orderings . . . . .                  | 184        |
| 15.5      | Transfinite Definition of the Wellordering on <i>L</i> . . . . . | 185        |
| 15.5.1    | The Corresponding Recursion Equations . . . . .                  | 185        |

|   |                |
|---|----------------|
| <b>16 Absoluteness for Order Types, Rank Functions and Well-Founded Relations</b>               | <b>186</b>     |
| 16.1 Order Types: A Direct Construction by Replacement . . . .                                  | 186            |
| 16.2 Kunen's theorem 5.4, page 127 . . . . .  | 190            |
| 16.3 Ordinal Arithmetic: Two Examples of Recursion . . . . .                                    | 190            |
| 16.3.1 Ordinal Addition . . . . .   | 190            |
| 16.3.2 Ordinal Multiplication . . . . .   | 193            |
| 16.4 Absoluteness of Well-Founded Relations . . . . .   | 194            |
| <br><b>17 Separation for Facts About Order Types, Rank Functions and Well-Founded Relations</b> | <br><b>197</b> |
| 17.1 The Locale $M_{ordertype}$ . . . . .   | 197            |
| 17.1.1 Separation for Order-Isomorphisms . . . . .  | 197            |
| 17.1.2 Separation for $obase$ . . . . .   | 197            |
| 17.1.3 Separation for a Theorem about $obase$ . . . . .   | 198            |
| 17.1.4 Replacement for $omap$ . . . . .   | 198            |
| 17.2 Instantiating the locale $M_{ordertype}$ . . . . .   | 199            |
| 17.3 The Locale $M_{wfrank}$ . . . . .  | 199            |
| 17.3.1 Separation for $wfrank$ . . . . .  | 199            |
| 17.3.2 Replacement for $wfrank$ . . . . .   | 199            |
| 17.3.3 Separation for Proving $Ord\_wfrank\_range$ . . . . .                                    | 200            |
| 17.3.4 Instantiating the locale $M_{wfrank}$ . . . . .  | 200            |





# 1 First-Order Formulas and the Definition of the Class L

**theory** *Formula* imports ZF begin

## 1.1 Internalized formulas of FOL

De Bruijn representation. Unbound variables get their denotations from an environment.

```
consts   formula :: i
datatype
  "formula" = Member ("x ∈ nat", "y ∈ nat")
              | Equal  ("x ∈ nat", "y ∈ nat")
              | Nand   ("p ∈ formula", "q ∈ formula")
              | Forall ("p ∈ formula")
```

**declare** *formula.intros* [TC]

**definition**

```
Neg :: "i ⇒ i" where
  "Neg(p) ≡ Nand(p,p)"
```

**definition**

```
And :: "[i,i] ⇒ i" where
  "And(p,q) ≡ Neg(Nand(p,q))"
```

**definition**

```
Or :: "[i,i] ⇒ i" where
  "Or(p,q) ≡ Nand(Neg(p),Neg(q))"
```

**definition**

```
Implies :: "[i,i] ⇒ i" where
  "Implies(p,q) ≡ Nand(p,Neg(q))"
```

**definition**

```
Iff :: "[i,i] ⇒ i" where
  "Iff(p,q) ≡ And(Implies(p,q), Implies(q,p))"
```

**definition**

```
Exists :: "i ⇒ i" where
  "Exists(p) ≡ Neg(Forall(Neg(p)))"
```

**lemma** *Neg\_type* [TC]: " $p \in \text{formula} \implies \text{Neg}(p) \in \text{formula}$ "  
⟨*proof*⟩

**lemma** *And\_type* [TC]: " $\llbracket p \in \text{formula}; q \in \text{formula} \rrbracket \implies \text{And}(p,q) \in \text{formula}$ "  
⟨*proof*⟩

**lemma** *Or\_type* [TC]: " $\llbracket p \in \text{formula}; q \in \text{formula} \rrbracket \implies \text{Or}(p,q) \in \text{formula}$ "  
 <proof>

**lemma** *Implies\_type* [TC]:  
 " $\llbracket p \in \text{formula}; q \in \text{formula} \rrbracket \implies \text{Implies}(p,q) \in \text{formula}$ "  
 <proof>

**lemma** *Iff\_type* [TC]:  
 " $\llbracket p \in \text{formula}; q \in \text{formula} \rrbracket \implies \text{Iff}(p,q) \in \text{formula}$ "  
 <proof>

**lemma** *Exists\_type* [TC]: " $p \in \text{formula} \implies \text{Exists}(p) \in \text{formula}$ "  
 <proof>

**consts** *satisfies* :: "[i,i] $\Rightarrow$ i"

**primrec**

"*satisfies*(A,Member(x,y)) =  
 ( $\lambda \text{env} \in \text{list}(A). \text{bool\_of\_o } (\text{nth}(x,\text{env}) \in \text{nth}(y,\text{env}))$ )"

"*satisfies*(A,Equal(x,y)) =  
 ( $\lambda \text{env} \in \text{list}(A). \text{bool\_of\_o } (\text{nth}(x,\text{env}) = \text{nth}(y,\text{env}))$ )"

"*satisfies*(A,Nand(p,q)) =  
 ( $\lambda \text{env} \in \text{list}(A). \text{not } ((\text{satisfies}(A,p)'\text{env}) \text{ and } (\text{satisfies}(A,q)'\text{env}))$ )"

"*satisfies*(A,Forall(p)) =  
 ( $\lambda \text{env} \in \text{list}(A). \text{bool\_of\_o } (\forall x \in A. \text{satisfies}(A,p) ' (\text{Cons}(x,\text{env})))$ )  
 = 1)"

**lemma** *satisfies\_type*: " $p \in \text{formula} \implies \text{satisfies}(A,p) \in \text{list}(A) \rightarrow \text{bool}$ "  
 <proof>

**abbreviation**

*sats* :: "[i,i,i]  $\Rightarrow$  o" **where**  
 "*sats*(A,p,env)  $\equiv$  *satisfies*(A,p)'env = 1"

**lemma** *sats\_Member\_iff* [simp]:  
 " $\text{env} \in \text{list}(A) \implies \text{sats}(A, \text{Member}(x,y), \text{env}) \longleftrightarrow \text{nth}(x,\text{env}) \in \text{nth}(y,\text{env})$ "  
 <proof>

**lemma** *sats\_Equal\_iff* [simp]:  
 " $\text{env} \in \text{list}(A) \implies \text{sats}(A, \text{Equal}(x,y), \text{env}) \longleftrightarrow \text{nth}(x,\text{env}) = \text{nth}(y,\text{env})$ "  
 <proof>

**lemma** *sats\_Nand\_iff* [simp]:  
 " $\text{env} \in \text{list}(A)$   
 $\implies (\text{sats}(A, \text{Nand}(p,q), \text{env})) \longleftrightarrow \neg (\text{sats}(A,p,\text{env}) \wedge \text{sats}(A,q,\text{env}))$ "

$\langle proof \rangle$

```
lemma sats_Forall_iff [simp]:  
  "env ∈ list(A)  
  ⇒ sats(A, Forall(p), env) ⇔ (∀ x∈A. sats(A, p, Cons(x,env)))"  
 $\langle proof \rangle$ 
```

declare satisfies.simps [simp del]

## 1.2 Dividing line between primitive and derived connectives

```
lemma sats_Neg_iff [simp]:  
  "env ∈ list(A)  
  ⇒ sats(A, Neg(p), env) ⇔ ¬ sats(A,p,env)"  
 $\langle proof \rangle$ 
```

```
lemma sats_And_iff [simp]:  
  "env ∈ list(A)  
  ⇒ (sats(A, And(p,q), env)) ⇔ sats(A,p,env) ∧ sats(A,q,env)"  
 $\langle proof \rangle$ 
```

```
lemma sats_Or_iff [simp]:  
  "env ∈ list(A)  
  ⇒ (sats(A, Or(p,q), env)) ⇔ sats(A,p,env) ∨ sats(A,q,env)"  
 $\langle proof \rangle$ 
```

```
lemma sats_Implies_iff [simp]:  
  "env ∈ list(A)  
  ⇒ (sats(A, Implies(p,q), env)) ⇔ (sats(A,p,env) ⇒ sats(A,q,env))"  
 $\langle proof \rangle$ 
```

```
lemma sats_Iff_iff [simp]:  
  "env ∈ list(A)  
  ⇒ (sats(A, Iff(p,q), env)) ⇔ (sats(A,p,env) ⇔ sats(A,q,env))"  
 $\langle proof \rangle$ 
```

```
lemma sats_Exists_iff [simp]:  
  "env ∈ list(A)  
  ⇒ sats(A, Exists(p), env) ⇔ (∃ x∈A. sats(A, p, Cons(x,env)))"  
 $\langle proof \rangle$ 
```

### 1.2.1 Derived rules to help build up formulas

```
lemma mem_iff_sats:  
  "⟦nth(i,env) = x; nth(j,env) = y; env ∈ list(A)⟧  
  ⇒ (x∈y) ⇔ sats(A, Member(i,j), env)"  
 $\langle proof \rangle$ 
```

```
lemma equal_iff_sats:  
  "⟦nth(i,env) = x; nth(j,env) = y; env ∈ list(A)⟧
```

$\langle proof \rangle \implies (x=y) \longleftrightarrow sats(A, Equal(i,j), env)"$

**lemma not\_iff\_sats:**  
 $"\llbracket P \longleftrightarrow sats(A,p,env); env \in list(A) \rrbracket$   
 $\implies (\neg P) \longleftrightarrow sats(A, Neg(p), env)"$   
 $\langle proof \rangle$

**lemma conj\_iff\_sats:**  
 $"\llbracket P \longleftrightarrow sats(A,p,env); Q \longleftrightarrow sats(A,q,env); env \in list(A) \rrbracket$   
 $\implies (P \wedge Q) \longleftrightarrow sats(A, And(p,q), env)"$   
 $\langle proof \rangle$

**lemma disj\_iff\_sats:**  
 $"\llbracket P \longleftrightarrow sats(A,p,env); Q \longleftrightarrow sats(A,q,env); env \in list(A) \rrbracket$   
 $\implies (P \vee Q) \longleftrightarrow sats(A, Or(p,q), env)"$   
 $\langle proof \rangle$

**lemma iff\_iff\_sats:**  
 $"\llbracket P \longleftrightarrow sats(A,p,env); Q \longleftrightarrow sats(A,q,env); env \in list(A) \rrbracket$   
 $\implies (P \longleftrightarrow Q) \longleftrightarrow sats(A, Iff(p,q), env)"$   
 $\langle proof \rangle$

**lemma imp\_iff\_sats:**  
 $"\llbracket P \longleftrightarrow sats(A,p,env); Q \longleftrightarrow sats(A,q,env); env \in list(A) \rrbracket$   
 $\implies (P \longrightarrow Q) \longleftrightarrow sats(A, Implies(p,q), env)"$   
 $\langle proof \rangle$

**lemma ball\_iff\_sats:**  
 $"\llbracket \bigwedge x. x \in A \implies P(x) \longleftrightarrow sats(A, p, Cons(x, env)); env \in list(A) \rrbracket$   
 $\implies (\forall x \in A. P(x)) \longleftrightarrow sats(A, Forall(p), env)"$   
 $\langle proof \rangle$

**lemma bex\_iff\_sats:**  
 $"\llbracket \bigwedge x. x \in A \implies P(x) \longleftrightarrow sats(A, p, Cons(x, env)); env \in list(A) \rrbracket$   
 $\implies (\exists x \in A. P(x)) \longleftrightarrow sats(A, Exists(p), env)"$   
 $\langle proof \rangle$

**lemmas FOL\_iff\_sats =**  
 $mem\_iff\_sats \ equal\_iff\_sats \ not\_iff\_sats \ conj\_iff\_sats$   
 $disj\_iff\_sats \ imp\_iff\_sats \ iff\_iff\_sats \ imp\_iff\_sats \ ball\_iff\_sats$   
 $bex\_iff\_sats$

### 1.3 Arity of a Formula: Maximum Free de Bruijn Index

**consts**     $arity :: "i \Rightarrow i"$   
**primrec**  
 $"arity(Member(x,y)) = succ(x) \cup succ(y)"$

```

"arity(Equal(x,y)) = succ(x) ∪ succ(y)"

"arity(Nand(p,q)) = arity(p) ∪ arity(q)"

"arity(Forall(p)) = Arith.pred(arity(p))"

lemma arity_type [TC]: "p ∈ formula ⇒ arity(p) ∈ nat"
⟨proof⟩

lemma arity_Neg [simp]: "arity(Neg(p)) = arity(p)"
⟨proof⟩

lemma arity_And [simp]: "arity(And(p,q)) = arity(p) ∪ arity(q)"
⟨proof⟩

lemma arity_Or [simp]: "arity(Or(p,q)) = arity(p) ∪ arity(q)"
⟨proof⟩

lemma arity_Implies [simp]: "arity(Implies(p,q)) = arity(p) ∪ arity(q)"
⟨proof⟩

lemma arity_Iff [simp]: "arity(Iff(p,q)) = arity(p) ∪ arity(q)"
⟨proof⟩

lemma arity_Exists [simp]: "arity(Exists(p)) = Arith.pred(arity(p))"
⟨proof⟩

lemma arity_sats_iff [rule_format]:
  "⟦p ∈ formula; extra ∈ list(A)⟧
  ⇒ ∀ env ∈ list(A).
    arity(p) ≤ length(env) →
    sats(A, p, env @ extra) ↔ sats(A, p, env)"
⟨proof⟩

lemma arity_sats1_iff:
  "⟦arity(p) ≤ succ(length(env)); p ∈ formula; x ∈ A; env ∈ list(A);
  extra ∈ list(A)⟧
  ⇒ sats(A, p, Cons(x, env @ extra)) ↔ sats(A, p, Cons(x, env))"
⟨proof⟩

```

## 1.4 Renaming Some de Bruijn Variables

**definition**

```

incr_var :: "[i,i]⇒i" where
  "incr_var(x,nq) ≡ if x<nq then x else succ(x)"

```

```

lemma incr_var_lt: "x<nq ⇒ incr_var(x,nq) = x"

```

$\langle proof \rangle$

**lemma** *incr\_var\_le*: " $nq \leq x \implies incr\_var(x, nq) = succ(x)$ "  
 $\langle proof \rangle$

**consts** *incr\_bv* :: " $i \Rightarrow i$ "

**primrec**

"*incr\_bv*(Member( $x, y$ )) =  
 $(\lambda nq \in nat. Member (incr\_var(x, nq), incr\_var(y, nq)))$ "

"*incr\_bv*(Equal( $x, y$ )) =  
 $(\lambda nq \in nat. Equal (incr\_var(x, nq), incr\_var(y, nq)))$ "

"*incr\_bv*(Nand( $p, q$ )) =  
 $(\lambda nq \in nat. Nand (incr\_bv(p) 'nq, incr\_bv(q) 'nq))$ "

"*incr\_bv*(Forall( $p$ )) =  
 $(\lambda nq \in nat. Forall (incr\_bv(p) ' succ(nq)))$ "

**lemma** [TC]: " $x \in nat \implies incr\_var(x, nq) \in nat$ "  
 $\langle proof \rangle$

**lemma** *incr\_bv\_type* [TC]: " $p \in formula \implies incr\_bv(p) \in nat \rightarrow formula$ "  
 $\langle proof \rangle$

Obviously, *DPow* is closed under complements and finite intersections and unions. Needs an inductive lemma to allow two lists of parameters to be combined.

**lemma** *sats\_incr\_bv\_iff* [rule\_format]:  
 $\llbracket p \in formula; env \in list(A); x \in A \rrbracket$   
 $\implies \forall bvs \in list(A).$   
 $sats(A, incr\_bv(p) ' length(bvs), bvs @ Cons(x, env)) \longleftrightarrow$   
 $sats(A, p, bvs @ env)$

$\langle proof \rangle$

**lemma** *incr\_var\_lemma*:  
 $\llbracket x \in nat; y \in nat; nq \leq x \rrbracket$   
 $\implies succ(x) \cup incr\_var(y, nq) = succ(x \cup y)$   
 $\langle proof \rangle$

**lemma** *incr\_And\_lemma*:  
 $y < x \implies y \cup succ(x) = succ(x \cup y)$   
 $\langle proof \rangle$

**lemma** *arity\_incr\_bv\_lemma* [rule\_format]:  
 $p \in formula$

$\implies \forall n \in \text{nat}. \text{arity}(\text{incr\_bv}(p) \text{ ' } n) =$   
 $(\text{if } n < \text{arity}(p) \text{ then succ}(\text{arity}(p)) \text{ else } \text{arity}(p))"$   
 $\langle \text{proof} \rangle$

## 1.5 Renaming all but the First de Bruijn Variable

**definition**

$\text{incr\_bv1} :: "i \Rightarrow i" \text{ where}$   
 $"\text{incr\_bv1}(p) \equiv \text{incr\_bv}(p) \text{ ' } 1"$

**lemma**  $\text{incr\_bv1\_type}$  [TC]:  $"p \in \text{formula} \implies \text{incr\_bv1}(p) \in \text{formula}"$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{sats\_incr\_bv1\_iff}$ :

$"[p \in \text{formula}; \text{env} \in \text{list}(A); x \in A; y \in A]$   
 $\implies \text{sats}(A, \text{incr\_bv1}(p), \text{Cons}(x, \text{Cons}(y, \text{env}))) \longleftrightarrow$   
 $\text{sats}(A, p, \text{Cons}(x, \text{env}))"$

$\langle \text{proof} \rangle$

**lemma**  $\text{formula\_add\_params1}$  [rule\_format]:

$"[p \in \text{formula}; n \in \text{nat}; x \in A]$   
 $\implies \forall \text{bvs} \in \text{list}(A). \forall \text{env} \in \text{list}(A).$   
 $\text{length}(\text{bvs}) = n \longrightarrow$   
 $\text{sats}(A, \text{iterates}(\text{incr\_bv1}, n, p), \text{Cons}(x, \text{bvs@env})) \longleftrightarrow$   
 $\text{sats}(A, p, \text{Cons}(x, \text{env}))"$

$\langle \text{proof} \rangle$

**lemma**  $\text{arity\_incr\_bv1\_eq}$ :

$"p \in \text{formula}$   
 $\implies \text{arity}(\text{incr\_bv1}(p)) =$   
 $(\text{if } 1 < \text{arity}(p) \text{ then succ}(\text{arity}(p)) \text{ else } \text{arity}(p))"$

$\langle \text{proof} \rangle$

**lemma**  $\text{arity\_iterates\_incr\_bv1\_eq}$ :

$"[p \in \text{formula}; n \in \text{nat}]$   
 $\implies \text{arity}(\text{incr\_bv1}^n(p)) =$   
 $(\text{if } 1 < \text{arity}(p) \text{ then } n \# + \text{arity}(p) \text{ else } \text{arity}(p))"$

$\langle \text{proof} \rangle$

## 1.6 Definable Powerset

The definable powerset operation: Kunen's definition VI 1.1, page 165.

**definition**

$\text{DPow} :: "i \Rightarrow i" \text{ where}$   
 $"\text{DPow}(A) \equiv \{X \in \text{Pow}(A).$   
 $\quad \exists \text{env} \in \text{list}(A). \exists p \in \text{formula}.$



$$\text{arity}(p) \leq \text{succ}(\text{length}(\text{env})) \wedge \\ X = \{x \in A. \text{sats}(A, p, \text{Cons}(x, \text{env}))\}$$

**lemma DPowI:**

" $\llbracket \text{env} \in \text{list}(A); p \in \text{formula}; \text{arity}(p) \leq \text{succ}(\text{length}(\text{env})) \rrbracket$   
 $\implies \{x \in A. \text{sats}(A, p, \text{Cons}(x, \text{env}))\} \in \text{DPow}(A)$ "  
 <proof>

With this rule we can specify  $p$  later.

**lemma DPowI2 [rule\_format]:**

" $\llbracket \forall x \in A. P(x) \longleftrightarrow \text{sats}(A, p, \text{Cons}(x, \text{env}));$   
 $\text{env} \in \text{list}(A); p \in \text{formula}; \text{arity}(p) \leq \text{succ}(\text{length}(\text{env})) \rrbracket$   
 $\implies \{x \in A. P(x)\} \in \text{DPow}(A)$ "  
 <proof>

**lemma DPowD:**

" $X \in \text{DPow}(A)$   
 $\implies X \subseteq A \wedge$   
 $(\exists \text{env} \in \text{list}(A). \exists p \in \text{formula}. \text{arity}(p) \leq \text{succ}(\text{length}(\text{env})) \wedge$   
 $X = \{x \in A. \text{sats}(A, p, \text{Cons}(x, \text{env}))\})$ "  
 <proof>

**lemmas DPow\_imp\_subset = DPowD [THEN conjunct1]**

**lemma** " $\llbracket p \in \text{formula}; \text{env} \in \text{list}(A); \text{arity}(p) \leq \text{succ}(\text{length}(\text{env})) \rrbracket$   
 $\implies \{x \in A. \text{sats}(A, p, \text{Cons}(x, \text{env}))\} \in \text{DPow}(A)$ "  
 <proof>

**lemma DPow\_subset\_Pow:** " $\text{DPow}(A) \subseteq \text{Pow}(A)$ "  
 <proof>

**lemma empty\_in\_DPow:** " $0 \in \text{DPow}(A)$ "  
 <proof>

**lemma Compl\_in\_DPow:** " $X \in \text{DPow}(A) \implies (A - X) \in \text{DPow}(A)$ "  
 <proof>

**lemma Int\_in\_DPow:** " $\llbracket X \in \text{DPow}(A); Y \in \text{DPow}(A) \rrbracket \implies X \cap Y \in \text{DPow}(A)$ "  
 <proof>

**lemma Un\_in\_DPow:** " $\llbracket X \in \text{DPow}(A); Y \in \text{DPow}(A) \rrbracket \implies X \cup Y \in \text{DPow}(A)$ "  
 <proof>

**lemma singleton\_in\_DPow:** " $a \in A \implies \{a\} \in \text{DPow}(A)$ "  
 <proof>

**lemma cons\_in\_DPow:** " $\llbracket a \in A; X \in \text{DPow}(A) \rrbracket \implies \text{cons}(a, X) \in \text{DPow}(A)$ "

*<proof>*

**lemma** *Fin\_into\_DPow*: " $X \in \text{Fin}(A) \implies X \in \text{DPow}(A)$ "

*<proof>*

*DPow* is not monotonic. For example, let  $A$  be some non-constructible set of natural numbers, and let  $B$  be  $\text{nat}$ . Then  $A \subseteq B$  and obviously  $A \in \text{DPow}(A)$  but  $A \notin \text{DPow}(B)$ .

**lemma** *Finite\_Pow\_subset\_Pow*: " $\text{Finite}(A) \implies \text{Pow}(A) \subseteq \text{DPow}(A)$ "

*<proof>*

**lemma** *Finite\_DPow\_eq\_Pow*: " $\text{Finite}(A) \implies \text{DPow}(A) = \text{Pow}(A)$ "

*<proof>*

## 1.7 Internalized Formulas for the Ordinals

The *sats* theorems below differ from the usual form in that they include an element of absoluteness. That is, they relate internalized formulas to real concepts such as the subset relation, rather than to the relativized concepts defined in theory *Relative*. This lets us prove the theorem as *Ords\_in\_DPow* without first having to instantiate the locale *M\_trivial*. Note that the present theory does not even take *Relative* as a parent.

### 1.7.1 The subset relation

**definition**

*subset\_fm* :: " $[i,i] \Rightarrow i$ " where  
 $\text{"subset\_fm}(x,y) \equiv \text{Forall}(\text{Implies}(\text{Member}(0,\text{succ}(x)), \text{Member}(0,\text{succ}(y))))$ "

**lemma** *subset\_type* [TC]: " $\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \implies \text{subset\_fm}(x,y) \in \text{formula}$ "

*<proof>*

**lemma** *arity\_subset\_fm* [simp]:

$\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \implies \text{arity}(\text{subset\_fm}(x,y)) = \text{succ}(x) \cup \text{succ}(y)$

*<proof>*

**lemma** *sats\_subset\_fm* [simp]:

$\llbracket x < \text{length}(\text{env}); y \in \text{nat}; \text{env} \in \text{list}(A); \text{Transset}(A) \rrbracket$   
 $\implies \text{sats}(A, \text{subset\_fm}(x,y), \text{env}) \longleftrightarrow \text{nth}(x,\text{env}) \subseteq \text{nth}(y,\text{env})$

*<proof>*

### 1.7.2 Transitive sets

**definition**

*transset\_fm* :: " $i \Rightarrow i$ " where  
 $\text{"transset\_fm}(x) \equiv \text{Forall}(\text{Implies}(\text{Member}(0,\text{succ}(x)), \text{subset\_fm}(0,\text{succ}(x))))$ "

**lemma** *transset\_type* [TC]: " $x \in \text{nat} \implies \text{transset\_fm}(x) \in \text{formula}$ "  
 <proof>

**lemma** *arity\_transset\_fm* [simp]:  
 " $x \in \text{nat} \implies \text{arity}(\text{transset\_fm}(x)) = \text{succ}(x)$ "  
 <proof>

**lemma** *sats\_transset\_fm* [simp]:  
 " $\llbracket x < \text{length}(\text{env}); \text{env} \in \text{list}(A); \text{Transset}(A) \rrbracket$   
 $\implies \text{sats}(A, \text{transset\_fm}(x), \text{env}) \longleftrightarrow \text{Transset}(\text{nth}(x, \text{env}))$ "  
 <proof>

### 1.7.3 Ordinals

**definition**  
*ordinal\_fm* :: " $i \Rightarrow i$ " where  
 "*ordinal\_fm*( $x$ )  $\equiv$   
 And(*transset\_fm*( $x$ ), Forall(Implies(Member(0, *succ*( $x$ )), *transset\_fm*(0))))"

**lemma** *ordinal\_type* [TC]: " $x \in \text{nat} \implies \text{ordinal\_fm}(x) \in \text{formula}$ "  
 <proof>

**lemma** *arity\_ordinal\_fm* [simp]:  
 " $x \in \text{nat} \implies \text{arity}(\text{ordinal\_fm}(x)) = \text{succ}(x)$ "  
 <proof>

**lemma** *sats\_ordinal\_fm*:  
 " $\llbracket x < \text{length}(\text{env}); \text{env} \in \text{list}(A); \text{Transset}(A) \rrbracket$   
 $\implies \text{sats}(A, \text{ordinal\_fm}(x), \text{env}) \longleftrightarrow \text{Ord}(\text{nth}(x, \text{env}))$ "  
 <proof>

The subset consisting of the ordinals is definable. Essential lemma for *Ord\_in\_Lset*. This result is the objective of the present subsection.

**theorem** *Ords\_in\_DPow*: " $\text{Transset}(A) \implies \{x \in A. \text{Ord}(x)\} \in \text{DPow}(A)$ "  
 <proof>

## 1.8 Constant Lset: Levels of the Constructible Universe

**definition**  
*Lset* :: " $i \Rightarrow i$ " where  
 "*Lset*( $i$ )  $\equiv \text{transrec}(i, \lambda x f. \bigcup_{y \in x. \text{DPow}(f'y))$ "

**definition**  
*L* :: " $i \Rightarrow o$ " where — Kunen's definition VI 1.5, page 167  
 "*L*( $x$ )  $\equiv \exists i. \text{Ord}(i) \wedge x \in \text{Lset}(i)$ "

NOT SUITABLE FOR REWRITING – RECURSIVE!

**lemma** *Lset*: "*Lset*( $i$ ) =  $(\bigcup_{j \in i. \text{DPow}(\text{Lset}(j)))$ "  
 <proof>

**lemma** *LsetI*: " $\llbracket y \in x; A \in \text{DPow}(\text{Lset}(y)) \rrbracket \implies A \in \text{Lset}(x)$ "  
 $\langle \text{proof} \rangle$

**lemma** *LsetD*: " $A \in \text{Lset}(x) \implies \exists y \in x. A \in \text{DPow}(\text{Lset}(y))$ "  
 $\langle \text{proof} \rangle$

### 1.8.1 Transitivity

**lemma** *elem\_subset\_in\_DPow*: " $\llbracket X \in A; X \subseteq A \rrbracket \implies X \in \text{DPow}(A)$ "  
 $\langle \text{proof} \rangle$

**lemma** *Transset\_subset\_DPow*: " $\text{Transset}(A) \implies A \subseteq \text{DPow}(A)$ "  
 $\langle \text{proof} \rangle$

**lemma** *Transset\_DPow*: " $\text{Transset}(A) \implies \text{Transset}(\text{DPow}(A))$ "  
 $\langle \text{proof} \rangle$

Kunen's VI 1.6 (a)

**lemma** *Transset\_Lset*: " $\text{Transset}(\text{Lset}(i))$ "  
 $\langle \text{proof} \rangle$

**lemma** *mem\_Lset\_imp\_subset\_Lset*: " $a \in \text{Lset}(i) \implies a \subseteq \text{Lset}(i)$ "  
 $\langle \text{proof} \rangle$

### 1.8.2 Monotonicity

Kunen's VI 1.6 (b)

**lemma** *Lset\_mono* [rule\_format]:  
 $\text{"}\forall j. i \leq j \longrightarrow \text{Lset}(i) \subseteq \text{Lset}(j)\text{"}$   
 $\langle \text{proof} \rangle$

This version lets us remove the premise  $\text{Ord}(i)$  sometimes.

**lemma** *Lset\_mono\_mem* [rule\_format]:  
 $\text{"}\forall j. i \in j \longrightarrow \text{Lset}(i) \subseteq \text{Lset}(j)\text{"}$   
 $\langle \text{proof} \rangle$

Useful with Reflection to bump up the ordinal

**lemma** *subset\_Lset\_ltD*: " $\llbracket A \subseteq \text{Lset}(i); i < j \rrbracket \implies A \subseteq \text{Lset}(j)$ "  
 $\langle \text{proof} \rangle$

### 1.8.3 0, successor and limit equations for Lset

**lemma** *Lset\_0* [simp]: " $\text{Lset}(0) = 0$ "  
 $\langle \text{proof} \rangle$

**lemma** *Lset\_succ\_subset1*: " $\text{DPow}(\text{Lset}(i)) \subseteq \text{Lset}(\text{succ}(i))$ "  
 $\langle \text{proof} \rangle$

**lemma** *Lset\_succ\_subset2*: " $Lset(succ(i)) \subseteq DPow(Lset(i))$ "  
 $\langle proof \rangle$

**lemma** *Lset\_succ*: " $Lset(succ(i)) = DPow(Lset(i))$ "  
 $\langle proof \rangle$

**lemma** *Lset\_Union [simp]*: " $Lset(\bigcup (X)) = (\bigcup_{y \in X}. Lset(y))$ "  
 $\langle proof \rangle$

#### 1.8.4 Lset applied to Limit ordinals

**lemma** *Limit\_Lset\_eq*:  
" $Limit(i) \implies Lset(i) = (\bigcup_{y \in i}. Lset(y))$ "  
 $\langle proof \rangle$

**lemma** *lt\_LsetI*: " $\llbracket a \in Lset(j); j < i \rrbracket \implies a \in Lset(i)$ "  
 $\langle proof \rangle$

**lemma** *Limit\_LsetE*:  
" $\llbracket a \in Lset(i); \neg R \implies Limit(i);$   
 $\bigwedge x. \llbracket x < i; a \in Lset(x) \rrbracket \implies R$   
 $\rrbracket \implies R$ "  
 $\langle proof \rangle$

#### 1.8.5 Basic closure properties

**lemma** *zero\_in\_Lset*: " $y \in x \implies 0 \in Lset(x)$ "  
 $\langle proof \rangle$

**lemma** *notin\_Lset*: " $x \notin Lset(x)$ "  
 $\langle proof \rangle$

### 1.9 Constructible Ordinals: Kunen's VI 1.9 (b)

**lemma** *Ords\_of\_Lset\_eq*: " $Ord(i) \implies \{x \in Lset(i). Ord(x)\} = i$ "  
 $\langle proof \rangle$

**lemma** *Ord\_subset\_Lset*: " $Ord(i) \implies i \subseteq Lset(i)$ "  
 $\langle proof \rangle$

**lemma** *Ord\_in\_Lset*: " $Ord(i) \implies i \in Lset(succ(i))$ "  
 $\langle proof \rangle$

**lemma** *Ord\_in\_L*: " $Ord(i) \implies L(i)$ "  
 $\langle proof \rangle$

### 1.9.1 Unions

**lemma** *Union\_in\_Lset*:

" $X \in \text{Lset}(i) \implies \bigcup (X) \in \text{Lset}(\text{succ}(i))$ "

*<proof>*

**theorem** *Union\_in\_L*: " $L(X) \implies L(\bigcup (X))$ "

*<proof>*

### 1.9.2 Finite sets and ordered pairs

**lemma** *singleton\_in\_Lset*: " $a \in \text{Lset}(i) \implies \{a\} \in \text{Lset}(\text{succ}(i))$ "

*<proof>*

**lemma** *doubleton\_in\_Lset*:

" $\llbracket a \in \text{Lset}(i); b \in \text{Lset}(i) \rrbracket \implies \{a, b\} \in \text{Lset}(\text{succ}(i))$ "

*<proof>*

**lemma** *Pair\_in\_Lset*:

" $\llbracket a \in \text{Lset}(i); b \in \text{Lset}(i); \text{Ord}(i) \rrbracket \implies \langle a, b \rangle \in \text{Lset}(\text{succ}(\text{succ}(i)))$ "

*<proof>*

**lemmas** *Lset\_UnI1* = *Un\_upper1* [THEN *Lset\_mono* [THEN *subsetD*]]

**lemmas** *Lset\_UnI2* = *Un\_upper2* [THEN *Lset\_mono* [THEN *subsetD*]]

Hard work is finding a single  $j \in i$  such that  $\{a, b\} \subseteq \text{Lset}(j)$

**lemma** *doubleton\_in\_LLimit*:

" $\llbracket a \in \text{Lset}(i); b \in \text{Lset}(i); \text{Limit}(i) \rrbracket \implies \{a, b\} \in \text{Lset}(i)$ "

*<proof>*

**theorem** *doubleton\_in\_L*: " $\llbracket L(a); L(b) \rrbracket \implies L(\{a, b\})$ "

*<proof>*

**lemma** *Pair\_in\_LLimit*:

" $\llbracket a \in \text{Lset}(i); b \in \text{Lset}(i); \text{Limit}(i) \rrbracket \implies \langle a, b \rangle \in \text{Lset}(i)$ "

Infer that  $a, b$  occur at ordinals  $x, x_a < i$ .

*<proof>*

The rank function for the constructible universe

**definition**

*lrank* :: " $i \Rightarrow i$ " **where** — Kunen's definition VI 1.7

" $\text{lrank}(x) \equiv \mu i. x \in \text{Lset}(\text{succ}(i))$ "

**lemma** *L\_I*: " $\llbracket x \in \text{Lset}(i); \text{Ord}(i) \rrbracket \implies L(x)$ "

*<proof>*

**lemma** *L\_D*: " $L(x) \implies \exists i. \text{Ord}(i) \wedge x \in \text{Lset}(i)$ "

*<proof>*

**lemma** *Ord\_lrank [simp]: "Ord(lrank(a))"*  
 $\langle proof \rangle$

**lemma** *Lset\_lrank\_lt [rule\_format]: "Ord(i)  $\implies x \in Lset(i) \longrightarrow lrank(x) < i$ "*  
 $\langle proof \rangle$

Kunen's VI 1.8. The proof is much harder than the text would suggest. For a start, it needs the previous lemma, which is proved by induction.

**lemma** *Lset\_iff\_lrank\_lt: "Ord(i)  $\implies x \in Lset(i) \longleftrightarrow L(x) \wedge lrank(x) < i$ "*  
 $\langle proof \rangle$

**lemma** *Lset\_succ\_lrank\_iff [simp]: "x  $\in Lset(succ(lrank(x))) \longleftrightarrow L(x)$ "*  
 $\langle proof \rangle$

Kunen's VI 1.9 (a)

**lemma** *lrank\_of\_Ord: "Ord(i)  $\implies lrank(i) = i$ "*  
 $\langle proof \rangle$

This is  $lrank(lrank(a)) = lrank(a)$

**declare** *Ord\_lrank [THEN lrank\_of\_Ord, simp]*

Kunen's VI 1.10

**lemma** *Lset\_in\_Lset\_succ: "Lset(i)  $\in Lset(succ(i))$ "*  
 $\langle proof \rangle$

**lemma** *lrank\_Lset: "Ord(i)  $\implies lrank(Lset(i)) = i$ "*  
 $\langle proof \rangle$

Kunen's VI 1.11

**lemma** *Lset\_subset\_Vset: "Ord(i)  $\implies Lset(i) \subseteq Vset(i)$ "*  
 $\langle proof \rangle$

Kunen's VI 1.12

**lemma** *Lset\_subset\_Vset': "i  $\in nat \implies Lset(i) = Vset(i)$ "*  
 $\langle proof \rangle$

Every set of constructible sets is included in some *Lset*

**lemma** *subset\_Lset:*  
 $"(\forall x \in A. L(x)) \implies \exists i. Ord(i) \wedge A \subseteq Lset(i)"$   
 $\langle proof \rangle$

**lemma** *subset\_LsetE:*  
 $"[\forall x \in A. L(x);$   
 $\bigwedge i. [Ord(i); A \subseteq Lset(i)] \implies P]$   
 $\implies P"$   
 $\langle proof \rangle$

### 1.9.3 For L to satisfy the Powerset axiom

**lemma** *LPow\_env\_typing*:  
 "⟦y ∈ Lset(i); Ord(i); y ⊆ X⟧  
 ⇒ ∃ z ∈ Pow(X). y ∈ Lset(succ(lrank(z)))"  
 ⟨proof⟩

**lemma** *LPow\_in\_Lset*:  
 "⟦X ∈ Lset(i); Ord(i)⟧ ⇒ ∃ j. Ord(j) ∧ {y ∈ Pow(X). L(y)} ∈ Lset(j)"  
 ⟨proof⟩

**theorem** *LPow\_in\_L*: "L(X) ⇒ L({y ∈ Pow(X). L(y)})"  
 ⟨proof⟩

### 1.10 Eliminating arity from the Definition of Lset

**lemma** *nth\_zero\_eq\_0*: "n ∈ nat ⇒ nth(n, [0]) = 0"  
 ⟨proof⟩

**lemma** *sats\_app\_0\_iff [rule\_format]*:  
 "⟦p ∈ formula; 0 ∈ A⟧  
 ⇒ ∀ env ∈ list(A). sats(A, p, env@[0]) ⟷ sats(A, p, env)"  
 ⟨proof⟩

**lemma** *sats\_app\_zeroes\_iff*:  
 "⟦p ∈ formula; 0 ∈ A; env ∈ list(A); n ∈ nat⟧  
 ⇒ sats(A, p, env @ repeat(0, n)) ⟷ sats(A, p, env)"  
 ⟨proof⟩

**lemma** *exists\_bigger\_env*:  
 "⟦p ∈ formula; 0 ∈ A; env ∈ list(A)⟧  
 ⇒ ∃ env' ∈ list(A). arity(p) ≤ succ(length(env')) ∧  
 (∀ a ∈ A. sats(A, p, Cons(a, env')) ⟷ sats(A, p, Cons(a, env)))"  
 ⟨proof⟩

A simpler version of *DPow*: no arity check!

**definition**

*DPow'* :: "i ⇒ i" where  
 "DPow'(A) ≡ {X ∈ Pow(A).  
 ∃ env ∈ list(A). ∃ p ∈ formula.  
 X = {x ∈ A. sats(A, p, Cons(x, env))}}"

**lemma** *DPow\_subset\_DPow'*: "DPow(A) ⊆ DPow'(A)"  
 ⟨proof⟩

**lemma** *DPow'\_0*: "DPow'(0) = {0}"  
 ⟨proof⟩

**lemma** *DPow'\_subset\_DPow*: "0 ∈ A ⇒ DPow'(A) ⊆ DPow(A)"  
 ⟨proof⟩



**lemma** *DPow\_eq\_DPow'*: " $\text{Transset}(A) \implies \text{DPow}(A) = \text{DPow}'(A)$ "  
 <proof>

And thus we can relativize *Lset* without bothering with *arity* and *length*

**lemma** *Lset\_eq\_transrec\_DPow'*: " $\text{Lset}(i) = \text{transrec}(i, \lambda x f. \bigcup_{y \in x} \text{DPow}'(f(y)))$ "  
 <proof>

With this rule we can specify *p* later and don't worry about arities at all!

**lemma** *DPow\_LsetI* [*rule\_format*]:  
 " $\llbracket \forall x \in \text{Lset}(i). P(x) \longleftrightarrow \text{sats}(\text{Lset}(i), p, \text{Cons}(x, \text{env})) ;$   
 $\text{env} \in \text{list}(\text{Lset}(i)); p \in \text{formula} \rrbracket$   
 $\implies \{x \in \text{Lset}(i). P(x)\} \in \text{DPow}(\text{Lset}(i))$ "  
 <proof>

end

## 2 Relativization and Absoluteness

**theory** *Relative* imports *ZF* begin

### 2.1 Relativized versions of standard set-theoretic concepts

**definition**

*empty* :: " $[i \Rightarrow o, i] \Rightarrow o$ " where  
 " $\text{empty}(M, z) \equiv \forall x[M]. x \notin z$ "

**definition**

*subset* :: " $[i \Rightarrow o, i, i] \Rightarrow o$ " where  
 " $\text{subset}(M, A, B) \equiv \forall x[M]. x \in A \longrightarrow x \in B$ "

**definition**

*upair* :: " $[i \Rightarrow o, i, i, i] \Rightarrow o$ " where  
 " $\text{upair}(M, a, b, z) \equiv a \in z \wedge b \in z \wedge (\forall x[M]. x \in z \longrightarrow x = a \vee x = b)$ "

**definition**

*pair* :: " $[i \Rightarrow o, i, i, i] \Rightarrow o$ " where  
 " $\text{pair}(M, a, b, z) \equiv \exists x[M]. \text{upair}(M, a, a, x) \wedge$   
 $(\exists y[M]. \text{upair}(M, a, b, y) \wedge \text{upair}(M, x, y, z))$ "

**definition**

*union* :: " $[i \Rightarrow o, i, i, i] \Rightarrow o$ " where  
 " $\text{union}(M, a, b, z) \equiv \forall x[M]. x \in z \longleftrightarrow x \in a \vee x \in b$ "

**definition**

*is\_cons* :: " $[i \Rightarrow o, i, i, i] \Rightarrow o$ " where

$\text{is\_cons}(M, a, b, z) \equiv \exists x[M]. \text{upair}(M, a, a, x) \wedge \text{union}(M, x, b, z)$ "

**definition**

$\text{successor} :: "[i \Rightarrow o, i, i] \Rightarrow o"$  where  
 $\text{"successor}(M, a, z) \equiv \text{is\_cons}(M, a, a, z)"$

**definition**

$\text{number1} :: "[i \Rightarrow o, i] \Rightarrow o"$  where  
 $\text{"number1}(M, a) \equiv \exists x[M]. \text{empty}(M, x) \wedge \text{successor}(M, x, a)"$

**definition**

$\text{number2} :: "[i \Rightarrow o, i] \Rightarrow o"$  where  
 $\text{"number2}(M, a) \equiv \exists x[M]. \text{number1}(M, x) \wedge \text{successor}(M, x, a)"$

**definition**

$\text{number3} :: "[i \Rightarrow o, i] \Rightarrow o"$  where  
 $\text{"number3}(M, a) \equiv \exists x[M]. \text{number2}(M, x) \wedge \text{successor}(M, x, a)"$

**definition**

$\text{powerset} :: "[i \Rightarrow o, i, i] \Rightarrow o"$  where  
 $\text{"powerset}(M, A, z) \equiv \forall x[M]. x \in z \longleftrightarrow \text{subset}(M, x, A)"$

**definition**

$\text{is\_Collect} :: "[i \Rightarrow o, i, i \Rightarrow o, i] \Rightarrow o"$  where  
 $\text{"is\_Collect}(M, A, P, z) \equiv \forall x[M]. x \in z \longleftrightarrow x \in A \wedge P(x)"$

**definition**

$\text{is\_Replace} :: "[i \Rightarrow o, i, [i, i] \Rightarrow o, i] \Rightarrow o"$  where  
 $\text{"is\_Replace}(M, A, P, z) \equiv \forall u[M]. u \in z \longleftrightarrow (\exists x[M]. x \in A \wedge P(x, u))"$

**definition**

$\text{inter} :: "[i \Rightarrow o, i, i, i] \Rightarrow o"$  where  
 $\text{"inter}(M, a, b, z) \equiv \forall x[M]. x \in z \longleftrightarrow x \in a \wedge x \in b"$

**definition**

$\text{setdiff} :: "[i \Rightarrow o, i, i, i] \Rightarrow o"$  where  
 $\text{"setdiff}(M, a, b, z) \equiv \forall x[M]. x \in z \longleftrightarrow x \in a \wedge x \notin b"$

**definition**

$\text{big\_union} :: "[i \Rightarrow o, i, i] \Rightarrow o"$  where  
 $\text{"big\_union}(M, A, z) \equiv \forall x[M]. x \in z \longleftrightarrow (\exists y[M]. y \in A \wedge x \in y)"$

**definition**

$\text{big\_inter} :: "[i \Rightarrow o, i, i] \Rightarrow o"$  where  
 $\text{"big\_inter}(M, A, z) \equiv$   
 $(A = 0 \longrightarrow z = 0) \wedge$   
 $(A \neq 0 \longrightarrow (\forall x[M]. x \in z \longleftrightarrow (\forall y[M]. y \in A \longrightarrow x \in y)))"$

**definition**

```

cartprod :: "[i⇒o,i,i,i] ⇒ o" where
  "cartprod(M,A,B,z) ≡
    ∀u[M]. u ∈ z ⟷ (∃x[M]. x∈A ∧ (∃y[M]. y∈B ∧ pair(M,x,y,u)))"

```

**definition**

```

is_sum :: "[i⇒o,i,i,i] ⇒ o" where
  "is_sum(M,A,B,Z) ≡
    ∃A0[M]. ∃n1[M]. ∃s1[M]. ∃B1[M].
    number1(M,n1) ∧ cartprod(M,n1,A,A0) ∧ upair(M,n1,n1,s1) ∧
    cartprod(M,s1,B,B1) ∧ union(M,A0,B1,Z)"

```

**definition**

```

is_Inl :: "[i⇒o,i,i] ⇒ o" where
  "is_Inl(M,a,z) ≡ ∃zero[M]. empty(M,zero) ∧ pair(M,zero,a,z)"

```

**definition**

```

is_Inr :: "[i⇒o,i,i] ⇒ o" where
  "is_Inr(M,a,z) ≡ ∃n1[M]. number1(M,n1) ∧ pair(M,n1,a,z)"

```

**definition**

```

is_converse :: "[i⇒o,i,i] ⇒ o" where
  "is_converse(M,r,z) ≡
    ∀x[M]. x ∈ z ⟷
    (∃w[M]. w∈r ∧ (∃u[M]. ∃v[M]. pair(M,u,v,w) ∧ pair(M,v,u,x)))"

```

**definition**

```

pre_image :: "[i⇒o,i,i,i] ⇒ o" where
  "pre_image(M,r,A,z) ≡
    ∀x[M]. x ∈ z ⟷ (∃w[M]. w∈r ∧ (∃y[M]. y∈A ∧ pair(M,x,y,w)))"

```

**definition**

```

is_domain :: "[i⇒o,i,i] ⇒ o" where
  "is_domain(M,r,z) ≡
    ∀x[M]. x ∈ z ⟷ (∃w[M]. w∈r ∧ (∃y[M]. pair(M,x,y,w)))"

```

**definition**

```

image :: "[i⇒o,i,i,i] ⇒ o" where
  "image(M,r,A,z) ≡
    ∀y[M]. y ∈ z ⟷ (∃w[M]. w∈r ∧ (∃x[M]. x∈A ∧ pair(M,x,y,w)))"

```

**definition**

```

is_range :: "[i⇒o,i,i] ⇒ o" where
  — the cleaner ∃r'[M]. is_converse(M, r, r') ∧ is_domain(M, r', z)
  unfortunately needs an instance of separation in order to prove M(converse(r)).
  "is_range(M,r,z) ≡
    ∀y[M]. y ∈ z ⟷ (∃w[M]. w∈r ∧ (∃x[M]. pair(M,x,y,w)))"

```

**definition**

```

is_field :: "[i⇒o,i,i] ⇒ o" where

```

```

"is_field(M,r,z) ≡
  ∃ dr[M]. ∃ rr[M]. is_domain(M,r,dr) ∧ is_range(M,r,rr) ∧
    union(M,dr,rr,z)"

```

**definition**

```

is_relation :: "[i⇒o,i] ⇒ o" where
  "is_relation(M,r) ≡
    (∀ z[M]. z∈r → (∃ x[M]. ∃ y[M]. pair(M,x,y,z)))"

```

**definition**

```

is_function :: "[i⇒o,i] ⇒ o" where
  "is_function(M,r) ≡
    ∀ x[M]. ∀ y[M]. ∀ y'[M]. ∀ p[M]. ∀ p'[M].
      pair(M,x,y,p) → pair(M,x,y',p') → p∈r → p'∈r → y=y'"

```

**definition**

```

fun_apply :: "[i⇒o,i,i,i] ⇒ o" where
  "fun_apply(M,f,x,y) ≡
    (∃ xs[M]. ∃ fxs[M].
      upair(M,x,x,xs) ∧ image(M,f,xs,fxs) ∧ big_union(M,fxs,y))"

```

**definition**

```

typed_function :: "[i⇒o,i,i,i] ⇒ o" where
  "typed_function(M,A,B,r) ≡
    is_function(M,r) ∧ is_relation(M,r) ∧ is_domain(M,r,A) ∧
    (∀ u[M]. u∈r → (∀ x[M]. ∀ y[M]. pair(M,x,y,u) → y∈B))"

```

**definition**

```

is_funspace :: "[i⇒o,i,i,i] ⇒ o" where
  "is_funspace(M,A,B,F) ≡
    ∀ f[M]. f ∈ F ↔ typed_function(M,A,B,f)"

```

**definition**

```

composition :: "[i⇒o,i,i,i] ⇒ o" where
  "composition(M,r,s,t) ≡
    ∀ p[M]. p ∈ t ↔
      (∃ x[M]. ∃ y[M]. ∃ z[M]. ∃ xy[M]. ∃ yz[M].
        pair(M,x,z,p) ∧ pair(M,x,y,xy) ∧ pair(M,y,z,yz) ∧
        xy ∈ s ∧ yz ∈ r)"

```

**definition**

```

injection :: "[i⇒o,i,i,i] ⇒ o" where
  "injection(M,A,B,f) ≡
    typed_function(M,A,B,f) ∧
    (∀ x[M]. ∀ x'[M]. ∀ y[M]. ∀ p[M]. ∀ p'[M].
      pair(M,x,y,p) → pair(M,x',y,p') → p∈f → p'∈f → x=x')"

```

**definition**

```

surjection :: "[i⇒o,i,i,i] ⇒ o" where

```

```

"surjection(M,A,B,f) ≡
  typed_function(M,A,B,f) ∧
  (∀ y[M]. y ∈ B → (∃ x[M]. x ∈ A ∧ fun_apply(M,f,x,y)))"

```

**definition**

```

bijection :: "[i⇒o,i,i,i] ⇒ o" where
  "bijection(M,A,B,f) ≡ injection(M,A,B,f) ∧ surjection(M,A,B,f)"

```

**definition**

```

restriction :: "[i⇒o,i,i,i] ⇒ o" where
  "restriction(M,r,A,z) ≡
    ∀ x[M]. x ∈ z ↔ (x ∈ r ∧ (∃ u[M]. u ∈ A ∧ (∃ v[M]. pair(M,u,v,x))))"

```

**definition**

```

transitive_set :: "[i⇒o,i] ⇒ o" where
  "transitive_set(M,a) ≡ ∀ x[M]. x ∈ a → subset(M,x,a)"

```

**definition**

```

ordinal :: "[i⇒o,i] ⇒ o" where
  — an ordinal is a transitive set of transitive sets
  "ordinal(M,a) ≡ transitive_set(M,a) ∧ (∀ x[M]. x ∈ a → transitive_set(M,x))"

```

**definition**

```

limit_ordinal :: "[i⇒o,i] ⇒ o" where
  — a limit ordinal is a non-empty, successor-closed ordinal
  "limit_ordinal(M,a) ≡
    ordinal(M,a) ∧ ¬ empty(M,a) ∧
    (∀ x[M]. x ∈ a → (∃ y[M]. y ∈ a ∧ successor(M,x,y)))"

```

**definition**

```

successor_ordinal :: "[i⇒o,i] ⇒ o" where
  — a successor ordinal is any ordinal that is neither empty nor limit
  "successor_ordinal(M,a) ≡
    ordinal(M,a) ∧ ¬ empty(M,a) ∧ ¬ limit_ordinal(M,a)"

```

**definition**

```

finite_ordinal :: "[i⇒o,i] ⇒ o" where
  — an ordinal is finite if neither it nor any of its elements are limit
  "finite_ordinal(M,a) ≡
    ordinal(M,a) ∧ ¬ limit_ordinal(M,a) ∧
    (∀ x[M]. x ∈ a → ¬ limit_ordinal(M,x))"

```

**definition**

```

omega :: "[i⇒o,i] ⇒ o" where
  — omega is a limit ordinal none of whose elements are limit
  "omega(M,a) ≡ limit_ordinal(M,a) ∧ (∀ x[M]. x ∈ a → ¬ limit_ordinal(M,x))"

```

**definition**

```

is_quasinat :: "[i⇒o,i] ⇒ o" where

```

"is\_quasinat(M,z)  $\equiv$  empty(M,z)  $\vee$  ( $\exists m[M].$  successor(M,m,z))"

**definition**

is\_nat\_case :: "[i $\Rightarrow$ o, i, [i,i] $\Rightarrow$ o, i, i]  $\Rightarrow$  o" where  
 "is\_nat\_case(M, a, is\_b, k, z)  $\equiv$   
 (empty(M,k)  $\longrightarrow$  z=a)  $\wedge$   
 ( $\forall m[M].$  successor(M,m,k)  $\longrightarrow$  is\_b(m,z))  $\wedge$   
 (is\_quasinat(M,k)  $\vee$  empty(M,z))"

**definition**

relation1 :: "[i $\Rightarrow$ o, [i,i] $\Rightarrow$ o, i $\Rightarrow$ i]  $\Rightarrow$  o" where  
 "relation1(M,is\_f,f)  $\equiv \forall x[M]. \forall y[M].$  is\_f(x,y)  $\longleftrightarrow$  y = f(x)"

**definition**

Relation1 :: "[i $\Rightarrow$ o, i, [i,i] $\Rightarrow$ o, i $\Rightarrow$ i]  $\Rightarrow$  o" where  
 — as above, but typed  
 "Relation1(M,A,is\_f,f)  $\equiv$   
 $\forall x[M]. \forall y[M].$  x $\in$ A  $\longrightarrow$  is\_f(x,y)  $\longleftrightarrow$  y = f(x)"

**definition**

relation2 :: "[i $\Rightarrow$ o, [i,i,i] $\Rightarrow$ o, [i,i] $\Rightarrow$ i]  $\Rightarrow$  o" where  
 "relation2(M,is\_f,f)  $\equiv \forall x[M]. \forall y[M]. \forall z[M].$  is\_f(x,y,z)  $\longleftrightarrow$  z = f(x,y)"

**definition**

Relation2 :: "[i $\Rightarrow$ o, i, i, [i,i,i] $\Rightarrow$ o, [i,i] $\Rightarrow$ i]  $\Rightarrow$  o" where  
 "Relation2(M,A,B,is\_f,f)  $\equiv$   
 $\forall x[M]. \forall y[M]. \forall z[M].$  x $\in$ A  $\longrightarrow$  y $\in$ B  $\longrightarrow$  is\_f(x,y,z)  $\longleftrightarrow$  z = f(x,y)"

**definition**

relation3 :: "[i $\Rightarrow$ o, [i,i,i,i] $\Rightarrow$ o, [i,i,i] $\Rightarrow$ i]  $\Rightarrow$  o" where  
 "relation3(M,is\_f,f)  $\equiv$   
 $\forall x[M]. \forall y[M]. \forall z[M]. \forall u[M].$  is\_f(x,y,z,u)  $\longleftrightarrow$  u = f(x,y,z)"

**definition**

Relation3 :: "[i $\Rightarrow$ o, i, i, i, [i,i,i,i] $\Rightarrow$ o, [i,i,i] $\Rightarrow$ i]  $\Rightarrow$  o" where  
 "Relation3(M,A,B,C,is\_f,f)  $\equiv$   
 $\forall x[M]. \forall y[M]. \forall z[M]. \forall u[M].$   
 x $\in$ A  $\longrightarrow$  y $\in$ B  $\longrightarrow$  z $\in$ C  $\longrightarrow$  is\_f(x,y,z,u)  $\longleftrightarrow$  u = f(x,y,z)"

**definition**

relation4 :: "[i $\Rightarrow$ o, [i,i,i,i,i] $\Rightarrow$ o, [i,i,i,i] $\Rightarrow$ i]  $\Rightarrow$  o" where  
 "relation4(M,is\_f,f)  $\equiv$   
 $\forall u[M]. \forall x[M]. \forall y[M]. \forall z[M]. \forall a[M].$  is\_f(u,x,y,z,a)  $\longleftrightarrow$  a = f(u,x,y,z)"

Useful when absoluteness reasoning has replaced the predicates by terms

**lemma** triv\_Relation1:

"Relation1(M, A,  $\lambda x y. y = f(x), f$ )"

*<proof>*

**lemma** *triv\_Relation2*:  
 "Relation2( $M, A, B, \lambda x y a. a = f(x,y), f$ )"  
 $\langle proof \rangle$

## 2.2 The relativized ZF axioms

### definition

*extensionality* :: "( $i \Rightarrow o$ )  $\Rightarrow$   $o$ " where  
 "extensionality( $M$ )  $\equiv$   
 $\forall x[M]. \forall y[M]. (\forall z[M]. z \in x \longleftrightarrow z \in y) \longrightarrow x=y$ "

### definition

*separation* :: "( $i \Rightarrow o, i \Rightarrow o$ )  $\Rightarrow$   $o$ " where  
 — The formula  $P$  should only involve parameters belonging to  $M$  and all its quantifiers must be relativized to  $M$ . We do not have separation as a scheme; every instance that we need must be assumed (and later proved) separately.  
 "separation( $M, P$ )  $\equiv$   
 $\forall z[M]. \exists y[M]. \forall x[M]. x \in y \longleftrightarrow x \in z \wedge P(x)$ "

### definition

*upair\_ax* :: "( $i \Rightarrow o$ )  $\Rightarrow$   $o$ " where  
 "upair\_ax( $M$ )  $\equiv \forall x[M]. \forall y[M]. \exists z[M]. \text{upair}(M, x, y, z)$ "

### definition

*Union\_ax* :: "( $i \Rightarrow o$ )  $\Rightarrow$   $o$ " where  
 "Union\_ax( $M$ )  $\equiv \forall x[M]. \exists z[M]. \text{big\_union}(M, x, z)$ "

### definition

*power\_ax* :: "( $i \Rightarrow o$ )  $\Rightarrow$   $o$ " where  
 "power\_ax( $M$ )  $\equiv \forall x[M]. \exists z[M]. \text{powerset}(M, x, z)$ "

### definition

*univalent* :: "( $i \Rightarrow o, i, [i, i] \Rightarrow o$ )  $\Rightarrow$   $o$ " where  
 "univalent( $M, A, P$ )  $\equiv$   
 $\forall x[M]. x \in A \longrightarrow (\forall y[M]. \forall z[M]. P(x, y) \wedge P(x, z) \longrightarrow y=z)$ "

### definition

*replacement* :: "( $i \Rightarrow o, [i, i] \Rightarrow o$ )  $\Rightarrow$   $o$ " where  
 "replacement( $M, P$ )  $\equiv$   
 $\forall A[M]. \text{univalent}(M, A, P) \longrightarrow$   
 $(\exists Y[M]. \forall b[M]. (\exists x[M]. x \in A \wedge P(x, b)) \longrightarrow b \in Y)$ "

### definition

*strong\_replacement* :: "( $i \Rightarrow o, [i, i] \Rightarrow o$ )  $\Rightarrow$   $o$ " where  
 "strong\_replacement( $M, P$ )  $\equiv$   
 $\forall A[M]. \text{univalent}(M, A, P) \longrightarrow$   
 $(\exists Y[M]. \forall b[M]. b \in Y \longleftrightarrow (\exists x[M]. x \in A \wedge P(x, b)))$ "

**definition**

`foundation_ax :: "(i⇒o) ⇒ o" where`  
`"foundation_ax(M) ≡`  
`∀ x[M]. (∃ y[M]. y∈x) → (∃ y[M]. y∈x ∧ ¬(∃ z[M]. z∈x ∧ z ∈`  
`y))"`

## 2.3 A trivial consistency proof for $V_\omega$

We prove that  $V_\omega$  (or *univ* in Isabelle) satisfies some ZF axioms. Kunen, Theorem IV 3.13, page 123.

**lemma** `univ0_downwards_mem`: " $\llbracket y \in x; x \in \text{univ}(0) \rrbracket \implies y \in \text{univ}(0)$ "  
`<proof>`

**lemma** `univ0_Ball_abs [simp]`:  
`"A ∈ univ(0) ⟹ (∀ x∈A. x ∈ univ(0) → P(x)) ⟷ (∀ x∈A. P(x))"`  
`<proof>`

**lemma** `univ0_Bex_abs [simp]`:  
`"A ∈ univ(0) ⟹ (∃ x∈A. x ∈ univ(0) ∧ P(x)) ⟷ (∃ x∈A. P(x))"`  
`<proof>`

Congruence rule for separation: can assume the variable is in  $M$

**lemma** `separation_cong [cong]`:  
`"(⋀ x. M(x) ⟹ P(x) ⟷ P'(x))`  
`⟹ separation(M, λx. P(x)) ⟷ separation(M, λx. P'(x))"`  
`<proof>`

**lemma** `univalent_cong [cong]`:  
`"⋈ A=A'; ⋀ x y. ⋈ x∈A; M(x); M(y) ⟹ P(x,y) ⟷ P'(x,y)⋈`  
`⟹ univalent(M, A, λx y. P(x,y)) ⟷ univalent(M, A', λx y. P'(x,y))"`  
`<proof>`

**lemma** `univalent_triv [intro,simp]`:  
`"univalent(M, A, λx y. y = f(x))"`  
`<proof>`

**lemma** `univalent_conjI2 [intro,simp]`:  
`"univalent(M,A,Q) ⟹ univalent(M, A, λx y. P(x,y) ∧ Q(x,y))"`  
`<proof>`

Congruence rule for replacement

**lemma** `strong_replacement_cong [cong]`:  
`"⋈ ⋀ x y. ⋈ M(x); M(y) ⟹ P(x,y) ⟷ P'(x,y)⋈`  
`⟹ strong_replacement(M, λx y. P(x,y)) ⟷`  
`strong_replacement(M, λx y. P'(x,y))"`  
`<proof>`

The extensionality axiom



**lemma** "extensionality( $\lambda x. x \in \text{univ}(0)$ )"  
 $\langle \text{proof} \rangle$

The separation axiom requires some lemmas

**lemma** *Collect\_in\_Vfrom*:  
 $\llbracket X \in \text{Vfrom}(A, j); \text{Transset}(A) \rrbracket \implies \text{Collect}(X, P) \in \text{Vfrom}(A, \text{succ}(j))$   
 $\langle \text{proof} \rangle$

**lemma** *Collect\_in\_VLimit*:  
 $\llbracket X \in \text{Vfrom}(A, i); \text{Limit}(i); \text{Transset}(A) \rrbracket$   
 $\implies \text{Collect}(X, P) \in \text{Vfrom}(A, i)$   
 $\langle \text{proof} \rangle$

**lemma** *Collect\_in\_univ*:  
 $\llbracket X \in \text{univ}(A); \text{Transset}(A) \rrbracket \implies \text{Collect}(X, P) \in \text{univ}(A)$   
 $\langle \text{proof} \rangle$

**lemma** "separation( $\lambda x. x \in \text{univ}(0), P$ )"  
 $\langle \text{proof} \rangle$

Unordered pairing axiom

**lemma** "upair\_ax( $\lambda x. x \in \text{univ}(0)$ )"  
 $\langle \text{proof} \rangle$

Union axiom

**lemma** "Union\_ax( $\lambda x. x \in \text{univ}(0)$ )"  
 $\langle \text{proof} \rangle$

Powerset axiom

**lemma** *Pow\_in\_univ*:  
 $\llbracket X \in \text{univ}(A); \text{Transset}(A) \rrbracket \implies \text{Pow}(X) \in \text{univ}(A)$   
 $\langle \text{proof} \rangle$

**lemma** "power\_ax( $\lambda x. x \in \text{univ}(0)$ )"  
 $\langle \text{proof} \rangle$

Foundation axiom

**lemma** "foundation\_ax( $\lambda x. x \in \text{univ}(0)$ )"  
 $\langle \text{proof} \rangle$

**lemma** "replacement( $\lambda x. x \in \text{univ}(0), P$ )"  
 $\langle \text{proof} \rangle$

no idea: maybe prove by induction on the rank of A?

Still missing: Replacement, Choice

## 2.4 Lemmas Needed to Reduce Some Set Constructions to Instances of Separation

**lemma** *image\_iff\_Collect*: "r ‘‘ A = {y ∈ ⋃ (⋃ (r)). ∃ p ∈ r. ∃ x ∈ A. p = ⟨x, y⟩}"  
 ⟨proof⟩

**lemma** *vimage\_iff\_Collect*:  
 "r -‘‘ A = {x ∈ ⋃ (⋃ (r)). ∃ p ∈ r. ∃ y ∈ A. p = ⟨x, y⟩}"  
 ⟨proof⟩

These two lemmas lets us prove *domain\_closed* and *range\_closed* without new instances of separation

**lemma** *domain\_eq\_vimage*: "domain(r) = r -‘‘ Union(Union(r))"  
 ⟨proof⟩

**lemma** *range\_eq\_image*: "range(r) = r ‘‘ Union(Union(r))"  
 ⟨proof⟩

**lemma** *replacementD*:  
 "⟦replacement(M,P); M(A); univalent(M,A,P)⟧  
 ⇒ ∃ Y[M]. (∀ b[M]. ((∃ x[M]. x ∈ A ∧ P(x,b)) → b ∈ Y))"  
 ⟨proof⟩

**lemma** *strong\_replacementD*:  
 "⟦strong\_replacement(M,P); M(A); univalent(M,A,P)⟧  
 ⇒ ∃ Y[M]. (∀ b[M]. (b ∈ Y ↔ (∃ x[M]. x ∈ A ∧ P(x,b))))"  
 ⟨proof⟩

**lemma** *separationD*:  
 "⟦separation(M,P); M(z)⟧ ⇒ ∃ y[M]. ∀ x[M]. x ∈ y ↔ x ∈ z ∧ P(x)"  
 ⟨proof⟩

More constants, for order types

**definition**

*order\_isomorphism* :: "[i ⇒ o, i, i, i, i, i] ⇒ o" where  
 "order\_isomorphism(M,A,r,B,s,f) ≡  
 bijection(M,A,B,f) ∧  
 (∀ x[M]. x ∈ A → (∀ y[M]. y ∈ A →  
 (∀ p[M]. ∀ fx[M]. ∀ fy[M]. ∀ q[M].  
 pair(M,x,y,p) → fun\_apply(M,f,x,fx) → fun\_apply(M,f,y,fy)  
 →  
 pair(M,fx,fy,q) → (p ∈ r ↔ q ∈ s))))"

**definition**

*pred\_set* :: "[i ⇒ o, i, i, i, i] ⇒ o" where  
 "pred\_set(M,A,x,r,B) ≡  
 ∀ y[M]. y ∈ B ↔ (∃ p[M]. p ∈ r ∧ y ∈ A ∧ pair(M,y,x,p))"

**definition**

```

membership :: "[i⇒o,i,i] ⇒ o" where — membership relation
"membership(M,A,r) ≡
  ∀p[M]. p ∈ r ⟷ (∃x[M]. x∈A ∧ (∃y[M]. y∈A ∧ x∈y ∧ pair(M,x,y,p)))"

```

## 2.5 Introducing a Transitive Class Model

The class M is assumed to be transitive and inhabited

```

locale M_trans =
  fixes M
  assumes transM: "[y∈x; M(x)] ⇒ M(y)"
  and M_inhabited: "∃x . M(x)"

```

```

lemma (in M_trans) nonempty [simp]: "M(0)"
<proof>

```

The class M is assumed to be transitive and to satisfy some relativized ZF axioms

```

locale M_trivial = M_trans +
  assumes upair_ax: "upair_ax(M)"
  and Union_ax: "Union_ax(M)"

```

```

lemma (in M_trans) rall_abs [simp]:
  "M(A) ⇒ (∀x[M]. x∈A ⟶ P(x)) ⟷ (∀x∈A. P(x))"
<proof>

```

```

lemma (in M_trans) rex_abs [simp]:
  "M(A) ⇒ (∃x[M]. x∈A ∧ P(x)) ⟷ (∃x∈A. P(x))"
<proof>

```

```

lemma (in M_trans) ball_iff_equiv:
  "M(A) ⇒ (∀x[M]. (x∈A ⟷ P(x))) ⟷
    (∀x∈A. P(x)) ∧ (∀x. P(x) ⟶ M(x) ⟶ x∈A)"
<proof>

```

Simplifies proofs of equalities when there's an iff-equality available for rewriting, universally quantified over M. But it's not the only way to prove such equalities: its premises  $M(A)$  and  $M(B)$  can be too strong.

```

lemma (in M_trans) M_equalityI:
  "[[∧x. M(x) ⇒ x∈A ⟷ x∈B; M(A); M(B)] ⇒ A=B"
<proof>

```

### 2.5.1 Trivial Absoluteness Proofs: Empty Set, Pairs, etc.

```

lemma (in M_trans) empty_abs [simp]:
  "M(z) ⇒ empty(M,z) ⟷ z=0"
<proof>

```

```

lemma (in M_trans) subset_abs [simp]:

```

$$M(A) \implies \text{subset}(M, A, B) \iff A \subseteq B$$
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trans$ )  $\text{upair\_abs}$  [simp]:  

$$M(z) \implies \text{upair}(M, a, b, z) \iff z = \{a, b\}$$
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ )  $\text{upair\_in\_MI}$  [intro!]:  

$$M(a) \wedge M(b) \implies M(\{a, b\})$$
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trans$ )  $\text{upair\_in\_MD}$  [dest!]:  

$$M(\{a, b\}) \implies M(a) \wedge M(b)$$
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ )  $\text{upair\_in\_M\_iff}$  [simp]:  

$$M(\{a, b\}) \iff M(a) \wedge M(b)$$
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ )  $\text{singleton\_in\_MI}$  [intro!]:  

$$M(a) \implies M(\{a\})$$
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trans$ )  $\text{singleton\_in\_MD}$  [dest!]:  

$$M(\{a\}) \implies M(a)$$
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ )  $\text{singleton\_in\_M\_iff}$  [simp]:  

$$M(\{a\}) \iff M(a)$$
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trans$ )  $\text{pair\_abs}$  [simp]:  

$$M(z) \implies \text{pair}(M, a, b, z) \iff z = \langle a, b \rangle$$
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trans$ )  $\text{pair\_in\_MD}$  [dest!]:  

$$M(\langle a, b \rangle) \implies M(a) \wedge M(b)$$
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ )  $\text{pair\_in\_MI}$  [intro!]:  

$$M(a) \wedge M(b) \implies M(\langle a, b \rangle)$$
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ )  $\text{pair\_in\_M\_iff}$  [simp]:  

$$M(\langle a, b \rangle) \iff M(a) \wedge M(b)$$
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trans$ )  $\text{pair\_components\_in\_M}$ :  

$$\llbracket \langle x, y \rangle \in A; M(A) \rrbracket \implies M(x) \wedge M(y)$$

$\langle proof \rangle$

**lemma** (in  $M\_trivial$ )  $cartprod\_abs$  [simp]:  
"  $\llbracket M(A); M(B); M(z) \rrbracket \implies cartprod(M, A, B, z) \longleftrightarrow z = A * B$  "  
 $\langle proof \rangle$

### 2.5.2 Absoluteness for Unions and Intersections

**lemma** (in  $M\_trans$ )  $union\_abs$  [simp]:  
"  $\llbracket M(a); M(b); M(z) \rrbracket \implies union(M, a, b, z) \longleftrightarrow z = a \cup b$  "  
 $\langle proof \rangle$

**lemma** (in  $M\_trans$ )  $inter\_abs$  [simp]:  
"  $\llbracket M(a); M(b); M(z) \rrbracket \implies inter(M, a, b, z) \longleftrightarrow z = a \cap b$  "  
 $\langle proof \rangle$

**lemma** (in  $M\_trans$ )  $setdiff\_abs$  [simp]:  
"  $\llbracket M(a); M(b); M(z) \rrbracket \implies setdiff(M, a, b, z) \longleftrightarrow z = a - b$  "  
 $\langle proof \rangle$

**lemma** (in  $M\_trans$ )  $Union\_abs$  [simp]:  
"  $\llbracket M(A); M(z) \rrbracket \implies big\_union(M, A, z) \longleftrightarrow z = \bigcup (A)$  "  
 $\langle proof \rangle$

**lemma** (in  $M\_trivial$ )  $Union\_closed$  [intro, simp]:  
"  $M(A) \implies M(\bigcup (A))$  "  
 $\langle proof \rangle$

**lemma** (in  $M\_trivial$ )  $Un\_closed$  [intro, simp]:  
"  $\llbracket M(A); M(B) \rrbracket \implies M(A \cup B)$  "  
 $\langle proof \rangle$

**lemma** (in  $M\_trivial$ )  $cons\_closed$  [intro, simp]:  
"  $\llbracket M(a); M(A) \rrbracket \implies M(cons(a, A))$  "  
 $\langle proof \rangle$

**lemma** (in  $M\_trivial$ )  $cons\_abs$  [simp]:  
"  $\llbracket M(b); M(z) \rrbracket \implies is\_cons(M, a, b, z) \longleftrightarrow z = cons(a, b)$  "  
 $\langle proof \rangle$

**lemma** (in  $M\_trivial$ )  $successor\_abs$  [simp]:  
"  $\llbracket M(a); M(z) \rrbracket \implies successor(M, a, z) \longleftrightarrow z = succ(a)$  "  
 $\langle proof \rangle$

**lemma** (in  $M\_trans$ )  $succ\_in\_MD$  [dest!]:  
"  $M(succ(a)) \implies M(a)$  "  
 $\langle proof \rangle$

**lemma** (in  $M\_trivial$ )  $succ\_in\_MI$  [intro!]:

" $M(a) \implies M(\text{succ}(a))$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ )  $\text{succ\_in\_M\_iff}$  [simp]:  
 " $M(\text{succ}(a)) \longleftrightarrow M(a)$ "  
 $\langle \text{proof} \rangle$

### 2.5.3 Absoluteness for Separation and Replacement

**lemma** (in  $M\_trans$ )  $\text{separation\_closed}$  [intro,simp]:  
 " $\llbracket \text{separation}(M,P); M(A) \rrbracket \implies M(\text{Collect}(A,P))$ "  
 $\langle \text{proof} \rangle$

**lemma**  $\text{separation\_iff}$ :  
 " $\text{separation}(M,P) \longleftrightarrow (\forall z[M]. \exists y[M]. \text{is\_Collect}(M,z,P,y))$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trans$ )  $\text{Collect\_abs}$  [simp]:  
 " $\llbracket M(A); M(z) \rrbracket \implies \text{is\_Collect}(M,A,P,z) \longleftrightarrow z = \text{Collect}(A,P)$ "  
 $\langle \text{proof} \rangle$

### 2.5.4 The Operator $\text{is\_Replace}$

**lemma**  $\text{is\_Replace\_cong}$  [cong]:  
 " $\llbracket A=A';$   
 $\bigwedge x y. \llbracket M(x); M(y) \rrbracket \implies P(x,y) \longleftrightarrow P'(x,y);$   
 $z=z' \rrbracket$   
 $\implies \text{is\_Replace}(M, A, \lambda x y. P(x,y), z) \longleftrightarrow$   
 $\text{is\_Replace}(M, A', \lambda x y. P'(x,y), z')$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trans$ )  $\text{univalent\_Replace\_iff}$ :  
 " $\llbracket M(A); \text{univalent}(M,A,P);$   
 $\bigwedge x y. \llbracket x \in A; P(x,y) \rrbracket \implies M(y) \rrbracket$   
 $\implies u \in \text{Replace}(A,P) \longleftrightarrow (\exists x. x \in A \wedge P(x,u))$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trans$ )  $\text{strong\_replacement\_closed}$  [intro,simp]:  
 " $\llbracket \text{strong\_replacement}(M,P); M(A); \text{univalent}(M,A,P);$   
 $\bigwedge x y. \llbracket x \in A; P(x,y) \rrbracket \implies M(y) \rrbracket \implies M(\text{Replace}(A,P))$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trans$ )  $\text{Replace\_abs}$ :  
 " $\llbracket M(A); M(z); \text{univalent}(M,A,P);$   
 $\bigwedge x y. \llbracket x \in A; P(x,y) \rrbracket \implies M(y) \rrbracket$   
 $\implies \text{is\_Replace}(M,A,P,z) \longleftrightarrow z = \text{Replace}(A,P)$ "  
 $\langle \text{proof} \rangle$

**lemma** (in *M\_trans*) *RepFun\_closed*:  
 "[strong\_replacement(*M*,  $\lambda x y. y = f(x)$ ); *M*(*A*);  $\forall x \in A. M(f(x))$ ]"  
 $\implies M(\text{RepFun}(A, f))$ "  
 <proof>

**lemma** *Replace\_conj\_eq*: "{*y* .  $x \in A, x \in A \wedge y = f(x)$ } = {*y* .  $x \in A, y = f(x)$ }"  
 <proof>

Better than *RepFun\_closed* when having the formula  $x \in A$  makes relativization easier.

**lemma** (in *M\_trans*) *RepFun\_closed2*:  
 "[strong\_replacement(*M*,  $\lambda x y. x \in A \wedge y = f(x)$ ); *M*(*A*);  $\forall x \in A. M(f(x))$ ]"  
 $\implies M(\text{RepFun}(A, \lambda x. f(x)))$ "  
 <proof>

### 2.5.5 Absoluteness for *Lambda*

**definition**

*is\_lambda* :: "[*i*  $\implies$  *o*, *i*, [*i*, *i*]  $\implies$  *o*, *i*]  $\implies$  *o*" where  
 "*is\_lambda*(*M*, *A*, *is\_b*, *z*)  $\equiv$   
 $\forall p[M]. p \in z \longleftrightarrow$   
 $(\exists u[M]. \exists v[M]. u \in A \wedge \text{pair}(M, u, v, p) \wedge \text{is\_b}(u, v))$ "

**lemma** (in *M\_trivial*) *lam\_closed*:  
 "[strong\_replacement(*M*,  $\lambda x y. y = \langle x, b(x) \rangle$ ); *M*(*A*);  $\forall x \in A. M(b(x))$ ]"  
 $\implies M(\lambda x \in A. b(x))$ "  
 <proof>

Better than *lam\_closed*: has the formula  $x \in A$

**lemma** (in *M\_trivial*) *lam\_closed2*:  
 "[strong\_replacement(*M*,  $\lambda x y. x \in A \wedge y = \langle x, b(x) \rangle$ );  
*M*(*A*);  $\forall m[M]. m \in A \longrightarrow M(b(m))$ ]  $\implies M(\text{Lambda}(A, b))$ "  
 <proof>

**lemma** (in *M\_trivial*) *lambda\_abs2*:  
 "[Relation1(*M*, *A*, *is\_b*, *b*); *M*(*A*);  $\forall m[M]. m \in A \longrightarrow M(b(m))$ ; *M*(*z*)]"  
 $\implies \text{is\_lambda}(M, A, \text{is\_b}, z) \longleftrightarrow z = \text{Lambda}(A, b)$ "  
 <proof>

**lemma** *is\_lambda\_cong* [*cong*]:  
 "[*A* = *A'*; *z* = *z'*;  
 $\bigwedge x y. [x \in A; M(x); M(y)] \implies \text{is\_b}(x, y) \longleftrightarrow \text{is\_b}'(x, y)$ ]"  
 $\implies \text{is\_lambda}(M, A, \lambda x y. \text{is\_b}(x, y), z) \longleftrightarrow$   
 $\text{is\_lambda}(M, A', \lambda x y. \text{is\_b}'(x, y), z')$ "  
 <proof>

**lemma** (in *M\_trans*) *image\_abs* [*simp*]:  
 "[*M*(*r*); *M*(*A*); *M*(*z*)]  $\implies \text{image}(M, r, A, z) \longleftrightarrow z = r `` A$ "

$\langle proof \rangle$

### 2.5.6 Relativization of Powerset

What about  $Pow\_abs$ ? Powerset is NOT absolute! This result is one direction of absoluteness.

```
lemma (in M_trans) powerset_Pow:
  "powerset(M, x, Pow(x))"
 $\langle proof \rangle$ 
```

But we can't prove that the powerset in  $M$  includes the real powerset.

```
lemma (in M_trans) powerset_imp_subset_Pow:
  "[powerset(M,x,y); M(y)]  $\implies y \subseteq Pow(x)$ "
 $\langle proof \rangle$ 
```

```
lemma (in M_trans) powerset_abs:
  assumes
    "M(y)"
  shows
    "powerset(M,x,y)  $\longleftrightarrow y = \{a \in Pow(x) \mid M(a)\}"$ 
 $\langle proof \rangle$ 
```

### 2.5.7 Absoluteness for the Natural Numbers

```
lemma (in M_trivial) nat_into_M [intro]:
  "n  $\in nat \implies M(n)$ "
 $\langle proof \rangle$ 
```

```
lemma (in M_trans) nat_case_closed [intro,simp]:
  "[M(k); M(a);  $\forall m[M]. M(b(m))$ ]  $\implies M(nat\_case(a,b,k))$ "
 $\langle proof \rangle$ 
```

```
lemma (in M_trivial) quasinat_abs [simp]:
  "M(z)  $\implies is\_quasinat(M,z) \longleftrightarrow quasinat(z)$ "
 $\langle proof \rangle$ 
```

```
lemma (in M_trivial) nat_case_abs [simp]:
  "[relation1(M,is_b,b); M(k); M(z)]
 $\implies is\_nat\_case(M,a,is_b,k,z) \longleftrightarrow z = nat\_case(a,b,k)$ "
 $\langle proof \rangle$ 
```

```
lemma is_nat_case_cong:
  "[a = a'; k = k'; z = z'; M(z');
 $\bigwedge x y. [M(x); M(y)] \implies is\_b(x,y) \longleftrightarrow is\_b'(x,y)$ ]
 $\implies is\_nat\_case(M, a, is\_b, k, z) \longleftrightarrow is\_nat\_case(M, a', is\_b',$ 
k', z')]"
 $\langle proof \rangle$ 
```



## 2.6 Absoluteness for Ordinals

These results constitute Theorem IV 5.1 of Kunen (page 126).

**lemma** (in *M\_trans*) *lt\_closed*:

" $\llbracket j < i; M(i) \rrbracket \implies M(j)$ "

*<proof>*

**lemma** (in *M\_trans*) *transitive\_set\_abs [simp]*:

" $M(a) \implies \text{transitive\_set}(M,a) \longleftrightarrow \text{Transset}(a)$ "

*<proof>*

**lemma** (in *M\_trans*) *ordinal\_abs [simp]*:

" $M(a) \implies \text{ordinal}(M,a) \longleftrightarrow \text{Ord}(a)$ "

*<proof>*

**lemma** (in *M\_trivial*) *limit\_ordinal\_abs [simp]*:

" $M(a) \implies \text{limit\_ordinal}(M,a) \longleftrightarrow \text{Limit}(a)$ "

*<proof>*

**lemma** (in *M\_trivial*) *successor\_ordinal\_abs [simp]*:

" $M(a) \implies \text{successor\_ordinal}(M,a) \longleftrightarrow \text{Ord}(a) \wedge (\exists b[M]. a = \text{succ}(b))$ "

*<proof>*

**lemma** *finite\_Ord\_is\_nat*:

" $\llbracket \text{Ord}(a); \neg \text{Limit}(a); \forall x \in a. \neg \text{Limit}(x) \rrbracket \implies a \in \text{nat}$ "

*<proof>*

**lemma** (in *M\_trivial*) *finite\_ordinal\_abs [simp]*:

" $M(a) \implies \text{finite\_ordinal}(M,a) \longleftrightarrow a \in \text{nat}$ "

*<proof>*

**lemma** *Limit\_non\_Limit\_implies\_nat*:

" $\llbracket \text{Limit}(a); \forall x \in a. \neg \text{Limit}(x) \rrbracket \implies a = \text{nat}$ "

*<proof>*

**lemma** (in *M\_trivial*) *omega\_abs [simp]*:

" $M(a) \implies \text{omega}(M,a) \longleftrightarrow a = \text{nat}$ "

*<proof>*

**lemma** (in *M\_trivial*) *number1\_abs [simp]*:

" $M(a) \implies \text{number1}(M,a) \longleftrightarrow a = 1$ "

*<proof>*

**lemma** (in *M\_trivial*) *number2\_abs [simp]*:

" $M(a) \implies \text{number2}(M,a) \longleftrightarrow a = \text{succ}(1)$ "

*<proof>*

**lemma** (in *M\_trivial*) *number3\_abs [simp]*:

" $M(a) \implies \text{number3}(M,a) \longleftrightarrow a = \text{succ}(\text{succ}(1))$ "

$\langle proof \rangle$

Kunen continued to 20...

## 2.7 Some instances of separation and strong replacement

```

locale M_basic = M_trivial +
assumes Inter_separation:
  " $M(A) \implies \text{separation}(M, \lambda x. \forall y[M]. y \in A \longrightarrow x \in y)$ "
and Diff_separation:
  " $M(B) \implies \text{separation}(M, \lambda x. x \notin B)$ "
and cartprod_separation:
  " $\llbracket M(A); M(B) \rrbracket$ 
     $\implies \text{separation}(M, \lambda z. \exists x[M]. x \in A \wedge (\exists y[M]. y \in B \wedge \text{pair}(M, x, y, z)))$ "
and image_separation:
  " $\llbracket M(A); M(r) \rrbracket$ 
     $\implies \text{separation}(M, \lambda y. \exists p[M]. p \in r \wedge (\exists x[M]. x \in A \wedge \text{pair}(M, x, y, p)))$ "
and converse_separation:
  " $M(r) \implies \text{separation}(M,$ 
     $\lambda z. \exists p[M]. p \in r \wedge (\exists x[M]. \exists y[M]. \text{pair}(M, x, y, p) \wedge \text{pair}(M, y, x, z)))$ "
and restrict_separation:
  " $M(A) \implies \text{separation}(M, \lambda z. \exists x[M]. x \in A \wedge (\exists y[M]. \text{pair}(M, x, y, z)))$ "
and comp_separation:
  " $\llbracket M(r); M(s) \rrbracket$ 
     $\implies \text{separation}(M, \lambda xz. \exists x[M]. \exists y[M]. \exists z[M]. \exists xy[M]. \exists yz[M].$ 
       $\text{pair}(M, x, z, xz) \wedge \text{pair}(M, x, y, xy) \wedge \text{pair}(M, y, z, yz) \wedge$ 
       $xy \in s \wedge yz \in r)$ "
and pred_separation:
  " $\llbracket M(r); M(x) \rrbracket \implies \text{separation}(M, \lambda y. \exists p[M]. p \in r \wedge \text{pair}(M, y, x, p))$ "
and Memrel_separation:
  " $\text{separation}(M, \lambda z. \exists x[M]. \exists y[M]. \text{pair}(M, x, y, z) \wedge x \in y)$ "
and funspace_succ_replacement:
  " $M(n) \implies$ 
     $\text{strong\_replacement}(M, \lambda p z. \exists f[M]. \exists b[M]. \exists nb[M]. \exists cnbf[M].$ 
       $\text{pair}(M, f, b, p) \wedge \text{pair}(M, n, b, nb) \wedge \text{is\_cons}(M, nb, f, cnbf)$ 
     $\wedge$ 
       $\text{upair}(M, cnbf, cnbf, z))$ "
and is_recfun_separation:
  — for well-founded recursion: used to prove is_recfun_equal
  " $\llbracket M(r); M(f); M(g); M(a); M(b) \rrbracket$ 
     $\implies \text{separation}(M,$ 
       $\lambda x. \exists xa[M]. \exists xb[M].$ 
       $\text{pair}(M, x, a, xa) \wedge xa \in r \wedge \text{pair}(M, x, b, xb) \wedge xb \in r \wedge$ 
       $(\exists fx[M]. \exists gx[M]. \text{fun\_apply}(M, f, x, fx) \wedge \text{fun\_apply}(M, g, x, gx)$ 
     $\wedge$ 
       $fx \neq gx))$ "
and power_ax:
  " $\text{power\_ax}(M)$ "

lemma (in M_trivial) cartprod_iff_lemma:
```

```

"[[M(C);  $\forall u[M]. u \in C \longleftrightarrow (\exists x \in A. \exists y \in B. u = \{\{x\}, \{x,y\}\}\}$ );
  powerset(M, A  $\cup$  B, p1); powerset(M, p1, p2); M(p2)]]
 $\implies C = \{u \in p2 \mid \exists x \in A. \exists y \in B. u = \{\{x\}, \{x,y\}\}\}$ "
<proof>

```

```

lemma (in M_basic) cartprod_iff:
  "[[M(A); M(B); M(C)]]
 $\implies \text{cartprod}(M, A, B, C) \longleftrightarrow$ 
  ( $\exists p1[M]. \exists p2[M]. \text{powerset}(M, A \cup B, p1) \wedge \text{powerset}(M, p1, p2) \wedge$ 
     $C = \{z \in p2 \mid \exists x \in A. \exists y \in B. z = \langle x, y \rangle\}$ )"
<proof>

```

```

lemma (in M_basic) cartprod_closed_lemma:
  "[[M(A); M(B)]]  $\implies \exists C[M]. \text{cartprod}(M, A, B, C)$ "
<proof>

```

All the lemmas above are necessary because Powerset is not absolute. I should have used Replacement instead!

```

lemma (in M_basic) cartprod_closed [intro,simp]:
  "[[M(A); M(B)]]  $\implies M(A*B)$ "
<proof>

```

```

lemma (in M_basic) sum_closed [intro,simp]:
  "[[M(A); M(B)]]  $\implies M(A+B)$ "
<proof>

```

```

lemma (in M_basic) sum_abs [simp]:
  "[[M(A); M(B); M(Z)]]  $\implies \text{is\_sum}(M, A, B, Z) \longleftrightarrow (Z = A+B)$ "
<proof>

```

```

lemma (in M_trivial) Inl_in_M_iff [iff]:
  " $M(\text{Inl}(a)) \longleftrightarrow M(a)$ "
<proof>

```

```

lemma (in M_trivial) Inl_abs [simp]:
  " $M(Z) \implies \text{is\_Inl}(M, a, Z) \longleftrightarrow (Z = \text{Inl}(a))$ "
<proof>

```

```

lemma (in M_trivial) Inr_in_M_iff [iff]:
  " $M(\text{Inr}(a)) \longleftrightarrow M(a)$ "
<proof>

```

```

lemma (in M_trivial) Inr_abs [simp]:
  " $M(Z) \implies \text{is\_Inr}(M, a, Z) \longleftrightarrow (Z = \text{Inr}(a))$ "
<proof>

```

### 2.7.1 converse of a relation

```

lemma (in M_basic) M_converse_iff:

```

```

    "M(r) ==>
    converse(r) =
    {z ∈ ∪ (∪ (r)) * ∪ (∪ (r)).
    ∃ p ∈ r. ∃ x[M]. ∃ y[M]. p = ⟨x,y⟩ ∧ z = ⟨y,x⟩}"
  <proof>

```

```

lemma (in M_basic) converse_closed [intro,simp]:
  "M(r) ==> M(converse(r))"
  <proof>

```

```

lemma (in M_basic) converse_abs [simp]:
  "⟦M(r); M(z)⟧ ==> is_converse(M,r,z) <=> z = converse(r)"
  <proof>

```

### 2.7.2 image, preimage, domain, range

```

lemma (in M_basic) image_closed [intro,simp]:
  "⟦M(A); M(r)⟧ ==> M(r-‘A)"
  <proof>

```

```

lemma (in M_basic) vimage_abs [simp]:
  "⟦M(r); M(A); M(z)⟧ ==> pre_image(M,r,A,z) <=> z = r-‘A"
  <proof>

```

```

lemma (in M_basic) vimage_closed [intro,simp]:
  "⟦M(A); M(r)⟧ ==> M(r-‘A)"
  <proof>

```

### 2.7.3 Domain, range and field

```

lemma (in M_trans) domain_abs [simp]:
  "⟦M(r); M(z)⟧ ==> is_domain(M,r,z) <=> z = domain(r)"
  <proof>

```

```

lemma (in M_basic) domain_closed [intro,simp]:
  "M(r) ==> M(domain(r))"
  <proof>

```

```

lemma (in M_trans) range_abs [simp]:
  "⟦M(r); M(z)⟧ ==> is_range(M,r,z) <=> z = range(r)"
  <proof>

```

```

lemma (in M_basic) range_closed [intro,simp]:
  "M(r) ==> M(range(r))"
  <proof>

```

```

lemma (in M_basic) field_abs [simp]:
  "⟦M(r); M(z)⟧ ==> is_field(M,r,z) <=> z = field(r)"
  <proof>

```

```

lemma (in M_basic) field_closed [intro,simp]:
  "M(r)  $\implies$  M(field(r))"
<proof>

```

#### 2.7.4 Relations, functions and application

```

lemma (in M_trans) relation_abs [simp]:
  "M(r)  $\implies$  is_relation(M,r)  $\longleftrightarrow$  relation(r)"
<proof>

```

```

lemma (in M_trivial) function_abs [simp]:
  "M(r)  $\implies$  is_function(M,r)  $\longleftrightarrow$  function(r)"
<proof>

```

```

lemma (in M_basic) apply_closed [intro,simp]:
  "M(f); M(a)  $\implies$  M(f'a)"
<proof>

```

```

lemma (in M_basic) apply_abs [simp]:
  "M(f); M(x); M(y)  $\implies$  fun_apply(M,f,x,y)  $\longleftrightarrow$  f'x = y"
<proof>

```

```

lemma (in M_trivial) typed_function_abs [simp]:
  "M(A); M(f)  $\implies$  typed_function(M,A,B,f)  $\longleftrightarrow$  f  $\in$  A  $\rightarrow$  B"
<proof>

```

```

lemma (in M_basic) injection_abs [simp]:
  "M(A); M(f)  $\implies$  injection(M,A,B,f)  $\longleftrightarrow$  f  $\in$  inj(A,B)"
<proof>

```

```

lemma (in M_basic) surjection_abs [simp]:
  "M(A); M(B); M(f)  $\implies$  surjection(M,A,B,f)  $\longleftrightarrow$  f  $\in$  surj(A,B)"
<proof>

```

```

lemma (in M_basic) bijection_abs [simp]:
  "M(A); M(B); M(f)  $\implies$  bijection(M,A,B,f)  $\longleftrightarrow$  f  $\in$  bij(A,B)"
<proof>

```

#### 2.7.5 Composition of relations

```

lemma (in M_basic) M_comp_iff:
  "M(r); M(s)
 $\implies$  r O s =
  {xz  $\in$  domain(s) * range(r).
 $\exists$  x[M].  $\exists$  y[M].  $\exists$  z[M]. xz = <x,z>  $\wedge$  <x,y>  $\in$  s  $\wedge$  <y,z>  $\in$  r}"
<proof>

```

```

lemma (in M_basic) comp_closed [intro,simp]:
  "M(r); M(s)  $\implies$  M(r O s)"
<proof>

```

```

lemma (in M_basic) composition_abs [simp]:
  "⟦M(r); M(s); M(t)⟧ ⇒ composition(M,r,s,t) ⟷ t = r ∘ s"
⟨proof⟩

no longer needed

lemma (in M_basic) restriction_is_function:
  "⟦restriction(M,f,A,z); function(f); M(f); M(A); M(z)⟧
  ⇒ function(z)"
⟨proof⟩

lemma (in M_trans) restriction_abs [simp]:
  "⟦M(f); M(A); M(z)⟧
  ⇒ restriction(M,f,A,z) ⟷ z = restrict(f,A)"
⟨proof⟩

lemma (in M_trans) M_restrict_iff:
  "M(r) ⇒ restrict(r,A) = {z ∈ r . ∃ x ∈ A. ∃ y[M]. z = ⟨x, y⟩}"
⟨proof⟩

lemma (in M_basic) restrict_closed [intro,simp]:
  "⟦M(A); M(r)⟧ ⇒ M(restrict(r,A))"
⟨proof⟩

lemma (in M_trans) Inter_abs [simp]:
  "⟦M(A); M(z)⟧ ⇒ big_inter(M,A,z) ⟷ z = ⋂ (A)"
⟨proof⟩

lemma (in M_basic) Inter_closed [intro,simp]:
  "M(A) ⇒ M(⋂ (A))"
⟨proof⟩

lemma (in M_basic) Int_closed [intro,simp]:
  "⟦M(A); M(B)⟧ ⇒ M(A ∩ B)"
⟨proof⟩

lemma (in M_basic) Diff_closed [intro,simp]:
  "⟦M(A); M(B)⟧ ⇒ M(A - B)"
⟨proof⟩

```

## 2.7.6 Some Facts About Separation Axioms

```

lemma (in M_basic) separation_conj:
  "⟦separation(M,P); separation(M,Q)⟧ ⇒ separation(M, λz. P(z) ∧
  Q(z))"
⟨proof⟩

```

**lemma** *Collect\_Un\_Collect\_eq*:  
 "Collect(A,P)  $\cup$  Collect(A,Q) = Collect(A,  $\lambda x. P(x) \vee Q(x)$ )"  
 <proof>

**lemma** *Diff\_Collect\_eq*:  
 "A - Collect(A,P) = Collect(A,  $\lambda x. \neg P(x)$ )"  
 <proof>

**lemma** (in *M\_trans*) *Collect\_rall\_eq*:  
 "M(Y)  $\implies$  Collect(A,  $\lambda x. \forall y[M]. y \in Y \longrightarrow P(x,y)$ ) =  
 (if Y=0 then A else ( $\bigcap_{y \in Y. \{x \in A. P(x,y)\}}$ ))"  
 <proof>

**lemma** (in *M\_basic*) *separation\_disj*:  
 "[separation(M,P); separation(M,Q)]  $\implies$  separation(M,  $\lambda z. P(z) \vee Q(z)$ )"  
 <proof>

**lemma** (in *M\_basic*) *separation\_neg*:  
 "separation(M,P)  $\implies$  separation(M,  $\lambda z. \neg P(z)$ )"  
 <proof>

**lemma** (in *M\_basic*) *separation\_imp*:  
 "[separation(M,P); separation(M,Q)]  
 $\implies$  separation(M,  $\lambda z. P(z) \longrightarrow Q(z)$ )"  
 <proof>

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*) *separation\_rall*:  
 "[M(Y);  $\forall y[M]. \text{separation}(M, \lambda x. P(x,y))$ ;  
 $\forall z[M]. \text{strong\_replacement}(M, \lambda x y. y = \{u \in z. P(u,x)\})$ ]  
 $\implies$  separation(M,  $\lambda x. \forall y[M]. y \in Y \longrightarrow P(x,y)$ )"  
 <proof>

## 2.7.7 Functions and function space

The assumption  $M(A \rightarrow B)$  is unusual, but essential: in all but trivial cases,  $A \rightarrow B$  cannot be expected to belong to  $M$ .

**lemma** (in *M\_trivial*) *is\_funspace\_abs [simp]*:  
 "[M(A); M(B); M(F); M(A  $\rightarrow$  B)]  $\implies$  is\_funspace(M,A,B,F)  $\longleftrightarrow$  F = A  $\rightarrow$  B"  
 <proof>

**lemma** (in *M\_basic*) *succ\_fun\_eq2*:  
 "[M(B); M(n  $\rightarrow$  B)]  $\implies$   
 succ(n)  $\rightarrow$  B =

$\bigcup \{z. p \in (n \rightarrow B) * B, \exists f[M]. \exists b[M]. p = \langle f, b \rangle \wedge z = \{ \text{cons}(\langle n, b \rangle, f) \} \}$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_basic$ ) *funspace\_succ*:  
 $\llbracket M(n); M(B); M(n \rightarrow B) \rrbracket \implies M(\text{succ}(n) \rightarrow B)$ "  
 $\langle \text{proof} \rangle$

$M$  contains all finite function spaces. Needed to prove the absoluteness of transitive closure. See the definition of *rtranc1\_alt* in *WF\_absolute.thy*.

**lemma** (in  $M\_basic$ ) *finite\_funspace\_closed* [intro,simp]:  
 $\llbracket n \in \text{nat}; M(B) \rrbracket \implies M(n \rightarrow B)$ "  
 $\langle \text{proof} \rangle$

## 2.8 Relativization and Absoluteness for Boolean Operators

**definition**

*is\_bool\_of\_o* :: "[ $i \Rightarrow o, o, i$ ]  $\Rightarrow o$ " where  
 $\text{is\_bool\_of\_o}(M, P, z) \equiv (P \wedge \text{number1}(M, z)) \vee (\neg P \wedge \text{empty}(M, z))$ "

**definition**

*is\_not* :: "[ $i \Rightarrow o, i, i$ ]  $\Rightarrow o$ " where  
 $\text{is\_not}(M, a, z) \equiv (\text{number1}(M, a) \wedge \text{empty}(M, z)) \mid$   
 $(\neg \text{number1}(M, a) \wedge \text{number1}(M, z))$ "

**definition**

*is\_and* :: "[ $i \Rightarrow o, i, i, i$ ]  $\Rightarrow o$ " where  
 $\text{is\_and}(M, a, b, z) \equiv (\text{number1}(M, a) \wedge z = b) \mid$   
 $(\neg \text{number1}(M, a) \wedge \text{empty}(M, z))$ "

**definition**

*is\_or* :: "[ $i \Rightarrow o, i, i, i$ ]  $\Rightarrow o$ " where  
 $\text{is\_or}(M, a, b, z) \equiv (\text{number1}(M, a) \wedge \text{number1}(M, z)) \mid$   
 $(\neg \text{number1}(M, a) \wedge z = b)$ "

**lemma** (in  $M\_trivial$ ) *bool\_of\_o\_abs* [simp]:  
 $M(z) \implies \text{is\_bool\_of\_o}(M, P, z) \longleftrightarrow z = \text{bool\_of\_o}(P)$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ ) *not\_abs* [simp]:  
 $\llbracket M(a); M(z) \rrbracket \implies \text{is\_not}(M, a, z) \longleftrightarrow z = \text{not}(a)$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ ) *and\_abs* [simp]:  
 $\llbracket M(a); M(b); M(z) \rrbracket \implies \text{is\_and}(M, a, b, z) \longleftrightarrow z = a \text{ and } b$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ ) *or\_abs* [simp]:



" $\llbracket M(a); M(b); M(z) \rrbracket \implies \text{is\_or}(M,a,b,z) \longleftrightarrow z = a \text{ or } b$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ )  $\text{bool\_of\_o\_closed}$  [intro,simp]:  
 " $M(\text{bool\_of\_o}(P))$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ )  $\text{and\_closed}$  [intro,simp]:  
 " $\llbracket M(p); M(q) \rrbracket \implies M(p \text{ and } q)$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ )  $\text{or\_closed}$  [intro,simp]:  
 " $\llbracket M(p); M(q) \rrbracket \implies M(p \text{ or } q)$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ )  $\text{not\_closed}$  [intro,simp]:  
 " $M(p) \implies M(\text{not}(p))$ "  
 $\langle \text{proof} \rangle$

## 2.9 Relativization and Absoluteness for List Operators

### definition

$\text{is\_Nil} :: "[i \Rightarrow o, i] \Rightarrow o$  where  
 — because  $[] \equiv \text{Inl}(0)$   
 " $\text{is\_Nil}(M, xs) \equiv \exists \text{zero}[M]. \text{empty}(M, \text{zero}) \wedge \text{is\_Inl}(M, \text{zero}, xs)$ "

### definition

$\text{is\_Cons} :: "[i \Rightarrow o, i, i, i] \Rightarrow o$  where  
 — because  $\text{Cons}(a, l) \equiv \text{Inr}(\langle a, l \rangle)$   
 " $\text{is\_Cons}(M, a, l, Z) \equiv \exists p[M]. \text{pair}(M, a, l, p) \wedge \text{is\_Inr}(M, p, Z)$ "

**lemma** (in  $M\_trivial$ )  $\text{Nil\_in\_M}$  [intro,simp]: " $M(\text{Nil})$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ )  $\text{Nil\_abs}$  [simp]: " $M(Z) \implies \text{is\_Nil}(M, Z) \longleftrightarrow (Z = \text{Nil})$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ )  $\text{Cons\_in\_M\_iff}$  [iff]: " $M(\text{Cons}(a, l)) \longleftrightarrow M(a) \wedge M(l)$ "  
 $\langle \text{proof} \rangle$

**lemma** (in  $M\_trivial$ )  $\text{Cons\_abs}$  [simp]:  
 " $\llbracket M(a); M(l); M(Z) \rrbracket \implies \text{is\_Cons}(M, a, l, Z) \longleftrightarrow (Z = \text{Cons}(a, l))$ "  
 $\langle \text{proof} \rangle$

### definition

```

quaselist :: "i  $\Rightarrow$  o" where
  "quaselist(xs)  $\equiv$  xs=Nil  $\vee$  ( $\exists$  x l. xs = Cons(x,l))"

```

**definition**

```

is_quaselist :: "[i $\Rightarrow$ o,i]  $\Rightarrow$  o" where
  "is_quaselist(M,z)  $\equiv$  is_Nil(M,z)  $\vee$  ( $\exists$  x[M].  $\exists$  l[M]. is_Cons(M,x,l,z))"

```

**definition**

```

list_case' :: "[i, [i,i] $\Rightarrow$ i, i]  $\Rightarrow$  i" where
  — A version of list_case that's always defined.
  "list_case'(a,b,xs)  $\equiv$ 
    if quaselist(xs) then list_case(a,b,xs) else 0"

```

**definition**

```

is_list_case :: "[i $\Rightarrow$ o, i, [i,i,i] $\Rightarrow$ o, i, i]  $\Rightarrow$  o" where
  — Returns 0 for non-lists
  "is_list_case(M, a, is_b, xs, z)  $\equiv$ 
    (is_Nil(M,xs)  $\longrightarrow$  z=a)  $\wedge$ 
    ( $\forall$  x[M].  $\forall$  l[M]. is_Cons(M,x,l,xs)  $\longrightarrow$  is_b(x,l,z))  $\wedge$ 
    (is_quaselist(M,xs)  $\vee$  empty(M,z))"

```

**definition**

```

hd' :: "i  $\Rightarrow$  i" where
  — A version of hd that's always defined.
  "hd'(xs)  $\equiv$  if quaselist(xs) then hd(xs) else 0"

```

**definition**

```

tl' :: "i  $\Rightarrow$  i" where
  — A version of tl that's always defined.
  "tl'(xs)  $\equiv$  if quaselist(xs) then tl(xs) else 0"

```

**definition**

```

is_hd :: "[i $\Rightarrow$ o,i,i]  $\Rightarrow$  o" where
  — hd([]) = 0 no constraints if not a list. Avoiding implication prevents the
  simplifier's looping.

```

```

  "is_hd(M,xs,H)  $\equiv$ 
    (is_Nil(M,xs)  $\longrightarrow$  empty(M,H))  $\wedge$ 
    ( $\forall$  x[M].  $\forall$  l[M].  $\neg$  is_Cons(M,x,l,xs)  $\vee$  H=x)  $\wedge$ 
    (is_quaselist(M,xs)  $\vee$  empty(M,H))"

```

**definition**

```

is_tl :: "[i $\Rightarrow$ o,i,i]  $\Rightarrow$  o" where
  — tl([]) = []; see comments about is_hd
  "is_tl(M,xs,T)  $\equiv$ 
    (is_Nil(M,xs)  $\longrightarrow$  T=xs)  $\wedge$ 
    ( $\forall$  x[M].  $\forall$  l[M].  $\neg$  is_Cons(M,x,l,xs)  $\vee$  T=l)  $\wedge$ 
    (is_quaselist(M,xs)  $\vee$  empty(M,T))"

```

### 2.9.1 *quaselist*: For Case-Splitting with *list\_case'*

**lemma** *[iff]*: "*quaselist*(Nil)"  
*<proof>*

**lemma** *[iff]*: "*quaselist*(Cons(*x*,*l*))"  
*<proof>*

**lemma** *list\_imp\_quaselist*: "*l* ∈ *list*(*A*) ⇒ *quaselist*(*l*)"  
*<proof>*

### 2.9.2 *list\_case'*, the Modified Version of *list\_case*

**lemma** *list\_case'\_Nil* *[simp]*: "*list\_case'*(*a*,*b*,Nil) = *a*"  
*<proof>*

**lemma** *list\_case'\_Cons* *[simp]*: "*list\_case'*(*a*,*b*,Cons(*x*,*l*)) = *b*(*x*,*l*)"  
*<proof>*

**lemma** *non\_list\_case*: " $\neg$  *quaselist*(*x*) ⇒ *list\_case'*(*a*,*b*,*x*) = 0"  
*<proof>*

**lemma** *list\_case'\_eq\_list\_case* *[simp]*:  
 "*xs* ∈ *list*(*A*) ⇒ *list\_case'*(*a*,*b*,*xs*) = *list\_case*(*a*,*b*,*xs*)"  
*<proof>*

**lemma** (in *M\_basic*) *list\_case'\_closed* *[intro,simp]*:  
 " $\llbracket M(k); M(a); \forall x[M]. \forall y[M]. M(b(x,y)) \rrbracket \Rightarrow M(\text{list\_case'}(a,b,k))$ "  
*<proof>*

**lemma** (in *M\_trivial*) *quaselist\_abs* *[simp]*:  
 "*M*(*z*) ⇒ *is\_quaselist*(*M*,*z*) ⇔ *quaselist*(*z*)"  
*<proof>*

**lemma** (in *M\_trivial*) *list\_case\_abs* *[simp]*:  
 " $\llbracket \text{relation2}(M, \text{is\_b}, b); M(k); M(z) \rrbracket$   
 ⇒ *is\_list\_case*(*M*,*a*,*is\_b*,*k*,*z*) ⇔ *z* = *list\_case'*(*a*,*b*,*k*)"  
*<proof>*

### 2.9.3 The Modified Operators *hd'* and *tl'*

**lemma** (in *M\_trivial*) *is\_hd\_Nil*: "*is\_hd*(*M*, [], *Z*) ⇔ *empty*(*M*, *Z*)"  
*<proof>*

**lemma** (in *M\_trivial*) *is\_hd\_Cons*:  
 " $\llbracket M(a); M(l) \rrbracket \Rightarrow \text{is\_hd}(M, \text{Cons}(a, l), Z) \Leftrightarrow Z = a$ "  
*<proof>*

**lemma** (in *M\_trivial*) *hd\_abs* *[simp]*:  
 " $\llbracket M(x); M(y) \rrbracket \Rightarrow \text{is\_hd}(M, x, y) \Leftrightarrow y = \text{hd'}(x)$ "

*<proof>*

**lemma** (in *M\_trivial*) *is\_tl\_Nil*: " $is\_tl(M, [], Z) \longleftrightarrow Z = []$ "  
*<proof>*

**lemma** (in *M\_trivial*) *is\_tl\_Cons*:  
" $\llbracket M(a); M(l) \rrbracket \implies is\_tl(M, Cons(a, l), Z) \longleftrightarrow Z = l$ "  
*<proof>*

**lemma** (in *M\_trivial*) *tl\_abs [simp]*:  
" $\llbracket M(x); M(y) \rrbracket \implies is\_tl(M, x, y) \longleftrightarrow y = tl'(x)$ "  
*<proof>*

**lemma** (in *M\_trivial*) *relation1\_tl*: " $relation1(M, is\_tl(M), tl')$ "  
*<proof>*

**lemma** *hd'\_Nil*: " $hd'([]) = 0$ "  
*<proof>*

**lemma** *hd'\_Cons*: " $hd'(Cons(a, l)) = a$ "  
*<proof>*

**lemma** *tl'\_Nil*: " $tl'([]) = []$ "  
*<proof>*

**lemma** *tl'\_Cons*: " $tl'(Cons(a, l)) = l$ "  
*<proof>*

**lemma** *iterates\_tl\_Nil*: " $n \in nat \implies tl'^{\sim n}([]) = []$ "  
*<proof>*

**lemma** (in *M\_basic*) *tl'\_closed*: " $M(x) \implies M(tl'(x))$ "  
*<proof>*

**end**

### 3 Relativized Wellorderings

**theory** *Wellorderings* imports *Relative* begin

We define functions analogous to *ordermap ordertype* but without using recursion. Instead, there is a direct appeal to Replacement. This will be the basis for a version relativized to some class *M*. The main result is Theorem I 7.6 in Kunen, page 17.

#### 3.1 Wellorderings

**definition**

**irreflexive** :: "[i⇒o,i,i]⇒o" where  
 "irreflexive(M,A,r) ≡ ∀x[M]. x∈A → ⟨x,x⟩ ∉ r"

**definition**

**transitive\_rel** :: "[i⇒o,i,i]⇒o" where  
 "transitive\_rel(M,A,r) ≡  
 ∀x[M]. x∈A → (∀y[M]. y∈A → (∀z[M]. z∈A →  
 ⟨x,y⟩∈r → ⟨y,z⟩∈r → ⟨x,z⟩∈r))"

**definition**

**linear\_rel** :: "[i⇒o,i,i]⇒o" where  
 "linear\_rel(M,A,r) ≡  
 ∀x[M]. x∈A → (∀y[M]. y∈A → ⟨x,y⟩∈r ∨ x=y ∨ ⟨y,x⟩∈r)"

**definition**

**wellfounded** :: "[i⇒o,i,i]⇒o" where  
 — EVERY non-empty set has an *r*-minimal element  
 "wellfounded(M,r) ≡  
 ∀x[M]. x≠0 → (∃y[M]. y∈x ∧ ¬(∃z[M]. z∈x ∧ ⟨z,y⟩ ∈ r))"

**definition**

**wellfounded\_on** :: "[i⇒o,i,i]⇒o" where  
 — every non-empty SUBSET OF A has an *r*-minimal element  
 "wellfounded\_on(M,A,r) ≡  
 ∀x[M]. x≠0 → x⊆A → (∃y[M]. y∈x ∧ ¬(∃z[M]. z∈x ∧ ⟨z,y⟩  
 ∈ r))"

**definition**

**wellordered** :: "[i⇒o,i,i]⇒o" where  
 — linear and wellfounded on A  
 "wellordered(M,A,r) ≡  
 transitive\_rel(M,A,r) ∧ linear\_rel(M,A,r) ∧ wellfounded\_on(M,A,r)"

### 3.1.1 Trivial absoluteness proofs

**lemma** (in *M\_basic*) **irreflexive\_abs** [simp]:  
 "M(A) ⇒ irreflexive(M,A,r) ↔ irrefl(A,r)"  
 ⟨proof⟩

**lemma** (in *M\_basic*) **transitive\_rel\_abs** [simp]:  
 "M(A) ⇒ transitive\_rel(M,A,r) ↔ trans[A](r)"  
 ⟨proof⟩

**lemma** (in *M\_basic*) **linear\_rel\_abs** [simp]:  
 "M(A) ⇒ linear\_rel(M,A,r) ↔ linear(A,r)"  
 ⟨proof⟩

**lemma** (in *M\_basic*) **wellordered\_is\_trans\_on**:  
 "⟦wellordered(M,A,r); M(A)⟧ ⇒ trans[A](r)"  
 ⟨proof⟩

```

lemma (in M_basic) wellordered_is_linear:
  "⟦wellordered(M,A,r); M(A)⟧  $\implies$  linear(A,r)"
  <proof>

lemma (in M_basic) wellordered_is_wellfounded_on:
  "⟦wellordered(M,A,r); M(A)⟧  $\implies$  wellfounded_on(M,A,r)"
  <proof>

lemma (in M_basic) wellfounded_imp_wellfounded_on:
  "⟦wellfounded(M,r); M(A)⟧  $\implies$  wellfounded_on(M,A,r)"
  <proof>

lemma (in M_basic) wellfounded_on_subset_A:
  "⟦wellfounded_on(M,A,r); B $\leq$ A⟧  $\implies$  wellfounded_on(M,B,r)"
  <proof>

```

### 3.1.2 Well-founded relations

```

lemma (in M_basic) wellfounded_on_iff_wellfounded:
  "wellfounded_on(M,A,r)  $\longleftrightarrow$  wellfounded(M, r  $\cap$  A*A)"
  <proof>

lemma (in M_basic) wellfounded_on_imp_wellfounded:
  "⟦wellfounded_on(M,A,r); r  $\subseteq$  A*A⟧  $\implies$  wellfounded(M,r)"
  <proof>

lemma (in M_basic) wellfounded_on_field_imp_wellfounded:
  "wellfounded_on(M, field(r), r)  $\implies$  wellfounded(M,r)"
  <proof>

lemma (in M_basic) wellfounded_iff_wellfounded_on_field:
  "M(r)  $\implies$  wellfounded(M,r)  $\longleftrightarrow$  wellfounded_on(M, field(r), r)"
  <proof>

lemma (in M_basic) wellfounded_induct:
  "⟦wellfounded(M,r); M(a); M(r); separation(M,  $\lambda x. \neg P(x)$ );
     $\forall x. M(x) \wedge (\forall y. \langle y,x \rangle \in r \longrightarrow P(y)) \longrightarrow P(x)$ ⟧
     $\implies P(a)$ "
  <proof>

lemma (in M_basic) wellfounded_on_induct:
  "⟦a $\in$ A; wellfounded_on(M,A,r); M(A);
    separation(M,  $\lambda x. x\in A \longrightarrow \neg P(x)$ );
     $\forall x\in A. M(x) \wedge (\forall y\in A. \langle y,x \rangle \in r \longrightarrow P(y)) \longrightarrow P(x)$ ⟧
     $\implies P(a)$ "
  <proof>

```

### 3.1.3 Kunen's lemma IV 3.14, page 123

```
lemma (in M_basic) linear_imp_relativized:
  "linear(A,r)  $\implies$  linear_rel(M,A,r)"
<proof>
```

```
lemma (in M_basic) trans_on_imp_relativized:
  "trans[A](r)  $\implies$  transitive_rel(M,A,r)"
<proof>
```

```
lemma (in M_basic) wf_on_imp_relativized:
  "wf[A](r)  $\implies$  wellfounded_on(M,A,r)"
<proof>
```

```
lemma (in M_basic) wf_imp_relativized:
  "wf(r)  $\implies$  wellfounded(M,r)"
<proof>
```

```
lemma (in M_basic) well_ord_imp_relativized:
  "well_ord(A,r)  $\implies$  wellordered(M,A,r)"
<proof>
```

The property being well founded (and hence of being well ordered) is not absolute: the set that doesn't contain a minimal element may not exist in the class M. However, every set that is well founded in a transitive model M is well founded (page 124).

## 3.2 Relativized versions of order-isomorphisms and order types

```
lemma (in M_basic) order_isomorphism_abs [simp]:
  "[M(A); M(B); M(f)]
 $\implies$  order_isomorphism(M,A,r,B,s,f)  $\longleftrightarrow$  f  $\in$  ord_iso(A,r,B,s)"
<proof>
```

```
lemma (in M_trans) pred_set_abs [simp]:
  "[M(r); M(B)]  $\implies$  pred_set(M,A,x,r,B)  $\longleftrightarrow$  B = Order.pred(A,x,r)"
<proof>
```

```
lemma (in M_basic) pred_closed [intro,simp]:
  "[M(A); M(r); M(x)]  $\implies$  M(Order.pred(A, x, r))"
<proof>
```

```
lemma (in M_basic) membership_abs [simp]:
  "[M(r); M(A)]  $\implies$  membership(M,A,r)  $\longleftrightarrow$  r = Memrel(A)"
<proof>
```

```
lemma (in M_basic) M_Memrel_iff:
  "M(A)  $\implies$  Memrel(A) = {z  $\in$  A*A.  $\exists$  x[M].  $\exists$  y[M]. z = <x,y>  $\wedge$  x  $\in$  y}"
```

$\langle proof \rangle$

```
lemma (in M_basic) Memrel_closed [intro,simp]:  
  "M(A)  $\implies$  M(Memrel(A))"  
 $\langle proof \rangle$ 
```

### 3.3 Main results of Kunen, Chapter 1 section 6

Subset properties– proved outside the locale

```
lemma linear_rel_subset:  
  "[linear_rel(M, A, r); B  $\subseteq$  A]  $\implies$  linear_rel(M, B, r)"  
 $\langle proof \rangle$ 
```

```
lemma transitive_rel_subset:  
  "[transitive_rel(M, A, r); B  $\subseteq$  A]  $\implies$  transitive_rel(M, B, r)"  
 $\langle proof \rangle$ 
```

```
lemma wellfounded_on_subset:  
  "[wellfounded_on(M, A, r); B  $\subseteq$  A]  $\implies$  wellfounded_on(M, B, r)"  
 $\langle proof \rangle$ 
```

```
lemma wellordered_subset:  
  "[wellordered(M, A, r); B  $\subseteq$  A]  $\implies$  wellordered(M, B, r)"  
 $\langle proof \rangle$ 
```

```
lemma (in M_basic) wellfounded_on_asym:  
  "[wellfounded_on(M,A,r);  $\langle a,x \rangle \in r$ ; a  $\in$  A; x  $\in$  A; M(A)]  $\implies$   $\langle x,a \rangle \notin r$ "  
 $\langle proof \rangle$ 
```

```
lemma (in M_basic) wellordered_asym:  
  "[wellordered(M,A,r);  $\langle a,x \rangle \in r$ ; a  $\in$  A; x  $\in$  A; M(A)]  $\implies$   $\langle x,a \rangle \notin r$ "  
 $\langle proof \rangle$ 
```

end

## 4 Relativized Well-Founded Recursion

theory WFreq imports Wellorderings begin

### 4.1 General Lemmas

```
lemma apply_recfun2:  
  "[is_recfun(r,a,H,f);  $\langle x,i \rangle : f$ ]  $\implies$  i = H(x, restrict(f,r- $\{x\}$ ))"  
 $\langle proof \rangle$ 
```

Expresses *is\_recfun* as a recursion equation

```
lemma is_recfun_iff_equation:  
  "is_recfun(r,a,H,f)  $\longleftrightarrow$ 
```



$$f \in r - \{a\} \rightarrow \text{range}(f) \wedge$$

$$(\forall x \in r - \{a\}. f'x = H(x, \text{restrict}(f, r - \{x\})))$$

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

**lemma** *is\_recfun\_imp\_in\_r*: " $\llbracket \text{is\_recfun}(r, a, H, f); \langle x, i \rangle \in f \rrbracket \implies \langle x, a \rangle \in r$ "  

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

**lemma** *trans\_Int\_eq*:  

$$\llbracket \text{trans}(r); \langle y, x \rangle \in r \rrbracket \implies r - \{x\} \cap r - \{y\} = r - \{y\}$$

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

**lemma** *is\_recfun\_restrict\_idem*:  

$$\text{is\_recfun}(r, a, H, f) \implies \text{restrict}(f, r - \{a\}) = f$$

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

**lemma** *is\_recfun\_cong\_lemma*:  

$$\llbracket \text{is\_recfun}(r, a, H, f); r = r'; a = a'; f = f';$$

$$\bigwedge x g. \llbracket \langle x, a' \rangle \in r'; \text{relation}(g); \text{domain}(g) \subseteq r' - \{x\} \rrbracket$$

$$\implies H(x, g) = H'(x, g) \rrbracket$$

$$\implies \text{is\_recfun}(r', a', H', f')$$

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

For *is\_recfun* we need only pay attention to functions whose domains are initial segments of *r*.

**lemma** *is\_recfun\_cong*:  

$$\llbracket r = r'; a = a'; f = f';$$

$$\bigwedge x g. \llbracket \langle x, a' \rangle \in r'; \text{relation}(g); \text{domain}(g) \subseteq r' - \{x\} \rrbracket$$

$$\implies H(x, g) = H'(x, g) \rrbracket$$

$$\implies \text{is\_recfun}(r, a, H, f) \longleftrightarrow \text{is\_recfun}(r', a', H', f')$$

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

## 4.2 Reworking of the Recursion Theory Within $\mathcal{M}$

**lemma** (in *M\_basic*) *is\_recfun\_separation'*:  

$$\llbracket f \in r - \{a\} \rightarrow \text{range}(f); g \in r - \{b\} \rightarrow \text{range}(g);$$

$$M(r); M(f); M(g); M(a); M(b) \rrbracket$$

$$\implies \text{separation}(M, \lambda x. \neg (\langle x, a \rangle \in r \longrightarrow \langle x, b \rangle \in r \longrightarrow f'x = g'x))$$

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

Stated using *trans*(*r*) rather than *transitive\_rel*(*M*, *A*, *r*) because the latter rewrites to the former anyway, by *transitive\_rel\_abs*. As always, theorems should be expressed in simplified form. The last three *M*-premises are redundant because of *M*(*r*), but without them we'd have to undertake more work to set up the induction formula.

**lemma** (in *M\_basic*) *is\_recfun\_equal* [*rule\_format*]:  

$$\llbracket \text{is\_recfun}(r, a, H, f); \text{is\_recfun}(r, b, H, g);$$

$$\begin{aligned} & \text{wellfounded}(M,r); \text{trans}(r); \\ & M(f); M(g); M(r); M(x); M(a); M(b) \parallel \\ \implies & \langle x,a \rangle \in r \longrightarrow \langle x,b \rangle \in r \longrightarrow f'x=g'x'' \\ \langle \text{proof} \rangle & \end{aligned}$$

**lemma** (in  $M\_basic$ )  $is\_recfun\_cut$ :  

$$\begin{aligned} & "[[is\_recfun(r,a,H,f); is\_recfun(r,b,H,g); \\ & \text{wellfounded}(M,r); \text{trans}(r); \\ & M(f); M(g); M(r); \langle b,a \rangle \in r]] \\ \implies & \text{restrict}(f, r - \langle \{b\} \rangle) = g'' \\ \langle \text{proof} \rangle & \end{aligned}$$

**lemma** (in  $M\_basic$ )  $is\_recfun\_functional$ :  

$$\begin{aligned} & "[[is\_recfun(r,a,H,f); is\_recfun(r,a,H,g); \\ & \text{wellfounded}(M,r); \text{trans}(r); M(f); M(g); M(r)]] \implies f=g'' \\ \langle \text{proof} \rangle & \end{aligned}$$

Tells us that  $is\_recfun$  can (in principle) be relativized.

**lemma** (in  $M\_basic$ )  $is\_recfun\_relativize$ :  

$$\begin{aligned} & "[[M(r); M(f); \forall x[M]. \forall g[M]. \text{function}(g) \longrightarrow M(H(x,g))]] \\ \implies & is\_recfun(r,a,H,f) \longleftrightarrow \\ & (\forall z[M]. z \in f \longleftrightarrow \\ & (\exists x[M]. \langle x,a \rangle \in r \wedge z = \langle x, H(x, \text{restrict}(f, r - \langle \{x\} \rangle)) \rangle))'' \\ \langle \text{proof} \rangle & \end{aligned}$$

**lemma** (in  $M\_basic$ )  $is\_recfun\_restrict$ :  

$$\begin{aligned} & "[[\text{wellfounded}(M,r); \text{trans}(r); is\_recfun(r,x,H,f); \langle y,x \rangle \in r; \\ & M(r); M(f); \\ & \forall x[M]. \forall g[M]. \text{function}(g) \longrightarrow M(H(x,g))]] \\ \implies & is\_recfun(r, y, H, \text{restrict}(f, r - \langle \{y\} \rangle))'' \\ \langle \text{proof} \rangle & \end{aligned}$$

**lemma** (in  $M\_basic$ )  $restrict\_Y\_lemma$ :  

$$\begin{aligned} & "[[\text{wellfounded}(M,r); \text{trans}(r); M(r); \\ & \forall x[M]. \forall g[M]. \text{function}(g) \longrightarrow M(H(x,g)); M(Y); \\ & \forall b[M]. \\ & b \in Y \longleftrightarrow \\ & (\exists x[M]. \langle x,a1 \rangle \in r \wedge \\ & (\exists y[M]. b = \langle x,y \rangle \wedge (\exists g[M]. is\_recfun(r,x,H,g) \wedge y = H(x,g))))); \\ & \langle x,a1 \rangle \in r; is\_recfun(r,x,H,f); M(f)]] \\ \implies & \text{restrict}(Y, r - \langle \{x\} \rangle) = f'' \\ \langle \text{proof} \rangle & \end{aligned}$$

For typical applications of Replacement for recursive definitions

**lemma** (in  $M\_basic$ )  $univalent\_is\_recfun$ :  

$$\begin{aligned} & "[[\text{wellfounded}(M,r); \text{trans}(r); M(r)] \\ \implies & \text{univalent}(M, A, \lambda x p. \\ & \exists y[M]. p = \langle x,y \rangle \wedge (\exists f[M]. is\_recfun(r,x,H,f) \wedge y = H(x,f)))'' \\ \langle \text{proof} \rangle & \end{aligned}$$

Proof of the inductive step for `exists_is_recfun`, since we must prove two versions.

**lemma** (in `M_basic`) `exists_is_recfun_indstep`:  

$$\begin{aligned} & \llbracket \forall y. \langle y, a1 \rangle \in r \longrightarrow (\exists f[M]. \text{is\_recfun}(r, y, H, f)); \\ & \quad \text{wellfounded}(M, r); \text{trans}(r); M(r); M(a1); \\ & \quad \text{strong\_replacement}(M, \lambda x z. \\ & \quad \quad \exists y[M]. \exists g[M]. \text{pair}(M, x, y, z) \wedge \text{is\_recfun}(r, x, H, g) \wedge y = \\ & \quad H(x, g)); \\ & \quad \forall x[M]. \forall g[M]. \text{function}(g) \longrightarrow M(H(x, g)) \rrbracket \\ & \implies \exists f[M]. \text{is\_recfun}(r, a1, H, f)'' \\ & \langle \text{proof} \rangle \end{aligned}$$

Relativized version, when we have the (currently weaker) premise `wellfounded`(`M`, `r`)

**lemma** (in `M_basic`) `wellfounded_exists_is_recfun`:  

$$\begin{aligned} & \llbracket \text{wellfounded}(M, r); \text{trans}(r); \\ & \quad \text{separation}(M, \lambda x. \neg (\exists f[M]. \text{is\_recfun}(r, x, H, f))); \\ & \quad \text{strong\_replacement}(M, \lambda x z. \\ & \quad \quad \exists y[M]. \exists g[M]. \text{pair}(M, x, y, z) \wedge \text{is\_recfun}(r, x, H, g) \wedge y = H(x, g)); \\ & \quad M(r); M(a); \\ & \quad \forall x[M]. \forall g[M]. \text{function}(g) \longrightarrow M(H(x, g)) \rrbracket \\ & \implies \exists f[M]. \text{is\_recfun}(r, a, H, f)'' \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** (in `M_basic`) `wf_exists_is_recfun [rule_format]`:  

$$\begin{aligned} & \llbracket \text{wf}(r); \text{trans}(r); M(r); \\ & \quad \text{strong\_replacement}(M, \lambda x z. \\ & \quad \quad \exists y[M]. \exists g[M]. \text{pair}(M, x, y, z) \wedge \text{is\_recfun}(r, x, H, g) \wedge y = H(x, g)); \\ & \quad \forall x[M]. \forall g[M]. \text{function}(g) \longrightarrow M(H(x, g)) \rrbracket \\ & \implies M(a) \longrightarrow (\exists f[M]. \text{is\_recfun}(r, a, H, f))'' \\ & \langle \text{proof} \rangle \end{aligned}$$

### 4.3 Relativization of the ZF Predicate `is_recfun`

**definition**

$M_{\text{is\_recfun}} :: "[i \Rightarrow o, [i, i, i] \Rightarrow o, i, i, i] \Rightarrow o"$  where  

$$\begin{aligned} & "M_{\text{is\_recfun}}(M, MH, r, a, f) \equiv \\ & \quad \forall z[M]. z \in f \longleftrightarrow \\ & \quad (\exists x[M]. \exists y[M]. \exists xa[M]. \exists sx[M]. \exists r\_sx[M]. \exists f\_r\_sx[M]. \\ & \quad \quad \text{pair}(M, x, y, z) \wedge \text{pair}(M, x, a, xa) \wedge \text{upair}(M, x, x, sx) \wedge \\ & \quad \quad \text{pre\_image}(M, r, sx, r\_sx) \wedge \text{restriction}(M, f, r\_sx, f\_r\_sx) \wedge \\ & \quad \quad xa \in r \wedge MH(x, f\_r\_sx, y))'' \end{aligned}$$

**definition**

$\text{is\_wfrec} :: "[i \Rightarrow o, [i, i, i] \Rightarrow o, i, i, i] \Rightarrow o"$  where  

$$\begin{aligned} & "is\_wfrec(M, MH, r, a, z) \equiv \\ & \quad \exists f[M]. M_{\text{is\_recfun}}(M, MH, r, a, f) \wedge MH(a, f, z)'' \end{aligned}$$

**definition**

```
wfrec_replacement :: "[i⇒o, [i,i,i]⇒o, i] ⇒ o" where
  "wfrec_replacement(M,MH,r) ≡
    strong_replacement(M,
      λx z. ∃y[M]. pair(M,x,y,z) ∧ is_wfrec(M,MH,r,x,y))"
```

**lemma** (in M\_basic) is\_recfun\_abs:

```
"[∀x[M]. ∀g[M]. function(g) ⟶ M(H(x,g)); M(r); M(a); M(f);
  relation2(M,MH,H)]
  ⟹ M_is_recfun(M,MH,r,a,f) ⟷ is_recfun(r,a,H,f)"
⟨proof⟩
```

**lemma** M\_is\_recfun\_cong [cong]:

```
"[r = r'; a = a'; f = f';
  ∧x g y. [M(x); M(g); M(y)] ⟹ MH(x,g,y) ⟷ MH'(x,g,y)]
  ⟹ M_is_recfun(M,MH,r,a,f) ⟷ M_is_recfun(M,MH',r',a',f')"
⟨proof⟩
```

**lemma** (in M\_basic) is\_wfrec\_abs:

```
"[∀x[M]. ∀g[M]. function(g) ⟶ M(H(x,g));
  relation2(M,MH,H); M(r); M(a); M(z)]
  ⟹ is_wfrec(M,MH,r,a,z) ⟷
    (∃g[M]. is_recfun(r,a,H,g) ∧ z = H(a,g))"
⟨proof⟩
```

Relating wfrec\_replacement to native constructs

**lemma** (in M\_basic) wfrec\_replacement':

```
"[wfrec_replacement(M,MH,r);
  ∀x[M]. ∀g[M]. function(g) ⟶ M(H(x,g));
  relation2(M,MH,H); M(r)]
  ⟹ strong_replacement(M, λx z. ∃y[M].
    pair(M,x,y,z) ∧ (∃g[M]. is_recfun(r,x,H,g) ∧ y = H(x,g)))"
⟨proof⟩
```

**lemma** wfrec\_replacement\_cong [cong]:

```
"[∧x y z. [M(x); M(y); M(z)] ⟹ MH(x,y,z) ⟷ MH'(x,y,z);
  r=r']
  ⟹ wfrec_replacement(M, λx y. MH(x,y), r) ⟷
    wfrec_replacement(M, λx y. MH'(x,y), r')"
⟨proof⟩
```

**end**

## 5 Absoluteness of Well-Founded Recursion

**theory** WF\_absolute imports WFrec begin

## 5.1 Transitive closure without fixedpoints

definition

```
rtranc1_alt :: "[i,i]⇒i" where
  "rtranc1_alt(A,r) ≡
    {p ∈ A*A. ∃n∈nat. ∃f ∈ succ(n) -> A.
      (∃x y. p = ⟨x,y⟩ ∧ f'0 = x ∧ f'n = y) ∧
      (∀i∈n. ⟨f'i, f'succ(i)⟩ ∈ r)}"
```

lemma alt\_rtranc1\_lemma1 [rule\_format]:

```
"n ∈ nat
  ⇒ ∀f ∈ succ(n) -> field(r).
    (∀i∈n. ⟨f'i, f'succ(i)⟩ ∈ r) → ⟨f'0, f'n⟩ ∈ r^*"
⟨proof⟩
```

lemma rtranc1\_alt\_subset\_rtranc1: "rtranc1\_alt(field(r),r) ⊆ r^\*"

⟨proof⟩

lemma rtranc1\_subset\_rtranc1\_alt: "r^\* ⊆ rtranc1\_alt(field(r),r)"

⟨proof⟩

lemma rtranc1\_alt\_eq\_rtranc1: "rtranc1\_alt(field(r),r) = r^\*"

⟨proof⟩

definition

```
rtran_closure_mem :: "[i⇒o,i,i,i] ⇒ o" where
  — The property of belonging to rtran_closure(r)
  "rtran_closure_mem(M,A,r,p) ≡
    ∃nnat[M]. ∃n[M]. ∃n'[M].
      omega(M,nnat) ∧ n∈nnat ∧ successor(M,n,n') ∧
      (∃f[M]. typed_function(M,n',A,f) ∧
      (∃x[M]. ∃y[M]. ∃zero[M]. pair(M,x,y,p) ∧ empty(M,zero)
      ∧
      fun_apply(M,f,zero,x) ∧ fun_apply(M,f,n,y)) ∧
      (∀j[M]. j∈n →
      (∃fj[M]. ∃sj[M]. ∃fsj[M]. ∃ffp[M].
      fun_apply(M,f,j,fj) ∧ successor(M,j,sj) ∧
      fun_apply(M,f,sj,fsj) ∧ pair(M,fj,fsj,ffp) ∧ ffp
      ∈ r)))"
```

definition

```
rtran_closure :: "[i⇒o,i,i] ⇒ o" where
  "rtran_closure(M,r,s) ≡
    ∀A[M]. is_field(M,r,A) →
    (∀p[M]. p ∈ s ↔ rtran_closure_mem(M,A,r,p))"
```

definition

```
tran_closure :: "[i⇒o,i,i] ⇒ o" where
  "tran_closure(M,r,t) ≡
```

```

       $\exists s[M]. \text{rtran\_closure}(M, r, s) \wedge \text{composition}(M, r, s, t)$ "

locale  $M\_tranc1 = M\_basic +$ 
  assumes  $\text{rtranc1\_separation}$ :
    " $\llbracket M(r); M(A) \rrbracket \implies \text{separation}(M, \text{rtran\_closure\_mem}(M, A, r))$ "
  and  $\text{wellfounded\_tranc1\_separation}$ :
    " $\llbracket M(r); M(Z) \rrbracket \implies$ 
       $\text{separation}(M, \lambda x.$ 
         $\exists w[M]. \exists wx[M]. \exists rp[M].$ 
         $w \in Z \wedge \text{pair}(M, w, x, wx) \wedge \text{tran\_closure}(M, r, rp) \wedge wx \in$ 
         $rp)$ "
    and  $M\_nat \text{ [iff] : } "M(\text{nat})"$ 

lemma (in  $M\_tranc1$ )  $\text{rtran\_closure\_mem\_iff}$ :
  " $\llbracket M(A); M(r); M(p) \rrbracket$ 
 $\implies \text{rtran\_closure\_mem}(M, A, r, p) \longleftrightarrow$ 
  ( $\exists n[M]. n \in \text{nat} \wedge$ 
    ( $\exists f[M]. f \in \text{succ}(n) \rightarrow A \wedge$ 
      ( $\exists x[M]. \exists y[M]. p = \langle x, y \rangle \wedge f'0 = x \wedge f'n = y \wedge$ 
        ( $\forall i \in n. \langle f'i, f'\text{succ}(i) \rangle \in r$ )))"
   $\langle \text{proof} \rangle$ 

lemma (in  $M\_tranc1$ )  $\text{rtran\_closure\_rtranc1}$ :
  " $M(r) \implies \text{rtran\_closure}(M, r, \text{rtranc1}(r))$ "
   $\langle \text{proof} \rangle$ 

lemma (in  $M\_tranc1$ )  $\text{rtranc1\_closed [intro, simp]}$ :
  " $M(r) \implies M(\text{rtranc1}(r))$ "
   $\langle \text{proof} \rangle$ 

lemma (in  $M\_tranc1$ )  $\text{rtranc1\_abs [simp]}$ :
  " $\llbracket M(r); M(z) \rrbracket \implies \text{rtran\_closure}(M, r, z) \longleftrightarrow z = \text{rtranc1}(r)$ "
   $\langle \text{proof} \rangle$ 

lemma (in  $M\_tranc1$ )  $\text{tranc1\_closed [intro, simp]}$ :
  " $M(r) \implies M(\text{tranc1}(r))$ "
   $\langle \text{proof} \rangle$ 

lemma (in  $M\_tranc1$ )  $\text{tranc1\_abs [simp]}$ :
  " $\llbracket M(r); M(z) \rrbracket \implies \text{tran\_closure}(M, r, z) \longleftrightarrow z = \text{tranc1}(r)$ "
   $\langle \text{proof} \rangle$ 

lemma (in  $M\_tranc1$ )  $\text{wellfounded\_tranc1\_separation'}$ :
  " $\llbracket M(r); M(Z) \rrbracket \implies \text{separation}(M, \lambda x. \exists w[M]. w \in Z \wedge \langle w, x \rangle \in r^+)$ "
   $\langle \text{proof} \rangle$ 

```

Alternative proof of  $\text{wf\_on\_tranc1}$ ; inspiration for the relativized version.  
 Original version is on theory WF.

```

lemma " $\llbracket \text{wf}[A](r); r^- 'A \subseteq A \rrbracket \implies \text{wf}[A](r^+)$ "

```

$\langle proof \rangle$

**lemma** (in  $M\_transcl$ )  $wellfounded\_on\_transcl$ :  
 $"\llbracket wellfounded\_on(M,A,r); \ r - 'A \subseteq A; M(r); M(A) \rrbracket$   
 $\implies wellfounded\_on(M,A,r^+)"$   
 $\langle proof \rangle$

**lemma** (in  $M\_transcl$ )  $wellfounded\_transcl$ :  
 $"\llbracket wellfounded(M,r); M(r) \rrbracket \implies wellfounded(M,r^+)"$   
 $\langle proof \rangle$

Absoluteness for wfrec-defined functions.

**lemma** (in  $M\_transcl$ )  $wfrec\_relativize$ :  
 $"\llbracket wf(r); M(a); M(r);$   
 $\quad strong\_replacement(M, \lambda x \ z. \exists y[M]. \exists g[M].$   
 $\quad \quad pair(M,x,y,z) \wedge$   
 $\quad \quad is\_recfun(r^+, x, \lambda x \ f. H(x, restrict(f, r - ' \{x\})), g) \wedge$   
 $\quad \quad y = H(x, restrict(g, r - ' \{x\})))$   
 $\quad \forall x[M]. \forall g[M]. function(g) \longrightarrow M(H(x,g)) \rrbracket$   
 $\implies wfrec(r,a,H) = z \longleftrightarrow$   
 $\quad (\exists f[M]. is\_recfun(r^+, a, \lambda x \ f. H(x, restrict(f, r - ' \{x\})), f)$   
 $\wedge$   
 $\quad \quad z = H(a, restrict(f, r - ' \{a\})))"$   
 $\langle proof \rangle$

Assuming  $r$  is transitive simplifies the occurrences of  $H$ . The premise  $relation(r)$  is necessary before we can replace  $r^+$  by  $r$ .

**theorem** (in  $M\_transcl$ )  $trans\_wfrec\_relativize$ :  
 $"\llbracket wf(r); \ trans(r); \ relation(r); \ M(r); \ M(a);$   
 $\quad wfrec\_replacement(M,MH,r); \ relation2(M,MH,H);$   
 $\quad \forall x[M]. \forall g[M]. function(g) \longrightarrow M(H(x,g)) \rrbracket$   
 $\implies wfrec(r,a,H) = z \longleftrightarrow (\exists f[M]. is\_recfun(r,a,H,f) \wedge z = H(a,f))"$   
 $\langle proof \rangle$

**theorem** (in  $M\_transcl$ )  $trans\_wfrec\_abs$ :  
 $"\llbracket wf(r); \ trans(r); \ relation(r); \ M(r); \ M(a); \ M(z);$   
 $\quad wfrec\_replacement(M,MH,r); \ relation2(M,MH,H);$   
 $\quad \forall x[M]. \forall g[M]. function(g) \longrightarrow M(H(x,g)) \rrbracket$   
 $\implies is\_wfrec(M,MH,r,a,z) \longleftrightarrow z = wfrec(r,a,H)"$   
 $\langle proof \rangle$

**lemma** (in  $M\_transcl$ )  $trans\_eq\_pair\_wfrec\_iff$ :  
 $"\llbracket wf(r); \ trans(r); \ relation(r); \ M(r); \ M(y);$   
 $\quad wfrec\_replacement(M,MH,r); \ relation2(M,MH,H);$   
 $\quad \forall x[M]. \forall g[M]. function(g) \longrightarrow M(H(x,g)) \rrbracket$   
 $\implies y = \langle x, wfrec(r, x, H) \rangle \longleftrightarrow$   
 $\quad (\exists f[M]. is\_recfun(r,x,H,f) \wedge y = \langle x, H(x,f) \rangle)"$

$\langle proof \rangle$

## 5.2 M is closed under well-founded recursion

Lemma with the awkward premise mentioning *wfrec*.

**lemma** (in *M\_trancl*) *wfrec\_closed\_lemma* [rule\_format]:  
 "[[wf(r); M(r);  
   strong\_replacement(M,  $\lambda x y. y = \langle x, wfrec(r, x, H) \rangle$ );  
    $\forall x[M]. \forall g[M]. function(g) \longrightarrow M(H(x, g))$ ]]  
 $\implies M(a) \longrightarrow M(wfrec(r, a, H))$ "  
 $\langle proof \rangle$

Eliminates one instance of replacement.

**lemma** (in *M\_trancl*) *wfrec\_replacement\_iff*:  
 "strong\_replacement(M,  $\lambda x z.$   
    $\exists y[M]. pair(M, x, y, z) \wedge (\exists g[M]. is\_recfun(r, x, H, g) \wedge y = H(x, g))$ )  
 $\longleftrightarrow$   
   strong\_replacement(M,  
      $\lambda x y. \exists f[M]. is\_recfun(r, x, H, f) \wedge y = \langle x, H(x, f) \rangle$ )"  
 $\langle proof \rangle$

Useful version for transitive relations

**theorem** (in *M\_trancl*) *trans\_wfrec\_closed*:  
 "[[wf(r); trans(r); relation(r); M(r); M(a);  
   wfrec\_replacement(M, MH, r); relation2(M, MH, H);  
    $\forall x[M]. \forall g[M]. function(g) \longrightarrow M(H(x, g))$ ]]  
 $\implies M(wfrec(r, a, H))$ "  
 $\langle proof \rangle$

## 5.3 Absoluteness without assuming transitivity

**lemma** (in *M\_trancl*) *eq\_pair\_wfrec\_iff*:  
 "[[wf(r); M(r); M(y);  
   strong\_replacement(M,  $\lambda x z. \exists y[M]. \exists g[M].$   
      $pair(M, x, y, z) \wedge$   
      $is\_recfun(r^+, x, \lambda x f. H(x, restrict(f, r - \{x\})), g) \wedge$   
      $y = H(x, restrict(g, r - \{x\}))$ );  
    $\forall x[M]. \forall g[M]. function(g) \longrightarrow M(H(x, g))$ ]]  
 $\implies y = \langle x, wfrec(r, x, H) \rangle \longleftrightarrow$   
    $(\exists f[M]. is\_recfun(r^+, x, \lambda x f. H(x, restrict(f, r - \{x\})), f)$   
 $\wedge$   
    $y = \langle x, H(x, restrict(f, r - \{x\})) \rangle$ "  
 $\langle proof \rangle$

Full version not assuming transitivity, but maybe not very useful.

**theorem** (in *M\_trancl*) *wfrec\_closed*:  
 "[[wf(r); M(r); M(a);  
   wfrec\_replacement(M, MH, r^+);



```

      relation2(M,MH,  $\lambda x f. H(x, \text{restrict}(f, r - \{x\}))$ );
       $\forall x[M]. \forall g[M]. \text{function}(g) \longrightarrow M(H(x,g))$ 
     $\implies M(\text{wfrec}(r,a,H))$ 
  <proof>

end

```

## 6 Absoluteness Properties for Recursive Datatypes

theory Datatype\_absolute imports Formula WF\_absolute begin

### 6.1 The lfp of a continuous function can be expressed as a union

definition

```

directed :: "i $\Rightarrow$ o" where
  "directed(A)  $\equiv A \neq 0 \wedge (\forall x \in A. \forall y \in A. x \cup y \in A)$ "

```

definition

```

contin :: "(i $\Rightarrow$ i)  $\Rightarrow$  o" where
  "contin(h)  $\equiv (\forall A. \text{directed}(A) \longrightarrow h(\bigcup A) = (\bigcup_{X \in A} h(X)))$ "

```

lemma bnd\_mono\_iterates\_subset: " $\llbracket \text{bnd\_mono}(D, h); n \in \text{nat} \rrbracket \implies h^{\wedge n}(0) \subseteq D$ "  
 <proof>

lemma bnd\_mono\_increasing [rule\_format]:  
 " $\llbracket i \in \text{nat}; j \in \text{nat}; \text{bnd\_mono}(D, h) \rrbracket \implies i \leq j \longrightarrow h^{\wedge i}(0) \subseteq h^{\wedge j}(0)$ "  
 <proof>

lemma directed\_iterates: " $\text{bnd\_mono}(D, h) \implies \text{directed}(\{h^{\wedge n}(0). n \in \text{nat}\})$ "  
 <proof>

lemma contin\_iterates\_eq:  
 " $\llbracket \text{bnd\_mono}(D, h); \text{contin}(h) \rrbracket$   
 $\implies h(\bigcup_{n \in \text{nat}} h^{\wedge n}(0)) = (\bigcup_{n \in \text{nat}} h^{\wedge n}(0))$ "  
 <proof>

lemma lfp\_subset\_Union:  
 " $\llbracket \text{bnd\_mono}(D, h); \text{contin}(h) \rrbracket \implies \text{lfp}(D, h) \subseteq (\bigcup_{n \in \text{nat}} h^{\wedge n}(0))$ "  
 <proof>

lemma Union\_subset\_lfp:  
 " $\text{bnd\_mono}(D, h) \implies (\bigcup_{n \in \text{nat}} h^{\wedge n}(0)) \subseteq \text{lfp}(D, h)$ "  
 <proof>

lemma lfp\_eq\_Union:  
 " $\llbracket \text{bnd\_mono}(D, h); \text{contin}(h) \rrbracket \implies \text{lfp}(D, h) = (\bigcup_{n \in \text{nat}} h^{\wedge n}(0))$ "

$\langle proof \rangle$

### 6.1.1 Some Standard Datatype Constructions Preserve Continuity

**lemma** *contin\_imp\_mono*: " $\llbracket X \subseteq Y; \text{contin}(F) \rrbracket \implies F(X) \subseteq F(Y)$ "  
 $\langle proof \rangle$

**lemma** *sum\_contin*: " $\llbracket \text{contin}(F); \text{contin}(G) \rrbracket \implies \text{contin}(\lambda X. F(X) + G(X))$ "  
 $\langle proof \rangle$

**lemma** *prod\_contin*: " $\llbracket \text{contin}(F); \text{contin}(G) \rrbracket \implies \text{contin}(\lambda X. F(X) * G(X))$ "  
 $\langle proof \rangle$

**lemma** *const\_contin*: " $\text{contin}(\lambda X. A)$ "  
 $\langle proof \rangle$

**lemma** *id\_contin*: " $\text{contin}(\lambda X. X)$ "  
 $\langle proof \rangle$

## 6.2 Absoluteness for "Iterates"

**definition**

*iterates\_MH* :: " $[i \Rightarrow o, [i, i] \Rightarrow o, i, i, i, i] \Rightarrow o$ " where  
 $\text{"iterates\_MH}(M, \text{isF}, v, n, g, z) \equiv$   
 $\text{is\_nat\_case}(M, v, \lambda m u. \exists gm[M]. \text{fun\_apply}(M, g, m, gm) \wedge \text{isF}(gm, u),$   
 $n, z)$ "

**definition**

*is\_iterates* :: " $[i \Rightarrow o, [i, i] \Rightarrow o, i, i, i] \Rightarrow o$ " where  
 $\text{"is\_iterates}(M, \text{isF}, v, n, Z) \equiv$   
 $\exists sn[M]. \exists msn[M]. \text{successor}(M, n, sn) \wedge \text{membership}(M, sn, msn) \wedge$   
 $\text{is\_wfrec}(M, \text{iterates\_MH}(M, \text{isF}, v), msn, n, Z)$ "

**definition**

*iterates\_replacement* :: " $[i \Rightarrow o, [i, i] \Rightarrow o, i] \Rightarrow o$ " where  
 $\text{"iterates\_replacement}(M, \text{isF}, v) \equiv$   
 $\forall n[M]. n \in \text{nat} \longrightarrow$   
 $\text{wfrec\_replacement}(M, \text{iterates\_MH}(M, \text{isF}, v), \text{Memrel}(\text{succ}(n)))$ "

**lemma** (in *M\_basic*) *iterates\_MH\_abs*:  
 $\llbracket \text{relation1}(M, \text{isF}, F); M(n); M(g); M(z) \rrbracket$   
 $\implies \text{iterates\_MH}(M, \text{isF}, v, n, g, z) \longleftrightarrow z = \text{nat\_case}(v, \lambda m. F(g'm), n)$   
 $\langle proof \rangle$

**lemma** (in *M\_trancl*) *iterates\_imp\_wfrec\_replacement*:  
 $\llbracket \text{relation1}(M, \text{isF}, F); n \in \text{nat}; \text{iterates\_replacement}(M, \text{isF}, v) \rrbracket$   
 $\implies \text{wfrec\_replacement}(M, \lambda n f z. z = \text{nat\_case}(v, \lambda m. F(f'm), n),$   
 $\text{Memrel}(\text{succ}(n)))$ "

$\langle proof \rangle$

**theorem** (in *M\_trancl*) *iterates\_abs*:

"[[*iterates\_replacement*(*M*,*isF*,*v*); *relation1*(*M*,*isF*,*F*);  
 $n \in \text{nat}; M(v); M(z); \forall x[M]. M(F(x))$ ]]  
 $\implies \text{is\_iterates}(M, \text{isF}, v, n, z) \iff z = \text{iterates}(F, n, v)$ "  
 $\langle proof \rangle$

**lemma** (in *M\_trancl*) *iterates\_closed [intro,simp]*:

"[[*iterates\_replacement*(*M*,*isF*,*v*); *relation1*(*M*,*isF*,*F*);  
 $n \in \text{nat}; M(v); \forall x[M]. M(F(x))$ ]]  
 $\implies M(\text{iterates}(F, n, v))$ "  
 $\langle proof \rangle$

### 6.3 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*),  $\lambda X. \{0\} + A * X$ )"  
 $\langle proof \rangle$

**lemma** *list\_fun\_contin*: "*contin*( $\lambda X. \{0\} + A * X$ )"  
 $\langle proof \rangle$

Re-expresses lists using sum and product

**lemma** *list\_eq\_lfp2*: "*list*(*A*) = *lfp*(*univ*(*A*),  $\lambda X. \{0\} + A * X$ )"  
 $\langle proof \rangle$

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

**lemma** *list\_eq\_Union*:  
"*list*(*A*) = ( $\bigcup_{n \in \text{nat}} (\lambda X. \{0\} + A * X) \hat{~} n (0)$ )"  
 $\langle proof \rangle$

**definition**

*is\_list\_functor* :: "*[i* $\Rightarrow$ *o, i, i, i*]  $\Rightarrow$  *o*" where  
"*is\_list\_functor*(*M*,*A*,*X*,*Z*)  $\equiv$   
 $\exists n1[M]. \exists AX[M].$   
 $\text{number1}(M, n1) \wedge \text{cartprod}(M, A, X, AX) \wedge \text{is\_sum}(M, n1, AX, Z)$ "

**lemma** (in *M\_basic*) *list\_functor\_abs [simp]*:

"[[*M*(*A*); *M*(*X*); *M*(*Z*)]  $\implies \text{is\_list\_functor}(M, A, X, Z) \iff (Z = \{0\} + A * X)$ "  
 $\langle proof \rangle$

### 6.4 formulas without univ

**lemma** *formula\_fun\_bnd\_mono*:

"bnd\_mono(univ(0),  $\lambda X. ((\text{nat} * \text{nat}) + (\text{nat} * \text{nat})) + (X * X + X))$ )"  
 $\langle \text{proof} \rangle$

**lemma** formula\_fun\_contin:  
 "contin( $\lambda X. ((\text{nat} * \text{nat}) + (\text{nat} * \text{nat})) + (X * X + X)$ )"  
 $\langle \text{proof} \rangle$

Re-expresses formulas using sum and product

**lemma** formula\_eq\_lfp2:  
 "formula = lfp(univ(0),  $\lambda X. ((\text{nat} * \text{nat}) + (\text{nat} * \text{nat})) + (X * X + X)$ )"  
 $\langle \text{proof} \rangle$

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

**lemma** formula\_eq\_Union:  
 "formula =  
 $(\bigcup_{n \in \text{nat}. } (\lambda X. ((\text{nat} * \text{nat}) + (\text{nat} * \text{nat})) + (X * X + X)) \wedge n (0))$ "  
 $\langle \text{proof} \rangle$

**definition**

is\_formula\_functor :: "[ $i \Rightarrow o, i, i$ ]  $\Rightarrow o$ " where  
 "is\_formula\_functor( $M, X, Z$ )  $\equiv$   
 $\exists \text{nat}'[M]. \exists \text{natnat}[M]. \exists \text{natnatsum}[M]. \exists XX[M]. \exists X3[M].$   
 $\text{omega}(M, \text{nat}') \wedge \text{cartprod}(M, \text{nat}', \text{nat}', \text{natnat}) \wedge$   
 $\text{is\_sum}(M, \text{natnat}, \text{natnat}, \text{natnatsum}) \wedge$   
 $\text{cartprod}(M, X, X, XX) \wedge \text{is\_sum}(M, XX, X, X3) \wedge$   
 $\text{is\_sum}(M, \text{natnatsum}, X3, Z)$ "

**lemma** (in M\_trancl) formula\_functor\_abs [simp]:  
 " $\llbracket M(X); M(Z) \rrbracket$   
 $\implies \text{is\_formula\_functor}(M, X, Z) \longleftrightarrow$   
 $Z = ((\text{nat} * \text{nat}) + (\text{nat} * \text{nat})) + (X * X + X)$ "  
 $\langle \text{proof} \rangle$

## 6.5 M Contains the List and Formula Datatypes

**definition**

list\_N :: "[ $i, i$ ]  $\Rightarrow i$ " where  
 "list\_N( $A, n$ )  $\equiv (\lambda X. \{0\} + A * X) \wedge n (0)$ "

**lemma** Nil\_in\_list\_N [simp]: " $[] \in \text{list\_N}(A, \text{succ}(n))$ "  
 $\langle \text{proof} \rangle$

**lemma** Cons\_in\_list\_N [simp]:  
 " $\text{Cons}(a, l) \in \text{list\_N}(A, \text{succ}(n)) \longleftrightarrow a \in A \wedge l \in \text{list\_N}(A, n)$ "  
 $\langle \text{proof} \rangle$

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

**lemma** *list\_N\_0*: " $\text{list}_N(A, 0) = 0$ "  
 <proof>

**lemma** *list\_N\_succ*: " $\text{list}_N(A, \text{succ}(n)) = \{0\} + A * (\text{list}_N(A, n))$ "  
 <proof>

**lemma** *list\_N\_imp\_list*:  
 " $\llbracket l \in \text{list}_N(A, n); n \in \text{nat} \rrbracket \implies l \in \text{list}(A)$ "  
 <proof>

**lemma** *list\_N\_imp\_length\_lt* [rule\_format]:  
 " $n \in \text{nat} \implies \forall l \in \text{list}_N(A, n). \text{length}(l) < n$ "  
 <proof>

**lemma** *list\_imp\_list\_N* [rule\_format]:  
 " $l \in \text{list}(A) \implies \forall n \in \text{nat}. \text{length}(l) < n \longrightarrow l \in \text{list}_N(A, n)$ "  
 <proof>

**lemma** *list\_N\_imp\_eq\_length*:  
 " $\llbracket n \in \text{nat}; l \notin \text{list}_N(A, n); l \in \text{list}_N(A, \text{succ}(n)) \rrbracket$   
 $\implies n = \text{length}(l)$ "  
 <proof>

Express *list\_rec* without using *rank* or *Vset*, neither of which is absolute.

**lemma** (in *M\_trivial*) *list\_rec\_eq*:  
 " $l \in \text{list}(A) \implies$   
 $\text{list\_rec}(a, g, l) =$   
 $\text{transrec}(\text{succ}(\text{length}(l)),$   
 $\lambda x h. \text{Lambda}(\text{list}(A),$   
 $\text{list\_case}'(a,$   
 $\lambda a l. g(a, l, h \text{ ' succ}(\text{length}(l)) \text{ ' } l)))$   
 $l$ "  
 <proof>

**definition**  
*is\_list\_N* :: " $[i \Rightarrow o, i, i, i] \Rightarrow o$ " where  
 " $\text{is\_list\_N}(M, A, n, Z) \equiv$   
 $\exists \text{zero}[M]. \text{empty}(M, \text{zero}) \wedge$   
 $\text{is\_iterates}(M, \text{is\_list\_functor}(M, A), \text{zero}, n, Z)$ "

**definition**  
*mem\_list* :: " $[i \Rightarrow o, i, i] \Rightarrow o$ " where  
 " $\text{mem\_list}(M, A, l) \equiv$   
 $\exists n[M]. \exists \text{listn}[M].$   
 $\text{finite\_ordinal}(M, n) \wedge \text{is\_list\_N}(M, A, n, \text{listn}) \wedge l \in \text{listn}$ "

**definition**  
*is\_list* :: " $[i \Rightarrow o, i, i] \Rightarrow o$ " where  
 " $\text{is\_list}(M, A, Z) \equiv \forall l[M]. l \in Z \longleftrightarrow \text{mem\_list}(M, A, l)$ "

### 6.5.1 Towards Absoluteness of *formula\_rec*

**consts**    *depth* :: "*i*  $\Rightarrow$  *i*"

**primrec**

"*depth*(*Member*(*x*,*y*)) = 0"  
 "*depth*(*Equal*(*x*,*y*)) = 0"  
 "*depth*(*Nand*(*p*,*q*)) = succ(*depth*(*p*)  $\cup$  *depth*(*q*))"  
 "*depth*(*Forall*(*p*)) = succ(*depth*(*p*))"

**lemma** *depth\_type* [TC]: "*p*  $\in$  *formula*  $\implies$  *depth*(*p*)  $\in$  *nat*"

*<proof>*

**definition**

*formula\_N* :: "*i*  $\Rightarrow$  *i*" **where**  
 "*formula\_N*(*n*)  $\equiv$  ( $\lambda X.$  ((*nat*\**nat*) + (*nat*\**nat*)) + (*X*\**X* + *X*))  $\wedge$  *n* (0)"

**lemma** *Member\_in\_formula\_N* [simp]:

"*Member*(*x*,*y*)  $\in$  *formula\_N*(succ(*n*))  $\longleftrightarrow$  *x*  $\in$  *nat*  $\wedge$  *y*  $\in$  *nat*"

*<proof>*

**lemma** *Equal\_in\_formula\_N* [simp]:

"*Equal*(*x*,*y*)  $\in$  *formula\_N*(succ(*n*))  $\longleftrightarrow$  *x*  $\in$  *nat*  $\wedge$  *y*  $\in$  *nat*"

*<proof>*

**lemma** *Nand\_in\_formula\_N* [simp]:

"*Nand*(*x*,*y*)  $\in$  *formula\_N*(succ(*n*))  $\longleftrightarrow$  *x*  $\in$  *formula\_N*(*n*)  $\wedge$  *y*  $\in$  *formula\_N*(*n*)"

*<proof>*

**lemma** *Forall\_in\_formula\_N* [simp]:

"*Forall*(*x*)  $\in$  *formula\_N*(succ(*n*))  $\longleftrightarrow$  *x*  $\in$  *formula\_N*(*n*)"

*<proof>*

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

**lemma** *formula\_N\_0*: "*formula\_N*(0) = 0"

*<proof>*

**lemma** *formula\_N\_succ*:

"*formula\_N*(succ(*n*)) =  
 ((*nat*\**nat*) + (*nat*\**nat*)) + (*formula\_N*(*n*) \* *formula\_N*(*n*) + *formula\_N*(*n*))"

*<proof>*

**lemma** *formula\_N\_imp\_formula*:

" $\llbracket p \in \text{formula\_N}(n); n \in \text{nat} \rrbracket \implies p \in \text{formula}$ "

*<proof>*

**lemma** *formula\_N\_imp\_depth\_lt* [rule\_format]:

"*n*  $\in$  *nat*  $\implies \forall p \in \text{formula\_N}(n). \text{depth}(p) < n$ "

*<proof>*

```

lemma formula_imp_formula_N [rule_format]:
  "p ∈ formula ⇒ ∀ n ∈ nat. depth(p) < n → p ∈ formula_N(n)"
⟨proof⟩

```

```

lemma formula_N_imp_eq_depth:
  "⌊n ∈ nat; p ∉ formula_N(n); p ∈ formula_N(succ(n))⌋
  ⇒ n = depth(p)"
⟨proof⟩

```

This result and the next are unused.

```

lemma formula_N_mono [rule_format]:
  "⌊m ∈ nat; n ∈ nat⌋ ⇒ m ≤ n → formula_N(m) ⊆ formula_N(n)"
⟨proof⟩

```

```

lemma formula_N_distrib:
  "⌊m ∈ nat; n ∈ nat⌋ ⇒ formula_N(m ∪ n) = formula_N(m) ∪ formula_N(n)"
⟨proof⟩

```

**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 ∈ formn"

```

**definition**

```

is_formula :: "[i ⇒ o, i] ⇒ o" where
  "is_formula(M, Z) ≡ ∀ p[M]. p ∈ Z ↔ mem_formula(M, p)"

```

```

locale M_datatypes = M_tranc1 +
  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(1) ⇒ iterates_replacement(M, λl t. is_tl(M, l, t), 1)"

```

### 6.5.2 Absoluteness of the List Construction

```
lemma (in M_datatypes) list_replacement2':
  "M(A)  $\implies$  strong_replacement(M,  $\lambda n y. n \in \text{nat} \wedge y = (\lambda X. \{0\} + A * X)^n$ 
  (0))"
<proof>
```

```
lemma (in M_datatypes) list_closed [intro,simp]:
  "M(A)  $\implies$  M(list(A))"
<proof>
```

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  $\in$  nat; M(Z)]
 $\implies$  is_list_N(M,A,n,Z)  $\longleftrightarrow$  Z = list_N(A,n)"
<proof>
```

```
lemma (in M_datatypes) list_N_closed [intro,simp]:
  "[M(A); n  $\in$  nat]  $\implies$  M(list_N(A,n))"
<proof>
```

```
lemma (in M_datatypes) mem_list_abs [simp]:
  "M(A)  $\implies$  mem_list(M,A,l)  $\longleftrightarrow$  l  $\in$  list(A)"
<proof>
```

```
lemma (in M_datatypes) list_abs [simp]:
  "[M(A); M(Z)]  $\implies$  is_list(M,A,Z)  $\longleftrightarrow$  Z = list(A)"
<proof>
```

### 6.5.3 Absoluteness of Formulas

```
lemma (in M_datatypes) formula_replacement2':
  "strong_replacement(M,  $\lambda n y. n \in \text{nat} \wedge y = (\lambda X. ((\text{nat} * \text{nat}) + (\text{nat} * \text{nat}))$ 
  + (X*X + X))^n (0))"
<proof>
```

```
lemma (in M_datatypes) formula_closed [intro,simp]:
  "M(formula)"
<proof>
```

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

```
lemma (in M_datatypes) formula_N_abs [simp]:
  "[n  $\in$  nat; M(Z)]
 $\implies$  is_formula_N(M,n,Z)  $\longleftrightarrow$  Z = formula_N(n)"
<proof>
```

```
lemma (in M_datatypes) formula_N_closed [intro,simp]:
```



" $n \in \text{nat} \implies M(\text{formula\_N}(n))$ "  
 <proof>

lemma (in M\_datatypes) mem\_formula\_abs [simp]:  
 "mem\_formula(M, l)  $\longleftrightarrow$  l  $\in$  formula"  
 <proof>

lemma (in M\_datatypes) formula\_abs [simp]:  
 " $\llbracket M(Z) \rrbracket \implies \text{is\_formula}(M, Z) \longleftrightarrow Z = \text{formula}$ "  
 <proof>

## 6.6 Absoluteness for $\varepsilon$ -Closure: the *eclose* Operator

Re-expresses *eclose* using "iterates"

lemma *eclose\_eq\_Union*:  
 " $\text{eclose}(A) = (\bigcup_{n \in \text{nat}} \text{Union}^n(A))$ "  
 <proof>

definition

*is\_eclose\_n* :: "[ $i \Rightarrow o, i, i, i$ ]  $\Rightarrow o$ " where  
 " $\text{is\_eclose\_n}(M, A, n, Z) \equiv \text{is\_iterates}(M, \text{big\_union}(M), A, n, Z)$ "

definition

*mem\_eclose* :: "[ $i \Rightarrow o, i, i$ ]  $\Rightarrow o$ " where  
 " $\text{mem\_eclose}(M, A, l) \equiv$   
 $\exists n[M]. \exists \text{eclosen}[M].$   
 $\text{finite\_ordinal}(M, n) \wedge \text{is\_eclose\_n}(M, A, n, \text{eclosen}) \wedge l \in \text{eclosen}$ "

definition

*is\_eclose* :: "[ $i \Rightarrow o, i, i$ ]  $\Rightarrow o$ " where  
 " $\text{is\_eclose}(M, A, Z) \equiv \forall u[M]. u \in Z \longleftrightarrow \text{mem\_eclose}(M, A, u)$ "

locale *M\_eclose* = *M\_datatypes* +

assumes *eclose\_replacement1*:

" $M(A) \implies \text{iterates\_replacement}(M, \text{big\_union}(M), A)$ "

and *eclose\_replacement2*:

" $M(A) \implies \text{strong\_replacement}(M,$   
 $\lambda n y. n \in \text{nat} \wedge \text{is\_iterates}(M, \text{big\_union}(M), A, n, y))$ "

lemma (in *M\_eclose*) *eclose\_replacement2'*:

" $M(A) \implies \text{strong\_replacement}(M, \lambda n y. n \in \text{nat} \wedge y = \text{Union}^n(A))$ "

<proof>

lemma (in *M\_eclose*) *eclose\_closed* [intro, simp]:

" $M(A) \implies M(\text{eclose}(A))$ "

<proof>

lemma (in *M\_eclose*) *is\_eclose\_n\_abs* [simp]:

$\llbracket M(A); n \in \text{nat}; M(Z) \rrbracket \implies \text{is\_eclose\_n}(M, A, n, Z) \longleftrightarrow Z = \text{Union}^n(A)$   
 $\langle \text{proof} \rangle$

**lemma** (in  $M_{\text{eclose}}$ )  $\text{mem\_eclose\_abs}$  [simp]:  
 $\llbracket M(A) \rrbracket \implies \text{mem\_eclose}(M, A, 1) \longleftrightarrow 1 \in \text{eclose}(A)$   
 $\langle \text{proof} \rangle$

**lemma** (in  $M_{\text{eclose}}$ )  $\text{eclose\_abs}$  [simp]:  
 $\llbracket M(A); M(Z) \rrbracket \implies \text{is\_eclose}(M, A, Z) \longleftrightarrow Z = \text{eclose}(A)$   
 $\langle \text{proof} \rangle$

## 6.7 Absoluteness for $\text{transrec}$

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

**definition**

$\text{is\_transrec} :: "[i \Rightarrow o, [i, i, i] \Rightarrow o, i, i] \Rightarrow o"$  where  
 $\text{"is\_transrec}(M, MH, a, z) \equiv$   
 $\exists sa[M]. \exists esa[M]. \exists mesa[M].$   
 $\text{upair}(M, a, a, sa) \wedge \text{is\_eclose}(M, sa, esa) \wedge \text{membership}(M, esa, mesa)$   
 $\wedge$   
 $\text{is\_wfrec}(M, MH, mesa, a, z)"$

**definition**

$\text{transrec\_replacement} :: "[i \Rightarrow o, [i, i, i] \Rightarrow o, i] \Rightarrow o"$  where  
 $\text{"transrec\_replacement}(M, MH, a) \equiv$   
 $\exists sa[M]. \exists esa[M]. \exists mesa[M].$   
 $\text{upair}(M, a, a, sa) \wedge \text{is\_eclose}(M, sa, esa) \wedge \text{membership}(M, esa, mesa)$   
 $\wedge$   
 $\text{wfrec\_replacement}(M, MH, mesa)"$

The condition  $\text{Ord}(i)$  lets us use the simpler  $\text{trans\_wfrec\_abs}$  rather than  $\text{trans\_wfrec\_abs}$ , which I haven't even proved yet.

**theorem** (in  $M_{\text{eclose}}$ )  $\text{transrec\_abs}$ :  
 $\llbracket \text{transrec\_replacement}(M, MH, i); \text{relation2}(M, MH, H);$   
 $\text{Ord}(i); M(i); M(z);$   
 $\forall x[M]. \forall g[M]. \text{function}(g) \longrightarrow M(H(x, g)) \rrbracket$   
 $\implies \text{is\_transrec}(M, MH, i, z) \longleftrightarrow z = \text{transrec}(i, H)$   
 $\langle \text{proof} \rangle$

**theorem** (in  $M_{\text{eclose}}$ )  $\text{transrec\_closed}$ :  
 $\llbracket \text{transrec\_replacement}(M, MH, i); \text{relation2}(M, MH, H);$   
 $\text{Ord}(i); M(i);$   
 $\forall x[M]. \forall g[M]. \text{function}(g) \longrightarrow M(H(x, g)) \rrbracket$   
 $\implies M(\text{transrec}(i, H))$   
 $\langle \text{proof} \rangle$

Helps to prove instances of  $\text{transrec\_replacement}$

```

lemma (in M_eclose) transrec_replacementI:
  "⌊M(a);
    strong_replacement (M,
      λx z. ∃y[M]. pair(M, x, y, z) ∧
        is_wfrec(M, MH, Memrel(eclose({a})), x, y))⌋
    ⇒ transrec_replacement(M, MH, a)"
  <proof>

```

## 6.8 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"

```

```

lemma (in M_datatypes) length_abs [simp]:
  "⌊M(A); l ∈ list(A); n ∈ nat⌋ ⇒ is_length(M, A, l, n) ⇔ n = length(l)"
  <proof>

```

Proof is trivial since *length* returns natural numbers.

```

lemma (in M_trivial) length_closed [intro, simp]:
  "l ∈ list(A) ⇒ M(length(l))"
  <proof>

```

## 6.9 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))"
  <proof>

```

```

lemma (in M_basic) iterates_tl'_closed:
  "⌊n ∈ nat; M(x)⌋ ⇒ M(tl' ^n (x))"
  <proof>

```

Immediate by type-checking

```

lemma (in M_datatypes) nth_closed [intro, simp]:
  "⌊xs ∈ list(A); n ∈ nat; M(A)⌋ ⇒ M(nth(n, xs))"
  <proof>

```

**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]*:  

$$\llbracket M(A); n \in \text{nat}; l \in \text{list}(A); M(Z) \rrbracket \implies \text{is\_nth}(M, n, l, Z) \longleftrightarrow Z = \text{nth}(n, l)$$
*<proof>*

## 6.10 Relativization and Absoluteness for the *formula* Constructors

**definition**

*is\_Member* :: "[*i* ⇒ *o*, *i*, *i*, *i*] ⇒ *o*" where  
— because  $\text{Member}(x, y) \equiv \text{Inl}(\text{Inl}(\langle x, y \rangle))$   

$$\text{"is\_Member}(M, x, y, Z) \equiv \exists p[M]. \exists u[M]. \text{pair}(M, x, y, p) \wedge \text{is\_Inl}(M, p, u) \wedge \text{is\_Inl}(M, u, Z)$$

**lemma** (in *M\_trivial*) *Member\_abs [simp]*:  

$$\llbracket M(x); M(y); M(Z) \rrbracket \implies \text{is\_Member}(M, x, y, Z) \longleftrightarrow (Z = \text{Member}(x, y))$$
*<proof>*

**lemma** (in *M\_trivial*) *Member\_in\_M\_iff [iff]*:  

$$M(\text{Member}(x, y)) \longleftrightarrow M(x) \wedge M(y)$$
*<proof>*

**definition**

*is\_Equal* :: "[*i* ⇒ *o*, *i*, *i*, *i*] ⇒ *o*" where  
— because  $\text{Equal}(x, y) \equiv \text{Inl}(\text{Inr}(\langle x, y \rangle))$   

$$\text{"is\_Equal}(M, x, y, Z) \equiv \exists p[M]. \exists u[M]. \text{pair}(M, x, y, p) \wedge \text{is\_Inr}(M, p, u) \wedge \text{is\_Inl}(M, u, Z)$$

**lemma** (in *M\_trivial*) *Equal\_abs [simp]*:  

$$\llbracket M(x); M(y); M(Z) \rrbracket \implies \text{is\_Equal}(M, x, y, Z) \longleftrightarrow (Z = \text{Equal}(x, y))$$
*<proof>*

**lemma** (in *M\_trivial*) *Equal\_in\_M\_iff [iff]*:  $M(\text{Equal}(x, y)) \longleftrightarrow M(x) \wedge M(y)$   
*<proof>*

**definition**

*is\_Nand* :: "[*i* ⇒ *o*, *i*, *i*, *i*] ⇒ *o*" where  
— because  $\text{Nand}(x, y) \equiv \text{Inr}(\text{Inl}(\langle x, y \rangle))$   

$$\text{"is\_Nand}(M, x, y, Z) \equiv \exists p[M]. \exists u[M]. \text{pair}(M, x, y, p) \wedge \text{is\_Inl}(M, p, u) \wedge \text{is\_Inr}(M, u, Z)$$

**lemma** (in *M\_trivial*) *Nand\_abs [simp]*:  

$$\llbracket M(x); M(y); M(Z) \rrbracket \implies \text{is\_Nand}(M, x, y, Z) \longleftrightarrow (Z = \text{Nand}(x, y))$$
*<proof>*

**lemma** (in *M\_trivial*) *Nand\_in\_M\_iff [iff]*:  $M(\text{Nand}(x, y)) \longleftrightarrow M(x) \wedge M(y)$   
*<proof>*

**definition**

```
is_Forall :: "[i⇒o,i,i] ⇒ o" where
  — because Forall(x) ≡ Inr(Inr(p))
  "is_Forall(M,p,Z) ≡ ∃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) ⟷ (Z = Forall(x))"
⟨proof⟩
```

**lemma** (in *M\_trivial*) *Forall\_in\_M\_iff* [iff]: "*M*(Forall(*x*)) ⟷ *M*(*x*)"

⟨proof⟩

## 6.11 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,
      λu v. c(u, v, h ' succ(depth(u)) ' u,
              h ' succ(depth(v)) ' v),
      λu. 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)),
    λx h. Lambda (formula, formula_rec_case(a,b,c,d,h))) ' p"
⟨proof⟩
```

### 6.11.1 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"
```

**lemma** (in *M\_datatypes*) *depth\_abs* [simp]:

```
"⌊p ∈ formula; n ∈ nat⌋ ⇒ is_depth(M,p,n) ⟷ n = depth(p)"
⟨proof⟩
```

Proof is trivial since *depth* returns natural numbers.

**lemma** (in *M\_trivial*) *depth\_closed* [intro,simp]:

"p ∈ formula ⇒ M(depth(p))"  
 ⟨proof⟩

### 6.11.2 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)"

⟨proof⟩

**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))"

⟨proof⟩

### 6.11.3 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, λu v. c(u, v, h'succ(depth(u))'u, h'succ(depth(v))'v));

```

      Relation1(M, formula,
        is_d, λu. 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))"
  <proof>

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 +
  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),
        λu 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), λu. 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, λx 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))"
  <proof>

lemma (in Formula_Rec) formula_rec_lam_closed:
  "M(g) ⇒ M(Lambda (formula, formula_rec_case(a,b,c,d,g)))"

```

$\langle proof \rangle$

**lemma** (in *Formula\_Rec*) *MH\_rel2*:

"relation2 (M, MH,  
 $\lambda x h. \text{Lambda } (formula, formula\_rec\_case(a,b,c,d,h))$ )"

$\langle proof \rangle$

**lemma** (in *Formula\_Rec*) *fr\_transrec\_closed*:

"n  $\in$  nat  
 $\implies M(\text{transrec}$   
 $(n, \lambda x h. \text{Lambda}(formula, formula\_rec\_case(a, b, c, d, h))))$ "

$\langle proof \rangle$

The main two results: *formula\_rec* is absolute for *M*.

**theorem** (in *Formula\_Rec*) *formula\_rec\_closed*:

"p  $\in$  formula  $\implies M(formula\_rec(a,b,c,d,p))$ "

$\langle proof \rangle$

**theorem** (in *Formula\_Rec*) *formula\_rec\_abs*:

"[p  $\in$  formula; M(z)]  
 $\implies is\_formula\_rec(M,MH,p,z) \longleftrightarrow z = formula\_rec(a,b,c,d,p)$ "

$\langle proof \rangle$

**end**

## 7 Closed Unbounded Classes and Normal Functions

**theory** *Normal* imports *ZF* begin

One source is the book

Frank R. Drake. *Set Theory: An Introduction to Large Cardinals*. North-Holland, 1974.

### 7.1 Closed and Unbounded (c.u.) Classes of Ordinals

**definition**

*Closed* :: "(i $\Rightarrow$ o)  $\Rightarrow$  o" where  
 $"Closed(P) \equiv \forall I. I \neq 0 \longrightarrow (\forall i \in I. Ord(i) \wedge P(i)) \longrightarrow P(\bigcup(I))"$

**definition**

*Unbounded* :: "(i $\Rightarrow$ o)  $\Rightarrow$  o" where  
 $"Unbounded(P) \equiv \forall i. Ord(i) \longrightarrow (\exists j. i < j \wedge P(j))"$

**definition**

*Closed\_Unbounded* :: "(i $\Rightarrow$ o)  $\Rightarrow$  o" where  
 $"Closed\_Unbounded(P) \equiv Closed(P) \wedge Unbounded(P)"$



### 7.1.1 Simple facts about c.u. classes

**lemma** *ClosedI*:

" $\llbracket \bigwedge I. \llbracket I \neq 0; \forall i \in I. \text{Ord}(i) \wedge P(i) \rrbracket \implies P(\bigcup(I)) \rrbracket$   
 $\implies \text{Closed}(P)$ "  
 $\langle \text{proof} \rangle$

**lemma** *ClosedD*:

" $\llbracket \text{Closed}(P); I \neq 0; \bigwedge i. i \in I \implies \text{Ord}(i); \bigwedge i. i \in I \implies P(i) \rrbracket$   
 $\implies P(\bigcup(I))$ "  
 $\langle \text{proof} \rangle$

**lemma** *UnboundedD*:

" $\llbracket \text{Unbounded}(P); \text{Ord}(i) \rrbracket \implies \exists j. i < j \wedge P(j)$ "  
 $\langle \text{proof} \rangle$

**lemma** *Closed\_Unbounded\_imp\_Unbounded*: " $\text{Closed\_Unbounded}(C) \implies \text{Unbounded}(C)$ "  
 $\langle \text{proof} \rangle$

The universal class, *V*, is closed and unbounded. A bit odd, since *C. U.* concerns only ordinals, but it's used below!

**theorem** *Closed\_Unbounded\_V* [simp]: " $\text{Closed\_Unbounded}(\lambda x. \text{True})$ "  
 $\langle \text{proof} \rangle$

The class of ordinals, *Ord*, is closed and unbounded.

**theorem** *Closed\_Unbounded\_Ord* [simp]: " $\text{Closed\_Unbounded}(\text{Ord})$ "  
 $\langle \text{proof} \rangle$

The class of limit ordinals, *Limit*, is closed and unbounded.

**theorem** *Closed\_Unbounded\_Limit* [simp]: " $\text{Closed\_Unbounded}(\text{Limit})$ "  
 $\langle \text{proof} \rangle$

The class of cardinals, *Card*, is closed and unbounded.

**theorem** *Closed\_Unbounded\_Card* [simp]: " $\text{Closed\_Unbounded}(\text{Card})$ "  
 $\langle \text{proof} \rangle$

### 7.1.2 The intersection of any set-indexed family of c.u. classes is c.u.

The constructions below come from Kunen, *Set Theory*, page 78.

**locale** *cub\_family* =

fixes *P* and *A*

fixes *next\_greater* — the next ordinal satisfying class *A*

fixes *sup\_greater* — sup of those ordinals over all *A*

assumes *closed*: " $a \in A \implies \text{Closed}(P(a))$ "

and *unbounded*: " $a \in A \implies \text{Unbounded}(P(a))$ "

and *A\_non0*: " $A \neq 0$ "

defines " $\text{next\_greater}(a, x) \equiv \mu y. x < y \wedge P(a, y)$ "

and "sup\_greater(x)  $\equiv \bigcup a \in A. \text{next\_greater}(a, x)$ "

begin

Trivial that the intersection is closed.

lemma Closed\_INT: "Closed( $\lambda x. \forall i \in A. P(i, x)$ )"  
 <proof>

All remaining effort goes to show that the intersection is unbounded.

lemma Ord\_sup\_greater:  
 "Ord(sup\_greater(x))"  
 <proof>

lemma Ord\_next\_greater:  
 "Ord(next\_greater(a, x))"  
 <proof>

next\_greater works as expected: it returns a larger value and one that belongs to class  $P(a)$ .

lemma  
 assumes "Ord(x)" "a  $\in A$ "  
 shows next\_greater\_in\_P: "P(a, next\_greater(a, x))"  
 and next\_greater\_gt: "x < next\_greater(a, x)"  
 <proof>

lemma sup\_greater\_gt:  
 "Ord(x)  $\implies x < \text{sup\_greater}(x)$ "  
 <proof>

lemma next\_greater\_le\_sup\_greater:  
 "a  $\in A \implies \text{next\_greater}(a, x) \leq \text{sup\_greater}(x)$ "  
 <proof>

lemma omega\_sup\_greater\_eq\_UN:  
 assumes "Ord(x)" "a  $\in A$ "  
 shows "sup\_greater $^\omega$  (x) =  
 ( $\bigcup n \in \text{nat}. \text{next\_greater}(a, \text{sup\_greater}^n(x))$ )"  
 <proof>

lemma P\_omega\_sup\_greater:  
 "[Ord(x); a  $\in A$ ]  $\implies P(a, \text{sup\_greater}^\omega(x))$ "  
 <proof>

lemma omega\_sup\_greater\_gt:  
 "Ord(x)  $\implies x < \text{sup\_greater}^\omega(x)$ "  
 <proof>

lemma Unbounded\_INT: "Unbounded( $\lambda x. \forall a \in A. P(a, x)$ )"  
 <proof>

```

lemma Closed_Unbounded_INT:
  "Closed_Unbounded( $\lambda x. \forall a \in A. P(a, x)$ )"
  <proof>

```

end

```

theorem Closed_Unbounded_INT:
  assumes " $\bigwedge a. a \in A \implies \text{Closed\_Unbounded}(P(a))$ "
  shows " $\text{Closed\_Unbounded}(\lambda x. \forall a \in A. P(a, x))$ "
  <proof>

```

```

lemma Int_iff_INT2:
  " $P(x) \wedge Q(x) \iff (\forall i \in 2. (i=0 \longrightarrow P(x)) \wedge (i=1 \longrightarrow Q(x)))$ "
  <proof>

```

```

theorem Closed_Unbounded_Int:
  " $\llbracket \text{Closed\_Unbounded}(P); \text{Closed\_Unbounded}(Q) \rrbracket$ 
 $\implies \text{Closed\_Unbounded}(\lambda x. P(x) \wedge Q(x))$ "
  <proof>

```

## 7.2 Normal Functions

```

definition
  mono_le_subset :: "( $i \Rightarrow i$ )  $\Rightarrow$  o" where
    "mono_le_subset(M)  $\equiv \forall i\ j. i \leq j \longrightarrow M(i) \subseteq M(j)$ "

```

```

definition
  mono_Ord :: "( $i \Rightarrow i$ )  $\Rightarrow$  o" where
    "mono_Ord(F)  $\equiv \forall i\ j. i < j \longrightarrow F(i) < F(j)$ "

```

```

definition
  cont_Ord :: "( $i \Rightarrow i$ )  $\Rightarrow$  o" where
    "cont_Ord(F)  $\equiv \forall l. \text{Limit}(l) \longrightarrow F(l) = (\bigcup i < l. F(i))$ "

```

```

definition
  Normal :: "( $i \Rightarrow i$ )  $\Rightarrow$  o" where
    "Normal(F)  $\equiv \text{mono\_Ord}(F) \wedge \text{cont\_Ord}(F)$ "

```

### 7.2.1 Immediate properties of the definitions

```

lemma NormalI:
  " $\llbracket \bigwedge i\ j. i < j \implies F(i) < F(j); \bigwedge l. \text{Limit}(l) \implies F(l) = (\bigcup i < l. F(i)) \rrbracket$ 
 $\implies \text{Normal}(F)$ "
  <proof>

```

```

lemma mono_Ord_imp_Ord: " $\llbracket \text{Ord}(i); \text{mono\_Ord}(F) \rrbracket \implies \text{Ord}(F(i))$ "
  <proof>

```

**lemma** *mono\_Ord\_imp\_mono*: " $\llbracket i < j; \text{mono\_Ord}(F) \rrbracket \implies F(i) < F(j)$ "  
 $\langle \text{proof} \rangle$

**lemma** *Normal\_imp\_Ord [simp]*: " $\llbracket \text{Normal}(F); \text{Ord}(i) \rrbracket \implies \text{Ord}(F(i))$ "  
 $\langle \text{proof} \rangle$

**lemma** *Normal\_imp\_cont*: " $\llbracket \text{Normal}(F); \text{Limit}(l) \rrbracket \implies F(l) = (\bigcup_{i < l}. F(i))$ "  
 $\langle \text{proof} \rangle$

**lemma** *Normal\_imp\_mono*: " $\llbracket i < j; \text{Normal}(F) \rrbracket \implies F(i) < F(j)$ "  
 $\langle \text{proof} \rangle$

**lemma** *Normal\_increasing*:  
 assumes  $i: " \text{Ord}(i) "$  and  $F: " \text{Normal}(F) "$  shows " $i \leq F(i)$ "  
 $\langle \text{proof} \rangle$

### 7.2.2 The class of fixedpoints is closed and unbounded

The proof is from Drake, pages 113–114.

**lemma** *mono\_Ord\_imp\_le\_subset*: " $\text{mono\_Ord}(F) \implies \text{mono\_le\_subset}(F)$ "  
 $\langle \text{proof} \rangle$

The following equation is taken for granted in any set theory text.

**lemma** *cont\_Ord\_Union*:  
 $\llbracket \text{cont\_Ord}(F); \text{mono\_le\_subset}(F); X=0 \longrightarrow F(0)=0; \forall x \in X. \text{Ord}(x) \rrbracket$   
 $\implies F(\bigcup(X)) = (\bigcup_{y \in X}. F(y))$   
 $\langle \text{proof} \rangle$

**lemma** *Normal\_Union*:  
 $\llbracket X \neq 0; \forall x \in X. \text{Ord}(x); \text{Normal}(F) \rrbracket \implies F(\bigcup(X)) = (\bigcup_{y \in X}. F(y))$   
 $\langle \text{proof} \rangle$

**lemma** *Normal\_imp\_fp\_Closed*: " $\text{Normal}(F) \implies \text{Closed}(\lambda i. F(i) = i)$ "  
 $\langle \text{proof} \rangle$

**lemma** *iterates\_Normal\_increasing*:  
 $\llbracket n \in \text{nat}; x < F(x); \text{Normal}(F) \rrbracket$   
 $\implies F^n(x) < F(\text{succ}(n))(x)$   
 $\langle \text{proof} \rangle$

**lemma** *Ord\_iterates\_Normal*:  
 $\llbracket n \in \text{nat}; \text{Normal}(F); \text{Ord}(x) \rrbracket \implies \text{Ord}(F^n(x))$   
 $\langle \text{proof} \rangle$

THIS RESULT IS UNUSED

**lemma** *iterates\_omega\_Limit*:

```

"[[Normal(F); x < F(x)]] ==> Limit(F^ω (x))"
⟨proof⟩

lemma iterates_omega_fixedpoint:
  "[[Normal(F); Ord(a)]] ==> F(F^ω (a)) = F^ω (a)"
⟨proof⟩

lemma iterates_omega_increasing:
  "[[Normal(F); Ord(a)]] ==> a ≤ F^ω (a)"
⟨proof⟩

lemma Normal_imp_fp_Unbounded: "Normal(F) ==> Unbounded(λi. F(i) = i)"
⟨proof⟩

theorem Normal_imp_fp_Closed_Unbounded:
  "Normal(F) ==> Closed_Unbounded(λi. F(i) = i)"
⟨proof⟩

7.2.3 Function normalize

Function normalize maps a function F to a normal function that bounds
it above. The result is normal if and only if F is continuous: succ is not
bounded above by any normal function, by Normal_imp_fp_Unbounded.

definition
  normalize :: "[i⇒i, i] ⇒ i" where
    "normalize(F,a) ≡ transrec2(a, F(0), λx r. F(succ(x)) ∪ succ(r))"

lemma Ord_normalize [simp, intro]:
  "[[Ord(a); ∧x. Ord(x) ==> Ord(F(x))]] ==> Ord(normalize(F, a))"
⟨proof⟩

lemma normalize_increasing:
  assumes ab: "a < b" and F: "∧x. Ord(x) ==> Ord(F(x))"
  shows "normalize(F,a) < normalize(F,b)"
⟨proof⟩

theorem Normal_normalize:
  assumes "∧x. Ord(x) ==> Ord(F(x))" shows "Normal(normalize(F))"
⟨proof⟩

theorem le_normalize:
  assumes a: "Ord(a)" and coF: "cont_Ord(F)" and F: "∧x. Ord(x) ==>
Ord(F(x))"
  shows "F(a) ≤ normalize(F,a)"
⟨proof⟩

```

### 7.3 The Alephs

This is the well-known transfinite enumeration of the cardinal numbers.

**definition**

```
Aleph :: "i  $\Rightarrow$  i"  (<(<open_block notation=<prefix  $\aleph$ >> $\aleph$ _)> [90] 90)
where
  " $\aleph$ a  $\equiv$  transrec2(a, nat,  $\lambda x$  r. csucc(r))"
```

**lemma** Card\_Aleph [simp, intro]:

```
"Ord(a)  $\implies$  Card(Aleph(a))"
<proof>
```

**lemma** Aleph\_increasing:

```
assumes ab: "a < b" shows "Aleph(a) < Aleph(b)"
<proof>
```

**theorem** Normal\_Aleph: "Normal(Aleph)"

<proof>

end

## 8 The Reflection Theorem

**theory** Reflection imports Normal begin

```
lemma all_iff_not_ex_not: "( $\forall x. P(x)$ )  $\longleftrightarrow$  ( $\neg$  ( $\exists x. \neg P(x)$ ))"
<proof>
```

```
lemma ball_iff_not_bex_not: "( $\forall x \in A. P(x)$ )  $\longleftrightarrow$  ( $\neg$  ( $\exists x \in A. \neg P(x)$ ))"
<proof>
```

From the notes of A. S. Kechris, page 6, and from Andrzej Mostowski, *Constructible Sets with Applications*, North-Holland, 1969, page 23.

### 8.1 Basic Definitions

First part: the cumulative hierarchy defining the class  $M$ . To avoid handling multiple arguments, we assume that  $Mset(1)$  is closed under ordered pairing provided 1 is limit. Possibly this could be avoided: the induction hypothesis *Cl\_reflects* (in locale *ex\_reflection*) could be weakened to  $\forall y \in Mset(a). \forall z \in Mset(a). P(\langle y, z \rangle) \longleftrightarrow Q(a, \langle y, z \rangle)$ , removing most uses of *Pair\_in\_Mset*. But there isn't much point in doing so, since ultimately the *ex\_reflection* proof is packaged up using the predicate *Reflects*.

**locale** reflection =

fixes Mset and M and Reflects

assumes Mset\_mono\_le : "mono\_le\_subset(Mset)"

and Mset\_cont : "cont\_Ord(Mset)"

```

    and Pair_in_Mset : "[x ∈ Mset(a); y ∈ Mset(a); Limit(a)]
                        ⇒ ⟨x,y⟩ ∈ Mset(a)"
  defines "M(x) ≡ ∃ a. Ord(a) ∧ x ∈ Mset(a)"
    and "Reflects(Cl,P,Q) ≡ Closed_Unbounded(Cl) ∧
        (∀ a. Cl(a) ⇒ (∀ x∈Mset(a). P(x) ⇔ Q(a,x)))"
  fixes F0 — ordinal for a specific value y
  fixes FF — sup over the whole level, y ∈ Mset(a)
  fixes ClEx — Reflecting ordinals for the formula ∃ z. P
  defines "F0(P,y) ≡ μ b. (∃ z. M(z) ∧ P(⟨y,z⟩)) ⇒
        (∃ z∈Mset(b). P(⟨y,z⟩))"
    and "FF(P) ≡ λa. ⋃ y∈Mset(a). F0(P,y)"
    and "ClEx(P,a) ≡ Limit(a) ∧ normalize(FF(P),a) = a"

```

begin

```

lemma Mset_mono: "i ≤ j ⇒ Mset(i) ⊆ Mset(j)"
  ⟨proof⟩

```

Awkward: we need a version of `ClEx_def` as an equality at the level of classes, which do not really exist

```

lemma ClEx_eq:
  "ClEx(P) ≡ λa. Limit(a) ∧ normalize(FF(P),a) = a"
  ⟨proof⟩

```

## 8.2 Easy Cases of the Reflection Theorem

```

theorem Triv_reflection [intro]:
  "Reflects(Ord, P, λa x. P(x))"
  ⟨proof⟩

```

```

theorem Not_reflection [intro]:
  "Reflects(Cl,P,Q) ⇒ Reflects(Cl, λx. ¬P(x), λa x. ¬Q(a,x))"
  ⟨proof⟩

```

```

theorem And_reflection [intro]:
  "[Reflects(Cl,P,Q); Reflects(C',P',Q')]"
  ⇒ Reflects(λa. Cl(a) ∧ C'(a), λx. P(x) ∧ P'(x),
    λa x. Q(a,x) ∧ Q'(a,x))"
  ⟨proof⟩

```

```

theorem Or_reflection [intro]:
  "[Reflects(Cl,P,Q); Reflects(C',P',Q')]"
  ⇒ Reflects(λa. Cl(a) ∧ C'(a), λx. P(x) ∨ P'(x),
    λa x. Q(a,x) ∨ Q'(a,x))"
  ⟨proof⟩

```

```

theorem Imp_reflection [intro]:
  "[Reflects(Cl,P,Q); Reflects(C',P',Q')]"
  ⇒ Reflects(λa. Cl(a) ∧ C'(a),

```

$\lambda x. P(x) \longrightarrow P'(x),$   
 $\lambda a \ x. Q(a,x) \longrightarrow Q'(a,x))"$

*<proof>*

**theorem** *Iff\_reflection* [intro]:  
 "⟦Reflects(Cl,P,Q); Reflects(C',P',Q')⟧  
 $\implies$  Reflects( $\lambda a. Cl(a) \wedge C'(a),$   
 $\lambda x. P(x) \longleftrightarrow P'(x),$   
 $\lambda a \ x. Q(a,x) \longleftrightarrow Q'(a,x))"$

*<proof>*

### 8.3 Reflection for Existential Quantifiers

**lemma** *F0\_works*:  
 "⟦ $y \in Mset(a); Ord(a); M(z); P(\langle y,z \rangle)$ ⟧  $\implies \exists z \in Mset(F0(P,y)). P(\langle y,z \rangle)"$   
*<proof>*

**lemma** *Ord\_F0* [intro,simp]: "*Ord*(*F0*(*P*,*y*))"

*<proof>*

**lemma** *Ord\_FF* [intro,simp]: "*Ord*(*FF*(*P*,*y*))"

*<proof>*

**lemma** *cont\_Ord\_FF*: "*cont\_Ord*(*FF*(*P*))"

*<proof>*

Recall that *F0* depends upon  $y \in Mset(a)$ , while *FF* depends only upon *a*.

**lemma** *FF\_works*:  
 "⟦ $M(z); y \in Mset(a); P(\langle y,z \rangle); Ord(a)$ ⟧  $\implies \exists z \in Mset(FF(P,a)). P(\langle y,z \rangle)"$   
*<proof>*

**lemma** *FFN\_works*:  
 "⟦ $M(z); y \in Mset(a); P(\langle y,z \rangle); Ord(a)$ ⟧  
 $\implies \exists z \in Mset(normalize(FF(P),a)). P(\langle y,z \rangle)"$   
*<proof>*

**end**

Locale for the induction hypothesis

**locale** *ex\_reflection* = *reflection* +  
**fixes** *P* — the original formula  
**fixes** *Q* — the reflected formula  
**fixes** *Cl* — the class of reflecting ordinals  
**assumes** *Cl\_reflects*: "⟦*Cl*(*a*); *Ord*(*a*)⟧  $\implies \forall x \in Mset(a). P(x) \longleftrightarrow Q(a,x)"$

**begin**

**lemma** *ClEx\_downward*:  
 "⟦ $M(z); y \in Mset(a); P(\langle y,z \rangle); Cl(a); ClEx(P,a)$ ⟧



$\implies \exists z \in \text{Mset}(a). Q(a, \langle y, z \rangle)$ "  
 $\langle \text{proof} \rangle$

**lemma** *ClEx\_upward*:  
 $\llbracket z \in \text{Mset}(a); y \in \text{Mset}(a); Q(a, \langle y, z \rangle); \text{Cl}(a); \text{ClEx}(P, a) \rrbracket$   
 $\implies \exists z. M(z) \wedge P(\langle y, z \rangle)$ "  
 $\langle \text{proof} \rangle$

Class *ClEx* indeed consists of reflecting ordinals...

**lemma** *ZF\_ClEx\_iff*:  
 $\llbracket y \in \text{Mset}(a); \text{Cl}(a); \text{ClEx}(P, a) \rrbracket$   
 $\implies (\exists z. M(z) \wedge P(\langle y, z \rangle)) \longleftrightarrow (\exists z \in \text{Mset}(a). Q(a, \langle y, z \rangle))$ "  
 $\langle \text{proof} \rangle$

...and it is closed and unbounded

**lemma** *ZF\_Closed\_Unbounded\_ClEx*:  
 $\text{"Closed\_Unbounded}(\text{ClEx}(P))$ "  
 $\langle \text{proof} \rangle$

**end**

The same two theorems, exported to locale *reflection*.

**context** *reflection*  
**begin**

Class *ClEx* indeed consists of reflecting ordinals...

**lemma** *ClEx\_iff*:  
 $\llbracket y \in \text{Mset}(a); \text{Cl}(a); \text{ClEx}(P, a);$   
 $\bigwedge a. \llbracket \text{Cl}(a); \text{Ord}(a) \rrbracket \implies \forall x \in \text{Mset}(a). P(x) \longleftrightarrow Q(a, x) \rrbracket$   
 $\implies (\exists z. M(z) \wedge P(\langle y, z \rangle)) \longleftrightarrow (\exists z \in \text{Mset}(a). Q(a, \langle y, z \rangle))$ "  
 $\langle \text{proof} \rangle$

**lemma** *Closed\_Unbounded\_ClEx*:  
 $\llbracket \bigwedge a. \llbracket \text{Cl}(a); \text{Ord}(a) \rrbracket \implies \forall x \in \text{Mset}(a). P(x) \longleftrightarrow Q(a, x) \rrbracket$   
 $\implies \text{Closed\_Unbounded}(\text{ClEx}(P))$ "  
 $\langle \text{proof} \rangle$

## 8.4 Packaging the Quantifier Reflection Rules

**lemma** *Ex\_reflection\_0*:  
 $\text{"Reflects}(\text{Cl}, P0, Q0)$   
 $\implies \text{Reflects}(\lambda a. \text{Cl}(a) \wedge \text{ClEx}(P0, a),$   
 $\lambda x. \exists z. M(z) \wedge P0(\langle x, z \rangle),$   
 $\lambda a x. \exists z \in \text{Mset}(a). Q0(a, \langle x, z \rangle))$ "  
 $\langle \text{proof} \rangle$

**lemma** *All\_reflection\_0*:

```

"Reflects(Cl,PO,QO)
 $\implies \text{Reflects}(\lambda a. \text{Cl}(a) \wedge \text{ClEx}(\lambda x. \neg \text{PO}(x), a),$ 
 $\lambda x. \forall z. M(z) \longrightarrow \text{PO}(\langle x, z \rangle),$ 
 $\lambda a \ x. \forall z \in \text{Mset}(a). \text{QO}(a, \langle x, z \rangle))"$ 
<proof>

theorem Ex_reflection [intro]:
"Reflects(Cl,  $\lambda x. P(\text{fst}(x), \text{snd}(x))$ ,  $\lambda a \ x. Q(a, \text{fst}(x), \text{snd}(x))$ )
 $\implies \text{Reflects}(\lambda a. \text{Cl}(a) \wedge \text{ClEx}(\lambda x. P(\text{fst}(x), \text{snd}(x))), a),$ 
 $\lambda x. \exists z. M(z) \wedge P(x, z),$ 
 $\lambda a \ x. \exists z \in \text{Mset}(a). Q(a, x, z))"$ 
<proof>

theorem All_reflection [intro]:
"Reflects(Cl,  $\lambda x. P(\text{fst}(x), \text{snd}(x))$ ,  $\lambda a \ x. Q(a, \text{fst}(x), \text{snd}(x))$ )
 $\implies \text{Reflects}(\lambda a. \text{Cl}(a) \wedge \text{ClEx}(\lambda x. \neg P(\text{fst}(x), \text{snd}(x))), a),$ 
 $\lambda x. \forall z. M(z) \longrightarrow P(x, z),$ 
 $\lambda a \ x. \forall z \in \text{Mset}(a). Q(a, x, z))"$ 
<proof>

And again, this time using class-bounded quantifiers

theorem Rex_reflection [intro]:
"Reflects(Cl,  $\lambda x. P(\text{fst}(x), \text{snd}(x))$ ,  $\lambda a \ x. Q(a, \text{fst}(x), \text{snd}(x))$ )
 $\implies \text{Reflects}(\lambda a. \text{Cl}(a) \wedge \text{ClEx}(\lambda x. P(\text{fst}(x), \text{snd}(x))), a),$ 
 $\lambda x. \exists z[M]. P(x, z),$ 
 $\lambda a \ x. \exists z \in \text{Mset}(a). Q(a, x, z))"$ 
<proof>

theorem Rall_reflection [intro]:
"Reflects(Cl,  $\lambda x. P(\text{fst}(x), \text{snd}(x))$ ,  $\lambda a \ x. Q(a, \text{fst}(x), \text{snd}(x))$ )
 $\implies \text{Reflects}(\lambda a. \text{Cl}(a) \wedge \text{ClEx}(\lambda x. \neg P(\text{fst}(x), \text{snd}(x))), a),$ 
 $\lambda x. \forall z[M]. P(x, z),$ 
 $\lambda a \ x. \forall z \in \text{Mset}(a). Q(a, x, z))"$ 
<proof>

```

No point considering bounded quantifiers, where reflection is trivial.

## 8.5 Simple Examples of Reflection

Example 1: reflecting a simple formula. The reflecting class is first given as the variable `?Cl` and later retrieved from the final proof state.

```

schematic_goal
"Reflects(?Cl,
 $\lambda x. \exists y. M(y) \wedge x \in y,$ 
 $\lambda a \ x. \exists y \in \text{Mset}(a). x \in y)"$ 
<proof>

```

Problem here: there needs to be a conjunction (class intersection) in the class of reflecting ordinals. The `Ord(a)` is redundant, though harmless.

**lemma**

```
"Reflects( $\lambda a. \text{Ord}(a) \wedge \text{ClEx}(\lambda x. \text{fst}(x) \in \text{snd}(x), a),$ 
 $\lambda x. \exists y. M(y) \wedge x \in y,$ 
 $\lambda a x. \exists y \in \text{Mset}(a). x \in y)$ "
```

$\langle \text{proof} \rangle$

Example 2

**schematic\_goal**

```
"Reflects( $?Cl,$ 
 $\lambda x. \exists y. M(y) \wedge (\forall z. M(z) \longrightarrow z \subseteq x \longrightarrow z \in y),$ 
 $\lambda a x. \exists y \in \text{Mset}(a). \forall z \in \text{Mset}(a). z \subseteq x \longrightarrow z \in y)$ "
```

$\langle \text{proof} \rangle$

Example 2'. We give the reflecting class explicitly.

**lemma**

```
"Reflects
( $\lambda a. (\text{Ord}(a) \wedge$ 
 $\text{ClEx}(\lambda x. \neg (\text{snd}(x) \subseteq \text{fst}(\text{fst}(x)) \longrightarrow \text{snd}(x) \in \text{snd}(\text{fst}(x))),$ 
 $a)) \wedge$ 
 $\text{ClEx}(\lambda x. \forall z. M(z) \longrightarrow z \subseteq \text{fst}(x) \longrightarrow z \in \text{snd}(x), a),$ 
 $\lambda x. \exists y. M(y) \wedge (\forall z. M(z) \longrightarrow z \subseteq x \longrightarrow z \in y),$ 
 $\lambda a x. \exists y \in \text{Mset}(a). \forall z \in \text{Mset}(a). z \subseteq x \longrightarrow z \in y)$ "
```

$\langle \text{proof} \rangle$

Example 2". We expand the subset relation.

**schematic\_goal**

```
"Reflects( $?Cl,$ 
 $\lambda x. \exists y. M(y) \wedge (\forall z. M(z) \longrightarrow (\forall w. M(w) \longrightarrow w \in z \longrightarrow w \in x) \longrightarrow$ 
 $z \in y),$ 
 $\lambda a x. \exists y \in \text{Mset}(a). \forall z \in \text{Mset}(a). (\forall w \in \text{Mset}(a). w \in z \longrightarrow w \in x) \longrightarrow$ 
 $z \in y)$ "
```

$\langle \text{proof} \rangle$

Example 2'''. Single-step version, to reveal the reflecting class.

**schematic\_goal**

```
"Reflects( $?Cl,$ 
 $\lambda x. \exists y. M(y) \wedge (\forall z. M(z) \longrightarrow z \subseteq x \longrightarrow z \in y),$ 
 $\lambda a x. \exists y \in \text{Mset}(a). \forall z \in \text{Mset}(a). z \subseteq x \longrightarrow z \in y)$ "
```

$\langle \text{proof} \rangle$

Example 3. Warning: the following examples make sense only if  $P$  is quantifier-free, since it is not being relativized.

**schematic\_goal**

```
"Reflects( $?Cl,$ 
 $\lambda x. \exists y. M(y) \wedge (\forall z. M(z) \longrightarrow z \in y \longleftrightarrow z \in x \wedge P(z)),$ 
 $\lambda a x. \exists y \in \text{Mset}(a). \forall z \in \text{Mset}(a). z \in y \longleftrightarrow z \in x \wedge P(z))"$ 
```

$\langle \text{proof} \rangle$

Example 3'

```

schematic_goal
  "Reflects(?Cl,
     $\lambda x. \exists y. M(y) \wedge y = \text{Collect}(x,P),$ 
     $\lambda a x. \exists y \in \text{Mset}(a). y = \text{Collect}(x,P))"$ 
  <proof>

```

Example 3"

```

schematic_goal
  "Reflects(?Cl,
     $\lambda x. \exists y. M(y) \wedge y = \text{Replace}(x,P),$ 
     $\lambda a x. \exists y \in \text{Mset}(a). y = \text{Replace}(x,P))"$ 
  <proof>

```

Example 4: Axiom of Choice. Possibly wrong, since  $\Pi$  needs to be relativized.

```

schematic_goal
  "Reflects(?Cl,
     $\lambda A. 0 \notin A \longrightarrow (\exists f. M(f) \wedge f \in (\prod X \in A. X)),$ 
     $\lambda a A. 0 \notin A \longrightarrow (\exists f \in \text{Mset}(a). f \in (\prod X \in A. X)))"$ 
  <proof>

```

**end**

**end**

## 9 The meta-existential quantifier

**theory** *MetaExists* **imports** *ZF* **begin**

Allows quantification over any term. Used to quantify over classes. Yields a proposition rather than a FOL formula.

**definition**

```

  ex :: "('a::t)  $\Rightarrow$  prop)  $\Rightarrow$  prop" (binder < $\bigvee$ > 0) where
  "ex(P)  $\equiv$  ( $\bigwedge Q. (\bigwedge x. \text{PROP } P(x) \Longrightarrow \text{PROP } Q) \Longrightarrow \text{PROP } Q$ )"
```

```

lemma meta_exI: "PROP P(x)  $\Longrightarrow$  ( $\bigvee x. \text{PROP } P(x)$ )"
  <proof>

```

```

lemma meta_exE: " $\llbracket \bigvee x. \text{PROP } P(x); \bigwedge x. \text{PROP } P(x) \Longrightarrow \text{PROP } R \rrbracket \Longrightarrow \text{PROP } R$ "
  <proof>

```

**end**

## 10 The ZF Axioms (Except Separation) in L

**theory** *L\_axioms* **imports** *Formula Relative Reflection MetaExists* **begin**

The class L satisfies the premises of locale *M\_trivial*

**lemma** *transL*: " $\llbracket y \in x; L(x) \rrbracket \implies L(y)$ "  
 $\langle proof \rangle$

**lemma** *nonempty*: " $L(0)$ "  
 $\langle proof \rangle$

**theorem** *upair\_ax*: "*upair\_ax*( $L$ )"  
 $\langle proof \rangle$

**theorem** *Union\_ax*: "*Union\_ax*( $L$ )"  
 $\langle proof \rangle$

**theorem** *power\_ax*: "*power\_ax*( $L$ )"  
 $\langle proof \rangle$

We don't actually need  $L$  to satisfy the foundation axiom.

**theorem** *foundation\_ax*: "*foundation\_ax*( $L$ )"  
 $\langle proof \rangle$

## 10.1 For $L$ to satisfy Replacement

**lemma** *LReplace\_in\_Lset*:  
 $\llbracket X \in Lset(i); \text{univalent}(L, X, Q); \text{Ord}(i) \rrbracket$   
 $\implies \exists j. \text{Ord}(j) \wedge \text{Replace}(X, \lambda x y. Q(x, y) \wedge L(y)) \subseteq Lset(j)$   
 $\langle proof \rangle$

**lemma** *LReplace\_in\_L*:  
 $\llbracket L(X); \text{univalent}(L, X, Q) \rrbracket$   
 $\implies \exists Y. L(Y) \wedge \text{Replace}(X, \lambda x y. Q(x, y) \wedge L(y)) \subseteq Y$   
 $\langle proof \rangle$

**theorem** *replacement*: "*replacement*( $L, P$ )"  
 $\langle proof \rangle$

**lemma** *strong\_replacementI* [*rule\_format*]:  
 $\llbracket \forall B[L]. \text{separation}(L, \lambda u. \exists x[L]. x \in B \wedge P(x, u)) \rrbracket$   
 $\implies \text{strong\_replacement}(L, P)$   
 $\langle proof \rangle$

## 10.2 Instantiating the locale $M_{\text{trivial}}$

No instances of Separation yet.

**lemma** *Lset\_mono\_le*: "*mono\_le\_subset*( $Lset$ )"  
 $\langle proof \rangle$

**lemma** *Lset\_cont*: "*cont\_Ord*( $Lset$ )"  
 $\langle proof \rangle$

**lemmas**  $L_{\text{nat}} = \text{Ord\_in\_L} \text{ [OF } \text{Ord\_nat}]$

**theorem** *M\_trivial\_L*: "*M\_trivial*(*L*)"  
 ⟨*proof*⟩

**interpretation** *L*: *M\_trivial* *L* ⟨*proof*⟩

### 10.3 Instantiation of the locale *reflection*

instances of locale constants

**definition**

*L\_F0* :: "[*i*⇒*o*,*i*] ⇒ *i*" **where**  
 "*L\_F0*(*P*,*y*) ≡ μ *b*. (∃ *z*. *L*(*z*) ∧ *P*(⟨*y*,*z*⟩)) → (∃ *z*∈*Lset*(*b*). *P*(⟨*y*,*z*⟩))"

**definition**

*L\_FF* :: "[*i*⇒*o*,*i*] ⇒ *i*" **where**  
 "*L\_FF*(*P*) ≡ λ*a*. ⋃*y*∈*Lset*(*a*). *L\_F0*(*P*,*y*)"

**definition**

*L\_ClEx* :: "[*i*⇒*o*,*i*] ⇒ *o*" **where**  
 "*L\_ClEx*(*P*) ≡ λ*a*. *Limit*(*a*) ∧ *normalize*(*L\_FF*(*P*),*a*) = *a*"

We must use the meta-existential quantifier; otherwise the reflection terms become enormous!

**definition**

*L\_Reflects* :: "[*i*⇒*o*, [*i*,*i*]⇒*o*] ⇒ *prop*" (⟨(3REFLECTS/ [\_,/ \_])⟩) **where**  
 "*REFLECTS*[*P*,*Q*] ≡ (⋀ *Cl*. *Closed\_Unbounded*(*Cl*) ∧  
 (⋀ *a*. *Cl*(*a*) → (⋀ *x* ∈ *Lset*(*a*). *P*(*x*) ↔ *Q*(*a*,*x*))))"

**theorem** *Triv\_reflection*:

"*REFLECTS*[*P*, λ*a* *x*. *P*(*x*)]"  
 ⟨*proof*⟩

**theorem** *Not\_reflection*:

"*REFLECTS*[*P*,*Q*] ⇒ *REFLECTS*[λ*x*. ¬*P*(*x*), λ*a* *x*. ¬*Q*(*a*,*x*)]"  
 ⟨*proof*⟩

**theorem** *And\_reflection*:

"[*REFLECTS*[*P*,*Q*]; *REFLECTS*[*P'*,*Q'*]]  
 ⇒ *REFLECTS*[λ*x*. *P*(*x*) ∧ *P'*(*x*), λ*a* *x*. *Q*(*a*,*x*) ∧ *Q'*(*a*,*x*)]"  
 ⟨*proof*⟩

**theorem** *Or\_reflection*:

"[*REFLECTS*[*P*,*Q*]; *REFLECTS*[*P'*,*Q'*]]  
 ⇒ *REFLECTS*[λ*x*. *P*(*x*) ∨ *P'*(*x*), λ*a* *x*. *Q*(*a*,*x*) ∨ *Q'*(*a*,*x*)]"  
 ⟨*proof*⟩

**theorem** *Imp\_reflection*:

```

"[[REFLECTS[P,Q]; REFLECTS[P',Q']]]
  ==> REFLECTS[λx. P(x) → P'(x), λa x. Q(a,x) → Q'(a,x)]"
⟨proof⟩

```

**theorem** *Iff\_reflection*:

```

"[[REFLECTS[P,Q]; REFLECTS[P',Q']]]
  ==> REFLECTS[λx. P(x) ↔ P'(x), λa x. Q(a,x) ↔ Q'(a,x)]"
⟨proof⟩

```

**lemma** *reflection\_Lset*: "reflection(Lset)"

⟨proof⟩

**theorem** *Ex\_reflection*:

```

"REFLECTS[λx. P(fst(x),snd(x)), λa x. Q(a,fst(x),snd(x))]
  ==> REFLECTS[λx. ∃z. L(z) ∧ P(x,z), λa x. ∃z∈Lset(a). Q(a,x,z)]"
⟨proof⟩

```

**theorem** *All\_reflection*:

```

"REFLECTS[λx. P(fst(x),snd(x)), λa x. Q(a,fst(x),snd(x))]
  ==> REFLECTS[λx. ∀z. L(z) → P(x,z), λa x. ∀z∈Lset(a). Q(a,x,z)]"
⟨proof⟩

```

**theorem** *Rex\_reflection*:

```

"REFLECTS[λx. P(fst(x),snd(x)), λa x. Q(a,fst(x),snd(x))]
  ==> REFLECTS[λx. ∃z[L]. P(x,z), λa x. ∃z∈Lset(a). Q(a,x,z)]"
⟨proof⟩

```

**theorem** *Rall\_reflection*:

```

"REFLECTS[λx. P(fst(x),snd(x)), λa x. Q(a,fst(x),snd(x))]
  ==> REFLECTS[λx. ∀z[L]. P(x,z), λa x. ∀z∈Lset(a). Q(a,x,z)]"
⟨proof⟩

```

This version handles an alternative form of the bounded quantifier in the second argument of *REFLECTS*.

**theorem** *Rex\_reflection'*:

```

"REFLECTS[λx. P(fst(x),snd(x)), λa x. Q(a,fst(x),snd(x))]
  ==> REFLECTS[λx. ∃z[L]. P(x,z), λa x. ∃z[##Lset(a)]. Q(a,x,z)]"
⟨proof⟩

```

As above.

**theorem** *Rall\_reflection'*:

```

"REFLECTS[λx. P(fst(x),snd(x)), λa x. Q(a,fst(x),snd(x))]
  ==> REFLECTS[λx. ∀z[L]. P(x,z), λa x. ∀z[##Lset(a)]. Q(a,x,z)]"
⟨proof⟩

```

**lemmas** *FOL\_reflections* =

*Triv\_reflection Not\_reflection And\_reflection Or\_reflection*

*Imp\_reflection Iff\_reflection Ex\_reflection All\_reflection  
 Rex\_reflection Rall\_reflection Rex\_reflection' Rall\_reflection'*

**lemma** *ReflectsD*:  
 "REFLECTS[P,Q]; Ord(i)  
 $\implies \exists j. i < j \wedge (\forall x \in \text{Lset}(j). P(x) \longleftrightarrow Q(j,x))$ "  
 <proof>

**lemma** *ReflectsE*:  
 "REFLECTS[P,Q]; Ord(i);  
 $\bigwedge j. [i < j; \forall x \in \text{Lset}(j). P(x) \longleftrightarrow Q(j,x)] \implies R$ "  
 <proof>

**lemma** *Collect\_mem\_eq*: "{x ∈ A. x ∈ B} = A ∩ B"  
 <proof>

## 10.4 Internalized Formulas for some Set-Theoretic Concepts

### 10.4.1 Some numbers to help write de Bruijn indices

**abbreviation**  
*digit3* :: i    (<3>) where "3 ≡ succ(2)"

**abbreviation**  
*digit4* :: i    (<4>) where "4 ≡ succ(3)"

**abbreviation**  
*digit5* :: i    (<5>) where "5 ≡ succ(4)"

**abbreviation**  
*digit6* :: i    (<6>) where "6 ≡ succ(5)"

**abbreviation**  
*digit7* :: i    (<7>) where "7 ≡ succ(6)"

**abbreviation**  
*digit8* :: i    (<8>) where "8 ≡ succ(7)"

**abbreviation**  
*digit9* :: i    (<9>) where "9 ≡ succ(8)"

### 10.4.2 The Empty Set, Internalized

**definition**  
*empty\_fm* :: "i ⇒ i" where  
 "empty\_fm(x) ≡ Forall(Neg(Member(0,succ(x))))"

**lemma** *empty\_type* [TC]:  
 "x ∈ nat  $\implies$  empty\_fm(x) ∈ formula"



*<proof>*

**lemma** *sats\_empty\_fm [simp]:*  
"[[ $x \in \text{nat}; \text{env} \in \text{list}(A)$ ]]  
 $\implies \text{sats}(A, \text{empty\_fm}(x), \text{env}) \longleftrightarrow \text{empty}(\#\#A, \text{nth}(x, \text{env}))$ "  
*<proof>*

**lemma** *empty\_iff\_sats:*  
"[[ $\text{nth}(i, \text{env}) = x; \text{nth}(j, \text{env}) = y;$   
 $i \in \text{nat}; \text{env} \in \text{list}(A)$ ]]  
 $\implies \text{empty}(\#\#A, x) \longleftrightarrow \text{sats}(A, \text{empty\_fm}(i), \text{env})$ "  
*<proof>*

**theorem** *empty\_reflection:*  
"REFLECTS[ $\lambda x. \text{empty}(L, f(x)),$   
 $\lambda i x. \text{empty}(\#\#L\text{set}(i), f(x))$ ]"  
*<proof>*

Not used. But maybe useful?

**lemma** *Transset\_sats\_empty\_fm\_eq\_0:*  
"[[ $n \in \text{nat}; \text{env} \in \text{list}(A); \text{Transset}(A)$ ]]  
 $\implies \text{sats}(A, \text{empty\_fm}(n), \text{env}) \longleftrightarrow \text{nth}(n, \text{env}) = 0$ "  
*<proof>*

### 10.4.3 Unordered Pairs, Internalized

**definition**  
*upair\_fm* :: "[ $i, i, i$ ] $\Rightarrow i$ " where  
"upair\_fm( $x, y, z$ )  $\equiv$   
And(Member( $x, z$ ),  
And(Member( $y, z$ ),  
Forall(Implies(Member(0, succ( $z$ )),  
Or(Equal(0, succ( $x$ )), Equal(0, succ( $y$ )))))))"

**lemma** *upair\_type [TC]:*  
"[[ $x \in \text{nat}; y \in \text{nat}; z \in \text{nat}$ ]]  $\implies \text{upair\_fm}(x, y, z) \in \text{formula}$ "  
*<proof>*

**lemma** *sats\_upair\_fm [simp]:*  
"[[ $x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; \text{env} \in \text{list}(A)$ ]]  
 $\implies \text{sats}(A, \text{upair\_fm}(x, y, z), \text{env}) \longleftrightarrow$   
 $\text{upair}(\#\#A, \text{nth}(x, \text{env}), \text{nth}(y, \text{env}), \text{nth}(z, \text{env}))$ "  
*<proof>*

**lemma** *upair\_iff\_sats:*  
"[[ $\text{nth}(i, \text{env}) = x; \text{nth}(j, \text{env}) = y; \text{nth}(k, \text{env}) = z;$   
 $i \in \text{nat}; j \in \text{nat}; k \in \text{nat}; \text{env} \in \text{list}(A)$ ]]  
 $\implies \text{upair}(\#\#A, x, y, z) \longleftrightarrow \text{sats}(A, \text{upair\_fm}(i, j, k), \text{env})$ "  
*<proof>*

Useful? At least it refers to "real" unordered pairs

```
lemma sats_upair_fm2 [simp]:
  "[x ∈ nat; y ∈ nat; z < length(env); env ∈ list(A); Transset(A)]
  ⇒ sats(A, upair_fm(x,y,z), env) ⇔
    nth(z,env) = {nth(x,env), nth(y,env)}"
⟨proof⟩
```

```
theorem upair_reflection:
  "REFLECTS[λx. upair(L,f(x),g(x),h(x)),
    λi x. upair(##Lset(i),f(x),g(x),h(x))]"
⟨proof⟩
```

#### 10.4.4 Ordered pairs, Internalized

**definition**

```
pair_fm :: "[i,i,i]⇒i" where
  "pair_fm(x,y,z) ≡
    Exists(And(upair_fm(succ(x),succ(x),0),
      Exists(And(upair_fm(succ(succ(x)),succ(succ(y)),0),
        upair_fm(1,0,succ(succ(z)))))))"
```

```
lemma pair_type [TC]:
  "[x ∈ nat; y ∈ nat; z ∈ nat] ⇒ pair_fm(x,y,z) ∈ formula"
⟨proof⟩
```

```
lemma sats_pair_fm [simp]:
  "[x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)]
  ⇒ sats(A, pair_fm(x,y,z), env) ⇔
    pair(##A, nth(x,env), nth(y,env), nth(z,env))"
⟨proof⟩
```

```
lemma pair_iff_sats:
  "[nth(i,env) = x; nth(j,env) = y; nth(k,env) = z;
    i ∈ nat; j ∈ nat; k ∈ nat; env ∈ list(A)]
  ⇒ pair(##A, x, y, z) ⇔ sats(A, pair_fm(i,j,k), env)"
⟨proof⟩
```

```
theorem pair_reflection:
  "REFLECTS[λx. pair(L,f(x),g(x),h(x)),
    λi x. pair(##Lset(i),f(x),g(x),h(x))]"
⟨proof⟩
```

#### 10.4.5 Binary Unions, Internalized

**definition**

```
union_fm :: "[i,i,i]⇒i" where
  "union_fm(x,y,z) ≡
    Forall(Iff(Member(0,succ(z)),
      Or(Member(0,succ(x)),Member(0,succ(y)))))"
```

**lemma union\_type [TC]:**  
 " $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \implies \text{union\_fm}(x,y,z) \in \text{formula}$ "  
 <proof>

**lemma sats\_union\_fm [simp]:**  
 " $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{sats}(A, \text{union\_fm}(x,y,z), \text{env}) \longleftrightarrow$   
 $\text{union}(\#\#A, \text{nth}(x,\text{env}), \text{nth}(y,\text{env}), \text{nth}(z,\text{env}))$ "  
 <proof>

**lemma union\_iff\_sats:**  
 " $\llbracket \text{nth}(i,\text{env}) = x; \text{nth}(j,\text{env}) = y; \text{nth}(k,\text{env}) = z;$   
 $i \in \text{nat}; j \in \text{nat}; k \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{union}(\#\#A, x, y, z) \longleftrightarrow \text{sats}(A, \text{union\_fm}(i,j,k), \text{env})$ "  
 <proof>

**theorem union\_reflection:**  
 "REFLECTS[ $\lambda x. \text{union}(L, f(x), g(x), h(x)),$   
 $\lambda i x. \text{union}(\#\#L\text{set}(i), f(x), g(x), h(x))$ ]"  
 <proof>

#### 10.4.6 Set "Cons," Internalized

**definition**  
 cons\_fm :: "[i,i,i] $\Rightarrow$ i" where  
 "cons\_fm(x,y,z)  $\equiv$   
 Exists(And(upair\_fm(succ(x),succ(x),0),  
 union\_fm(0,succ(y),succ(z))))"

**lemma cons\_type [TC]:**  
 " $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \implies \text{cons\_fm}(x,y,z) \in \text{formula}$ "  
 <proof>

**lemma sats\_cons\_fm [simp]:**  
 " $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{sats}(A, \text{cons\_fm}(x,y,z), \text{env}) \longleftrightarrow$   
 $\text{is\_cons}(\#\#A, \text{nth}(x,\text{env}), \text{nth}(y,\text{env}), \text{nth}(z,\text{env}))$ "  
 <proof>

**lemma cons\_iff\_sats:**  
 " $\llbracket \text{nth}(i,\text{env}) = x; \text{nth}(j,\text{env}) = y; \text{nth}(k,\text{env}) = z;$   
 $i \in \text{nat}; j \in \text{nat}; k \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{is\_cons}(\#\#A, x, y, z) \longleftrightarrow \text{sats}(A, \text{cons\_fm}(i,j,k), \text{env})$ "  
 <proof>

**theorem cons\_reflection:**  
 "REFLECTS[ $\lambda x. \text{is\_cons}(L, f(x), g(x), h(x)),$

$\lambda i \ x. \text{is\_cons}(\#\#Lset(i), f(x), g(x), h(x))]$ "

*<proof>*

#### 10.4.7 Successor Function, Internalized

**definition**

$\text{succ\_fm} :: "[i, i] \Rightarrow i"$  where  
 $\text{"succ\_fm}(x, y) \equiv \text{cons\_fm}(x, x, y)"$

**lemma**  $\text{succ\_type} \ [TC]:$

$\text{"}\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \Longrightarrow \text{succ\_fm}(x, y) \in \text{formula}"$

*<proof>*

**lemma**  $\text{sats\_succ\_fm} \ [simp]:$

$\text{"}\llbracket x \in \text{nat}; y \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\Longrightarrow \text{sats}(A, \text{succ\_fm}(x, y), \text{env}) \longleftrightarrow$   
 $\text{successor}(\#\#A, \text{nth}(x, \text{env}), \text{nth}(y, \text{env}))"$

*<proof>*

**lemma**  $\text{successor\_iff\_sats}:$

$\text{"}\llbracket \text{nth}(i, \text{env}) = x; \text{nth}(j, \text{env}) = y;$   
 $i \in \text{nat}; j \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\Longrightarrow \text{successor}(\#\#A, x, y) \longleftrightarrow \text{sats}(A, \text{succ\_fm}(i, j), \text{env})"$

*<proof>*

**theorem**  $\text{successor\_reflection}:$

$\text{"REFLECTS}[\lambda x. \text{successor}(L, f(x), g(x)),$   
 $\lambda i \ x. \text{successor}(\#\#Lset(i), f(x), g(x))]$ "

*<proof>*

#### 10.4.8 The Number 1, Internalized

**definition**

$\text{number1\_fm} :: "i \Rightarrow i"$  where  
 $\text{"number1\_fm}(a) \equiv \text{Exists}(\text{And}(\text{empty\_fm}(0), \text{succ\_fm}(0, \text{succ}(a))))"$

**lemma**  $\text{number1\_type} \ [TC]:$

$\text{"}x \in \text{nat} \Longrightarrow \text{number1\_fm}(x) \in \text{formula}"$

*<proof>*

**lemma**  $\text{sats\_number1\_fm} \ [simp]:$

$\text{"}\llbracket x \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\Longrightarrow \text{sats}(A, \text{number1\_fm}(x), \text{env}) \longleftrightarrow \text{number1}(\#\#A, \text{nth}(x, \text{env}))"$

*<proof>*

**lemma**  $\text{number1\_iff\_sats}:$

$\text{"}\llbracket \text{nth}(i, \text{env}) = x; \text{nth}(j, \text{env}) = y;$   
 $i \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\Longrightarrow \text{number1}(\#\#A, x) \longleftrightarrow \text{sats}(A, \text{number1\_fm}(i), \text{env})"$

*<proof>*

```

theorem number1_reflection:
  "REFLECTS[ $\lambda x. \text{number1}(L, f(x)),$ 
              $\lambda i x. \text{number1}(\#\text{Lset}(i), f(x))]$ "
<proof>

```

#### 10.4.9 Big Union, Internalized

**definition**

```

big_union_fm :: "[i,i] $\Rightarrow$ i" where
  "big_union_fm(A,z)  $\equiv$ 
    Forall(Iff(Member(0,succ(z)),
                Exists(And(Member(0,succ(succ(A))), Member(1,0))))))"

```

**lemma** big\_union\_type [TC]:

```

  " $\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \Rightarrow \text{big\_union\_fm}(x,y) \in \text{formula}$ "
<proof>

```

**lemma** sats\_big\_union\_fm [simp]:

```

  " $\llbracket x \in \text{nat}; y \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$ 
 $\Rightarrow \text{sats}(A, \text{big\_union\_fm}(x,y), \text{env}) \longleftrightarrow$ 
    big_union( $\#\text{A}$ , nth(x,env), nth(y,env))"
<proof>

```

**lemma** big\_union\_iff\_sats:

```

  " $\llbracket \text{nth}(i,\text{env}) = x; \text{nth}(j,\text{env}) = y;
    i \in \text{nat}; j \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$ 
 $\Rightarrow \text{big\_union}(\#\text{A}, x, y) \longleftrightarrow \text{sats}(A, \text{big\_union\_fm}(i,j), \text{env})$ "
<proof>

```

**theorem** big\_union\_reflection:

```

  "REFLECTS[ $\lambda x. \text{big\_union}(L, f(x), g(x)),$ 
              $\lambda i x. \text{big\_union}(\#\text{Lset}(i), f(x), g(x))]$ "
<proof>

```

#### 10.4.10 Variants of Satisfaction Definitions for Ordinals, etc.

The *sats* theorems below are standard versions of the ones proved in theory *Formula*. They relate elements of type *formula* to relativized concepts such as *subset* or *ordinal* rather than to real concepts such as *Ord*. Now that we have instantiated the locale *M\_trivial*, we no longer require the earlier versions.

**lemma** sats\_subset\_fm':

```

  " $\llbracket x \in \text{nat}; y \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$ 
 $\Rightarrow \text{sats}(A, \text{subset\_fm}(x,y), \text{env}) \longleftrightarrow \text{subset}(\#\text{A}, \text{nth}(x,\text{env}), \text{nth}(y,\text{env}))$ "
<proof>

```

**theorem** subset\_reflection:

```

    "REFLECTS[ $\lambda x. \text{subset}(L, f(x), g(x)),$ 
       $\lambda i x. \text{subset}(\text{\#\#Lset}(i), f(x), g(x))]$ "
  <proof>

lemma sats_transset_fm':
  "[ $x \in \text{nat}; \text{env} \in \text{list}(A)$ ]"
   $\implies \text{sats}(A, \text{transset\_fm}(x), \text{env}) \longleftrightarrow \text{transitive\_set}(\text{\#\#A}, \text{nth}(x, \text{env}))$ "
  <proof>

theorem transitive_set_reflection:
  "REFLECTS[ $\lambda x. \text{transitive\_set}(L, f(x)),$ 
     $\lambda i x. \text{transitive\_set}(\text{\#\#Lset}(i), f(x))]$ "
  <proof>

lemma sats_ordinal_fm':
  "[ $x \in \text{nat}; \text{env} \in \text{list}(A)$ ]"
   $\implies \text{sats}(A, \text{ordinal\_fm}(x), \text{env}) \longleftrightarrow \text{ordinal}(\text{\#\#A}, \text{nth}(x, \text{env}))$ "
  <proof>

lemma ordinal_iff_sats:
  "[ $\text{nth}(i, \text{env}) = x; i \in \text{nat}; \text{env} \in \text{list}(A)$ ]"
   $\implies \text{ordinal}(\text{\#\#A}, x) \longleftrightarrow \text{sats}(A, \text{ordinal\_fm}(i), \text{env})$ "
  <proof>

theorem ordinal_reflection:
  "REFLECTS[ $\lambda x. \text{ordinal}(L, f(x)), \lambda i x. \text{ordinal}(\text{\#\#Lset}(i), f(x))]$ "
  <proof>

```

#### 10.4.11 Membership Relation, Internalized

**definition**

```

Memrel_fm :: "[i,i] $\Rightarrow$ i" where
  "Memrel_fm(A,r)  $\equiv$ 
    Forall(Iff(Member(0,succ(r)),
      Exists(And(Member(0,succ(succ(A))),
        Exists(And(Member(0,succ(succ(succ(A)))),
          And(Member(1,0),
            pair_fm(1,0,2))))))))))"

```

```

lemma Memrel_type [TC]:
  "[ $x \in \text{nat}; y \in \text{nat}$ ]  $\implies \text{Memrel\_fm}(x,y) \in \text{formula}$ "
  <proof>

```

```

lemma sats_Memrel_fm [simp]:
  "[ $x \in \text{nat}; y \in \text{nat}; \text{env} \in \text{list}(A)$ ]"
   $\implies \text{sats}(A, \text{Memrel\_fm}(x,y), \text{env}) \longleftrightarrow$ 
     $\text{membership}(\text{\#\#A}, \text{nth}(x, \text{env}), \text{nth}(y, \text{env}))$ "
  <proof>

```

**lemma** *Memrel\_iff\_sats*:  
 "⟦nth(i,env) = x; nth(j,env) = y;  
   i ∈ nat; j ∈ nat; env ∈ list(A)⟧  
 ⇒ membership(##A, x, y) ⇔ sats(A, Memrel\_fm(i,j), env)"  
 <proof>

**theorem** *membership\_reflection*:  
 "REFLECTS[λx. membership(L,f(x),g(x)),  
   λi x. membership(##Lset(i),f(x),g(x))]"  
 <proof>

#### 10.4.12 Predecessor Set, Internalized

**definition**  
*pred\_set\_fm* :: "[i,i,i,i]⇒i" where  
 "pred\_set\_fm(A,x,r,B) ≡  
   Forall(Iff(Member(0,succ(B)),  
     Exists(And(Member(0,succ(succ(r))),  
       And(Member(1,succ(succ(A))),  
       pair\_fm(1,succ(succ(x)),0))))))"

**lemma** *pred\_set\_type* [TC]:  
 "⟦A ∈ nat; x ∈ nat; r ∈ nat; B ∈ nat⟧  
 ⇒ pred\_set\_fm(A,x,r,B) ∈ formula"  
 <proof>

**lemma** *sats\_pred\_set\_fm* [simp]:  
 "⟦U ∈ nat; x ∈ nat; r ∈ nat; B ∈ nat; env ∈ list(A)⟧  
 ⇒ sats(A, pred\_set\_fm(U,x,r,B), env) ⇔  
   pred\_set(##A, nth(U,env), nth(x,env), nth(r,env), nth(B,env))"  
 <proof>

**lemma** *pred\_set\_iff\_sats*:  
 "⟦nth(i,env) = U; nth(j,env) = x; nth(k,env) = r; nth(l,env) = B;  
   i ∈ nat; j ∈ nat; k ∈ nat; l ∈ nat; env ∈ list(A)⟧  
 ⇒ pred\_set(##A,U,x,r,B) ⇔ sats(A, pred\_set\_fm(i,j,k,l), env)"  
 <proof>

**theorem** *pred\_set\_reflection*:  
 "REFLECTS[λx. pred\_set(L,f(x),g(x),h(x),b(x)),  
   λi x. pred\_set(##Lset(i),f(x),g(x),h(x),b(x))]"  
 <proof>

#### 10.4.13 Domain of a Relation, Internalized

**definition**  
*domain\_fm* :: "[i,i]⇒i" where  
 "domain\_fm(r,z) ≡  
   Forall(Iff(Member(0,succ(z)),

$$\text{Exists}(\text{And}(\text{Member}(0, \text{succ}(\text{succ}(r))),$$

$$\text{Exists}(\text{pair\_fm}(2, 0, 1))))))"$$

**lemma** *domain\_type* [TC]:  

$$\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \implies \text{domain\_fm}(x, y) \in \text{formula}$$
 $\langle \text{proof} \rangle$

**lemma** *sats\_domain\_fm* [simp]:  

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

$$\implies \text{sats}(A, \text{domain\_fm}(x, y), \text{env}) \longleftrightarrow$$

$$\text{is\_domain}(\#\#A, \text{nth}(x, \text{env}), \text{nth}(y, \text{env}))"$$
 $\langle \text{proof} \rangle$

**lemma** *domain\_iff\_sats*:  

$$\llbracket \text{nth}(i, \text{env}) = x; \text{nth}(j, \text{env}) = y;$$

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

$$\implies \text{is\_domain}(\#\#A, x, y) \longleftrightarrow \text{sats}(A, \text{domain\_fm}(i, j), \text{env})"$$
 $\langle \text{proof} \rangle$

**theorem** *domain\_reflection*:  

$$\text{"REFLECTS}[\lambda x. \text{is\_domain}(L, f(x), g(x)),$$

$$\lambda i x. \text{is\_domain}(\#\#L\text{set}(i), f(x), g(x))]"$$
 $\langle \text{proof} \rangle$

#### 10.4.14 Range of a Relation, Internalized

**definition**  
 $\text{range\_fm} :: "[i, i] \Rightarrow i"$  where  

$$\text{"range\_fm}(r, z) \equiv$$

$$\text{Forall}(\text{Iff}(\text{Member}(0, \text{succ}(z)),$$

$$\text{Exists}(\text{And}(\text{Member}(0, \text{succ}(\text{succ}(r))),$$

$$\text{Exists}(\text{pair\_fm}(0, 2, 1))))))"$$

**lemma** *range\_type* [TC]:  

$$\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \implies \text{range\_fm}(x, y) \in \text{formula}$$
 $\langle \text{proof} \rangle$

**lemma** *sats\_range\_fm* [simp]:  

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

$$\implies \text{sats}(A, \text{range\_fm}(x, y), \text{env}) \longleftrightarrow$$

$$\text{is\_range}(\#\#A, \text{nth}(x, \text{env}), \text{nth}(y, \text{env}))"$$
 $\langle \text{proof} \rangle$

**lemma** *range\_iff\_sats*:  

$$\llbracket \text{nth}(i, \text{env}) = x; \text{nth}(j, \text{env}) = y;$$

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

$$\implies \text{is\_range}(\#\#A, x, y) \longleftrightarrow \text{sats}(A, \text{range\_fm}(i, j), \text{env})"$$
 $\langle \text{proof} \rangle$



```

theorem range_reflection:
  "REFLECTS[ $\lambda x. \text{is\_range}(L, f(x), g(x)),$ 
     $\lambda i x. \text{is\_range}(\#\text{Lset}(i), f(x), g(x))]$ "
  <proof>

```

#### 10.4.15 Field of a Relation, Internalized

**definition**

```

field_fm :: "[i,i] $\Rightarrow$ i" where
  "field_fm(r,z)  $\equiv$ 
    Exists(And(domain_fm(succ(r),0),
      Exists(And(range_fm(succ(succ(r)),0),
        union_fm(1,0,succ(succ(z)))))))"

```

```

lemma field_type [TC]:
  " $\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \Rightarrow \text{field\_fm}(x,y) \in \text{formula}$ "
  <proof>

```

```

lemma sats_field_fm [simp]:
  " $\llbracket x \in \text{nat}; y \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$ 
 $\Rightarrow \text{sats}(A, \text{field\_fm}(x,y), \text{env}) \longleftrightarrow$ 
  is_field( $\#\#A$ , nth(x,env), nth(y,env))"
  <proof>

```

```

lemma field_iff_sats:
  " $\llbracket \text{nth}(i,\text{env}) = x; \text{nth}(j,\text{env}) = y;
    i \in \text{nat}; j \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$ 
 $\Rightarrow \text{is\_field}(\#\#A, x, y) \longleftrightarrow \text{sats}(A, \text{field\_fm}(i,j), \text{env})$ "
  <proof>

```

```

theorem field_reflection:
  "REFLECTS[ $\lambda x. \text{is\_field}(L, f(x), g(x)),$ 
     $\lambda i x. \text{is\_field}(\#\text{Lset}(i), f(x), g(x))]$ "
  <proof>

```

#### 10.4.16 Image under a Relation, Internalized

**definition**

```

image_fm :: "[i,i,i] $\Rightarrow$ i" where
  "image_fm(r,A,z)  $\equiv$ 
    Forall(Iff(Member(0,succ(z)),
      Exists(And(Member(0,succ(succ(r))),
        Exists(And(Member(0,succ(succ(succ(A)))),
          pair_fm(0,2,1)))))))"

```

```

lemma image_type [TC]:
  " $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \Rightarrow \text{image\_fm}(x,y,z) \in \text{formula}$ "
  <proof>

```

```

lemma sats_image_fm [simp]:

```

```

"[[x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)]
  ⇒ sats(A, image_fm(x,y,z), env) ⇔
    image(##A, nth(x,env), nth(y,env), nth(z,env))]"
⟨proof⟩

```

```

lemma image_iff_sats:
  "[[nth(i,env) = x; nth(j,env) = y; nth(k,env) = z;
    i ∈ nat; j ∈ nat; k ∈ nat; env ∈ list(A)]
    ⇒ image(##A, x, y, z) ⇔ sats(A, image_fm(i,j,k), env)]"
⟨proof⟩

```

```

theorem image_reflection:
  "REFLECTS[λx. image(L,f(x),g(x),h(x)),
    λi x. image(##Lset(i),f(x),g(x),h(x))]"
⟨proof⟩

```

#### 10.4.17 Pre-Image under a Relation, Internalized

**definition**

```

pre_image_fm :: "[i,i,i]⇒i" where
  "pre_image_fm(r,A,z) ≡
    Forall(Iff(Member(0,succ(z)),
      Exists(And(Member(0,succ(succ(r))),
        Exists(And(Member(0,succ(succ(succ(A)))),
          pair_fm(2,0,1)))))))"

```

```

lemma pre_image_type [TC]:
  "[[x ∈ nat; y ∈ nat; z ∈ nat] ⇒ pre_image_fm(x,y,z) ∈ formula]"
⟨proof⟩

```

```

lemma sats_pre_image_fm [simp]:
  "[[x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)]
    ⇒ sats(A, pre_image_fm(x,y,z), env) ⇔
      pre_image(##A, nth(x,env), nth(y,env), nth(z,env))]"
⟨proof⟩

```

```

lemma pre_image_iff_sats:
  "[[nth(i,env) = x; nth(j,env) = y; nth(k,env) = z;
    i ∈ nat; j ∈ nat; k ∈ nat; env ∈ list(A)]
    ⇒ pre_image(##A, x, y, z) ⇔ sats(A, pre_image_fm(i,j,k), env)]"
⟨proof⟩

```

```

theorem pre_image_reflection:
  "REFLECTS[λx. pre_image(L,f(x),g(x),h(x)),
    λi x. pre_image(##Lset(i),f(x),g(x),h(x))]"
⟨proof⟩

```

#### 10.4.18 Function Application, Internalized

**definition**

```

fun_apply_fm :: "[i,i,i]⇒i" where
  "fun_apply_fm(f,x,y) ≡
    Exists(Exists(And(upair_fm(succ(succ(x)), succ(succ(x)), 1),
      And(image_fm(succ(succ(f)), 1, 0),
        big_union_fm(0,succ(succ(y)))))))"

lemma fun_apply_type [TC]:
  "⟦x ∈ nat; y ∈ nat; z ∈ nat⟧ ⇒ fun_apply_fm(x,y,z) ∈ formula"
  <proof>

lemma sats_fun_apply_fm [simp]:
  "⟦x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)⟧
  ⇒ sats(A, fun_apply_fm(x,y,z), env) ⟷
    fun_apply(##A, nth(x,env), nth(y,env), nth(z,env))"
  <proof>

lemma fun_apply_iff_sats:
  "⟦nth(i,env) = x; nth(j,env) = y; nth(k,env) = z;
    i ∈ nat; j ∈ nat; k ∈ nat; env ∈ list(A)⟧
  ⇒ fun_apply(##A, x, y, z) ⟷ sats(A, fun_apply_fm(i,j,k), env)"
  <proof>

theorem fun_apply_reflection:
  "REFLECTS[λx. fun_apply(L,f(x),g(x),h(x)),
    λi x. fun_apply(##Lset(i),f(x),g(x),h(x))]"
  <proof>



### 10.4.19 The Concept of Relation, Internalized



definition
  relation_fm :: "i⇒i" where
    "relation_fm(r) ≡
      Forall(Implies(Member(0,succ(r)), Exists(Exists(pair_fm(1,0,2)))))"

lemma relation_type [TC]:
  "⟦x ∈ nat⟧ ⇒ relation_fm(x) ∈ formula"
  <proof>

lemma sats_relation_fm [simp]:
  "⟦x ∈ nat; env ∈ list(A)⟧
  ⇒ sats(A, relation_fm(x), env) ⟷ is_relation(##A, nth(x,env))"
  <proof>

lemma relation_iff_sats:
  "⟦nth(i,env) = x; nth(j,env) = y;
    i ∈ nat; env ∈ list(A)⟧
  ⇒ is_relation(##A, x) ⟷ sats(A, relation_fm(i), env)"
  <proof>

```

```

theorem is_relation_reflection:
  "REFLECTS[ $\lambda x. \text{is\_relation}(L, f(x)),$ 
     $\lambda i x. \text{is\_relation}(\#\#L\text{set}(i), f(x))]$ "
  <proof>

```

#### 10.4.20 The Concept of Function, Internalized

```

definition
  function_fm :: "i  $\Rightarrow$  i" where
    "function_fm(r)  $\equiv$ 
      Forall(Forall(Forall(Forall(Forall(
        Implies(pair_fm(4,3,1),
          Implies(pair_fm(4,2,0),
            Implies(Member(1,r#+5),
              Implies(Member(0,r#+5), Equal(3,2))))))))))"

```

```

lemma function_type [TC]:
  " $\llbracket x \in \text{nat} \rrbracket \Rightarrow \text{function\_fm}(x) \in \text{formula}$ "
  <proof>

```

```

lemma sats_function_fm [simp]:
  " $\llbracket x \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$ 
 $\Rightarrow \text{sats}(A, \text{function\_fm}(x), \text{env}) \longleftrightarrow \text{is\_function}(\#\#A, \text{nth}(x, \text{env}))$ "
  <proof>

```

```

lemma is_function_iff_sats:
  " $\llbracket \text{nth}(i, \text{env}) = x; \text{nth}(j, \text{env}) = y; i \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$ 
 $\Rightarrow \text{is\_function}(\#\#A, x) \longleftrightarrow \text{sats}(A, \text{function\_fm}(i), \text{env})$ "
  <proof>

```

```

theorem is_function_reflection:
  "REFLECTS[ $\lambda x. \text{is\_function}(L, f(x)),$ 
     $\lambda i x. \text{is\_function}(\#\#L\text{set}(i), f(x))]$ "
  <proof>

```

#### 10.4.21 Typed Functions, Internalized

```

definition
  typed_function_fm :: "[i,i,i]  $\Rightarrow$  i" where
    "typed_function_fm(A,B,r)  $\equiv$ 
      And(function_fm(r),
        And(relation_fm(r),
          And(domain_fm(r,A),
            Forall(Implies(Member(0,succ(r)),
              Forall(Forall(Implies(pair_fm(1,0,2), Member(0,B#+3))))))))"

```

```

lemma typed_function_type [TC]:
  " $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \Rightarrow \text{typed\_function\_fm}(x,y,z) \in \text{formula}$ "
  <proof>

```

```

lemma sats_typed_function_fm [simp]:
  "[x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)]
  ⇒ sats(A, typed_function_fm(x,y,z), env) ⟷
    typed_function(##A, nth(x,env), nth(y,env), nth(z,env))"
⟨proof⟩

lemma typed_function_iff_sats:
  "[nth(i,env) = x; nth(j,env) = y; nth(k,env) = z;
   i ∈ nat; j ∈ nat; k ∈ nat; env ∈ list(A)]
  ⇒ typed_function(##A, x, y, z) ⟷ sats(A, typed_function_fm(i,j,k),
env)"
⟨proof⟩

lemmas function_reflections =
  empty_reflection number1_reflection
  upair_reflection pair_reflection union_reflection
  big_union_reflection cons_reflection successor_reflection
  fun_apply_reflection subset_reflection
  transitive_set_reflection membership_reflection
  pred_set_reflection domain_reflection range_reflection field_reflection
  image_reflection pre_image_reflection
  is_relation_reflection is_function_reflection

lemmas function_iff_sats =
  empty_iff_sats number1_iff_sats
  upair_iff_sats pair_iff_sats union_iff_sats
  big_union_iff_sats cons_iff_sats successor_iff_sats
  fun_apply_iff_sats Memrel_iff_sats
  pred_set_iff_sats domain_iff_sats range_iff_sats field_iff_sats
  image_iff_sats pre_image_iff_sats
  relation_iff_sats is_function_iff_sats

theorem typed_function_reflection:
  "REFLECTS[λx. typed_function(L,f(x),g(x),h(x)),
    λi x. typed_function(##Lset(i),f(x),g(x),h(x))]"
⟨proof⟩

```

#### 10.4.22 Composition of Relations, Internalized

**definition**

```

composition_fm :: "[i,i,i]⇒i" where
  "composition_fm(r,s,t) ≡
    Forall(Iff(Member(0,succ(t)),
      Exists(Exists(Exists(Exists(Exists(
        And(pair_fm(4,2,5),
        And(pair_fm(4,3,1),
        And(pair_fm(3,2,0),

```

And(Member(1,s#+6), Member(0,r#+6))))))))))"

**lemma** *composition\_type* [TC]:  
 "[x ∈ nat; y ∈ nat; z ∈ nat] ⇒ composition\_fm(x,y,z) ∈ formula"  
 <proof>

**lemma** *sats\_composition\_fm* [simp]:  
 "[x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)]  
 ⇒ sats(A, composition\_fm(x,y,z), env) ⇔  
 composition(##A, nth(x,env), nth(y,env), nth(z,env))"  
 <proof>

**lemma** *composition\_iff\_sats*:  
 "[nth(i,env) = x; nth(j,env) = y; nth(k,env) = z;  
 i ∈ nat; j ∈ nat; k ∈ nat; env ∈ list(A)]  
 ⇒ composition(##A, x, y, z) ⇔ sats(A, composition\_fm(i,j,k),  
 env)"  
 <proof>

**theorem** *composition\_reflection*:  
 "REFLECTS[λx. composition(L,f(x),g(x),h(x)),  
 λi x. composition(##Lset(i),f(x),g(x),h(x))]"  
 <proof>

#### 10.4.23 Injections, Internalized

**definition**  
*injection\_fm* :: "[i,i,i]⇒i" where  
 "injection\_fm(A,B,f) ≡  
 And(typed\_function\_fm(A,B,f),  
 Forall(Forall(Forall(Forall(Forall(  
 Implies(pair\_fm(4,2,1),  
 Implies(pair\_fm(3,2,0),  
 Implies(Member(1,f#+5),  
 Implies(Member(0,f#+5), Equal(4,3))))))))))"

**lemma** *injection\_type* [TC]:  
 "[x ∈ nat; y ∈ nat; z ∈ nat] ⇒ injection\_fm(x,y,z) ∈ formula"  
 <proof>

**lemma** *sats\_injection\_fm* [simp]:  
 "[x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)]  
 ⇒ sats(A, injection\_fm(x,y,z), env) ⇔  
 injection(##A, nth(x,env), nth(y,env), nth(z,env))"  
 <proof>

**lemma** *injection\_iff\_sats*:  
 "[nth(i,env) = x; nth(j,env) = y; nth(k,env) = z;

```

      i ∈ nat; j ∈ nat; k ∈ nat; env ∈ list(A)]
    ⇒ injection(##A, x, y, z) ⇔ sats(A, injection_fm(i,j,k), env)"
  ⟨proof⟩

```

```

theorem injection_reflection:
  "REFLECTS[λx. injection(L,f(x),g(x),h(x)),
    λi x. injection(##Lset(i),f(x),g(x),h(x))]"
  ⟨proof⟩

```

#### 10.4.24 Surjections, Internalized

**definition**

```

surjection_fm :: "[i,i,i]⇒i" where
  "surjection_fm(A,B,f) ≡
    And(typed_function_fm(A,B,f),
      Forall(Implies(Member(0,succ(B)),
        Exists(And(Member(0,succ(succ(A))),
          fun_apply_fm(succ(succ(f)),0,1))))))"

```

```

lemma surjection_type [TC]:
  "⌊x ∈ nat; y ∈ nat; z ∈ nat⌋ ⇒ surjection_fm(x,y,z) ∈ formula"
  ⟨proof⟩

```

```

lemma sats_surjection_fm [simp]:
  "⌊x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)⌋
    ⇒ sats(A, surjection_fm(x,y,z), env) ⇔
      surjection(##A, nth(x,env), nth(y,env), nth(z,env))"
  ⟨proof⟩

```

```

lemma surjection_iff_sats:
  "⌊nth(i,env) = x; nth(j,env) = y; nth(k,env) = z;
    i ∈ nat; j ∈ nat; k ∈ nat; env ∈ list(A)⌋
    ⇒ surjection(##A, x, y, z) ⇔ sats(A, surjection_fm(i,j,k), env)"
  ⟨proof⟩

```

```

theorem surjection_reflection:
  "REFLECTS[λx. surjection(L,f(x),g(x),h(x)),
    λi x. surjection(##Lset(i),f(x),g(x),h(x))]"
  ⟨proof⟩

```

#### 10.4.25 Bijections, Internalized

**definition**

```

bijection_fm :: "[i,i,i]⇒i" where
  "bijection_fm(A,B,f) ≡ And(injection_fm(A,B,f), surjection_fm(A,B,f))"

```

```

lemma bijection_type [TC]:
  "⌊x ∈ nat; y ∈ nat; z ∈ nat⌋ ⇒ bijection_fm(x,y,z) ∈ formula"
  ⟨proof⟩

```

```

lemma sats_bijection_fm [simp]:
  "[x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)]
  ⇒ sats(A, bijection_fm(x,y,z), env) ↔
    bijection(##A, nth(x,env), nth(y,env), nth(z,env))"
  <proof>

lemma bijection_iff_sats:
  "[nth(i,env) = x; nth(j,env) = y; nth(k,env) = z;
    i ∈ nat; j ∈ nat; k ∈ nat; env ∈ list(A)]
  ⇒ bijection(##A, x, y, z) ↔ sats(A, bijection_fm(i,j,k), env)"
  <proof>

theorem bijection_reflection:
  "REFLECTS[λx. bijection(L,f(x),g(x),h(x)),
    λi x. bijection(##Lset(i),f(x),g(x),h(x))]"
  <proof>

10.4.26 Restriction of a Relation, Internalized

definition
  restriction_fm :: "[i,i,i]⇒i" where
    "restriction_fm(r,A,z) ≡
      Forall(Iff(Member(0,succ(z)),
        And(Member(0,succ(r)),
          Exists(And(Member(0,succ(succ(A))),
            Exists(pair_fm(1,0,2)))))))"

lemma restriction_type [TC]:
  "[x ∈ nat; y ∈ nat; z ∈ nat] ⇒ restriction_fm(x,y,z) ∈ formula"
  <proof>

lemma sats_restriction_fm [simp]:
  "[x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)]
  ⇒ sats(A, restriction_fm(x,y,z), env) ↔
    restriction(##A, nth(x,env), nth(y,env), nth(z,env))"
  <proof>

lemma restriction_iff_sats:
  "[nth(i,env) = x; nth(j,env) = y; nth(k,env) = z;
    i ∈ nat; j ∈ nat; k ∈ nat; env ∈ list(A)]
  ⇒ restriction(##A, x, y, z) ↔ sats(A, restriction_fm(i,j,k),
env)"
  <proof>

theorem restriction_reflection:
  "REFLECTS[λx. restriction(L,f(x),g(x),h(x)),
    λi x. restriction(##Lset(i),f(x),g(x),h(x))]"
  <proof>

```



#### 10.4.27 Order-Isomorphisms, Internalized

**definition**

```

order_isomorphism_fm :: "[i,i,i,i,i]⇒i" where
"order_isomorphism_fm(A,r,B,s,f) ≡
And(bijection_fm(A,B,f),
Forall(Implies(Member(0,succ(A)),
Forall(Implies(Member(0,succ(succ(A))),
Forall(Forall(Forall(Forall(
Implies(pair_fm(5,4,3),
Implies(fun_apply_fm(f#+6,5,2),
Implies(fun_apply_fm(f#+6,4,1),
Implies(pair_fm(2,1,0),
Iff(Member(3,r#+6), Member(0,s#+6))))))))))))))"

```

**lemma** order\_isomorphism\_type [TC]:

```

"[[A ∈ nat; r ∈ nat; B ∈ nat; s ∈ nat; f ∈ nat]]
⇒ order_isomorphism_fm(A,r,B,s,f) ∈ formula"

```

⟨proof⟩

**lemma** sats\_order\_isomorphism\_fm [simp]:

```

"[[U ∈ nat; r ∈ nat; B ∈ nat; s ∈ nat; f ∈ nat; env ∈ list(A)]
⇒ sats(A, order_isomorphism_fm(U,r,B,s,f), env) ⇔
order_isomorphism(##A, nth(U,env), nth(r,env), nth(B,env),
nth(s,env), nth(f,env))]"

```

⟨proof⟩

**lemma** order\_isomorphism\_iff\_sats:

```

"[[nth(i,env) = U; nth(j,env) = r; nth(k,env) = B; nth(j',env) = s;
nth(k',env) = f;
i ∈ nat; j ∈ nat; k ∈ nat; j' ∈ nat; k' ∈ nat; env ∈ list(A)]
⇒ order_isomorphism(##A,U,r,B,s,f) ⇔
sats(A, order_isomorphism_fm(i,j,k,j',k'), env)"

```

⟨proof⟩

**theorem** order\_isomorphism\_reflection:

```

"REFLECTS[λx. order_isomorphism(L,f(x),g(x),h(x),g'(x),h'(x)),
λi x. order_isomorphism(##Lset(i),f(x),g(x),h(x),g'(x),h'(x))]"

```

⟨proof⟩

#### 10.4.28 Limit Ordinals, Internalized

A limit ordinal is a non-empty, successor-closed ordinal

**definition**

```

limit_ordinal_fm :: "i⇒i" where
"limit_ordinal_fm(x) ≡
And(ordinal_fm(x),
And(Neg(empty_fm(x)),
Forall(Implies(Member(0,succ(x)),

```

Exists(And(Member(0,succ(succ(x))),  
succ\_fm(1,0))))))"

**lemma** *limit\_ordinal\_type* [TC]:  
 "x ∈ nat ⇒ limit\_ordinal\_fm(x) ∈ formula"  
 <proof>

**lemma** *sats\_limit\_ordinal\_fm* [simp]:  
 "⌈x ∈ nat; env ∈ list(A)⌋  
 ⇒ sats(A, limit\_ordinal\_fm(x), env) ⇔ limit\_ordinal(##A, nth(x,env))"  
 <proof>

**lemma** *limit\_ordinal\_iff\_sats*:  
 "⌈nth(i,env) = x; nth(j,env) = y;  
 i ∈ nat; env ∈ list(A)⌋  
 ⇒ limit\_ordinal(##A, x) ⇔ sats(A, limit\_ordinal\_fm(i), env)"  
 <proof>

**theorem** *limit\_ordinal\_reflection*:  
 "REFLECTS[λx. limit\_ordinal(L,f(x)),  
 λi x. limit\_ordinal(##Lset(i),f(x))]"  
 <proof>

#### 10.4.29 Finite Ordinals: The Predicate "Is A Natural Number"

**definition**  
*finite\_ordinal\_fm* :: "i⇒i" where  
 "finite\_ordinal\_fm(x) ≡  
 And(ordinal\_fm(x),  
 And(Neg(limit\_ordinal\_fm(x)),  
 Forall(Implies(Member(0,succ(x)),  
 Neg(limit\_ordinal\_fm(0))))))"

**lemma** *finite\_ordinal\_type* [TC]:  
 "x ∈ nat ⇒ finite\_ordinal\_fm(x) ∈ formula"  
 <proof>

**lemma** *sats\_finite\_ordinal\_fm* [simp]:  
 "⌈x ∈ nat; env ∈ list(A)⌋  
 ⇒ sats(A, finite\_ordinal\_fm(x), env) ⇔ finite\_ordinal(##A, nth(x,env))"  
 <proof>

**lemma** *finite\_ordinal\_iff\_sats*:  
 "⌈nth(i,env) = x; nth(j,env) = y;  
 i ∈ nat; env ∈ list(A)⌋  
 ⇒ finite\_ordinal(##A, x) ⇔ sats(A, finite\_ordinal\_fm(i), env)"  
 <proof>

**theorem** *finite\_ordinal\_reflection*:

```

      "REFLECTS[ $\lambda x. \text{finite\_ordinal}(L, f(x)),$ 
                 $\lambda i x. \text{finite\_ordinal}(\#\#L\text{set}(i), f(x))]$ "
    <proof>

```

#### 10.4.30 Omega: The Set of Natural Numbers

**definition**

```

  omega_fm :: "i  $\Rightarrow$  i" where
    "omega_fm(x)  $\equiv$ 
      And(limit_ordinal_fm(x),
          Forall(Implies(Member(0, succ(x)),
                          Neg(limit_ordinal_fm(0)))))"

```

**lemma** omega\_type [TC]:

```

  "x  $\in$  nat  $\implies$  omega_fm(x)  $\in$  formula"
  <proof>

```

**lemma** sats\_omega\_fm [simp]:

```

  "[x  $\in$  nat; env  $\in$  list(A)]
   $\implies$  sats(A, omega_fm(x), env)  $\longleftrightarrow$  omega( $\#\#A$ , nth(x, env))"
  <proof>

```

**lemma** omega\_iff\_sats:

```

  "[nth(i, env) = x; nth(j, env) = y;
   i  $\in$  nat; env  $\in$  list(A)]
   $\implies$  omega( $\#\#A$ , x)  $\longleftrightarrow$  sats(A, omega_fm(i), env)"
  <proof>

```

**theorem** omega\_reflection:

```

  "REFLECTS[ $\lambda x. \text{omega}(L, f(x)),$ 
             $\lambda i x. \text{omega}(\#\#L\text{set}(i), f(x))]$ "
  <proof>

```

**lemmas** fun\_plus\_reflections =

```

  typed_function_reflection composition_reflection
  injection_reflection surjection_reflection
  bijection_reflection restriction_reflection
  order_isomorphism_reflection finite_ordinal_reflection
  ordinal_reflection limit_ordinal_reflection omega_reflection

```

**lemmas** fun\_plus\_iff\_sats =

```

  typed_function_iff_sats composition_iff_sats
  injection_iff_sats surjection_iff_sats
  bijection_iff_sats restriction_iff_sats
  order_isomorphism_iff_sats finite_ordinal_iff_sats
  ordinal_iff_sats limit_ordinal_iff_sats omega_iff_sats

```

**end**

## 11 Early Instances of Separation and Strong Replacement

**theory Separation imports L\_axioms WF\_absolute begin**

This theory proves all instances needed for locale *M\_basic*

Helps us solve for de Bruijn indices!

**lemma nth\_ConsI:** " $\llbracket \text{nth}(n, l) = x; n \in \text{nat} \rrbracket \implies \text{nth}(\text{succ}(n), \text{Cons}(a, l)) = x$ "  
 $\langle \text{proof} \rangle$

**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*

**lemma Collect\_conj\_in\_DPow:**  
 $\llbracket \{x \in A. P(x)\} \in \text{DPow}(A); \{x \in A. Q(x)\} \in \text{DPow}(A) \rrbracket$   
 $\implies \{x \in A. P(x) \wedge Q(x)\} \in \text{DPow}(A)$   
 $\langle \text{proof} \rangle$

**lemma Collect\_conj\_in\_DPow\_Lset:**  
 $\llbracket z \in \text{Lset}(j); \{x \in \text{Lset}(j). P(x)\} \in \text{DPow}(\text{Lset}(j)) \rrbracket$   
 $\implies \{x \in \text{Lset}(j). x \in z \wedge P(x)\} \in \text{DPow}(\text{Lset}(j))$   
 $\langle \text{proof} \rangle$

**lemma separation\_CollectI:**  
 $\llbracket \bigwedge z. L(z) \implies L(\{x \in z. P(x)\}) \rrbracket \implies \text{separation}(L, \lambda x. P(x))$   
 $\langle \text{proof} \rangle$

Reduces the original comprehension to the reflected one

**lemma reflection\_imp\_L\_separation:**  
 $\llbracket \forall x \in \text{Lset}(j). P(x) \longleftrightarrow Q(x);$   
 $\{x \in \text{Lset}(j). Q(x)\} \in \text{DPow}(\text{Lset}(j));$   
 $\text{Ord}(j); z \in \text{Lset}(j) \rrbracket \implies L(\{x \in z. P(x)\})$   
 $\langle \text{proof} \rangle$

Encapsulates the standard proof script for proving instances of Separation.

**lemma gen\_separation:**  
**assumes** *reflection:* "*REFLECTS [P,Q]*"  
**and** *Lu:* "*L(u)*"  
**and** *collI:* " $\bigwedge j. u \in \text{Lset}(j)$   
 $\implies \text{Collect}(\text{Lset}(j), Q(j)) \in \text{DPow}(\text{Lset}(j))$ "  
**shows** "*separation(L,P)*"  
 $\langle \text{proof} \rangle$

As above, but typically *u* is a finite enumeration such as  $\{a, b\}$ ; thus the new subgoal gets the assumption  $\{a, b\} \subseteq \text{Lset}(i)$ , which is logically equivalent to  $a \in \text{Lset}(i)$  and  $b \in \text{Lset}(i)$ .

```

lemma gen_separation_multi:
  assumes reflection: "REFLECTS [P,Q]"
    and Lu:          "L(u)"
    and collI: "∧j. u ⊆ Lset(j)
                ⇒ Collect(Lset(j), Q(j)) ∈ DPow(Lset(j))"
  shows "separation(L,P)"
<proof>

```

### 11.1 Separation for Intersection

```

lemma Inter_Reflects:
  "REFLECTS[λx. ∀y[L]. y∈A → x ∈ y,
            λi x. ∀y∈Lset(i). y∈A → x ∈ y]"
<proof>

```

```

lemma Inter_separation:
  "L(A) ⇒ separation(L, λx. ∀y[L]. y∈A → x∈y)"
<proof>

```

### 11.2 Separation for Set Difference

```

lemma Diff_Reflects:
  "REFLECTS[λx. x ∉ B, λi x. x ∉ B]"
<proof>

```

```

lemma Diff_separation:
  "L(B) ⇒ separation(L, λx. x ∉ B)"
<proof>

```

### 11.3 Separation for Cartesian Product

```

lemma cartprod_Reflects:
  "REFLECTS[λz. ∃x[L]. x∈A ∧ (∃y[L]. y∈B ∧ pair(L,x,y,z)),
            λi z. ∃x∈Lset(i). x∈A ∧ (∃y∈Lset(i). y∈B ∧
                                     pair(##Lset(i),x,y,z))]"
<proof>

```

```

lemma cartprod_separation:
  "[[L(A); L(B)]]
  ⇒ separation(L, λz. ∃x[L]. x∈A ∧ (∃y[L]. y∈B ∧ pair(L,x,y,z)))"
<proof>

```

### 11.4 Separation for Image

```

lemma image_Reflects:
  "REFLECTS[λy. ∃p[L]. p∈r ∧ (∃x[L]. x∈A ∧ pair(L,x,y,p)),
            λi y. ∃p∈Lset(i). p∈r ∧ (∃x∈Lset(i). x∈A ∧ pair(##Lset(i),x,y,p))]"
<proof>

```

```

lemma image_separation:

```

$$\llbracket L(A); L(r) \rrbracket$$

$$\implies \text{separation}(L, \lambda y. \exists p[L]. p \in r \wedge (\exists x[L]. x \in A \wedge \text{pair}(L, x, y, p)))$$

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

## 11.5 Separation for Converse

**lemma converse\_Reflects:**  

$$\text{"REFLECTS}[\lambda z. \exists p[L]. p \in r \wedge (\exists x[L]. \exists y[L]. \text{pair}(L, x, y, p) \wedge \text{pair}(L, y, x, z)),$$

$$\lambda i z. \exists p \in \text{Lset}(i). p \in r \wedge (\exists x \in \text{Lset}(i). \exists y \in \text{Lset}(i).$$

$$\text{pair}(\#\text{Lset}(i), x, y, p) \wedge \text{pair}(\#\text{Lset}(i), y, x, z))]"$$

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

**lemma converse\_separation:**  

$$\text{"}L(r) \implies \text{separation}(L,$$

$$\lambda z. \exists p[L]. p \in r \wedge (\exists x[L]. \exists y[L]. \text{pair}(L, x, y, p) \wedge \text{pair}(L, y, x, z)))"$$

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

## 11.6 Separation for Restriction

**lemma restrict\_Reflects:**  

$$\text{"REFLECTS}[\lambda z. \exists x[L]. x \in A \wedge (\exists y[L]. \text{pair}(L, x, y, z)),$$

$$\lambda i z. \exists x \in \text{Lset}(i). x \in A \wedge (\exists y \in \text{Lset}(i). \text{pair}(\#\text{Lset}(i), x, y, z))]"$$

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

**lemma restrict\_separation:**  

$$\text{"}L(A) \implies \text{separation}(L, \lambda z. \exists x[L]. x \in A \wedge (\exists y[L]. \text{pair}(L, x, y, z)))"$$

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

## 11.7 Separation for Composition

**lemma comp\_Reflects:**  

$$\text{"REFLECTS}[\lambda xz. \exists x[L]. \exists y[L]. \exists z[L]. \exists xy[L]. \exists yz[L].$$

$$\text{pair}(L, x, z, xz) \wedge \text{pair}(L, x, y, xy) \wedge \text{pair}(L, y, z, yz) \wedge$$

$$xy \in s \wedge yz \in r,$$

$$\lambda i xz. \exists x \in \text{Lset}(i). \exists y \in \text{Lset}(i). \exists z \in \text{Lset}(i). \exists xy \in \text{Lset}(i). \exists yz \in \text{Lset}(i).$$

$$\text{pair}(\#\text{Lset}(i), x, z, xz) \wedge \text{pair}(\#\text{Lset}(i), x, y, xy) \wedge$$

$$\text{pair}(\#\text{Lset}(i), y, z, yz) \wedge xy \in s \wedge yz \in r]"$$

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

**lemma comp\_separation:**  

$$\text{"} \llbracket L(r); L(s) \rrbracket$$

$$\implies \text{separation}(L, \lambda xz. \exists x[L]. \exists y[L]. \exists z[L]. \exists xy[L]. \exists yz[L].$$

$$\text{pair}(L, x, z, xz) \wedge \text{pair}(L, x, y, xy) \wedge \text{pair}(L, y, z, yz) \wedge$$

$$xy \in s \wedge yz \in r)"$$

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

## 11.8 Separation for Predecessors in an Order

**lemma pred\_Reflects:**  

$$\text{"REFLECTS}[\lambda y. \exists p[L]. p \in r \wedge \text{pair}(L, y, x, p),$$

$\lambda i y. \exists p \in Lset(i). p \in r \wedge pair(\#\#Lset(i), y, x, p)]"$

$\langle proof \rangle$

**lemma** *pred\_separation*:  
 $"[L(x); L(x)] \implies separation(L, \lambda y. \exists p[L]. p \in r \wedge pair(L, y, x, p))"$   
 $\langle proof \rangle$

## 11.9 Separation for the Membership Relation

**lemma** *Memrel\_Reflects*:  
 $"REFLECTS[\lambda z. \exists x[L]. \exists y[L]. pair(L, x, y, z) \wedge x \in y,$   
 $\lambda i z. \exists x \in Lset(i). \exists y \in Lset(i). pair(\#\#Lset(i), x, y, z)$   
 $\wedge x \in y]"$   
 $\langle proof \rangle$

**lemma** *Memrel\_separation*:  
 $"separation(L, \lambda z. \exists x[L]. \exists y[L]. pair(L, x, y, z) \wedge x \in y)"$   
 $\langle proof \rangle$

### 11.10 Replacement for FunSpace

**lemma** *funspace\_succ\_Reflects*:  
 $"REFLECTS[\lambda z. \exists p[L]. p \in A \wedge (\exists f[L]. \exists b[L]. \exists nb[L]. \exists cnbf[L].$   
 $pair(L, f, b, p) \wedge pair(L, n, b, nb) \wedge is\_cons(L, nb, f, cnbf) \wedge$   
 $upair(L, cnbf, cnbf, z)),$   
 $\lambda i z. \exists p \in Lset(i). p \in A \wedge (\exists f \in Lset(i). \exists b \in Lset(i).$   
 $\exists nb \in Lset(i). \exists cnbf \in Lset(i).$   
 $pair(\#\#Lset(i), f, b, p) \wedge pair(\#\#Lset(i), n, b, nb) \wedge$   
 $is\_cons(\#\#Lset(i), nb, f, cnbf) \wedge upair(\#\#Lset(i), cnbf, cnbf, z))]"$   
 $\langle proof \rangle$

**lemma** *funspace\_succ\_replacement*:  
 $"L(n) \implies$   
 $strong\_replacement(L, \lambda p z. \exists f[L]. \exists b[L]. \exists nb[L]. \exists cnbf[L].$   
 $pair(L, f, b, p) \wedge pair(L, n, b, nb) \wedge is\_cons(L, nb, f, cnbf)$   
 $\wedge$   
 $upair(L, cnbf, cnbf, z))"$   
 $\langle proof \rangle$

### 11.11 Separation for a Theorem about *is\_recfun*

**lemma** *is\_recfun\_reflects*:  
 $"REFLECTS[\lambda x. \exists xa[L]. \exists xb[L].$   
 $pair(L, x, a, xa) \wedge xa \in r \wedge pair(L, x, b, xb) \wedge xb \in r \wedge$   
 $(\exists fx[L]. \exists gx[L]. fun\_apply(L, f, x, fx) \wedge fun\_apply(L, g, x, gx)$   
 $\wedge$   
 $fx \neq gx),$   
 $\lambda i x. \exists xa \in Lset(i). \exists xb \in Lset(i).$   
 $pair(\#\#Lset(i), x, a, xa) \wedge xa \in r \wedge pair(\#\#Lset(i), x, b, xb) \wedge$   
 $xb \in r \wedge$

```

      (∃ fx ∈ Lset(i). ∃ gx ∈ Lset(i). fun_apply(##Lset(i), f, x, fx)
    ^
      fun_apply(##Lset(i), g, x, gx) ∧ fx ≠ gx)]"
  <proof>

lemma is_refun_separation:
  — for well-founded recursion
  "⟦L(x); L(f); L(g); L(a); L(b)⟧
  ⇒ separation(L,
    λx. ∃ xa[L]. ∃ xb[L].
      pair(L, x, a, xa) ∧ xa ∈ r ∧ pair(L, x, b, xb) ∧ xb ∈ r ∧
      (∃ fx[L]. ∃ gx[L]. fun_apply(L, f, x, fx) ∧ fun_apply(L, g, x, gx)
    ^
      fx ≠ gx))"
  <proof>

```

### 11.12 Instantiating the locale $M_{\text{basic}}$

Separation (and Strong Replacement) for basic set-theoretic constructions such as intersection, Cartesian Product and image.

```

lemma M_basic_axioms_L: "M_basic_axioms(L)"
  <proof>

```

```

theorem M_basic_L: " M_basic(L)"
  <proof>

```

```

interpretation L: M_basic L <proof>

```

**end**

```

theory Internalize imports L_axioms Datatype_absolute begin

```

### 11.13 Internalized Forms of Data Structuring Operators

#### 11.13.1 The Formula $\text{is\_Inl}$ , 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 ∈ nat; z ∈ nat⟧ ⇒ Inl_fm(x,z) ∈ formula"
  <proof>

```

**lemma** sats\_Inl\_fm [simp]:

```

  "⟦x ∈ nat; z ∈ nat; env ∈ list(A)⟧
  ⇒ sats(A, Inl_fm(x,z), env) ⟷ is_Inl(##A, nth(x,env), nth(z,env))"

```



$\langle proof \rangle$

**lemma** *Inl\_iff\_sats*:

" $\llbracket nth(i, env) = x; nth(k, env) = z; \\ i \in nat; k \in nat; env \in list(A) \rrbracket \\ \implies is\_Inl(\#\#A, x, z) \longleftrightarrow sats(A, Inl\_fm(i, k), env) \rrbracket$ "

$\langle proof \rangle$

**theorem** *Inl\_reflection*:

" $REFLECTS[\lambda x. is\_Inl(L, f(x), h(x)), \\ \lambda i x. is\_Inl(\#\#Lset(i), f(x), h(x))]$ "

$\langle proof \rangle$

### 11.13.2 The Formula *is\_Inr*, Internalized

**definition**

*Inr\_fm* :: " $i, i \Rightarrow i$ " where  
" $Inr\_fm(a, z) \equiv Exists(And(number1\_fm(0), pair\_fm(0, succ(a), succ(z))))$ "

**lemma** *Inr\_type* [TC]:

" $\llbracket x \in nat; z \in nat \rrbracket \implies Inr\_fm(x, z) \in formula$ "

$\langle proof \rangle$

**lemma** *sats\_Inr\_fm* [simp]:

" $\llbracket x \in nat; z \in nat; env \in list(A) \rrbracket \\ \implies sats(A, Inr\_fm(x, z), env) \longleftrightarrow is\_Inr(\#\#A, nth(x, env), nth(z, env)) \rrbracket$ "

$\langle proof \rangle$

**lemma** *Inr\_iff\_sats*:

" $\llbracket nth(i, env) = x; nth(k, env) = z; \\ i \in nat; k \in nat; env \in list(A) \rrbracket \\ \implies is\_Inr(\#\#A, x, z) \longleftrightarrow sats(A, Inr\_fm(i, k), env) \rrbracket$ "

$\langle proof \rangle$

**theorem** *Inr\_reflection*:

" $REFLECTS[\lambda x. is\_Inr(L, f(x), h(x)), \\ \lambda i x. is\_Inr(\#\#Lset(i), f(x), h(x))]$ "

$\langle proof \rangle$

### 11.13.3 The Formula *is\_Nil*, Internalized

**definition**

*Nil\_fm* :: " $i \Rightarrow i$ " where  
" $Nil\_fm(x) \equiv Exists(And(empty\_fm(0), Inl\_fm(0, succ(x))))$ "

**lemma** *Nil\_type* [TC]: " $x \in nat \implies Nil\_fm(x) \in formula$ "

$\langle proof \rangle$

**lemma** *sats\_Nil\_fm* [simp]:

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

$\implies \text{sats}(A, \text{Nil\_fm}(x), \text{env}) \longleftrightarrow \text{is\_Nil}(\#\#A, \text{nth}(x, \text{env}))$ "  
 $\langle \text{proof} \rangle$

**lemma Nil\_iff\_sats:**  
 $\llbracket \text{nth}(i, \text{env}) = x; i \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{is\_Nil}(\#\#A, x) \longleftrightarrow \text{sats}(A, \text{Nil\_fm}(i), \text{env})$ "  
 $\langle \text{proof} \rangle$

**theorem Nil\_reflection:**  
 $\text{"REFLECTS}[\lambda x. \text{is\_Nil}(L, f(x)),$   
 $\lambda i x. \text{is\_Nil}(\#\#L\text{set}(i), f(x))]\text{"}$   
 $\langle \text{proof} \rangle$

#### 11.13.4 The Formula *is\_Cons*, Internalized

**definition**  
 $\text{Cons\_fm} :: "[i, i, i] \Rightarrow i$ " where  
 $\text{"Cons\_fm}(a, l, Z) \equiv$   
 $\text{Exists}(\text{And}(\text{pair\_fm}(\text{succ}(a), \text{succ}(l), 0), \text{Inr\_fm}(0, \text{succ}(Z))))$ "

**lemma Cons\_type [TC]:**  
 $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \implies \text{Cons\_fm}(x, y, z) \in \text{formula}$ "  
 $\langle \text{proof} \rangle$

**lemma sats\_Cons\_fm [simp]:**  
 $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{sats}(A, \text{Cons\_fm}(x, y, z), \text{env}) \longleftrightarrow$   
 $\text{is\_Cons}(\#\#A, \text{nth}(x, \text{env}), \text{nth}(y, \text{env}), \text{nth}(z, \text{env}))$ "  
 $\langle \text{proof} \rangle$

**lemma Cons\_iff\_sats:**  
 $\llbracket \text{nth}(i, \text{env}) = x; \text{nth}(j, \text{env}) = y; \text{nth}(k, \text{env}) = z;$   
 $i \in \text{nat}; j \in \text{nat}; k \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{is\_Cons}(\#\#A, x, y, z) \longleftrightarrow \text{sats}(A, \text{Cons\_fm}(i, j, k), \text{env})$ "  
 $\langle \text{proof} \rangle$

**theorem Cons\_reflection:**  
 $\text{"REFLECTS}[\lambda x. \text{is\_Cons}(L, f(x), g(x), h(x)),$   
 $\lambda i x. \text{is\_Cons}(\#\#L\text{set}(i), f(x), g(x), h(x))]\text{"}$   
 $\langle \text{proof} \rangle$

#### 11.13.5 The Formula *is\_quasilist*, Internalized

**definition**  
 $\text{quasilist\_fm} :: "i \Rightarrow i$ " where  
 $\text{"quasilist\_fm}(x) \equiv$   
 $\text{Or}(\text{Nil\_fm}(x), \text{Exists}(\text{Exists}(\text{Cons\_fm}(1, 0, \text{succ}(\text{succ}(x)))))$ "

**lemma quasilist\_type [TC]:**  $x \in \text{nat} \implies \text{quasilist\_fm}(x) \in \text{formula}$ "  
 $\langle \text{proof} \rangle$

```

lemma sats_quaselist_fm [simp]:
  "[[x ∈ nat; env ∈ list(A)]]
  ⇒ sats(A, quaselist_fm(x), env) ↔ is_quaselist(##A, nth(x,env))"
⟨proof⟩

```

```

lemma quaselist_iff_sats:
  "[[nth(i,env) = x; i ∈ nat; env ∈ list(A)]]
  ⇒ is_quaselist(##A, x) ↔ sats(A, quaselist_fm(i), env)"
⟨proof⟩

```

```

theorem quaselist_reflection:
  "REFLECTS[λx. is_quaselist(L,f(x)),
    λi x. is_quaselist(##Lset(i),f(x))]"
⟨proof⟩

```

## 11.14 Absoluteness for the Function *nth*

### 11.14.1 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(quaselist_fm(xs), empty_fm(H))))"

```

```

lemma hd_type [TC]:
  "[[x ∈ nat; y ∈ nat]] ⇒ hd_fm(x,y) ∈ formula"
⟨proof⟩

```

```

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))"
⟨proof⟩

```

```

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)"
⟨proof⟩

```

```

theorem hd_reflection:
  "REFLECTS[λx. is_hd(L,f(x),g(x)),
    λi x. is_hd(##Lset(i),f(x),g(x))]"
⟨proof⟩

```

### 11.14.2 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"
⟨proof⟩

```

```

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))"
⟨proof⟩

```

```

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)"
⟨proof⟩

```

```

theorem tl_reflection:
  "REFLECTS[λx. is_tl(L,f(x),g(x)),
    λi x. is_tl(##Lset(i),f(x),g(x))]"
⟨proof⟩

```

### 11.14.3 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"
⟨proof⟩

```

```

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))"
⟨proof⟩

```

```

lemma is_bool_of_o_iff_sats:
  "⌊P ⟷ sats(A, p, env); nth(k,env) = z; k ∈ nat; env ∈ list(A)⌋

```

$\implies \text{is\_bool\_of\_o}(\#\#A, P, z) \longleftrightarrow \text{sats}(A, \text{bool\_of\_o\_fm}(p,k), \text{env})"$   
 $\langle \text{proof} \rangle$

**theorem** *bool\_of\_o\_reflection*:  
 $"\text{REFLECTS } [P(L), \lambda i. P(\#\#Lset(i))] \implies$   
 $\text{REFLECTS}[\lambda x. \text{is\_bool\_of\_o}(L, P(L,x), f(x)),$   
 $\lambda i x. \text{is\_bool\_of\_o}(\#\#Lset(i), P(\#\#Lset(i),x), f(x))]"$   
 $\langle \text{proof} \rangle$

## 11.15 More Internalizations

### 11.15.1 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]  $\Rightarrow$  i" where  
 $"\text{lambda\_fm}(p,A,z) \equiv$   
 $\text{Forall}(\text{Iff}(\text{Member}(0,\text{succ}(z)),$   
 $\text{Exists}(\text{Exists}(\text{And}(\text{Member}(1,A\#+3),$   
 $\text{And}(\text{pair\_fm}(1,0,2), p))))))"$

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]:  
 $"\llbracket p \in \text{formula}; x \in \text{nat}; y \in \text{nat} \rrbracket$   
 $\implies \text{lambda\_fm}(p,x,y) \in \text{formula}"$   
 $\langle \text{proof} \rangle$

**lemma** *sats\_lambda\_fm*:  
**assumes** *is\_b\_iff\_sats*:  
 $"\bigwedge a0 a1 a2.$   
 $\llbracket a0 \in A; a1 \in A; a2 \in A \rrbracket$   
 $\implies \text{is\_b}(a1, a0) \longleftrightarrow \text{sats}(A, p, \text{Cons}(a0, \text{Cons}(a1, \text{Cons}(a2, \text{env}))))"$   
**shows**  
 $"\llbracket x \in \text{nat}; y \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{sats}(A, \text{lambda\_fm}(p,x,y), \text{env}) \longleftrightarrow$   
 $\text{is\_lambda}(\#\#A, \text{nth}(x,\text{env}), \text{is\_b}, \text{nth}(y,\text{env}))"$   
 $\langle \text{proof} \rangle$

**theorem** *is\_lambda\_reflection*:  
**assumes** *is\_b\_reflection*:  
 $"\bigwedge f g h. \text{REFLECTS}[\lambda x. \text{is\_b}(L, f(x), g(x), h(x)),$   
 $\lambda i x. \text{is\_b}(\#\#Lset(i), f(x), g(x), h(x))]"$   
**shows**  $"\text{REFLECTS}[\lambda x. \text{is\_lambda}(L, A(x), \text{is\_b}(L,x), f(x)),$   
 $\lambda i x. \text{is\_lambda}(\#\#Lset(i), A(x), \text{is\_b}(\#\#Lset(i),x), f(x))]"$   
 $\langle \text{proof} \rangle$

### 11.15.2 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"
⟨proof⟩
```

**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))"
⟨proof⟩
```

**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)"
⟨proof⟩
```

**theorem** *Member\_reflection*:

```
"REFLECTS[λx. is_Member(L,f(x),g(x),h(x)),
  λi x. is_Member(##Lset(i),f(x),g(x),h(x))]"
⟨proof⟩
```

### 11.15.3 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"
⟨proof⟩
```

**lemma** *sats\_Equal\_fm* [simp]:

```
"[x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)]
⇒ sats(A, Equal_fm(x,y,z), env) ⟷
  is_Equal(##A, nth(x,env), nth(y,env), nth(z,env))"
⟨proof⟩
```

**lemma** *Equal\_iff\_sats*:

```
"[nth(i,env) = x; nth(j,env) = y; nth(k,env) = z;
  i ∈ nat; j ∈ nat; k ∈ nat; env ∈ list(A)]
```

$\implies \text{is\_Equal}(\#\#A, x, y, z) \longleftrightarrow \text{sats}(A, \text{Equal\_fm}(i, j, k), \text{env})"$   
 $\langle \text{proof} \rangle$

**theorem** *Equal\_reflection*:

"REFLECTS $[\lambda x. \text{is\_Equal}(L, f(x), g(x), h(x)),$   
 $\lambda i x. \text{is\_Equal}(\#\#L\text{set}(i), f(x), g(x), h(x))]$ "

$\langle \text{proof} \rangle$

#### 11.15.4 The Operator *is\_Nand*, Internalized

**definition**

*Nand\_fm* :: "[i,i,i] $\Rightarrow$ i" where  
*Nand\_fm*(x,y,Z)  $\equiv$   
 $\text{Exists}(\text{Exists}(\text{And}(\text{pair\_fm}(x\#+2, y\#+2, 1),$   
 $\text{And}(\text{Inl\_fm}(1, 0), \text{Inr\_fm}(0, Z\#+2))))"$

**lemma** *is\_Nand\_type* [TC]:

" $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \implies \text{Nand\_fm}(x, y, z) \in \text{formula}"$

$\langle \text{proof} \rangle$

**lemma** *sats\_Nand\_fm* [simp]:

" $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{sats}(A, \text{Nand\_fm}(x, y, z), \text{env}) \longleftrightarrow$   
 $\text{is\_Nand}(\#\#A, \text{nth}(x, \text{env}), \text{nth}(y, \text{env}), \text{nth}(z, \text{env}))"$

$\langle \text{proof} \rangle$

**lemma** *Nand\_iff\_sats*:

" $\llbracket \text{nth}(i, \text{env}) = x; \text{nth}(j, \text{env}) = y; \text{nth}(k, \text{env}) = z;$   
 $i \in \text{nat}; j \in \text{nat}; k \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{is\_Nand}(\#\#A, x, y, z) \longleftrightarrow \text{sats}(A, \text{Nand\_fm}(i, j, k), \text{env})"$

$\langle \text{proof} \rangle$

**theorem** *Nand\_reflection*:

"REFLECTS $[\lambda x. \text{is\_Nand}(L, f(x), g(x), h(x)),$   
 $\lambda i x. \text{is\_Nand}(\#\#L\text{set}(i), f(x), g(x), h(x))]$ "

$\langle \text{proof} \rangle$

#### 11.15.5 The Operator *is\_Forall*, Internalized

**definition**

*Forall\_fm* :: "[i,i] $\Rightarrow$ i" where  
*Forall\_fm*(x,Z)  $\equiv$   
 $\text{Exists}(\text{And}(\text{Inr\_fm}(\text{succ}(x), 0), \text{Inr\_fm}(0, \text{succ}(Z))))"$

**lemma** *is\_Forall\_type* [TC]:

" $\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \implies \text{Forall\_fm}(x, y) \in \text{formula}"$

$\langle \text{proof} \rangle$

**lemma** *sats\_Forall\_fm* [simp]:

" $\llbracket x \in \text{nat}; y \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$

$\implies \text{sats}(A, \text{Forall\_fm}(x,y), \text{env}) \longleftrightarrow$   
 $\text{is\_Forall}(\#\#A, \text{nth}(x,\text{env}), \text{nth}(y,\text{env}))"$   
 $\langle \text{proof} \rangle$

**lemma** *Forall\_iff\_sats*:  
 $"\llbracket \text{nth}(i,\text{env}) = x; \text{nth}(j,\text{env}) = y;$   
 $i \in \text{nat}; j \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{is\_Forall}(\#\#A, x, y) \longleftrightarrow \text{sats}(A, \text{Forall\_fm}(i,j), \text{env})"$   
 $\langle \text{proof} \rangle$

**theorem** *Forall\_reflection*:  
 $"\text{REFLECTS}[\lambda x. \text{is\_Forall}(L, f(x), g(x)),$   
 $\lambda i x. \text{is\_Forall}(\#\#L\text{set}(i), f(x), g(x))]"$   
 $\langle \text{proof} \rangle$

### 11.15.6 The Operator *is\_and*, Internalized

**definition**  
 $\text{and\_fm} :: "[i,i,i] \Rightarrow i"$  where  
 $"\text{and\_fm}(a,b,z) \equiv$   
 $\text{Or}(\text{And}(\text{number1\_fm}(a), \text{Equal}(z,b)),$   
 $\text{And}(\text{Neg}(\text{number1\_fm}(a)), \text{empty\_fm}(z)))"$

**lemma** *is\_and\_type [TC]*:  
 $"\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \implies \text{and\_fm}(x,y,z) \in \text{formula}"$   
 $\langle \text{proof} \rangle$

**lemma** *sats\_and\_fm [simp]*:  
 $"\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{sats}(A, \text{and\_fm}(x,y,z), \text{env}) \longleftrightarrow$   
 $\text{is\_and}(\#\#A, \text{nth}(x,\text{env}), \text{nth}(y,\text{env}), \text{nth}(z,\text{env}))"$   
 $\langle \text{proof} \rangle$

**lemma** *is\_and\_iff\_sats*:  
 $"\llbracket \text{nth}(i,\text{env}) = x; \text{nth}(j,\text{env}) = y; \text{nth}(k,\text{env}) = z;$   
 $i \in \text{nat}; j \in \text{nat}; k \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{is\_and}(\#\#A, x, y, z) \longleftrightarrow \text{sats}(A, \text{and\_fm}(i,j,k), \text{env})"$   
 $\langle \text{proof} \rangle$

**theorem** *is\_and\_reflection*:  
 $"\text{REFLECTS}[\lambda x. \text{is\_and}(L, f(x), g(x), h(x)),$   
 $\lambda i x. \text{is\_and}(\#\#L\text{set}(i), f(x), g(x), h(x))]"$   
 $\langle \text{proof} \rangle$

### 11.15.7 The Operator *is\_or*, Internalized

**definition**  
 $\text{or\_fm} :: "[i,i,i] \Rightarrow i"$  where  
 $"\text{or\_fm}(a,b,z) \equiv$   
 $\text{Or}(\text{And}(\text{number1\_fm}(a), \text{number1\_fm}(z)),$



$\text{And}(\text{Neg}(\text{number1\_fm}(a)), \text{Equal}(z, b)))$ "

**lemma** *is\_or\_type* [TC]:

" $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \implies \text{or\_fm}(x, y, z) \in \text{formula}$ "  
 $\langle \text{proof} \rangle$

**lemma** *sats\_or\_fm* [simp]:

" $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{sats}(A, \text{or\_fm}(x, y, z), \text{env}) \longleftrightarrow$   
 $\text{is\_or}(\#\#A, \text{nth}(x, \text{env}), \text{nth}(y, \text{env}), \text{nth}(z, \text{env}))$ "  
 $\langle \text{proof} \rangle$

**lemma** *is\_or\_iff\_sats*:

" $\llbracket \text{nth}(i, \text{env}) = x; \text{nth}(j, \text{env}) = y; \text{nth}(k, \text{env}) = z;$   
 $i \in \text{nat}; j \in \text{nat}; k \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{is\_or}(\#\#A, x, y, z) \longleftrightarrow \text{sats}(A, \text{or\_fm}(i, j, k), \text{env})$ "  
 $\langle \text{proof} \rangle$

**theorem** *is\_or\_reflection*:

" $\text{REFLECTS}[\lambda x. \text{is\_or}(L, f(x), g(x), h(x)),$   
 $\lambda i x. \text{is\_or}(\#\#L\text{set}(i), f(x), g(x), h(x))]$ "  
 $\langle \text{proof} \rangle$

### 11.15.8 The Operator *is\_not*, Internalized

**definition**

*not\_fm* :: " $[i, i] \Rightarrow i$ " where  
 $\text{"not\_fm}(a, z) \equiv$   
 $\text{Or}(\text{And}(\text{number1\_fm}(a), \text{empty\_fm}(z)),$   
 $\text{And}(\text{Neg}(\text{number1\_fm}(a)), \text{number1\_fm}(z)))$ "

**lemma** *is\_not\_type* [TC]:

" $\llbracket x \in \text{nat}; z \in \text{nat} \rrbracket \implies \text{not\_fm}(x, z) \in \text{formula}$ "  
 $\langle \text{proof} \rangle$

**lemma** *sats\_is\_not\_fm* [simp]:

" $\llbracket x \in \text{nat}; z \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{sats}(A, \text{not\_fm}(x, z), \text{env}) \longleftrightarrow \text{is\_not}(\#\#A, \text{nth}(x, \text{env}), \text{nth}(z, \text{env}))$ "  
 $\langle \text{proof} \rangle$

**lemma** *is\_not\_iff\_sats*:

" $\llbracket \text{nth}(i, \text{env}) = x; \text{nth}(k, \text{env}) = z;$   
 $i \in \text{nat}; k \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{is\_not}(\#\#A, x, z) \longleftrightarrow \text{sats}(A, \text{not\_fm}(i, k), \text{env})$ "  
 $\langle \text{proof} \rangle$

**theorem** *is\_not\_reflection*:

" $\text{REFLECTS}[\lambda x. \text{is\_not}(L, f(x), g(x)),$   
 $\lambda i x. \text{is\_not}(\#\#L\text{set}(i), f(x), g(x))]$ "

*<proof>*

```
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
```

## 11.16 Well-Founded Recursion!

### 11.16.1 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)  $\wedge$  pair(M,x,y,z)  $\wedge$ 
    ( $\exists xa[M]. \exists sx[M]. \exists r\_sx[M].$ 
      pair(M,x,a,xa)  $\wedge$  upair(M,x,x,sx)  $\wedge$ 
      pre_image(M,r,sx,r_sx)  $\wedge$  restriction(M,f,r_sx,f_r_sx)  $\wedge$ 
      xa  $\in$  r)))"
```

*<proof>*

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

**definition**

```
is_recfun_fm :: "[i, i, i, i] $\Rightarrow$ i" where
  "is_recfun_fm(p,r,a,f)  $\equiv$ 
  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  $\in$  formula; x  $\in$  nat; y  $\in$  nat; z  $\in$  nat]
 $\impl$  is_recfun_fm(p,x,y,z)  $\in$  formula"
<proof>
```

```
lemma sats_is_recfun_fm:
  assumes MH_iff_sats:
    " $\bigwedge a0 a1 a2 a3.$ 
     $\llbracket a0 \in A; a1 \in A; a2 \in A; a3 \in A \rrbracket$ "
```

```

    ⇒ 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))"
  ⟨proof⟩

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)"
  ⟨proof⟩

```

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))]"
  ⟨proof⟩

```

### 11.16.2 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⟧

```

```

     $\Rightarrow is\_wfrec\_fm(p,x,y,z) \in formula$ 
  <proof>

lemma sats_is_wfrec_fm:
  assumes MH_iff_sats:
    " $\bigwedge a0\ a1\ a2\ a3\ a4.$ 
       $\llbracket a0 \in A; a1 \in A; a2 \in A; a3 \in A; a4 \in A \rrbracket$ 
       $\Rightarrow MH(a2, a1, a0) \longleftrightarrow sats(A, p, Cons(a0, Cons(a1, Cons(a2, Cons(a3, Cons(a4, env))))))$ "
  shows
    " $\llbracket x \in nat; y < length(env); z < length(env); env \in list(A) \rrbracket$ 
       $\Rightarrow sats(A, is\_wfrec\_fm(p,x,y,z), env) \longleftrightarrow$ 
       $is\_wfrec(\#\#A, MH, nth(x,env), nth(y,env), nth(z,env))$ "
  <proof>

```

```

lemma is_wfrec_iff_sats:
  assumes MH_iff_sats:
    " $\bigwedge a0\ a1\ a2\ a3\ a4.$ 
       $\llbracket a0 \in A; a1 \in A; a2 \in A; a3 \in A; a4 \in A \rrbracket$ 
       $\Rightarrow MH(a2, a1, a0) \longleftrightarrow sats(A, p, Cons(a0, Cons(a1, Cons(a2, Cons(a3, Cons(a4, env))))))$ "
  shows
    " $\llbracket nth(i,env) = x; nth(j,env) = y; nth(k,env) = z;$ 
       $i \in nat; j < length(env); k < length(env); env \in list(A) \rrbracket$ 
       $\Rightarrow is\_wfrec(\#\#A, MH, x, y, z) \longleftrightarrow sats(A, is\_wfrec\_fm(p,i,j,k), env)$ "
  <proof>

```

```

theorem is_wfrec_reflection:
  assumes MH_reflection:
    " $\bigwedge f' f g h. REFLECTS[\lambda x. MH(L, f'(x), f(x), g(x), h(x)),$ 
       $\lambda i x. MH(\#\#Lset(i), f'(x), f(x), g(x), h(x))]$ "
  shows "REFLECTS $[\lambda x. is\_wfrec(L, MH(L,x), f(x), g(x), h(x)),$ 
       $\lambda i x. is\_wfrec(\#\#Lset(i), MH(\#\#Lset(i),x), f(x), g(x),$ 
       $h(x))]$ "
  <proof>

```

## 11.17 For Datatypes

### 11.17.1 Binary Products, Internalized

definition

cartprod\_fm :: "[i,i,i] $\Rightarrow$ i" where

```

  "cartprod_fm(A,B,z)  $\equiv$ 
    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]:

" $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \implies \text{cartprod\_fm}(x,y,z) \in \text{formula}$ "  
 <proof>

**lemma** *sats\_cartprod\_fm [simp]*:  
 " $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{sats}(A, \text{cartprod\_fm}(x,y,z), \text{env}) \longleftrightarrow$   
 $\text{cartprod}(\#\#A, \text{nth}(x,\text{env}), \text{nth}(y,\text{env}), \text{nth}(z,\text{env}))$ "  
 <proof>

**lemma** *cartprod\_iff\_sats*:  
 " $\llbracket \text{nth}(i,\text{env}) = x; \text{nth}(j,\text{env}) = y; \text{nth}(k,\text{env}) = z;$   
 $i \in \text{nat}; j \in \text{nat}; k \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{cartprod}(\#\#A, x, y, z) \longleftrightarrow \text{sats}(A, \text{cartprod\_fm}(i,j,k), \text{env})$ "  
 <proof>

**theorem** *cartprod\_reflection*:  
 "REFLECTS $[\lambda x. \text{cartprod}(L, f(x), g(x), h(x)),$   
 $\lambda i x. \text{cartprod}(\#\#L\text{set}(i), f(x), g(x), h(x))]$ "  
 <proof>

### 11.17.2 Binary Sums, Internalized

**definition**  
*sum\_fm* :: " $[i, i, i] \Rightarrow i$ " where  
 "*sum\_fm*(A,B,Z)  $\equiv$   
 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]*:  
 " $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \implies \text{sum\_fm}(x,y,z) \in \text{formula}$ "  
 <proof>

**lemma** *sats\_sum\_fm [simp]*:  
 " $\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{sats}(A, \text{sum\_fm}(x,y,z), \text{env}) \longleftrightarrow$   
 $\text{is\_sum}(\#\#A, \text{nth}(x,\text{env}), \text{nth}(y,\text{env}), \text{nth}(z,\text{env}))$ "  
 <proof>

**lemma** *sum\_iff\_sats*:  
 " $\llbracket \text{nth}(i,\text{env}) = x; \text{nth}(j,\text{env}) = y; \text{nth}(k,\text{env}) = z;$   
 $i \in \text{nat}; j \in \text{nat}; k \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{is\_sum}(\#\#A, x, y, z) \longleftrightarrow \text{sats}(A, \text{sum\_fm}(i,j,k), \text{env})$ "  
 <proof>

**theorem** *sum\_reflection*:  
 "REFLECTS $[\lambda x. \text{is\_sum}(L, f(x), g(x), h(x)),$

$\lambda i \ x. \text{is\_sum}(\#\#Lset(i), f(x), g(x), h(x))]$ "

*<proof>*

### 11.17.3 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*"

*<proof>*

**lemma** *sats\_quasinat\_fm* [simp]:

"[*x* ∈ *nat*; *env* ∈ *list*(*A*)]  
 ⇒ *sats*(*A*, *quasinat\_fm*(*x*), *env*) ⇔ *is\_quasinat*(*##A*, *nth*(*x*, *env*))"

*<proof>*

**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*)"

*<proof>*

**theorem** *quasinat\_reflection*:

"REFLECTS[ $\lambda x. \text{is\_quasinat}(L, f(x))$ ,  
 $\lambda i \ x. \text{is\_quasinat}(\#\#Lset(i), f(x))]$ "

*<proof>*

### 11.17.4 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*"

*<proof>*

```

lemma sats_is_nat_case_fm:
  assumes is_b_iff_sats:
    " $\bigwedge a. a \in A \implies is\_b(a, nth(z, env)) \longleftrightarrow$ 
       $sats(A, p, Cons(nth(z, env), Cons(a, env)))$ "
  shows
    " $\llbracket x \in nat; y \in nat; z < length(env); env \in list(A) \rrbracket$ 
       $\implies sats(A, is\_nat\_case\_fm(x, p, y, z), env) \longleftrightarrow$ 
       $is\_nat\_case(\#A, nth(x, env), is\_b, nth(y, env), nth(z, env))$ "
  <proof>

lemma is_nat_case_iff_sats:
  " $\llbracket (\bigwedge a. a \in A \implies is\_b(a, z) \longleftrightarrow$ 
     $sats(A, p, Cons(z, Cons(a, env))));$ 
     $nth(i, env) = x; nth(j, env) = y; nth(k, env) = z;$ 
     $i \in nat; j \in nat; k < length(env); env \in list(A) \rrbracket$ 
     $\implies is\_nat\_case(\#A, x, is\_b, y, z) \longleftrightarrow sats(A, is\_nat\_case\_fm(i, p, j, k),$ 
  env)"
  <proof>

```

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:
    " $\bigwedge h f g. REFLECTS[\lambda x. is\_b(L, h(x), f(x), g(x)),$ 
       $\lambda i x. is\_b(\#Lset(i), h(x), f(x), g(x))]$ "
  shows " $REFLECTS[\lambda x. is\_nat\_case(L, f(x), is\_b(L, x), g(x), h(x)),$ 
     $\lambda i x. is\_nat\_case(\#Lset(i), f(x), is\_b(\#Lset(i), x),$ 
     $g(x), h(x))]$ "
  <proof>

```

## 11.18 The Operator *iterates\_MH*, Needed for Iteration

**definition**

```

iterates_MH_fm :: "[i, i, i, i, i]  $\Rightarrow$  i" where
  "iterates_MH_fm(isF, v, n, g, z)  $\equiv$ 
    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]:

```

  " $\llbracket p \in formula;$ 
     $v \in nat; x \in nat; y \in nat; z \in nat \rrbracket$ 
     $\implies iterates\_MH\_fm(p, v, x, y, z) \in formula$ "
  <proof>

```

**lemma** *sats\_iterates\_MH\_fm*:

```

assumes is_F_iff_sats:
  "∧a b c d. [a ∈ A; b ∈ A; c ∈ A; d ∈ A]
    ⇒ is_F(a,b) ⇔
      sats(A, p, Cons(b, Cons(a, Cons(c, Cons(d,env))))))"

shows
  "[v ∈ nat; x ∈ nat; y ∈ nat; z < length(env); env ∈ list(A)]
    ⇒ sats(A, iterates_MH_fm(p,v,x,y,z), env) ⇔
      iterates_MH(##A, is_F, nth(v,env), nth(x,env), nth(y,env),
nth(z,env))"
  <proof>

lemma iterates_MH_iff_sats:
  assumes is_F_iff_sats:
    "∧a b c d. [a ∈ A; b ∈ A; c ∈ A; d ∈ A]
      ⇒ is_F(a,b) ⇔
        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' ∈ nat; i ∈ nat; j ∈ nat; k < length(env); env ∈ list(A)]
      ⇒ iterates_MH(##A, is_F, v, x, y, z) ⇔
        sats(A, iterates_MH_fm(p,i',i,j,k), env)"
    <proof>

```

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))]"
  <proof>

```

### 11.18.1 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]:*

" $\llbracket p \in \text{formula}; x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket$   
 $\implies \text{is\_iterates\_fm}(p, x, y, z) \in \text{formula}$ "

*<proof>*

**lemma** *sats\_is\_iterates\_fm:*

**assumes** *is\_F\_iff\_sats:*

" $\bigwedge a\ b\ c\ d\ e\ f\ g\ h\ i\ j\ k.$

$\llbracket a \in A; b \in A; c \in A; d \in A; e \in A; f \in A;$   
 $g \in A; h \in A; i \in A; j \in A; k \in A \rrbracket$

$\implies \text{is\_F}(a, b) \longleftrightarrow$

$\text{sats}(A, p, \text{Cons}(b, \text{Cons}(a, \text{Cons}(c, \text{Cons}(d, \text{Cons}(e, \text{Cons}(f,$

$\text{Cons}(g, \text{Cons}(h, \text{Cons}(i, \text{Cons}(j, \text{Cons}(k, \text{env}}))))))))))$ "

**shows**

" $\llbracket x \in \text{nat}; y < \text{length}(\text{env}); z < \text{length}(\text{env}); \text{env} \in \text{list}(A) \rrbracket$

$\implies \text{sats}(A, \text{is\_iterates\_fm}(p, x, y, z), \text{env}) \longleftrightarrow$

$\text{is\_iterates}(\#\#A, \text{is\_F}, \text{nth}(x, \text{env}), \text{nth}(y, \text{env}), \text{nth}(z, \text{env}))$ "

*<proof>*

**lemma** *is\_iterates\_iff\_sats:*

**assumes** *is\_F\_iff\_sats:*

" $\bigwedge a\ b\ c\ d\ e\ f\ g\ h\ i\ j\ k.$

$\llbracket a \in A; b \in A; c \in A; d \in A; e \in A; f \in A;$   
 $g \in A; h \in A; i \in A; j \in A; k \in A \rrbracket$

$\implies \text{is\_F}(a, b) \longleftrightarrow$

$\text{sats}(A, p, \text{Cons}(b, \text{Cons}(a, \text{Cons}(c, \text{Cons}(d, \text{Cons}(e, \text{Cons}(f,$

$\text{Cons}(g, \text{Cons}(h, \text{Cons}(i, \text{Cons}(j, \text{Cons}(k, \text{env}}))))))))))$ "

**shows**

" $\llbracket \text{nth}(i, \text{env}) = x; \text{nth}(j, \text{env}) = y; \text{nth}(k, \text{env}) = z;$

$i \in \text{nat}; j < \text{length}(\text{env}); k < \text{length}(\text{env}); \text{env} \in \text{list}(A) \rrbracket$

$\implies \text{is\_iterates}(\#\#A, \text{is\_F}, x, y, z) \longleftrightarrow$

$\text{sats}(A, \text{is\_iterates\_fm}(p, i, j, k), \text{env})$ "

*<proof>*

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  $\text{list}_N$ .

**theorem** *is\_iterates\_reflection:*

**assumes** *p\_reflection:*

" $\bigwedge f\ g\ h. \text{REFLECTS}[\lambda x. p(L, h(x), f(x), g(x)),$

$\lambda i\ x. p(\#\#L\text{set}(i), h(x), f(x), g(x))]$ "

**shows** " $\text{REFLECTS}[\lambda x. \text{is\_iterates}(L, p(L, x), f(x), g(x), h(x)),$

$\lambda i\ x. \text{is\_iterates}(\#\#L\text{set}(i), p(\#\#L\text{set}(i), x), f(x), g(x),$

$h(x))]$ "

*<proof>*

### 11.18.2 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"
<proof>
```

**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))"
<proof>
```

**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)"
<proof>
```

**theorem**  $eclose\_n\_reflection$ :

```
"REFLECTS[λx. is_eclose_n(L, f(x), g(x), h(x)),
  λi x. is_eclose_n(##Lset(i), f(x), g(x), h(x))]"
<proof>
```

### 11.18.3 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"
<proof>
```

**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))"
<proof>
```

**lemma**  $mem\_eclose\_iff\_sats$ :

```
"[nth(i,env) = x; nth(j,env) = y;
```

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

$$\implies \text{mem\_eclose}(\#\#A, x, y) \longleftrightarrow \text{sats}(A, \text{mem\_eclose\_fm}(i, j), \text{env})"$$

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

**theorem** *mem\_eclose\_reflection*:  

$$\text{"REFLECTS}[\lambda x. \text{mem\_eclose}(L, f(x), g(x)),$$

$$\lambda i x. \text{mem\_eclose}(\#\#L\text{set}(i), f(x), g(x))]"$$

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

#### 11.18.4 The Predicate “Is *eclose*(A)”

**definition**  

$$\text{is\_eclose\_fm} :: "[i, i] \Rightarrow i" \text{ where}$$

$$\text{"is\_eclose\_fm}(A, Z) \equiv$$

$$\text{Forall}(\text{Iff}(\text{Member}(0, \text{succ}(Z)), \text{mem\_eclose\_fm}(\text{succ}(A), 0)))"$$

**lemma** *is\_eclose\_type* [TC]:  

$$\text{"}\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \implies \text{is\_eclose\_fm}(x, y) \in \text{formula}"$$

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

**lemma** *sats\_is\_eclose\_fm* [simp]:  

$$\text{"}\llbracket x \in \text{nat}; y \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$$

$$\implies \text{sats}(A, \text{is\_eclose\_fm}(x, y), \text{env}) \longleftrightarrow \text{is\_eclose}(\#\#A, \text{nth}(x, \text{env}),$$

$$\text{nth}(y, \text{env}))"$$

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

**lemma** *is\_eclose\_iff\_sats*:  

$$\text{"}\llbracket \text{nth}(i, \text{env}) = x; \text{nth}(j, \text{env}) = y;$$

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

$$\implies \text{is\_eclose}(\#\#A, x, y) \longleftrightarrow \text{sats}(A, \text{is\_eclose\_fm}(i, j), \text{env})"$$

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

**theorem** *is\_eclose\_reflection*:  

$$\text{"REFLECTS}[\lambda x. \text{is\_eclose}(L, f(x), g(x)),$$

$$\lambda i x. \text{is\_eclose}(\#\#L\text{set}(i), f(x), g(x))]"$$

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

#### 11.18.5 The List Functor, Internalized

**definition**  

$$\text{list\_functor\_fm} :: "[i, i, i] \Rightarrow i" \text{ where}$$

$$\text{"list\_functor\_fm}(A, X, Z) \equiv$$

$$\text{Exists}(\text{Exists}(\text{And}(\text{number1\_fm}(1),$$

$$\text{And}(\text{cartprod\_fm}(A\#+2, X\#+2, 0), \text{sum\_fm}(1, 0, Z\#+2))))"$$

**lemma** *list\_functor\_type* [TC]:  

$$\text{"}\llbracket x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket \implies \text{list\_functor\_fm}(x, y, z) \in \text{formula}"$$

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

```

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))"
  <proof>

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)"
  <proof>

theorem list_functor_reflection:
  "REFLECTS[λx. is_list_functor(L,f(x),g(x),h(x)),
    λi x. is_list_functor(##Lset(i),f(x),g(x),h(x))]"
  <proof>

11.18.6 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"
  <proof>

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))"
  <proof>

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)"
  <proof>

theorem list_N_reflection:
  "REFLECTS[λx. is_list_N(L, f(x), g(x), h(x)),
    λi x. is_list_N(##Lset(i), f(x), g(x), h(x))]"
  <proof>

```

### 11.18.7 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"
⟨proof⟩
```

**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))"
⟨proof⟩
```

**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)"
⟨proof⟩
```

**theorem** mem\_list\_reflection:

```
"REFLECTS[λx. mem_list(L,f(x),g(x)),
  λi x. mem_list(##Lset(i),f(x),g(x))]"
⟨proof⟩
```

### 11.18.8 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"
⟨proof⟩
```

**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))"
⟨proof⟩
```

**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)"
⟨proof⟩
```

```

theorem is_list_reflection:
  "REFLECTS[ $\lambda x. \text{is\_list}(L, f(x), g(x)),$ 
     $\lambda i x. \text{is\_list}(\#\#L\text{set}(i), f(x), g(x))]$ ]"
<proof>

```

### 11.18.9 The Formula Functor, Internalized

definition formula\_functor\_fm :: "[i,i] $\Rightarrow$ i" where

```

"formula_functor_fm(X,Z)  $\equiv$ 
  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]:
  " $\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \Rightarrow \text{formula\_functor\_fm}(x,y) \in \text{formula}$ "
<proof>

```

```

lemma sats_formula_functor_fm [simp]:
  " $\llbracket x \in \text{nat}; y \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$ 
 $\Rightarrow \text{sats}(A, \text{formula\_functor\_fm}(x,y), \text{env}) \longleftrightarrow$ 
   $\text{is\_formula\_functor}(\#\#A, \text{nth}(x,\text{env}), \text{nth}(y,\text{env}))$ "
<proof>

```

```

lemma formula_functor_iff_sats:
  " $\llbracket \text{nth}(i,\text{env}) = x; \text{nth}(j,\text{env}) = y;
    i \in \text{nat}; j \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$ 
 $\Rightarrow \text{is\_formula\_functor}(\#\#A, x, y) \longleftrightarrow \text{sats}(A, \text{formula\_functor\_fm}(i,j),$ 
 $\text{env})$ "
<proof>

```

```

theorem formula_functor_reflection:
  "REFLECTS[ $\lambda x. \text{is\_formula\_functor}(L, f(x), g(x)),$ 
     $\lambda i x. \text{is\_formula\_functor}(\#\#L\text{set}(i), f(x), g(x))]$ ]"
<proof>

```

### 11.18.10 The Formula is\_formula\_N, Internalized

definition

```

formula_N_fm :: "[i,i] $\Rightarrow$ i" where
  "formula_N_fm(n,Z)  $\equiv$ 
    Exists(
      And(empty_fm(0),
        is_iterates_fm(formula_functor_fm(1,0), 0, n#+1, Z#+1)))"

```

```

lemma formula_N_fm_type [TC]:

```

" $\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \implies \text{formula\_N\_fm}(x,y) \in \text{formula}$ "  
 <proof>

**lemma** *sats\_formula\_N\_fm [simp]:*  
 " $\llbracket x < \text{length}(\text{env}); y < \text{length}(\text{env}); \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{sats}(A, \text{formula\_N\_fm}(x,y), \text{env}) \longleftrightarrow$   
 $\text{is\_formula\_N}(\#\#A, \text{nth}(x,\text{env}), \text{nth}(y,\text{env}))$ "  
 <proof>

**lemma** *formula\_N\_iff\_sats:*  
 " $\llbracket \text{nth}(i,\text{env}) = x; \text{nth}(j,\text{env}) = y;$   
 $i < \text{length}(\text{env}); j < \text{length}(\text{env}); \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{is\_formula\_N}(\#\#A, x, y) \longleftrightarrow \text{sats}(A, \text{formula\_N\_fm}(i,j), \text{env})$ "  
 <proof>

**theorem** *formula\_N\_reflection:*  
 "REFLECTS $[\lambda x. \text{is\_formula\_N}(L, f(x), g(x)),$   
 $\lambda i x. \text{is\_formula\_N}(\#\#L\text{set}(i), f(x), g(x))]$ "  
 <proof>

### 11.18.11 The Predicate "Is A Formula"

**definition**  
*mem\_formula\_fm* :: " $i \Rightarrow i$ " where  
 "*mem\_formula\_fm*( $x$ )  $\equiv$   
 Exists(Exists(  
 And(finite\_ordinal\_fm(1),  
 And(formula\_N\_fm(1,0), Member( $x\#+2,0$ ))))"

**lemma** *mem\_formula\_type [TC]:*  
 " $x \in \text{nat} \implies \text{mem\_formula\_fm}(x) \in \text{formula}$ "  
 <proof>

**lemma** *sats\_mem\_formula\_fm [simp]:*  
 " $\llbracket x \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{sats}(A, \text{mem\_formula\_fm}(x), \text{env}) \longleftrightarrow \text{mem\_formula}(\#\#A, \text{nth}(x,\text{env}))$ "  
 <proof>

**lemma** *mem\_formula\_iff\_sats:*  
 " $\llbracket \text{nth}(i,\text{env}) = x; i \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$   
 $\implies \text{mem\_formula}(\#\#A, x) \longleftrightarrow \text{sats}(A, \text{mem\_formula\_fm}(i), \text{env})$ "  
 <proof>

**theorem** *mem\_formula\_reflection:*  
 "REFLECTS $[\lambda x. \text{mem\_formula}(L,f(x)),$   
 $\lambda i x. \text{mem\_formula}(\#\#L\text{set}(i),f(x))]$ "  
 <proof>

### 11.18.12 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"
```

*<proof>*

**lemma** *sats\_is\_formula\_fm* [simp]:

```
"[x ∈ nat; env ∈ list(A)]
 ⇒ sats(A, is_formula_fm(x), env) ⟷ is_formula(##A, nth(x,env))"
```

*<proof>*

**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)"
```

*<proof>*

**theorem** *is\_formula\_reflection*:

```
"REFLECTS[λx. is_formula(L,f(x)),
 λi x. is_formula(##Lset(i),f(x))]"
```

*<proof>*

### 11.18.13 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"
```

*<proof>*

**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))"
⟨proof⟩

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)"
⟨proof⟩

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))]"
⟨proof⟩

end

```

## 12 Separation for Facts About Recursion

theory *Rec\_Separation* imports *Separation Internalize* begin

This theory proves all instances needed for locales  $M\_tranc1$  and  $M\_datatypes$

```

lemma eq_succ_imp_lt: "[i = succ(j); Ord(i)] ⇒ j < i"
⟨proof⟩

```

### 12.1 The Locale $M\_tranc1$

#### 12.1.1 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 ∈ nat; y ∈ nat; z ∈ nat⌋ ⇒ rtran_closure_mem_fm(x,y,z) ∈ formula"
⟨proof⟩

```

```

lemma sats_rtran_closure_mem_fm [simp]:
  "⌊x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)⌋
  ⇒ sats(A, rtran_closure_mem_fm(x,y,z), env) ⟷
    rtran_closure_mem(##A, nth(x,env), nth(y,env), nth(z,env))"
⟨proof⟩

```

```

lemma rtran_closure_mem_iff_sats:
  "⌊nth(i,env) = x; nth(j,env) = y; nth(k,env) = z;
    i ∈ nat; j ∈ nat; k ∈ nat; env ∈ list(A)⌋
  ⇒ rtran_closure_mem(##A, x, y, z) ⟷ sats(A, rtran_closure_mem_fm(i,j,k),
env)"
⟨proof⟩

```

```

lemma rtran_closure_mem_reflection:
  "REFLECTS[λx. rtran_closure_mem(L,f(x),g(x),h(x)),
    λi x. rtran_closure_mem(##Lset(i),f(x),g(x),h(x))]"
⟨proof⟩

```

Separation for  $r^*$ .

```

lemma rtranc1_separation:
  "⌊L(r); L(A)⌋ ⇒ separation (L, rtran_closure_mem(L,A,r))"
⟨proof⟩

```

### 12.1.2 Reflexive/Transitive Closure, Internalized

**definition**

$rtran\_closure\_fm :: "[i,i] \Rightarrow i$  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 ∈ nat; y ∈ nat] ⇒ rtran_closure_fm(x,y) ∈ formula"
⟨proof⟩

```

```

lemma sats_rtran_closure_fm [simp]:
  "[x ∈ nat; y ∈ nat; env ∈ list(A)]
  ⇒ sats(A, rtran_closure_fm(x,y), env) ⟷
    rtran_closure(##A, nth(x,env), nth(y,env))"
⟨proof⟩

```

```

lemma rtran_closure_iff_sats:
  "[nth(i,env) = x; nth(j,env) = y;
    i ∈ nat; j ∈ nat; env ∈ list(A)]
  ⇒ rtran_closure(##A, x, y) ⟷ sats(A, rtran_closure_fm(i,j),
env)"
⟨proof⟩

```

```

theorem rtran_closure_reflection:
  "REFLECTS[λx. rtran_closure(L,f(x),g(x)),
    λi x. rtran_closure(##Lset(i),f(x),g(x))]"
⟨proof⟩

```

### 12.1.3 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 ∈ nat; y ∈ nat] ⇒ tran_closure_fm(x,y) ∈ formula"
⟨proof⟩

```

```

lemma sats_tran_closure_fm [simp]:
  "[x ∈ nat; y ∈ nat; env ∈ list(A)]
  ⇒ sats(A, tran_closure_fm(x,y), env) ⟷
    tran_closure(##A, nth(x,env), nth(y,env))"
⟨proof⟩

```

```

lemma tran_closure_iff_sats:
  "[nth(i,env) = x; nth(j,env) = y;
    i ∈ nat; j ∈ nat; env ∈ list(A)]
  ⇒ tran_closure(##A, x, y) ⟷ sats(A, tran_closure_fm(i,j), env)"
⟨proof⟩

```

```

theorem tran_closure_reflection:
  "REFLECTS[ $\lambda x. \text{tran\_closure}(L, f(x), g(x)),$ 
     $\lambda i x. \text{tran\_closure}(\#\text{Lset}(i), f(x), g(x))]$ "
  <proof>

```

#### 12.1.4 Separation for the Proof of *wellfounded\_on\_trancl*

```

lemma wellfounded_trancl_reflects:
  "REFLECTS[ $\lambda x. \exists w[L]. \exists wx[L]. \exists rp[L].$ 
     $w \in Z \wedge \text{pair}(L, w, x, wx) \wedge \text{tran\_closure}(L, r, rp) \wedge wx \in$ 
     $rp,$ 
     $\lambda i x. \exists w \in \text{Lset}(i). \exists wx \in \text{Lset}(i). \exists rp \in \text{Lset}(i).$ 
     $w \in Z \wedge \text{pair}(\#\text{Lset}(i), w, x, wx) \wedge \text{tran\_closure}(\#\text{Lset}(i), r, rp)$ 
     $\wedge$ 
     $wx \in rp]$ "
  <proof>

```

```

lemma wellfounded_trancl_separation:
  " $\llbracket L(r); L(Z) \rrbracket \implies$ 
    separation  $(L, \lambda x.$ 
       $\exists w[L]. \exists wx[L]. \exists rp[L].$ 
       $w \in Z \wedge \text{pair}(L, w, x, wx) \wedge \text{tran\_closure}(L, r, rp) \wedge wx \in$ 
       $rp)$ "
  <proof>

```

#### 12.1.5 Instantiating the locale *M\_trancl*

```

lemma M_trancl_axioms_L: "M_trancl_axioms(L)"
  <proof>

```

```

theorem M_trancl_L: "M_trancl(L)"
  <proof>

```

```

interpretation L: M_trancl L <proof>

```

### 12.2 *L* is Closed Under the Operator *list*

#### 12.2.1 Instances of Replacement for Lists

```

lemma list_replacement1_Reflects:
  "REFLECTS
    [ $\lambda x. \exists u[L]. u \in B \wedge (\exists y[L]. \text{pair}(L, u, y, x) \wedge$ 
       $\text{is\_wfrec}(L, \text{iterates\_MH}(L, \text{is\_list\_functor}(L, A), 0), \text{memsn}, u,$ 
       $y)),$ 
     $\lambda i x. \exists u \in \text{Lset}(i). u \in B \wedge (\exists y \in \text{Lset}(i). \text{pair}(\#\text{Lset}(i), u, y,$ 
     $x) \wedge$ 
       $\text{is\_wfrec}(\#\text{Lset}(i),$ 
       $\text{iterates\_MH}(\#\text{Lset}(i),$ 

```

```

is_list_functor(##Lset(i), A), 0), memsn, u,
y))]"
<proof>

```

```

lemma list_replacement1:
  "L(A)  $\implies$  iterates_replacement(L, is_list_functor(L,A), 0)"
<proof>

```

```

lemma list_replacement2_Reflects:
  "REFLECTS
    [ $\lambda x. \exists u[L]. u \in B \wedge u \in \text{nat} \wedge$ 
      is_iterates(L, is_list_functor(L, A), 0, u, x),
      $\lambda i x. \exists u \in \text{Lset}(i). u \in B \wedge u \in \text{nat} \wedge$ 
      is_iterates(##Lset(i), is_list_functor(##Lset(i), A), 0,
u, x)]"
<proof>

```

```

lemma list_replacement2:
  "L(A)  $\implies$  strong_replacement(L,
     $\lambda n y. n \in \text{nat} \wedge \text{is\_iterates}(L, \text{is\_list\_functor}(L,A), 0, n, y))"$ 
<proof>

```

## 12.3 L is Closed Under the Operator *formula*

### 12.3.1 Instances of Replacement for Formulas

```

lemma formula_replacement1_Reflects:
  "REFLECTS
    [ $\lambda x. \exists u[L]. u \in B \wedge (\exists y[L]. \text{pair}(L,u,y,x) \wedge$ 
      is_wfrec(L, iterates_MH(L, is_formula_functor(L), 0), memsn,
u, y)),
      $\lambda i x. \exists u \in \text{Lset}(i). u \in B \wedge (\exists y \in \text{Lset}(i). \text{pair}(\text{##Lset}(i), u, y,
x) \wedge$ 
      is_wfrec(##Lset(i),
        iterates_MH(##Lset(i),
          is_formula_functor(##Lset(i)), 0), memsn, u,
y))]"
<proof>

```

```

lemma formula_replacement1:
  "iterates_replacement(L, is_formula_functor(L), 0)"
<proof>

```

```

lemma formula_replacement2_Reflects:
  "REFLECTS
    [ $\lambda x. \exists u[L]. u \in B \wedge u \in \text{nat} \wedge$ 
      is_iterates(L, is_formula_functor(L), 0, u, x),
      $\lambda i x. \exists u \in \text{Lset}(i). u \in B \wedge u \in \text{nat} \wedge$ 

```

```

                                is_iterates(##Lset(i), is_formula_functor(##Lset(i)), 0,
u, x)]"
⟨proof⟩

```

```

lemma formula_replacement2:
  "strong_replacement(L,
    λn y. n ∈ nat ∧ is_iterates(L, is_formula_functor(L), 0, n, y))"
⟨proof⟩

```

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

### 12.3.2 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(1), 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"
⟨proof⟩

```

```

lemma sats_nth_fm [simp]:
  "⌊x < length(env); y ∈ nat; z ∈ nat; env ∈ list(A)⌋
    ⇒ sats(A, nth_fm(x,y,z), env) ⇔
      is_nth(##A, nth(x,env), nth(y,env), nth(z,env))"
⟨proof⟩

```

```

lemma nth_iff_sats:
  "⌊nth(i,env) = x; nth(j,env) = y; nth(k,env) = z;
    i < length(env); j ∈ nat; k ∈ nat; env ∈ list(A)⌋
    ⇒ is_nth(##A, x, y, z) ⇔ sats(A, nth_fm(i,j,k), env)"
⟨proof⟩

```

```

theorem nth_reflection:
  "REFLECTS[λx. is_nth(L, f(x), g(x), h(x)),
    λi x. is_nth(##Lset(i), f(x), g(x), h(x))]"
⟨proof⟩

```

### 12.3.3 An Instance of Replacement for *nth*

```

lemma nth_replacement_Reflects:
  "REFLECTS
    [λx. ∃u[L]. u ∈ B ∧ (∃y[L]. pair(L,u,y,x) ∧
      is_wfrec(L, iterates_MH(L, is_tl(L), z), 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_tl(##Lset(i)), z), memsn, u, y))]
⟨proof⟩

```

```

lemma nth_replacement:
  "L(w) ⇒ iterates_replacement(L, is_tl(L), w)"
⟨proof⟩

```

### 12.3.4 Instantiating the locale $M\_datatypes$

```

lemma M_datatypes_axioms_L: "M_datatypes_axioms(L)"
⟨proof⟩

```

```

theorem M_datatypes_L: "M_datatypes(L)"
⟨proof⟩

```

```

interpretation L: M_datatypes L ⟨proof⟩

```

## 12.4 $L$ is Closed Under the Operator $eclose$

### 12.4.1 Instances of Replacement for $eclose$

```

lemma eclose_replacement1_Reflects:
  "REFLECTS
   [λx. ∃u[L]. u ∈ B ∧ (∃y[L]. pair(L,u,y,x) ∧
    is_wfrec(L, iterates_MH(L, big_union(L), A), 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), big_union(##Lset(i)), A),
             memsn, u, y))]"
⟨proof⟩

```

```

lemma eclose_replacement1:
  "L(A) ⇒ iterates_replacement(L, big_union(L), A)"
⟨proof⟩

```

```

lemma eclose_replacement2_Reflects:
  "REFLECTS
   [λx. ∃u[L]. u ∈ B ∧ u ∈ nat ∧
    is_iterates(L, big_union(L), A, u, x),
    λi x. ∃u ∈ Lset(i). u ∈ B ∧ u ∈ nat ∧
    is_iterates(##Lset(i), big_union(##Lset(i)), A, u, x)]"
⟨proof⟩

```

```

lemma eclose_replacement2:
  "L(A) ⇒ strong_replacement(L,
    λn y. n ∈ nat ∧ is_iterates(L, big_union(L), A, n, y))"

```

*<proof>*

#### 12.4.2 Instantiating the locale $M_{\text{eclose}}$

**lemma**  $M_{\text{eclose\_axioms\_L}}$ : " $M_{\text{eclose\_axioms}}(L)$ "  
*<proof>*

**theorem**  $M_{\text{eclose\_L}}$ : " $M_{\text{eclose}}(L)$ "  
*<proof>*

**interpretation**  $L$ :  $M_{\text{eclose}} L$  *<proof>*

**end**

### 13 Absoluteness for the Satisfies Relation on Formulas

**theory**  $Satisfies\_absolute$  imports  $Datatype\_absolute$   $Rec\_Separation$  **begin**

#### 13.1 More Internalization

##### 13.1.1 The Formula $is\_depth$ , Internalized

**definition**

$depth\_fm :: "[i,i] \Rightarrow i$ " **where**  
" $depth\_fm(p,n) \equiv$   
     $Exists(Exists(Exists($   
         $And(formula\_N\_fm(n\#+3,1),$   
         $And(Neg(Member(p\#+3,1)),$   
         $And(succ\_fm(n\#+3,2),$   
         $And(formula\_N\_fm(2,0), Member(p\#+3,0))))))$ "

**lemma**  $depth\_fm\_type$  [TC]:  
" $\llbracket x \in nat; y \in nat \rrbracket \implies depth\_fm(x,y) \in formula$ "  
*<proof>*

**lemma**  $sats\_depth\_fm$  [simp]:  
" $\llbracket x \in nat; y < length(env); env \in list(A) \rrbracket$   
     $\implies sats(A, depth\_fm(x,y), env) \longleftrightarrow$   
         $is\_depth(\#A, nth(x,env), nth(y,env))$ "  
*<proof>*

**lemma**  $depth\_iff\_sats$ :  
" $\llbracket nth(i,env) = x; nth(j,env) = y;$   
     $i \in nat; j < length(env); env \in list(A) \rrbracket$   
     $\implies is\_depth(\#A, x, y) \longleftrightarrow sats(A, depth\_fm(i,j), env)$ "  
*<proof>*



```

theorem depth_reflection:
  "REFLECTS[ $\lambda x. is\_depth(L, f(x), g(x)),$ 
     $\lambda i x. is\_depth(\#Lset(i), f(x), g(x))]$ "
<proof>

```

### 13.1.2 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]  $\Rightarrow$  i" where
  "formula_case_fm(is_a, is_b, is_c, is_d, v, z)  $\equiv$ 
    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  $\in$  formula; is_b  $\in$  formula; is_c  $\in$  formula; is_d  $\in$  formula;
    x  $\in$  nat; y  $\in$  nat]
 $\Rightarrow$  formula_case_fm(is_a, is_b, is_c, is_d, x, y)  $\in$  formula"
<proof>

```

**lemma** *sats\_formula\_case\_fm*:

assumes *is\_a\_iff\_sats*:

" $\bigwedge a0\ a1\ a2.$

$\llbracket a0 \in A; a1 \in A; a2 \in A \rrbracket$

$\Rightarrow ISA(a2, a1, a0) \longleftrightarrow sats(A, is\_a, Cons(a0, Cons(a1, Cons(a2, env))))$ "

and *is\_b\_iff\_sats*:

" $\bigwedge a0\ a1\ a2.$

$\llbracket a0 \in A; a1 \in A; a2 \in A \rrbracket$

$\Rightarrow ISB(a2, a1, a0) \longleftrightarrow sats(A, is\_b, Cons(a0, Cons(a1, Cons(a2, env))))$ "

and *is\_c\_iff\_sats*:

" $\bigwedge a0\ a1\ a2.$

$\llbracket a0 \in A; a1 \in A; a2 \in A \rrbracket$

```

     $\Rightarrow \text{ISC}(a2, a1, a0) \longleftrightarrow \text{sats}(A, \text{is\_c}, \text{Cons}(a0, \text{Cons}(a1, \text{Cons}(a2, \text{env}))))$ "
and is_d_iff_sats:
    " $\bigwedge a0\ a1.$ 
     $\llbracket a0 \in A; a1 \in A \rrbracket$ 
     $\Rightarrow \text{ISD}(a1, a0) \longleftrightarrow \text{sats}(A, \text{is\_d}, \text{Cons}(a0, \text{Cons}(a1, \text{env})))$ "
shows
    " $\llbracket x \in \text{nat}; y < \text{length}(\text{env}); \text{env} \in \text{list}(A) \rrbracket$ 
     $\Rightarrow \text{sats}(A, \text{formula\_case\_fm}(\text{is\_a}, \text{is\_b}, \text{is\_c}, \text{is\_d}, x, y), \text{env}) \longleftrightarrow$ 
     $\text{is\_formula\_case}(\#\#A, \text{ISA}, \text{ISB}, \text{ISC}, \text{ISD}, \text{nth}(x, \text{env}), \text{nth}(y, \text{env}))$ "
<proof>

lemma formula_case_iff_sats:
  assumes is_a_iff_sats:
    " $\bigwedge a0\ a1\ a2.$ 
     $\llbracket a0 \in A; a1 \in A; a2 \in A \rrbracket$ 
     $\Rightarrow \text{ISA}(a2, a1, a0) \longleftrightarrow \text{sats}(A, \text{is\_a}, \text{Cons}(a0, \text{Cons}(a1, \text{Cons}(a2, \text{env}))))$ "
  and is_b_iff_sats:
    " $\bigwedge a0\ a1\ a2.$ 
     $\llbracket a0 \in A; a1 \in A; a2 \in A \rrbracket$ 
     $\Rightarrow \text{ISB}(a2, a1, a0) \longleftrightarrow \text{sats}(A, \text{is\_b}, \text{Cons}(a0, \text{Cons}(a1, \text{Cons}(a2, \text{env}))))$ "
  and is_c_iff_sats:
    " $\bigwedge a0\ a1\ a2.$ 
     $\llbracket a0 \in A; a1 \in A; a2 \in A \rrbracket$ 
     $\Rightarrow \text{ISC}(a2, a1, a0) \longleftrightarrow \text{sats}(A, \text{is\_c}, \text{Cons}(a0, \text{Cons}(a1, \text{Cons}(a2, \text{env}))))$ "
  and is_d_iff_sats:
    " $\bigwedge a0\ a1.$ 
     $\llbracket a0 \in A; a1 \in A \rrbracket$ 
     $\Rightarrow \text{ISD}(a1, a0) \longleftrightarrow \text{sats}(A, \text{is\_d}, \text{Cons}(a0, \text{Cons}(a1, \text{env})))$ "
  shows
    " $\llbracket \text{nth}(i, \text{env}) = x; \text{nth}(j, \text{env}) = y; i \in \text{nat}; j < \text{length}(\text{env}); \text{env} \in \text{list}(A) \rrbracket$ 
     $\Rightarrow \text{is\_formula\_case}(\#\#A, \text{ISA}, \text{ISB}, \text{ISC}, \text{ISD}, x, y) \longleftrightarrow$ 
     $\text{sats}(A, \text{formula\_case\_fm}(\text{is\_a}, \text{is\_b}, \text{is\_c}, \text{is\_d}, i, j), \text{env})$ "
<proof>

```

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:
    " $\bigwedge h\ f\ g\ g'. \text{REFLECTS}[\lambda x. \text{is\_a}(L, h(x), f(x), g(x), g'(x)),$ 
     $\lambda i\ x. \text{is\_a}(\#\#L\text{set}(i), h(x), f(x), g(x), g'(x))]$ "
  and is_b_reflection:
    " $\bigwedge h\ f\ g\ g'. \text{REFLECTS}[\lambda x. \text{is\_b}(L, h(x), f(x), g(x), g'(x)),$ 
     $\lambda i\ x. \text{is\_b}(\#\#L\text{set}(i), h(x), f(x), g(x), g'(x))]$ "
  and is_c_reflection:
    " $\bigwedge h\ f\ g\ g'. \text{REFLECTS}[\lambda x. \text{is\_c}(L, h(x), f(x), g(x), g'(x)),$ 
     $\lambda i\ x. \text{is\_c}(\#\#L\text{set}(i), h(x), f(x), g(x), g'(x))]$ "
  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))]"
<proof>

```

### 13.2 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) ≡
  ∃ dp[M]. ∃ sdp[M]. ∃ 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 ∈ formula; M(z)]
  ⇒ is_depth_apply(M,h,p,z) ⟷ z = h ` succ(depth(p)) ` p"
<proof>

```

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) ≡
  \x y. \env ∈ list(A). bool_of_o (nth(x,env) ∈ nth(y,env))"

```

#### definition

```

satisfies_is_a :: "[i⇒o,i,i,i,i]⇒o" where
  "satisfies_is_a(M,A) ≡
  \x y zz. ∀ lA[M]. is_list(M,A,lA) ⟶
    is_lambda(M, lA,
      \env z. is_bool_of_o(M,
        ∃ nx[M]. ∃ ny[M].
          is_nth(M,x,env,nx) ∧ is_nth(M,y,env,ny) ∧ nx ∈
ny, z),
      zz)"

```

#### definition

```

satisfies_b :: "[i,i,i]⇒i" where
  "satisfies_b(A) ≡
  \x y. \env ∈ 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) ≡*  
 $\lambda x\ y\ zz. \forall lA[M]. is\_list(M,A,lA) \longrightarrow$   
 $is\_lambda(M, lA,$   
 $\lambda env\ z. is\_bool\_of\_o(M,$   
 $\exists nx[M]. is\_nth(M,x,env,nx) \wedge is\_nth(M,y,env,nx),$   
 $z),$   
 $zz)"$

**definition**

*satisfies\_c* :: "[i,i,i,i,i]⇒i" **where**  
*"satisfies\_c(A) ≡ λp q rp rq. λenv ∈ list(A). not(rp ‘ env and rq ‘ env)"*

**definition**

*satisfies\_is\_c* :: "[i⇒o,i,i,i,i,i]⇒o" **where**  
*"satisfies\_is\_c(M,A,h) ≡*  
 $\lambda p\ q\ zz. \forall lA[M]. is\_list(M,A,lA) \longrightarrow$   
 $is\_lambda(M, lA, \lambda env\ z. \exists hp[M]. \exists hq[M].$   
 $(\exists rp[M]. is\_depth\_apply(M,h,p,rp) \wedge fun\_apply(M,rp,env,hp))$   
 $\wedge$   
 $(\exists rq[M]. is\_depth\_apply(M,h,q,rq) \wedge fun\_apply(M,rq,env,hq))$   
 $\wedge$   
 $(\exists pq[M]. is\_and(M,hp,hq,pq) \wedge is\_not(M,pq,z)),$   
 $zz)"$

**definition**

*satisfies\_d* :: "[i,i,i]⇒i" **where**  
*"satisfies\_d(A)*  
 $\equiv \lambda p\ rp. \lambda env \in list(A). bool\_of\_o(\forall x \in A. rp\ ' (Cons(x,env)) = 1)"$

**definition**

*satisfies\_is\_d* :: "[i⇒o,i,i,i,i,i]⇒o" **where**  
*"satisfies\_is\_d(M,A,h) ≡*  
 $\lambda p\ zz. \forall lA[M]. is\_list(M,A,lA) \longrightarrow$   
 $is\_lambda(M, lA,$   
 $\lambda env\ z. \exists rp[M]. is\_depth\_apply(M,h,p,rp) \wedge$   
 $is\_bool\_of\_o(M,$   
 $\forall x[M]. \forall xenv[M]. \forall hp[M].$   
 $x \in A \longrightarrow is\_Cons(M,x,env,xenv) \longrightarrow$   
 $fun\_apply(M,rp,xenv,hp) \longrightarrow number1(M,hp),$   
 $z),$   
 $zz)"$

**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 ≡
  λM A u f z.
    ∀ fml[M]. is_formula(M,fml) →
      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,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)"

```

*<proof>*

Further constraints on the class *M* in order to prove absoluteness for the constants defined above. The ultimate goal is the absoluteness of the function *satisfies*.

**locale *M\_satisfies* = *M\_eclose* +**

**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)]]"
```

```

     $\Rightarrow$  strong_replacement
      (M,  $\lambda$ env z.  $\exists$ rpe[M].  $\exists$ rqe[M].  $\exists$ andpq[M].  $\exists$ notpq[M].
        fun_apply(M,rp,env,rpe)  $\wedge$  fun_apply(M,rq,env,rqe)  $\wedge$ 
        is_and(M,rpe,rqe, andpq)  $\wedge$  is_not(M, andpq, notpq)  $\wedge$ 
        env  $\in$  list(A)  $\wedge$  pair(M, env, notpq, z)))"
  and
  Forall_replacement:
    "[M(A); M(rp)]
     $\Rightarrow$  strong_replacement
      (M,  $\lambda$ env z.  $\exists$ bo[M].
        env  $\in$  list(A)  $\wedge$ 
        is_bool_of_o (M,
           $\forall$ a[M].  $\forall$ co[M].  $\forall$ rpco[M].
            a  $\in$  A  $\rightarrow$  is_Cons(M,a,env,co)  $\rightarrow$ 
            fun_apply(M,rp,co,rpco)  $\rightarrow$  number1(M,
rpco),
          bo)  $\wedge$ 
          pair(M,env,bo,z)))"
  and
  formula_rec_replacement:
    — For the transrec
    "[n  $\in$  nat; M(A)]  $\Rightarrow$  transrec_replacement(M, satisfies_MH(M,A), n)"
  and
  formula_rec_lambda_replacement:
    — For the  $\lambda$ -abstraction in the transrec body
    "[M(g); M(A)]  $\Rightarrow$ 
      strong_replacement (M,
         $\lambda$ x y. mem_formula(M,x)  $\wedge$ 
          ( $\exists$ 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)  $\wedge$ 
            pair(M, x, c, y)))"

lemma (in M_satisfies) Member_replacement':
  "[M(A); x  $\in$  nat; y  $\in$  nat]
   $\Rightarrow$  strong_replacement
    (M,  $\lambda$ env z. env  $\in$  list(A)  $\wedge$ 
      z =  $\langle$ env, bool_of_o(nth(x, env)  $\in$  nth(y, env)) $\rangle$ )"
  <proof>

lemma (in M_satisfies) Equal_replacement':
  "[M(A); x  $\in$  nat; y  $\in$  nat]
   $\Rightarrow$  strong_replacement
    (M,  $\lambda$ env z. env  $\in$  list(A)  $\wedge$ 
      z =  $\langle$ env, bool_of_o(nth(x, env) = nth(y, env)) $\rangle$ )"
  <proof>

```

```

lemma (in M_satisfies) Nand_replacement':
  "⌊M(A); M(rp); M(rq)⌋
  ⇒ strong_replacement
    (M, λenv z. env ∈ list(A) ∧ z = ⟨env, not(rp'env and rq'env)⟩)"
⟨proof⟩

lemma (in M_satisfies) Forall_replacement':
  "⌊M(A); M(rp)⌋
  ⇒ strong_replacement
    (M, λenv z.
      env ∈ list(A) ∧
      z = ⟨env, bool_of_o (∀a∈A. rp ' Cons(a,env) = 1)⟩)"
⟨proof⟩

lemma (in M_satisfies) a_closed:
  "⌊M(A); x∈nat; y∈nat⌋ ⇒ M(satisfies_a(A,x,y))"
⟨proof⟩

lemma (in M_satisfies) a_rel:
  "M(A) ⇒ Relation2(M, nat, nat, satisfies_is_a(M,A), satisfies_a(A))"
⟨proof⟩

lemma (in M_satisfies) b_closed:
  "⌊M(A); x∈nat; y∈nat⌋ ⇒ M(satisfies_b(A,x,y))"
⟨proof⟩

lemma (in M_satisfies) b_rel:
  "M(A) ⇒ Relation2(M, nat, nat, satisfies_is_b(M,A), satisfies_b(A))"
⟨proof⟩

lemma (in M_satisfies) c_closed:
  "⌊M(A); x ∈ formula; y ∈ formula; M(rx); M(ry)⌋
  ⇒ M(satisfies_c(A,x,y,rx,ry))"
⟨proof⟩

lemma (in M_satisfies) c_rel:
  "⌊M(A); M(f)⌋ ⇒
    Relation2 (M, formula, formula,
      satisfies_is_c(M,A,f),
      λu v. satisfies_c(A, u, v, f ' succ(depth(u)) ' u,
        f ' succ(depth(v)) ' v))"
⟨proof⟩

lemma (in M_satisfies) d_closed:
  "⌊M(A); x ∈ formula; M(rx)⌋ ⇒ M(satisfies_d(A,x,rx))"
⟨proof⟩

lemma (in M_satisfies) d_rel:
  "⌊M(A); M(f)⌋ ⇒

```

```

    Relation1(M, formula, satisfies_is_d(M,A,f),
      λu. satisfies_d(A, u, f ' succ(depth(u)) ' u))"
  ⟨proof⟩

```

```

lemma (in M_satisfies) fr_replace:
  "⌊n ∈ nat; M(A)⌋ ⇒ transrec_replacement(M,satisfies_MH(M,A),n)"
  ⟨proof⟩

```

```

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))"
  ⟨proof⟩

```

```

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)⟩)"
  ⟨proof⟩

```

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))"
  ⟨proof⟩

```

```

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))"
  ⟨proof⟩

```

```

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))"
<proof>

```

```

lemma (in M_satisfies) satisfies_abs:
  "[M(A); M(z); p ∈ formula]
  ⇒ is_satisfies(M,A,p,z) ⇔ z = satisfies(A,p)"
<proof>

```

### 13.3 Internalizations Needed to Instantiate $M_{satisfies}$

#### 13.3.1 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"
<proof>

```

```

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))"
<proof>

```

```

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)"
<proof>

```

```

lemma depth_apply_reflection:
  "REFLECTS[λx. is_depth_apply(L,f(x),g(x),h(x)),
    λi x. is_depth_apply(##Lset(i),f(x),g(x),h(x))]"
<proof>

```

### 13.3.2 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"
⟨proof⟩
```

**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))"
⟨proof⟩
```

**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)"
⟨proof⟩
```

**theorem** *satisfies\_is\_a\_reflection*:

```
"REFLECTS[λx. satisfies_is_a(L,f(x),g(x),h(x),g'(x)),
  λi x. satisfies_is_a(##Lset(i),f(x),g(x),h(x),g'(x))]"
⟨proof⟩
```

### 13.3.3 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]:

$$\llbracket A \in \text{nat}; x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket$$

$$\implies \text{satisfies\_is\_b\_fm}(A, x, y, z) \in \text{formula}$$

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

**lemma** *sats\_satisfies\_is\_b\_fm [simp]:*  

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

$$\implies \text{sats}(A, \text{satisfies\_is\_b\_fm}(u, x, y, z), \text{env}) \longleftrightarrow$$

$$\text{satisfies\_is\_b}(\#\#A, \text{nth}(u, \text{env}), \text{nth}(x, \text{env}), \text{nth}(y, \text{env}), \text{nth}(z, \text{env}))$$

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

**lemma** *satisfies\_is\_b\_iff\_sats:*  

$$\llbracket \text{nth}(u, \text{env}) = \text{nu}; \text{nth}(x, \text{env}) = \text{nx}; \text{nth}(y, \text{env}) = \text{ny}; \text{nth}(z, \text{env}) = \text{nz};$$

$$u \in \text{nat}; x < \text{length}(\text{env}); y < \text{length}(\text{env}); z \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$$

$$\implies \text{satisfies\_is\_b}(\#\#A, \text{nu}, \text{nx}, \text{ny}, \text{nz}) \longleftrightarrow$$

$$\text{sats}(A, \text{satisfies\_is\_b\_fm}(u, x, y, z), \text{env})$$

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

**theorem** *satisfies\_is\_b\_reflection:*  

$$\text{REFLECTS}[\lambda x. \text{satisfies\_is\_b}(L, f(x), g(x), h(x), g'(x)),$$

$$\lambda i x. \text{satisfies\_is\_b}(\#\#\text{Lset}(i), f(x), g(x), h(x), g'(x))]$$

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

### 13.3.4 The Operator *satisfies\_is\_c*, Internalized

**definition**  

$$\text{satisfies\_is\_c\_fm} :: "[i, i, i, i, i] \Rightarrow i" \text{ where}$$

$$\text{"satisfies\_is\_c\_fm}(A, h, p, q, zz) \equiv$$

$$\text{Forall}(\text{Implies}(\text{is\_list\_fm}(\text{succ}(A), 0),$$

$$\text{lambda\_fm}(\text{Exists}(\text{Exists}(\text{And}(\text{Exists}(\text{And}(\text{depth\_apply\_fm}(h\#+7, p\#+7, 0), \text{fun\_apply\_fm}(0, 4, 2))),$$

$$\text{And}(\text{Exists}(\text{And}(\text{depth\_apply\_fm}(h\#+7, q\#+7, 0), \text{fun\_apply\_fm}(0, 4, 1))),$$

$$\text{Exists}(\text{And}(\text{and\_fm}(2, 1, 0), \text{not\_fm}(0, 3)))))$$

$$0, \text{succ}(zz)))$$

**lemma** *satisfies\_is\_c\_type [TC]:*  

$$\llbracket A \in \text{nat}; h \in \text{nat}; x \in \text{nat}; y \in \text{nat}; z \in \text{nat} \rrbracket$$

$$\implies \text{satisfies\_is\_c\_fm}(A, h, x, y, z) \in \text{formula}$$

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

**lemma** *sats\_satisfies\_is\_c\_fm [simp]:*  

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

$$\implies \text{sats}(A, \text{satisfies\_is\_c\_fm}(u, v, x, y, z), \text{env}) \longleftrightarrow$$

$$\text{satisfies\_is\_c}(\#\#A, \text{nth}(u, \text{env}), \text{nth}(v, \text{env}), \text{nth}(x, \text{env}),$$

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

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

**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)"
⟨proof⟩

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))]"
  ⟨proof⟩

```

### 13.3.5 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"
⟨proof⟩

```

```

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))"
  ⟨proof⟩

```

```

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)"
  ⟨proof⟩

```

```

theorem satisfies_is_d_reflection:
  "REFLECTS[ $\lambda x. \text{satisfies\_is\_d}(L, f(x), g(x), h(x), g'(x)),$ 
     $\lambda i x. \text{satisfies\_is\_d}(\#\#L\text{set}(i), f(x), g(x), h(x), g'(x))]$ ]"
  <proof>

```

### 13.3.6 The Operator *satisfies\_MH*, Internalized

**definition**

```

satisfies_MH_fm :: "[i,i,i,i] $\Rightarrow$ i" where
"satisfies_MH_fm(A,u,f,zz)  $\equiv$ 
  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]:

```

" $\llbracket A \in \text{nat}; u \in \text{nat}; x \in \text{nat}; z \in \text{nat} \rrbracket$ 
 $\Rightarrow \text{satisfies\_MH\_fm}(A,u,x,z) \in \text{formula}$ "
<proof>

```

**lemma** sats\_satisfies\_MH\_fm [simp]:

```

" $\llbracket u \in \text{nat}; x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$ 
 $\Rightarrow \text{sats}(A, \text{satisfies\_MH\_fm}(u,x,y,z), \text{env}) \longleftrightarrow$ 
   $\text{satisfies\_MH}(\#\#A, \text{nth}(u,\text{env}), \text{nth}(x,\text{env}), \text{nth}(y,\text{env}), \text{nth}(z,\text{env}))$ "
<proof>

```

**lemma** satisfies\_MH\_iff\_sats:

```

" $\llbracket \text{nth}(u,\text{env}) = \text{nu}; \text{nth}(x,\text{env}) = \text{nx}; \text{nth}(y,\text{env}) = \text{ny}; \text{nth}(z,\text{env}) = \text{nz};$ 
   $u \in \text{nat}; x \in \text{nat}; y \in \text{nat}; z \in \text{nat}; \text{env} \in \text{list}(A) \rrbracket$ 
 $\Rightarrow \text{satisfies\_MH}(\#\#A, \text{nu}, \text{nx}, \text{ny}, \text{nz}) \longleftrightarrow$ 
   $\text{sats}(A, \text{satisfies\_MH\_fm}(u,x,y,z), \text{env})$ "
<proof>

```

**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[ $\lambda x. \text{satisfies\_MH}(L, f(x), g(x), h(x), g'(x)),$ 
   $\lambda i x. \text{satisfies\_MH}(\#\#L\text{set}(i), f(x), g(x), h(x), g'(x))]$ ]"
  <proof>

```

## 13.4 Lemmas for Instantiating the Locale $M_{satisfies}$

### 13.4.1 The Member Case

**lemma** *Member\_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))]]"
⟨proof⟩
```

**lemma** *Member\_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))"
⟨proof⟩
```

### 13.4.2 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))]]"
⟨proof⟩
```

**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))"

```

$\langle proof \rangle$

### 13.4.3 The Nand Case

**lemma** *Nand\_Reflects*:

```
"REFLECTS [ $\lambda x. \exists u[L]. u \in B \wedge$ 
  ( $\exists rpe[L]. \exists rqe[L]. \exists andpq[L]. \exists notpq[L].$ 
     $fun\_apply(L, rp, u, rpe) \wedge fun\_apply(L, rq, u, rqe) \wedge$ 
     $is\_and(L, rpe, rqe, andpq) \wedge is\_not(L, andpq, notpq)$ 
 $\wedge$ 
     $u \in list(A) \wedge pair(L, u, notpq, x)$ ),
 $\lambda i x. \exists u \in Lset(i). u \in B \wedge$ 
  ( $\exists rpe \in Lset(i). \exists rqe \in Lset(i). \exists andpq \in Lset(i). \exists notpq \in Lset(i).$ 
     $fun\_apply(\#\#Lset(i), rp, u, rpe) \wedge fun\_apply(\#\#Lset(i), rq, u,$ 
 $rqe) \wedge$ 
     $is\_and(\#\#Lset(i), rpe, rqe, andpq) \wedge is\_not(\#\#Lset(i), andpq, notpq)$ 
 $\wedge$ 
     $u \in list(A) \wedge pair(\#\#Lset(i), u, notpq, x)$ )]"
```

$\langle proof \rangle$

**lemma** *Nand\_replacement*:

```
"[ $L(A); L(rp); L(rq)$ ]"
 $\implies strong\_replacement$ 
  ( $L, \lambda env z. \exists rpe[L]. \exists rqe[L]. \exists andpq[L]. \exists notpq[L].$ 
     $fun\_apply(L, rp, env, rpe) \wedge fun\_apply(L, rq, env, rqe) \wedge$ 
     $is\_and(L, rpe, rqe, andpq) \wedge is\_not(L, andpq, notpq) \wedge$ 
     $env \in list(A) \wedge pair(L, env, notpq, z)$ )"
```

$\langle proof \rangle$

### 13.4.4 The Forall Case

**lemma** *Forall\_Reflects*:

```
"REFLECTS [ $\lambda x. \exists u[L]. u \in B \wedge (\exists bo[L]. u \in list(A) \wedge$ 
   $is\_bool\_of\_o(L,$ 
     $\forall a[L]. \forall co[L]. \forall rpco[L]. a \in A \implies$ 
     $is\_Cons(L, a, u, co) \implies fun\_apply(L, rp, co, rpco) \implies$ 
     $number1(L, rpco),$ 
     $bo) \wedge pair(L, u, bo, x)$ ),
 $\lambda i x. \exists u \in Lset(i). u \in B \wedge (\exists bo \in Lset(i). u \in list(A) \wedge$ 
   $is\_bool\_of\_o(\#\#Lset(i),$ 
     $\forall a \in Lset(i). \forall co \in Lset(i). \forall rpco \in Lset(i). a \in A \implies$ 
     $is\_Cons(\#\#Lset(i), a, u, co) \implies fun\_apply(\#\#Lset(i), rp, co, rpco)$ 
 $\implies$ 
     $number1(\#\#Lset(i), rpco),$ 
     $bo) \wedge pair(\#\#Lset(i), u, bo, x)$ )]"
```

$\langle proof \rangle$

**lemma** *Forall\_replacement*:

```
"[ $L(A); L(rp)$ ]"
 $\implies strong\_replacement$ 
```

```

(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))"
⟨proof⟩

```

#### 13.4.5 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))]"]"
⟨proof⟩

```

```

lemma formula_rec_replacement:
  — For the transrec
  "[n ∈ nat; L(A)] ⇒ transrec_replacement(L, satisfies_MH(L, A), n)"
⟨proof⟩

```

#### 13.4.6 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))]"]"
⟨proof⟩

```

```

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)))"
⟨proof⟩

```

### 13.5 Instantiating $M_{\text{satisfies}}$

```

lemma M_satisfies_axioms_L: "M_satisfies_axioms(L)"
⟨proof⟩

```

```

theorem M_satisfies_L: "M_satisfies(L)"
⟨proof⟩

```

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

## 14 Absoluteness for the Definable Powerset Function

```

theory DPow_absolute imports Satisfies_absolute begin

```

### 14.1 Preliminary Internalizations

#### 14.1.1 The Operator $\text{is\_formula\_rec}$

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"
⟨proof⟩

```

```

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))"
⟨proof⟩

```

```

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)"
    ⟨proof⟩

```

```

theorem formula_rec_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_formula_rec(L, MH(L,x), f(x), h(x)),
    λi x. is_formula_rec(##Lset(i), MH(##Lset(i),x), f(x),
h(x))]"
  ⟨proof⟩

```

### 14.1.2 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"
  ⟨proof⟩

```

```

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))"
  <proof>

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)"
  <proof>

theorem satisfies_reflection:
  "REFLECTS[λx. is_satisfies(L,f(x),g(x),h(x)),
    λi x. is_satisfies(##Lset(i),f(x),g(x),h(x))]"
  <proof>

```

## 14.2 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))}}}"
  <proof>

```

Relativize the use of  $\lambda A \ p \ env. \text{sats}(A, p, env)$  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))"
  <proof>

```

```

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))}"
  <proof>

```

### 14.2.1 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"
  <proof>

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))"
  <proof>

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)"
  <proof>

theorem DPow_sats_reflection:
  "REFLECTS[λx. is_DPow_sats(L,f(x),g(x),h(x),g'(x)),
    λi x. is_DPow_sats(##Lset(i),f(x),g(x),h(x),g'(x))]"
  <proof>



### 14.3 A Locale for Relativizing the Operator DPow'



locale M_DPow = M_satisfies +
  assumes sep:
    "⟦M(A); env ∈ list(A); p ∈ formula⟧
      ⇒ separation(M, λx. is_DPow_sats(M,A,env,p,x))"
  and rep:
    "M(A)
      ⇒ strong_replacement (M,
        λep z. ∃ env[M]. ∃ p[M]. mem_formula(M,p) ∧ mem_list(M,A,env)
        ∧
          pair(M,env,p,ep) ∧
          is_Collect(M, A, λx. is_DPow_sats(M,A,env,p,x), z))"

lemma (in M_DPow) sep':
  "⟦M(A); env ∈ list(A); p ∈ formula⟧
    ⇒ separation(M, λx. sats(A, p, Cons(x,env)))"
  <proof>

```

```

lemma (in M_DPow) rep':
  "M(A)
  ⇒ strong_replacement (M,
    λep z. ∃ env ∈ list(A). ∃ p ∈ formula.
      ep = ⟨env, p⟩ ∧ z = {x ∈ A . sats(A, p, Cons(x, env))})"
  ⟨proof⟩

```

```

lemma univalent_pair_eq:
  "univalent (M, A, λxy z. ∃ x ∈ B. ∃ y ∈ C. xy = ⟨x, y⟩ ∧ z = f(x, y))"
  ⟨proof⟩

```

```

lemma (in M_DPow) DPow'_closed: "M(A) ⇒ M(DPow'(A))"
  ⟨proof⟩

```

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)"
  ⟨proof⟩

```

## 14.4 Instantiating the Locale $M\_DPow$

### 14.4.1 The Instance of Separation

```

lemma DPow_separation:
  "⌊L(A); env ∈ list(A); p ∈ formula⌋
  ⇒ separation (L, λx. is_DPow_sats (L, A, env, p, x))"
  ⟨proof⟩

```

### 14.4.2 The Instance of Replacement

```

lemma DPow_replacement_Reflects:
  "REFLECTS [λx. ∃ 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)),
  λi 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))]"
⟨proof⟩

```

```

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))"
⟨proof⟩

```

### 14.4.3 Actually Instantiating the Locale

```

lemma M_DPow_axioms_L: "M_DPow_axioms(L)"
⟨proof⟩

```

```

theorem M_DPow_L: "M_DPow(L)"
⟨proof⟩

```

```

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]

```

### 14.4.4 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"
⟨proof⟩

```

```

lemma sats_Collect_fm:
  assumes is_P_iff_sats:
    "∧a. a ∈ A ⇒ is_P(a) ↔ sats(A, p, Cons(a, env))"
  shows
    "[[x ∈ nat; y ∈ nat; env ∈ list(A)]
    ⇒ sats(A, Collect_fm(x,p,y), env) ↔
      is_Collect(##A, nth(x,env), is_P, nth(y,env))"
⟨proof⟩

```

```

lemma Collect_iff_sats:
  assumes is_P_iff_sats:
    " $\bigwedge a. a \in A \implies is\_P(a) \longleftrightarrow sats(A, p, Cons(a, env))$ "
  shows
    " $\llbracket nth(i, env) = x; nth(j, env) = y; i \in nat; j \in nat; env \in list(A) \rrbracket \implies is\_Collect(\#A, x, is\_P, y) \longleftrightarrow sats(A, Collect\_fm(i, p, j), env)$ "
  <proof>

```

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:
    " $\bigwedge h f g. REFLECTS[\lambda x. is\_P(L, f(x), g(x)), \lambda i x. is\_P(\#Lset(i), f(x), g(x))]$ "
  shows " $REFLECTS[\lambda x. is\_Collect(L, f(x), is\_P(L, x), g(x)), \lambda i x. is\_Collect(\#Lset(i), f(x), is\_P(\#Lset(i), x), g(x))]$ "
  <proof>

```

#### 14.4.5 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]  $\Rightarrow$  i" where
    "Replace_fm(A, is_P, z)  $\equiv$ 
      Forall(Iff(Member(0, succ(z)),
        Exists(And(Member(0, A#+2), is_P))))"

```

```

lemma is_Replace_type [TC]:
  " $\llbracket is\_P \in formula; x \in nat; y \in nat \rrbracket \implies Replace\_fm(x, is\_P, y) \in formula$ "
  <proof>

```

```

lemma sats_Replace_fm:
  assumes is_P_iff_sats:
    " $\bigwedge a b. \llbracket a \in A; b \in A \rrbracket \implies is\_P(a, b) \longleftrightarrow sats(A, p, Cons(a, Cons(b, env)))$ "
  shows
    " $\llbracket x \in nat; y \in nat; env \in list(A) \rrbracket \implies sats(A, Replace\_fm(x, p, y), env) \longleftrightarrow is\_Replace(\#A, nth(x, env), is\_P, nth(y, env))$ "
  <proof>

```

```

lemma Replace_iff_sats:
  assumes is_P_iff_sats:
    " $\bigwedge a b. \llbracket a \in A; b \in A \rrbracket \implies is\_P(a, b) \longleftrightarrow sats(A, p, Cons(a, Cons(b, env)))$ "

```

```

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)"
⟨proof⟩

```

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))]"
⟨proof⟩

```

#### 14.4.6 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"
⟨proof⟩

```

```

lemma sats_DPow'_fm [simp]:
  "[x ∈ nat; y ∈ nat; env ∈ list(A)]
  ⇒ sats(A, DPow'_fm(x,y), env) ⇔
     is_DPow'(##A, nth(x,env), nth(y,env))"
⟨proof⟩

```

```

lemma DPow'_iff_sats:
  "[nth(i,env) = x; nth(j,env) = y;
   i ∈ nat; j ∈ nat; env ∈ list(A)]
  ⇒ is_DPow'(##A, x, y) ⇔ sats(A, DPow'_fm(i,j), env)"
⟨proof⟩

```

```

theorem DPow'_reflection:
  "REFLECTS[λx. is_DPow'(L,f(x),g(x)),
            λi x. is_DPow'(##Lset(i),f(x),g(x))]"

```



$\langle proof \rangle$

## 14.5 A Locale for Relativizing the Operator $Lset$

**definition**

```
transrec_body :: "[i $\Rightarrow$ o,i,i,i,i]  $\Rightarrow$  o" where
  "transrec_body(M,g,x)  $\equiv$ 
     $\lambda y z. \exists gy[M]. y \in x \wedge \text{fun\_apply}(M,g,y,gy) \wedge \text{is\_DPow}'(M,gy,z)"$ 
```

**lemma** (in  $M\_DPow$ ) *transrec\_body\_abs*:

```
" $\llbracket M(x); M(g); M(z) \rrbracket$ 
 $\implies \text{transrec\_body}(M,g,x,y,z) \longleftrightarrow y \in x \wedge z = \text{DPow}'(g'y)"$ 
```

$\langle proof \rangle$

**locale**  $M\_Lset = M\_DPow +$

**assumes** *strong\_rep*:

```
" $\llbracket M(x); M(g) \rrbracket \implies \text{strong\_replacement}(M, \lambda y z. \text{transrec\_body}(M,g,x,y,z))"$ 
```

**and** *transrec\_rep*:

```
" $M(i) \implies \text{transrec\_replacement}(M, \lambda x f u.
  \exists r[M]. \text{is\_Replace}(M, x, \text{transrec\_body}(M,f,x), r) \wedge
  \text{big\_union}(M, r, u), i)"$ 
```

**lemma** (in  $M\_Lset$ ) *strong\_rep'*:

```
" $\llbracket M(x); M(g) \rrbracket
\implies \text{strong\_replacement}(M, \lambda y z. y \in x \wedge z = \text{DPow}'(g'y))"$ 
```

$\langle proof \rangle$

**lemma** (in  $M\_Lset$ ) *DPow\_apply\_closed*:

```
" $\llbracket M(f); M(x); y \in x \rrbracket \implies M(\text{DPow}'(f'y))"$ 
```

$\langle proof \rangle$

**lemma** (in  $M\_Lset$ ) *RepFun\_DPow\_apply\_closed*:

```
" $\llbracket M(f); M(x) \rrbracket \implies M(\{\text{DPow}'(f'y). y \in x\})"$ 
```

$\langle proof \rangle$

**lemma** (in  $M\_Lset$ ) *RepFun\_DPow\_abs*:

```
" $\llbracket M(x); M(f); M(r) \rrbracket
\implies \text{is\_Replace}(M, x, \lambda y z. \text{transrec\_body}(M,f,x,y,z), r) \longleftrightarrow
  r = \{\text{DPow}'(f'y). y \in x\}"$ 
```

$\langle proof \rangle$

**lemma** (in  $M\_Lset$ ) *transrec\_rep'*:

```
" $M(i) \implies \text{transrec\_replacement}(M, \lambda x f u. u = (\bigcup_{y \in x. \text{DPow}'(f'y)),
i)"$ 
```

$\langle proof \rangle$

Relativization of the Operator  $Lset$

**definition**

$is\_Lset :: "[i \Rightarrow o, i, i] \Rightarrow o"$  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.  
 $"is\_Lset(M,a,z) \equiv is\_transrec(M, \lambda x f u. u = (\bigcup_{y \in x}. DPow'(f'y)), a, z)"$

**lemma** (in  $M\_Lset$ )  $Lset\_abs$ :  
 $"[Ord(i); M(i); M(z)] \implies is\_Lset(M,i,z) \longleftrightarrow z = Lset(i)"$   
 $\langle proof \rangle$

**lemma** (in  $M\_Lset$ )  $Lset\_closed$ :  
 $"[Ord(i); M(i)] \implies M(Lset(i))"$   
 $\langle proof \rangle$

## 14.6 Instantiating the Locale $M\_Lset$

### 14.6.1 The First Instance of Replacement

**lemma**  $strong\_rep\_Reflects$ :  
 $"REFLECTS [\lambda u. \exists v[L]. v \in B \wedge (\exists gy[L]. v \in x \wedge fun\_apply(L,g,v,gy) \wedge is\_DPow'(L,gy,u)), \lambda i u. \exists v \in Lset(i). v \in B \wedge (\exists gy \in Lset(i). v \in x \wedge fun\_apply(##Lset(i),g,v,gy) \wedge is\_DPow'(##Lset(i),gy,u))]"$   
 $\langle proof \rangle$

**lemma**  $strong\_rep$ :  
 $"[L(x); L(g)] \implies strong\_replacement(L, \lambda y z. transrec\_body(L,g,x,y,z))"$   
 $\langle proof \rangle$

### 14.6.2 The Second Instance of Replacement

**lemma**  $transrec\_rep\_Reflects$ :  
 $"REFLECTS [\lambda x. \exists v[L]. v \in B \wedge (\exists y[L]. pair(L,v,y,x) \wedge is\_wfrec(L, \lambda x f u. \exists r[L]. is\_Replace(L, x, \lambda y z. \exists gy[L]. y \in x \wedge fun\_apply(L,f,y,gy) \wedge is\_DPow'(L,gy,z), r) \wedge big\_union(L,r,u), mr, v, y)), \lambda i x. \exists v \in Lset(i). v \in B \wedge (\exists y \in Lset(i). pair(##Lset(i),v,y,x) \wedge is\_wfrec(##Lset(i), \lambda x f u. \exists r \in Lset(i). is\_Replace(##Lset(i), x, \lambda y z. \exists gy \in Lset(i). y \in x \wedge fun\_apply(##Lset(i),f,y,gy) \wedge is\_DPow'(##Lset(i),gy,z), r) \wedge big\_union(##Lset(i),r,u), mr, v, y))]"$   
 $\langle proof \rangle$

```

lemma transrec_rep:
  "[[L(j)]]
  ⇒ transrec_replacement(L, λx f u.
    ∃ r[L]. is_Replace(L, x, transrec_body(L,f,x), r) ∧
    big_union(L, r, u), j)"
⟨proof⟩

```

### 14.6.3 Actually Instantiating $M\_Lset$

```

lemma M_Lset_axioms_L: "M_Lset_axioms(L)"
⟨proof⟩

```

```

theorem M_Lset_L: "M_Lset(L)"
⟨proof⟩

```

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]

```

## 14.7 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)"
⟨proof⟩

```

end

## 15 The Axiom of Choice Holds in L!

```

theory AC_in_L imports Formula Separation begin

```

### 15.1 Extending a Wellordering over a List – Lexicographic Power

This could be moved into a library.

```

consts
  rlist    :: "[i,i]⇒i"

inductive
  domains "rlist(A,r)" ⊆ "list(A) * list(A)"
  intros
    shorterI:

```

```
"[[length(l') < length(l); l' ∈ list(A); l ∈ list(A)]]
⇒ <l', l> ∈ rlist(A,r)"
```

```
sameI:
"[[<l',l> ∈ rlist(A,r); a ∈ A]]
⇒ <Cons(a,l'), Cons(a,l)> ∈ rlist(A,r)"
```

```
diffI:
"[[length(l') = length(l); <a',a> ∈ r;
  l' ∈ list(A); l ∈ list(A); a' ∈ A; a ∈ A]]
⇒ <Cons(a',l'), Cons(a,l)> ∈ rlist(A,r)"
```

```
type_intros list.intros
```

### 15.1.1 Type checking

```
lemmas rlist_type = rlist.dom_subset
```

```
lemmas field_rlist = rlist_type [THEN field_rel_subset]
```

### 15.1.2 Linearity

```
lemma rlist_Nil_Cons [intro]:
"[[a ∈ A; l ∈ list(A)]] ⇒ <[], Cons(a,l)> ∈ rlist(A, r)"
<proof>
```

```
lemma linear_rlist:
  assumes r: "linear(A,r)" shows "linear(list(A),rlist(A,r))"
<proof>
```

### 15.1.3 Well-foundedness

Nothing preceeds Nil in this ordering.

```
inductive_cases rlist_NilE: " <l, []> ∈ rlist(A,r)"
```

```
inductive_cases rlist_ConsE: " <l', Cons(x,l)> ∈ rlist(A,r)"
```

```
lemma not_rlist_Nil [simp]: " <l, []> ∉ rlist(A,r)"
<proof>
```

```
lemma rlist_imp_length_le: "<l',l> ∈ rlist(A,r) ⇒ length(l') ≤ length(l)"
<proof>
```

```
lemma wf_on_rlist_n:
"[[n ∈ nat; wf[A](r)]] ⇒ wf[{l ∈ list(A). length(l) = n}](rlist(A,r))"
<proof>
```

```
lemma list_eq_UN_length: "list(A) = (⋃ n ∈ nat. {l ∈ list(A). length(l)
= n})"
<proof>
```

```
lemma wf_on_rlist: "wf[A](r)  $\implies$  wf[list(A)](rlist(A,r))"
<proof>
```

```
lemma wf_rlist: "wf(r)  $\implies$  wf(rlist(field(r),r))"
<proof>
```

```
lemma well_ord_rlist:
  "well_ord(A,r)  $\implies$  well_ord(list(A), rlist(A,r))"
<proof>
```

## 15.2 An Injection from Formulas into the Natural Numbers

There is a well-known bijection between  $\text{nat} \times \text{nat}$  and  $\text{nat}$  given by the expression  $f(m,n) = \text{triangle}(m+n) + m$ , where  $\text{triangle}(k)$  enumerates the triangular numbers and can be defined by  $\text{triangle}(0)=0$ ,  $\text{triangle}(\text{succ}(k)) = \text{succ}(k + \text{triangle}(k))$ . Some small amount of effort is needed to show that  $f$  is a bijection. We already know that such a bijection exists by the theorem `well_ord_InfCard_square_eq`:

$$\llbracket \text{well\_ord}(A, r); \text{InfCard}(|A|) \rrbracket \implies A \times A \approx A$$

However, this result merely states that there is a bijection between the two sets. It provides no means of naming a specific bijection. Therefore, we conduct the proofs under the assumption that a bijection exists. The simplest way to organize this is to use a locale.

Locale for any arbitrary injection between  $\text{nat} \times \text{nat}$  and  $\text{nat}$

```
locale Nat_Times_Nat =
  fixes fn
  assumes fn_inj: "fn  $\in$  inj(nat*nat, nat)"
```

```
consts   enum :: "[i,i] $\Rightarrow$ i"
primrec
  "enum(f, Member(x,y)) = f ' <0, f ' <x,y>"
  "enum(f, Equal(x,y)) = f ' <1, f ' <x,y>"
  "enum(f, Nand(p,q)) = f ' <2, f ' <enum(f,p), enum(f,q)>>"
  "enum(f, Forall(p)) = f ' <succ(2), enum(f,p)>"
```

```
lemma (in Nat_Times_Nat) fn_type [TC,simp]:
  " $\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \implies \text{fn}'\langle x,y \rangle \in \text{nat}$ "
<proof>
```

```
lemma (in Nat_Times_Nat) fn_iff:
  " $\llbracket x \in \text{nat}; y \in \text{nat}; u \in \text{nat}; v \in \text{nat} \rrbracket$ "
```

$\implies (fn\langle x,y \rangle = fn\langle u,v \rangle) \longleftrightarrow (x=u \wedge y=v)"$   
 $\langle proof \rangle$

**lemma** (in Nat\_Times\_Nat) enum\_type [TC,simp]:  
 "p ∈ formula  $\implies$  enum(fn,p) ∈ nat"  
 $\langle proof \rangle$

**lemma** (in Nat\_Times\_Nat) enum\_inject [rule\_format]:  
 "p ∈ formula  $\implies \forall q \in formula. enum(fn,p) = enum(fn,q) \longrightarrow p=q"$   
 $\langle proof \rangle$

**lemma** (in Nat\_Times\_Nat) inj\_formula\_nat:  
 " $(\lambda p \in formula. enum(fn,p)) \in inj(formula, nat)"$   
 $\langle proof \rangle$

**lemma** (in Nat\_Times\_Nat) well\_ord\_formula:  
 "well\_ord(formula, measure(formula, enum(fn)))"  
 $\langle proof \rangle$

**lemmas** nat\_times\_nat\_lepoll\_nat =  
 InfCard\_nat [THEN InfCard\_square\_eqpoll, THEN eqpoll\_imp\_lepoll]

Not needed—but interesting?

**theorem** formula\_lepoll\_nat: "formula  $\lesssim$  nat"  
 $\langle proof \rangle$

### 15.3 Defining the Wellordering on $DPow(A)$

The objective is to build a wellordering on  $DPow(A)$  from a given one on  $A$ . We first introduce wellorderings for environments, which are lists built over  $A$ . We combine it with the enumeration of formulas. The order type of the resulting wellordering gives us a map from (environment, formula) pairs into the ordinals. For each member of  $DPow(A)$ , we take the minimum such ordinal.

**definition**

env\_form\_r :: "[i,i,i] $\Rightarrow$ i" where  
 — wellordering on (environment, formula) pairs  
 "env\_form\_r(f,r,A)  $\equiv$   
 rmult(list(A), rlist(A, r),  
 formula, measure(formula, enum(f)))"

**definition**

env\_form\_map :: "[i,i,i,i] $\Rightarrow$ i" where  
 — map from (environment, formula) pairs to ordinals  
 "env\_form\_map(f,r,A,z)  
 $\equiv$  ordermap(list(A) \* formula, env\_form\_r(f,r,A)) ‘ z"

**definition**

```

DPow_ord :: "[i,i,i,i,i]⇒o" where
  — predicate that holds if k is a valid index for X
  "DPow_ord(f,r,A,X,k) ≡
    ∃ env ∈ list(A). ∃ p ∈ formula.
      arity(p) ≤ succ(length(env)) ∧
      X = {x∈A. sats(A, p, Cons(x,env))} ∧
      env_form_map(f,r,A,⟨env,p⟩) = k"

```

**definition**

```

DPow_least :: "[i,i,i,i]⇒i" where
  — function yielding the smallest index for X
  "DPow_least(f,r,A,X) ≡ μ k. DPow_ord(f,r,A,X,k)"

```

**definition**

```

DPow_r :: "[i,i,i]⇒i" where
  — a wellordering on DPow(A)
  "DPow_r(f,r,A) ≡ measure(DPow(A), DPow_least(f,r,A))"

```

**lemma** (in Nat\_Times\_Nat) well\_ord\_env\_form\_r:

```

  "well_ord(A,r)
    ⇒ well_ord(list(A) * formula, env_form_r(fn,r,A))"
⟨proof⟩

```

**lemma** (in Nat\_Times\_Nat) Ord\_env\_form\_map:

```

  "⟦well_ord(A,r); z ∈ list(A) * formula⟧
    ⇒ Ord(env_form_map(fn,r,A,z))"
⟨proof⟩

```

**lemma** DPow\_imp\_ex\_DPow\_ord:

```

  "X ∈ DPow(A) ⇒ ∃ k. DPow_ord(fn,r,A,X,k)"
⟨proof⟩

```

**lemma** (in Nat\_Times\_Nat) DPow\_ord\_imp\_Ord:

```

  "⟦DPow_ord(fn,r,A,X,k); well_ord(A,r)⟧ ⇒ Ord(k)"
⟨proof⟩

```

**lemma** (in Nat\_Times\_Nat) DPow\_imp\_DPow\_least:

```

  "⟦X ∈ DPow(A); well_ord(A,r)⟧
    ⇒ DPow_ord(fn, r, A, X, DPow_least(fn,r,A,X))"
⟨proof⟩

```

**lemma** (in Nat\_Times\_Nat) env\_form\_map\_inject:

```

  "⟦env_form_map(fn,r,A,u) = env_form_map(fn,r,A,v); well_ord(A,r);
    u ∈ list(A) * formula; v ∈ list(A) * formula⟧
    ⇒ u=v"
⟨proof⟩

```

**lemma** (in Nat\_Times\_Nat) DPow\_ord\_unique:

$$\llbracket \text{DPow\_ord}(\text{fn}, r, A, X, k); \text{DPow\_ord}(\text{fn}, r, A, Y, k); \text{well\_ord}(A, r) \rrbracket$$

$$\implies X=Y$$

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

**lemma** (in Nat\_Times\_Nat) well\_ord\_DPow\_r:  

$$\text{"well\_ord}(A, r) \implies \text{well\_ord}(\text{DPow}(A), \text{DPow\_r}(\text{fn}, r, A))"$$

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

**lemma** (in Nat\_Times\_Nat) DPow\_r\_type:  

$$\text{"DPow\_r}(\text{fn}, r, A) \subseteq \text{DPow}(A) * \text{DPow}(A)"$$

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

## 15.4 Limit Construction for Well-Orderings

Now we work towards the transfinite definition of wellorderings for  $Lset(i)$ . We assume as an inductive hypothesis that there is a family of wellorderings for smaller ordinals.

**definition**

$$\text{rlimit} :: "[i, i \Rightarrow i] \Rightarrow i" \text{ where}$$
 — Expresses the wellordering at limit ordinals. The conditional lets us remove the premise  $Limit(i)$  from some theorems.
 
$$\text{"rlimit}(i, r) \equiv$$

$$\text{if } Limit(i) \text{ then}$$

$$\{z: Lset(i) * Lset(i). \exists x' x. z = \langle x', x \rangle \wedge$$

$$(lrank(x') < lrank(x) \mid$$

$$(lrank(x') = lrank(x) \wedge \langle x', x \rangle \in r(\text{succ}(lrank(x))))\}$$

$$\text{else } 0"$$

**definition**

$$Lset\_new :: "i \Rightarrow i" \text{ where}$$
 — This constant denotes the set of elements introduced at level  $\text{succ}(i)$ 

$$\text{"Lset\_new}(i) \equiv \{x \in Lset(\text{succ}(i)). lrank(x) = i\}"$$

**lemma** Limit\_Lset\_eq2:  

$$\text{"Limit}(i) \implies Lset(i) = (\bigcup j < i. Lset\_new(j))"$$

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

**lemma** wf\_on\_Lset:  

$$\text{"wf}[Lset(\text{succ}(j))](r(\text{succ}(j))) \implies \text{wf}[Lset\_new(j)](\text{rlimit}(i, r))"$$

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

**lemma** wf\_on\_rlimit:  

$$\text{"}(\forall j < i. \text{wf}[Lset(j)](r(j))) \implies \text{wf}[Lset(i)](\text{rlimit}(i, r))"$$

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

**lemma** linear\_rlimit:  

$$\text{"}[\text{Limit}(i); \forall j < i. \text{linear}(Lset(j), r(j))]\implies \text{linear}(Lset(i), \text{rlimit}(i, r))"$$



*<proof>*

**lemma** *well\_ord\_rlimit*:  
"[[Limit(i);  $\forall j < i. \text{well\_ord}(\text{Lset}(j), r(j))$ ]]  
 $\implies \text{well\_ord}(\text{Lset}(i), \text{rlimit}(i, r))$ "  
*<proof>*

**lemma** *rlimit\_cong*:  
" $(\bigwedge j. j < i \implies r'(j) = r(j)) \implies \text{rlimit}(i, r) = \text{rlimit}(i, r')$ "  
*<proof>*

## 15.5 Transfinite Definition of the Wellordering on $L$

**definition**

$L_r :: "[i, i] \Rightarrow i$  where  
" $L_r(f) \equiv \lambda i. \text{transrec3}(i, 0, \lambda x r. \text{DPow}_r(f, r, \text{Lset}(x)), \lambda x r. \text{rlimit}(x, \lambda y. r'y))$ "

### 15.5.1 The Corresponding Recursion Equations

**lemma** [*simp*]: " $L_r(f, 0) = 0$ "  
*<proof>*

**lemma** [*simp*]: " $L_r(f, \text{succ}(i)) = \text{DPow}_r(f, L_r(f, i), \text{Lset}(i))$ "  
*<proof>*

The limit case is non-trivial because of the distinction between object-level and meta-level abstraction.

**lemma** [*simp*]: " $\text{Limit}(i) \implies L_r(f, i) = \text{rlimit}(i, L_r(f))$ "  
*<proof>*

**lemma** (in *Nat\_Times\_Nat*)  $L_r$ \_type:  
" $\text{Ord}(i) \implies L_r(\text{fn}, i) \subseteq \text{Lset}(i) * \text{Lset}(i)$ "  
*<proof>*

**lemma** (in *Nat\_Times\_Nat*) *well\_ord\_L\_r*:  
" $\text{Ord}(i) \implies \text{well\_ord}(\text{Lset}(i), L_r(\text{fn}, i))$ "  
*<proof>*

**lemma** *well\_ord\_L\_r*:  
" $\text{Ord}(i) \implies \exists r. \text{well\_ord}(\text{Lset}(i), r)$ "  
*<proof>*

Every constructible set is well-ordered! Therefore the Wellordering Theorem and the Axiom of Choice hold in  $L$ !

**theorem** *L\_implies\_AC*: assumes  $x: "L(x)"$  shows " $\exists r. \text{well\_ord}(x, r)$ "  
*<proof>*

**interpretation**  $L: M\_basic\ L\ \langle proof \rangle$

**theorem** " $\forall x[L]. \exists r. wellordered(L, x, r)$ "  
 $\langle proof \rangle$

In order to prove  $\exists r[L]. wellordered(L, x, r)$ , it's necessary to know that  $r$  is actually constructible. It follows from the assumption " $\forall$  equals  $L$ '", but this reasoning doesn't appear to work in Isabelle.

**end**

## 16 Absoluteness for Order Types, Rank Functions and Well-Founded Relations

**theory** *Rank* imports *WF\_absolute* begin

### 16.1 Order Types: A Direct Construction by Replacement

```

locale  $M\_ordertype = M\_basic +$ 
assumes well_ord_iso_separation:
  " $\llbracket M(A); M(f); M(r) \rrbracket$ 
 $\implies separation\ (M, \lambda x. x \in A \longrightarrow (\exists y[M]. (\exists p[M].$ 
 $fun\_apply(M, f, x, y) \wedge pair(M, y, x, p) \wedge p \in r)))$ "
and obase_separation:
  — part of the order type formalization
  " $\llbracket M(A); M(r) \rrbracket$ 
 $\implies separation(M, \lambda a. \exists x[M]. \exists g[M]. \exists mx[M]. \exists par[M].$ 
 $ordinal(M, x) \wedge membership(M, x, mx) \wedge pred\_set(M, A, a, r, par)$ 
 $\wedge$ 
 $order\_isomorphism(M, par, r, x, mx, g))$ "
and obase_equals_separation:
  " $\llbracket M(A); M(r) \rrbracket$ 
 $\implies separation\ (M, \lambda x. x \in A \longrightarrow \neg(\exists y[M]. \exists g[M].$ 
 $ordinal(M, y) \wedge (\exists my[M]. \exists pxr[M].$ 
 $membership(M, y, my) \wedge pred\_set(M, A, x, r, pxr)$ 
 $\wedge$ 
 $order\_isomorphism(M, pxr, r, y, my, g))))$ "
and omap_replacement:
  " $\llbracket M(A); M(r) \rrbracket$ 
 $\implies strong\_replacement(M,$ 
 $\lambda a\ z. \exists x[M]. \exists g[M]. \exists mx[M]. \exists par[M].$ 
 $ordinal(M, x) \wedge pair(M, a, x, z) \wedge membership(M, x, mx) \wedge$ 
 $pred\_set(M, A, a, r, par) \wedge order\_isomorphism(M, par, r, x, mx, g))$ "

```

Inductive argument for Kunen's Lemma I 6.1, etc. Simple proof from Hal-  
 mos, page 72

**lemma** (in  $M\_ordertype$ ) *wellordered\_iso\_subset\_lemma*:  
 " $\llbracket wellordered(M, A, r); f \in ord\_iso(A, r, A', r); A' \leq A; y \in A;$

$$M(A); M(f); M(r) \implies \neg \langle f'y, y \rangle \in r$$
  
 $\langle proof \rangle$

Kunen's Lemma I 6.1, page 14: there's no order-isomorphism to an initial segment of a well-ordering

**lemma** (in  $M\_ordertype$ ) *wellordered\_iso\_predD*:  

$$\llbracket \text{wellordered}(M,A,r); f \in \text{ord\_iso}(A, r, \text{Order.pred}(A,x,r), r); \\ M(A); M(f); M(r) \rrbracket \implies x \notin A$$
  
 $\langle proof \rangle$

**lemma** (in  $M\_ordertype$ ) *wellordered\_iso\_pred\_eq\_lemma*:  

$$\llbracket f \in \langle \text{Order.pred}(A,y,r), r \rangle \cong \langle \text{Order.pred}(A,x,r), r \rangle; \\ \text{wellordered}(M,A,r); x \in A; y \in A; M(A); M(f); M(r) \rrbracket \implies \langle x,y \rangle \notin r$$
  
 $\langle proof \rangle$

Simple consequence of Lemma 6.1

**lemma** (in  $M\_ordertype$ ) *wellordered\_iso\_pred\_eq*:  

$$\llbracket \text{wellordered}(M,A,r); \\ f \in \text{ord\_iso}(\text{Order.pred}(A,a,r), r, \text{Order.pred}(A,c,r), r); \\ M(A); M(f); M(r); a \in A; c \in A \rrbracket \implies a=c$$
  
 $\langle proof \rangle$

Following Kunen's Theorem I 7.6, page 17. Note that this material is not required elsewhere.

Can't use *well\_ord\_iso\_preserving* because it needs the strong premise *well\_ord*(A, r)

**lemma** (in  $M\_ordertype$ ) *ord\_iso\_pred\_imp\_lt*:  

$$\llbracket f \in \text{ord\_iso}(\text{Order.pred}(A,x,r), r, i, \text{Memrel}(i)); \\ g \in \text{ord\_iso}(\text{Order.pred}(A,y,r), r, j, \text{Memrel}(j)); \\ \text{wellordered}(M,A,r); x \in A; y \in A; M(A); M(r); M(f); M(g); \\ M(j); \\ \text{Ord}(i); \text{Ord}(j); \langle x,y \rangle \in r \rrbracket \\ \implies i < j$$
  
 $\langle proof \rangle$

**lemma** *ord\_iso\_converse1*:  

$$\llbracket f: \text{ord\_iso}(A,r,B,s); \langle b, f'a \rangle: s; a:A; b:B \rrbracket \\ \implies \langle \text{converse}(f) ' b, a \rangle \in r$$
  
 $\langle proof \rangle$

**definition**

*obase* :: " $[i \Rightarrow o, i, i] \Rightarrow i$ " where  
— the domain of *om*, eventually shown to equal A  

$$obase(M,A,r) \equiv \{a \in A. \exists x[M]. \exists g[M]. \text{Ord}(x) \wedge \\ g \in \text{ord\_iso}(\text{Order.pred}(A,a,r), r, x, \text{Memrel}(x))\}$$

**definition**

```
omap :: "[i⇒o,i,i,i] ⇒ o" where
  — the function that maps wosets to order types
  "omap(M,A,r,f) ≡
    ∀ z[M].
      z ∈ f ⟷ (∃ a∈A. ∃ x[M]. ∃ g[M]. z = ⟨a,x⟩ ∧ Ord(x) ∧
                g ∈ ord_iso(Order.pred(A,a,r),r,x,Memrel(x)))"
```

**definition**

```
otype :: "[i⇒o,i,i,i] ⇒ o" where — the order types themselves
  "otype(M,A,r,i) ≡ ∃ f[M]. omap(M,A,r,f) ∧ is_range(M,f,i)"
```

Can also be proved with the premise  $M(z)$  instead of  $M(f)$ , but that version is less useful. This lemma is also more useful than the definition, `omap_def`.

**lemma** (in `M_ordertype`) `omap_iff`:

```
"[omap(M,A,r,f); M(A); M(f)]
⇒ z ∈ f ⟷
  (∃ a∈A. ∃ x[M]. ∃ g[M]. z = ⟨a,x⟩ ∧ Ord(x) ∧
    g ∈ ord_iso(Order.pred(A,a,r),r,x,Memrel(x)))"
```

*<proof>*

**lemma** (in `M_ordertype`) `omap_unique`:

```
"[omap(M,A,r,f); omap(M,A,r,f'); M(A); M(r); M(f); M(f')] ⇒ f'
= f"
```

*<proof>*

**lemma** (in `M_ordertype`) `omap_yields_Ord`:

```
"[omap(M,A,r,f); ⟨a,x⟩ ∈ f; M(a); M(x)] ⇒ Ord(x)"
```

*<proof>*

**lemma** (in `M_ordertype`) `otype_iff`:

```
"[otype(M,A,r,i); M(A); M(r); M(i)]
⇒ x ∈ i ⟷
  (M(x) ∧ Ord(x) ∧
    (∃ a∈A. ∃ g[M]. g ∈ ord_iso(Order.pred(A,a,r),r,x,Memrel(x))))"
```

*<proof>*

**lemma** (in `M_ordertype`) `otype_eq_range`:

```
"[omap(M,A,r,f); otype(M,A,r,i); M(A); M(r); M(f); M(i)]
⇒ i = range(f)"
```

*<proof>*

**lemma** (in `M_ordertype`) `Ord_otype`:

```
"[otype(M,A,r,i); trans[A](r); M(A); M(r); M(i)] ⇒ Ord(i)"
```

*<proof>*

**lemma** (in `M_ordertype`) `domain_omap`:

```

    "⟦omap(M,A,r,f); M(A); M(r); M(B); M(f)⟧
      ⟹ domain(f) = obase(M,A,r)"
  <proof>

lemma (in M_ordertype) omap_subset:
  "⟦omap(M,A,r,f); otype(M,A,r,i);
    M(A); M(r); M(f); M(B); M(i)⟧ ⟹ f ⊆ obase(M,A,r) * i"
  <proof>

lemma (in M_ordertype) omap_funtype:
  "⟦omap(M,A,r,f); otype(M,A,r,i);
    M(A); M(r); M(f); M(i)⟧ ⟹ f ∈ obase(M,A,r) -> i"
  <proof>

```

```

lemma (in M_ordertype) wellordered_omap_bij:
  "⟦wellordered(M,A,r); omap(M,A,r,f); otype(M,A,r,i);
    M(A); M(r); M(f); M(i)⟧ ⟹ f ∈ bij(obase(M,A,r),i)"
  <proof>

```

This is not the final result: we must show  $oB(A, r) = A$

```

lemma (in M_ordertype) omap_ord_iso:
  "⟦wellordered(M,A,r); omap(M,A,r,f); otype(M,A,r,i);
    M(A); M(r); M(f); M(i)⟧ ⟹ f ∈ ord_iso(obase(M,A,r),r,i,Memrel(i))"
  <proof>

```

```

lemma (in M_ordertype) Ord_omap_image_pred:
  "⟦wellordered(M,A,r); omap(M,A,r,f); otype(M,A,r,i);
    M(A); M(r); M(f); M(i); b ∈ A⟧ ⟹ Ord(f ‘‘ Order.pred(A,b,r))"
  <proof>

```

```

lemma (in M_ordertype) restrict_omap_ord_iso:
  "⟦wellordered(M,A,r); omap(M,A,r,f); otype(M,A,r,i);
    D ⊆ obase(M,A,r); M(A); M(r); M(f); M(i)⟧
    ⟹ restrict(f,D) ∈ (⟨D,r⟩ ≅ ⟨f‘‘D, Memrel(f‘‘D)⟩)"
  <proof>

```

```

lemma (in M_ordertype) obase_equals:
  "⟦wellordered(M,A,r); omap(M,A,r,f); otype(M,A,r,i);
    M(A); M(r); M(f); M(i)⟧ ⟹ obase(M,A,r) = A"
  <proof>

```

Main result:  $om$  gives the order-isomorphism  $\langle A, r \rangle \cong \langle i, Memrel(i) \rangle$

```

theorem (in M_ordertype) omap_ord_iso_otype:
  "⟦wellordered(M,A,r); omap(M,A,r,f); otype(M,A,r,i);
    M(A); M(r); M(f); M(i)⟧ ⟹ f ∈ ord_iso(A, r, i, Memrel(i))"
  <proof>

```

```

lemma (in M_ordertype) obase_exists:

```

" $\llbracket M(A); M(r) \rrbracket \implies M(\text{obase}(M, A, r))$ "  
 <proof>

**lemma** (in *M\_ordertype*) *omap\_exists*:  
 " $\llbracket M(A); M(r) \rrbracket \implies \exists z[M]. \text{omap}(M, A, r, z)$ "  
 <proof>

**lemma** (in *M\_ordertype*) *otype\_exists*:  
 " $\llbracket \text{wellordered}(M, A, r); M(A); M(r) \rrbracket \implies \exists i[M]. \text{otype}(M, A, r, i)$ "  
 <proof>

**lemma** (in *M\_ordertype*) *ordertype\_exists*:  
 " $\llbracket \text{wellordered}(M, A, r); M(A); M(r) \rrbracket$   
 $\implies \exists f[M]. (\exists i[M]. \text{Ord}(i) \wedge f \in \text{ord\_iso}(A, r, i, \text{Memrel}(i)))$ "  
 <proof>

**lemma** (in *M\_ordertype*) *relativized\_imp\_well\_ord*:  
 " $\llbracket \text{wellordered}(M, A, r); M(A); M(r) \rrbracket \implies \text{well\_ord}(A, r)$ "  
 <proof>

## 16.2 Kunen's theorem 5.4, page 127

(a) The notion of Wellordering is absolute

**theorem** (in *M\_ordertype*) *well\_ord\_abs [simp]*:  
 " $\llbracket M(A); M(r) \rrbracket \implies \text{wellordered}(M, A, r) \longleftrightarrow \text{well\_ord}(A, r)$ "  
 <proof>

(b) Order types are absolute

**theorem** (in *M\_ordertype*) *ordertypes\_are\_absolute*:  
 " $\llbracket \text{wellordered}(M, A, r); f \in \text{ord\_iso}(A, r, i, \text{Memrel}(i));$   
 $M(A); M(r); M(f); M(i); \text{Ord}(i) \rrbracket \implies i = \text{ordertype}(A, r)$ "  
 <proof>

## 16.3 Ordinal Arithmetic: Two Examples of Recursion

Note: the remainder of this theory is not needed elsewhere.

### 16.3.1 Ordinal Addition

**definition**

*is\_oadd\_fun* :: " $[i \Rightarrow o, i, i, i, i] \Rightarrow o$ " where  
 "*is\_oadd\_fun*(*M, i, j, x, f*)  $\equiv$   
 ( $\forall sj \ msj. M(sj) \longrightarrow M(msj) \longrightarrow$   
 $\text{successor}(M, j, sj) \longrightarrow \text{membership}(M, sj, msj) \longrightarrow$   
 $M\_is\_recfun}(M,$   
 $\lambda x \ g \ y. \exists gx[M]. \text{image}(M, g, x, gx) \wedge \text{union}(M, i, gx, y),$   
 $msj, x, f))$ "

**definition**

```
is_oadd :: "[i⇒o,i,i,i] ⇒ o" where
  "is_oadd(M,i,j,k) ≡
    (¬ ordinal(M,i) ∧ ¬ ordinal(M,j) ∧ k=0) |
    (¬ ordinal(M,i) ∧ ordinal(M,j) ∧ k=j) |
    (ordinal(M,i) ∧ ¬ ordinal(M,j) ∧ k=i) |
    (ordinal(M,i) ∧ ordinal(M,j) ∧
      (∃ f fj sj. M(f) ∧ M(fj) ∧ M(sj) ∧
        successor(M,j,sj) ∧ is_oadd_fun(M,i,sj,sj,f) ∧
        fun_apply(M,f,j,fj) ∧ fj = k))"
```

**definition**

```
omult_eqns :: "[i,i,i,i] ⇒ o" where
  "omult_eqns(i,x,g,z) ≡
    Ord(x) ∧
    (x=0 ⟶ z=0) ∧
    (∀ j. x = succ(j) ⟶ z = g'j ++ i) ∧
    (Limit(x) ⟶ z = ⋃ (g'x))"
```

**definition**

```
is_omult_fun :: "[i⇒o,i,i,i] ⇒ o" where
  "is_omult_fun(M,i,j,f) ≡
    (∃ df. M(df) ∧ is_function(M,f) ∧
      is_domain(M,f,df) ∧ subset(M, j, df)) ∧
    (∀ x∈j. omult_eqns(i,x,f,f'x))"
```

**definition**

```
is_omult :: "[i⇒o,i,i,i] ⇒ o" where
  "is_omult(M,i,j,k) ≡
    ∃ f fj sj. M(f) ∧ M(fj) ∧ M(sj) ∧
      successor(M,j,sj) ∧ is_omult_fun(M,i,sj,f) ∧
      fun_apply(M,f,j,fj) ∧ fj = k"
```

**locale**  $M\_ord\_arith = M\_ordertype +$

**assumes**  $oadd\_strong\_replacement:$

```
"[M(i); M(j)] ⟹
  strong_replacement(M,
    λx z. ∃ y[M]. pair(M,x,y,z) ∧
      (∃ f[M]. ∃ fx[M]. is_oadd_fun(M,i,j,x,f) ∧
        image(M,f,x,fx) ∧ y = i ∪ fx))"
```

**and**  $omult\_strong\_replacement':$

```
"[M(i); M(j)] ⟹
  strong_replacement(M,
    λx z. ∃ y[M]. z = ⟨x,y⟩ ∧
      (∃ g[M]. is_recfun(Memrel(succ(j)),x,λx g. THE z. omult_eqns(i,x,g,z),g))"
```

$\wedge$   
 $y = (THE\ z.\ omult\_eqns(i,\ x,\ g,\ z)))$ "

**is\_oadd\_fun**: Relating the pure "language of set theory" to Isabelle/ZF

**lemma** (in *M\_ord\_arith*) *is\_oadd\_fun\_iff*:  

$$\begin{aligned}
& "[a \leq j; M(i); M(j); M(a); M(f)] \\
& \implies is\_oadd\_fun(M, i, j, a, f) \longleftrightarrow \\
& \quad f \in a \rightarrow range(f) \wedge (\forall x. M(x) \rightarrow x < a \rightarrow f'x = i \cup f'x)"
\end{aligned}$$
*<proof>*

**lemma** (in *M\_ord\_arith*) *oadd\_strong\_replacement'*:  

$$\begin{aligned}
& "[M(i); M(j)] \implies \\
& \quad strong\_replacement(M, \\
& \quad \quad \lambda x\ z.\ \exists y[M].\ z = \langle x, y \rangle \wedge \\
& \quad \quad (\exists g[M].\ is\_recfun(Memrel(succ(j)), x, \lambda x\ g.\ i \cup g'x, g))
\end{aligned}$$
 $\wedge$   
 $y = i \cup g'x)$ "
*<proof>*

**lemma** (in *M\_ord\_arith*) *exists\_oadd*:  

$$\begin{aligned}
& "[Ord(j); M(i); M(j)] \\
& \implies \exists f[M].\ is\_recfun(Memrel(succ(j)), j, \lambda x\ g.\ i \cup g'x, f)"
\end{aligned}$$
*<proof>*

**lemma** (in *M\_ord\_arith*) *exists\_oadd\_fun*:  

$$\begin{aligned}
& "[Ord(j); M(i); M(j)] \implies \exists f[M].\ is\_oadd\_fun(M, i, succ(j), succ(j), f)"
\end{aligned}$$
*<proof>*

**lemma** (in *M\_ord\_arith*) *is\_oadd\_fun\_apply*:  

$$\begin{aligned}
& "[x < j; M(i); M(j); M(f); is\_oadd\_fun(M, i, j, j, f)] \\
& \implies f'x = i \cup (\bigcup_{k \in x} \{f'k\})"
\end{aligned}$$
*<proof>*

**lemma** (in *M\_ord\_arith*) *is\_oadd\_fun\_iff\_oadd [rule\_format]*:  

$$\begin{aligned}
& "[is\_oadd\_fun(M, i, J, J, f); M(i); M(J); M(f); Ord(i); Ord(j)] \\
& \implies j < J \rightarrow f'j = i++j"
\end{aligned}$$
*<proof>*

**lemma** (in *M\_ord\_arith*) *Ord\_oadd\_abs*:  

$$\begin{aligned}
& "[M(i); M(j); M(k); Ord(i); Ord(j)] \implies is\_oadd(M, i, j, k) \longleftrightarrow k = i++j"
\end{aligned}$$
*<proof>*

**lemma** (in *M\_ord\_arith*) *oadd\_abs*:  

$$\begin{aligned}
& "[M(i); M(j); M(k)] \implies is\_oadd(M, i, j, k) \longleftrightarrow k = i++j"
\end{aligned}$$
*<proof>*

**lemma** (in *M\_ord\_arith*) *oadd\_closed [intro, simp]*:



" $\llbracket M(i); M(j) \rrbracket \implies M(i++j)$ "  
 <proof>

### 16.3.2 Ordinal Multiplication

**lemma** *omult\_eqns\_unique*:  
 " $\llbracket \text{omult\_eqns}(i, x, g, z); \text{omult\_eqns}(i, x, g, z') \rrbracket \implies z = z'$ "  
 <proof>

**lemma** *omult\_eqns\_0*: " $\text{omult\_eqns}(i, 0, g, z) \longleftrightarrow z = 0$ "  
 <proof>

**lemma** *the\_omult\_eqns\_0*: " $(\text{THE } z. \text{omult\_eqns}(i, 0, g, z)) = 0$ "  
 <proof>

**lemma** *omult\_eqns\_succ*: " $\text{omult\_eqns}(i, \text{succ}(j), g, z) \longleftrightarrow \text{Ord}(j) \wedge z = g'j ++ i$ "  
 <proof>

**lemma** *the\_omult\_eqns\_succ*:  
 " $\text{Ord}(j) \implies (\text{THE } z. \text{omult\_eqns}(i, \text{succ}(j), g, z)) = g'j ++ i$ "  
 <proof>

**lemma** *omult\_eqns\_Limit*:  
 " $\text{Limit}(x) \implies \text{omult\_eqns}(i, x, g, z) \longleftrightarrow z = \bigcup (g'x)$ "  
 <proof>

**lemma** *the\_omult\_eqns\_Limit*:  
 " $\text{Limit}(x) \implies (\text{THE } z. \text{omult\_eqns}(i, x, g, z)) = \bigcup (g'x)$ "  
 <proof>

**lemma** *omult\_eqns\_Not*: " $\neg \text{Ord}(x) \implies \neg \text{omult\_eqns}(i, x, g, z)$ "  
 <proof>

**lemma** (in *M\_ord\_arith*) *the\_omult\_eqns\_closed*:  
 " $\llbracket M(i); M(x); M(g); \text{function}(g) \rrbracket$   
 $\implies M(\text{THE } z. \text{omult\_eqns}(i, x, g, z))$ "  
 <proof>

**lemma** (in *M\_ord\_arith*) *exists\_omult*:  
 " $\llbracket \text{Ord}(j); M(i); M(j) \rrbracket$   
 $\implies \exists f[M]. \text{is\_recfun}(\text{Memrel}(\text{succ}(j)), j, \lambda x g. \text{THE } z. \text{omult\_eqns}(i, x, g, z), f)$ "  
 <proof>

**lemma** (in *M\_ord\_arith*) *exists\_omult\_fun*:  
 " $\llbracket \text{Ord}(j); M(i); M(j) \rrbracket \implies \exists f[M]. \text{is\_omult\_fun}(M, i, \text{succ}(j), f)$ "  
 <proof>

```

lemma (in M_ord_arith) is_omult_fun_apply_0:
  "⟦0 < j; is_omult_fun(M,i,j,f)⟧ ⇒ f'0 = 0"
  <proof>

lemma (in M_ord_arith) is_omult_fun_apply_succ:
  "⟦succ(x) < j; is_omult_fun(M,i,j,f)⟧ ⇒ f'succ(x) = f'x ++ i"
  <proof>

lemma (in M_ord_arith) is_omult_fun_apply_Limit:
  "⟦x < j; Limit(x); M(j); M(f); is_omult_fun(M,i,j,f)⟧
    ⇒ f' x = (⋃ y∈x. f'y)"
  <proof>

lemma (in M_ord_arith) is_omult_fun_eq_omult:
  "⟦is_omult_fun(M,i,J,f); M(J); M(f); Ord(i); Ord(j)⟧
    ⇒ j < J → f'j = i**j"
  <proof>

lemma (in M_ord_arith) omult_abs:
  "⟦M(i); M(j); M(k); Ord(i); Ord(j)⟧ ⇒ is_omult(M,i,j,k) ↔ k =
    i**j"
  <proof>

```

## 16.4 Absoluteness of Well-Founded Relations

Relativized to  $M$ : Every well-founded relation is a subset of some inverse image of an ordinal. Key step is the construction (in  $M$ ) of a rank function.

```

locale M_wfrank = M_trancl +
  assumes wfrank_separation:
    "M(r) ⇒
      separation (M, λx.
        ∀ rplus[M]. tran_closure(M,r,rplus) →
        ¬ (∃ f[M]. M_is_recfun(M, λx f y. is_range(M,f,y), rplus, x,
        f)))"
  and wfrank_strong_replacement:
    "M(r) ⇒
      strong_replacement(M, λx z.
        ∀ rplus[M]. tran_closure(M,r,rplus) →
        (∃ y[M]. ∃ f[M]. pair(M,x,y,z) ∧
          M_is_recfun(M, λx f y. is_range(M,f,y), rplus,
          x, f) ∧
            is_range(M,f,y)))"
  and Ord_wfrank_separation:
    "M(r) ⇒
      separation (M, λx.
        ∀ rplus[M]. tran_closure(M,r,rplus) →
        ¬ (∀ f[M]. ∀ rangef[M].
          is_range(M,f,rangef) →

```

$$M\_is\_recfun(M, \lambda x f y. is\_range(M, f, y), rplus, x, f) \longrightarrow ordinal(M, range f))"$$

Proving that the relativized instances of Separation or Replacement agree with the "real" ones.

**lemma** (in  $M\_wfrank$ )  $wfrank\_separation'$ :  
 $"M(r) \implies$   
 $separation$   
 $(M, \lambda x. \neg (\exists f[M]. is\_recfun(r^+, x, \lambda x f. range(f), f)))"$   
 $\langle proof \rangle$

**lemma** (in  $M\_wfrank$ )  $wfrank\_strong\_replacement'$ :  
 $"M(r) \implies$   
 $strong\_replacement(M, \lambda x z. \exists y[M]. \exists f[M].$   
 $pair(M, x, y, z) \wedge is\_recfun(r^+, x, \lambda x f. range(f), f)$   
 $\wedge$   
 $y = range(f))"$   
 $\langle proof \rangle$

**lemma** (in  $M\_wfrank$ )  $Ord\_wfrank\_separation'$ :  
 $"M(r) \implies$   
 $separation (M, \lambda x.$   
 $\neg (\forall f[M]. is\_recfun(r^+, x, \lambda x. range, f) \longrightarrow Ord(range(f))))"$   
 $\langle proof \rangle$

This function, defined using replacement, is a rank function for well-founded relations within the class M.

**definition**  
 $wellfoundedrank :: "[i \Rightarrow o, i, i] \Rightarrow i"$  where  
 $"wellfoundedrank(M, r, A) \equiv$   
 $\{p. x \in A, \exists y[M]. \exists f[M].$   
 $p = \langle x, y \rangle \wedge is\_recfun(r^+, x, \lambda x f. range(f), f)$   
 $\wedge$   
 $y = range(f)\}"$

**lemma** (in  $M\_wfrank$ )  $exists\_wfrank$ :  
 $"[wellfounded(M, r); M(a); M(r)]$   
 $\implies \exists f[M]. is\_recfun(r^+, a, \lambda x f. range(f), f)"$   
 $\langle proof \rangle$

**lemma** (in  $M\_wfrank$ )  $M\_wellfoundedrank$ :  
 $"[wellfounded(M, r); M(r); M(A)] \implies M(wellfoundedrank(M, r, A))"$   
 $\langle proof \rangle$

**lemma** (in  $M\_wfrank$ )  $Ord\_wfrank\_range [rule\_format]$ :  
 $"[wellfounded(M, r); a \in A; M(r); M(A)]$   
 $\implies \forall f[M]. is\_recfun(r^+, a, \lambda x f. range(f), f) \longrightarrow Ord(range(f))"$   
 $\langle proof \rangle$

```

lemma (in M_wfrank) Ord_range_wellfoundedrank:
  "[[wellfounded(M,r); r ⊆ A*A; M(r); M(A)]]
  ⇒ Ord (range(wellfoundedrank(M,r,A)))"
⟨proof⟩

lemma (in M_wfrank) function_wellfoundedrank:
  "[[wellfounded(M,r); M(r); M(A)]]
  ⇒ function(wellfoundedrank(M,r,A))"
⟨proof⟩

lemma (in M_wfrank) domain_wellfoundedrank:
  "[[wellfounded(M,r); M(r); M(A)]]
  ⇒ domain(wellfoundedrank(M,r,A)) = A"
⟨proof⟩

lemma (in M_wfrank) wellfoundedrank_type:
  "[[wellfounded(M,r); M(r); M(A)]]
  ⇒ wellfoundedrank(M,r,A) ∈ A -> range(wellfoundedrank(M,r,A))"
⟨proof⟩

lemma (in M_wfrank) Ord_wellfoundedrank:
  "[[wellfounded(M,r); a ∈ A; r ⊆ A*A; M(r); M(A)]]
  ⇒ Ord(wellfoundedrank(M,r,A) ‘ a)"
⟨proof⟩

lemma (in M_wfrank) wellfoundedrank_eq:
  "[[is_recfun(r^+, a, λx. range, f);
  wellfounded(M,r); a ∈ A; M(f); M(r); M(A)]]
  ⇒ wellfoundedrank(M,r,A) ‘ a = range(f)"
⟨proof⟩

lemma (in M_wfrank) wellfoundedrank_lt:
  "[[⟨a,b⟩ ∈ r;
  wellfounded(M,r); r ⊆ A*A; M(r); M(A)]]
  ⇒ wellfoundedrank(M,r,A) ‘ a < wellfoundedrank(M,r,A) ‘ b"
⟨proof⟩

lemma (in M_wfrank) wellfounded_imp_subset_rvimage:
  "[[wellfounded(M,r); r ⊆ A*A; M(r); M(A)]]
  ⇒ ∃ i f. Ord(i) ∧ r ⊆ rvimage(A, f, Memrel(i))"
⟨proof⟩

lemma (in M_wfrank) wellfounded_imp_wf:
  "[[wellfounded(M,r); relation(r); M(r)]] ⇒ wf(r)"
⟨proof⟩

```

```

lemma (in  $M\_wfrank$ ) wellfounded_on_imp_wf_on:
  "[[wellfounded_on( $M, A, r$ ); relation( $r$ );  $M(r)$ ;  $M(A)$ ]]  $\implies wf[A](r)$ "
  <proof>

theorem (in  $M\_wfrank$ ) wf_abs:
  "[[relation( $r$ );  $M(r)$ ]]  $\implies wellfounded(M, r) \longleftrightarrow wf(r)$ "
  <proof>

theorem (in  $M\_wfrank$ ) wf_on_abs:
  "[[relation( $r$ );  $M(r)$ ;  $M(A)$ ]]  $\implies wellfounded\_on(M, A, r) \longleftrightarrow wf[A](r)$ "
  <proof>

end

```

## 17 Separation for Facts About Order Types, Rank Functions and Well-Founded Relations

**theory** *Rank\_Separation* **imports** *Rank Rec\_Separation* **begin**

This theory proves all instances needed for locales  $M\_ordertype$  and  $M\_wfrank$ .  
But the material is not needed for proving the relative consistency of AC.

### 17.1 The Locale $M\_ordertype$

#### 17.1.1 Separation for Order-Isomorphisms

```

lemma well_ord_iso_Reflects:
  "REFLECTS[ $\lambda x. x \in A \longrightarrow$ 
     $(\exists y[L]. \exists p[L]. fun\_apply(L, f, x, y) \wedge pair(L, y, x, p) \wedge p$ 
 $\in r),$ 
     $\lambda i x. x \in A \longrightarrow (\exists y \in Lset(i). \exists p \in Lset(i).$ 
     $fun\_apply(\#\#Lset(i), f, x, y) \wedge pair(\#\#Lset(i), y, x, p) \wedge p$ 
 $\in r)]$ "
  <proof>

```

```

lemma well_ord_iso_separation:
  "[[ $L(A)$ ;  $L(f)$ ;  $L(r)$ ]]
 $\implies separation(L, \lambda x. x \in A \longrightarrow (\exists y[L]. (\exists p[L].$ 
     $fun\_apply(L, f, x, y) \wedge pair(L, y, x, p) \wedge p \in r)))$ "
  <proof>

```

#### 17.1.2 Separation for *obase*

```

lemma obase_reflects:
  "REFLECTS[ $\lambda a. \exists x[L]. \exists g[L]. \exists mx[L]. \exists par[L].$ 
     $ordinal(L, x) \wedge membership(L, x, mx) \wedge pred\_set(L, A, a, r, par)$ 
 $\wedge$ 
     $order\_isomorphism(L, par, r, x, mx, g),$ 

```

$\lambda i \ a. \exists x \in \text{Lset}(i). \exists g \in \text{Lset}(i). \exists mx \in \text{Lset}(i). \exists par \in \text{Lset}(i).$   
 $\text{ordinal}(\#\text{Lset}(i), x) \wedge \text{membership}(\#\text{Lset}(i), x, mx) \wedge \text{pred\_set}(\#\text{Lset}(i), A, a, r, p$   
 $\wedge$   
 $\text{order\_isomorphism}(\#\text{Lset}(i), par, r, x, mx, g)]"$   
 $\langle proof \rangle$

**lemma** *obase\_separation*:  
 — part of the order type formalization  
 $"\llbracket L(A); L(r) \rrbracket$   
 $\implies \text{separation}(L, \lambda a. \exists x[L]. \exists g[L]. \exists mx[L]. \exists par[L].$   
 $\text{ordinal}(L, x) \wedge \text{membership}(L, x, mx) \wedge \text{pred\_set}(L, A, a, r, par)$   
 $\wedge$   
 $\text{order\_isomorphism}(L, par, r, x, mx, g))"$   
 $\langle proof \rangle$

### 17.1.3 Separation for a Theorem about obase

**lemma** *obase\_equals\_reflects*:  
 $"\text{REFLECTS}[\lambda x. x \in A \longrightarrow \neg(\exists y[L]. \exists g[L].$   
 $\text{ordinal}(L, y) \wedge (\exists my[L]. \exists pxx[L].$   
 $\text{membership}(L, y, my) \wedge \text{pred\_set}(L, A, x, r, pxx) \wedge$   
 $\text{order\_isomorphism}(L, pxx, r, y, my, g))],$   
 $\lambda i \ x. x \in A \longrightarrow \neg(\exists y \in \text{Lset}(i). \exists g \in \text{Lset}(i).$   
 $\text{ordinal}(\#\text{Lset}(i), y) \wedge (\exists my \in \text{Lset}(i). \exists pxx \in \text{Lset}(i).$   
 $\text{membership}(\#\text{Lset}(i), y, my) \wedge \text{pred\_set}(\#\text{Lset}(i), A, x, r, pxx)$   
 $\wedge$   
 $\text{order\_isomorphism}(\#\text{Lset}(i), pxx, r, y, my, g)))]"$   
 $\langle proof \rangle$

**lemma** *obase\_equals\_separation*:  
 $"\llbracket L(A); L(r) \rrbracket$   
 $\implies \text{separation}(L, \lambda x. x \in A \longrightarrow \neg(\exists y[L]. \exists g[L].$   
 $\text{ordinal}(L, y) \wedge (\exists my[L]. \exists pxx[L].$   
 $\text{membership}(L, y, my) \wedge \text{pred\_set}(L, A, x, r, pxx)$   
 $\wedge$   
 $\text{order\_isomorphism}(L, pxx, r, y, my, g)))]"$   
 $\langle proof \rangle$

### 17.1.4 Replacement for omap

**lemma** *omap\_reflects*:  
 $"\text{REFLECTS}[\lambda z. \exists a[L]. a \in B \wedge (\exists x[L]. \exists g[L]. \exists mx[L]. \exists par[L].$   
 $\text{ordinal}(L, x) \wedge \text{pair}(L, a, x, z) \wedge \text{membership}(L, x, mx) \wedge$   
 $\text{pred\_set}(L, A, a, r, par) \wedge \text{order\_isomorphism}(L, par, r, x, mx, g)],$   
 $\lambda i \ z. \exists a \in \text{Lset}(i). a \in B \wedge (\exists x \in \text{Lset}(i). \exists g \in \text{Lset}(i). \exists mx \in \text{Lset}(i).$   
 $\exists par \in \text{Lset}(i).$   
 $\text{ordinal}(\#\text{Lset}(i), x) \wedge \text{pair}(\#\text{Lset}(i), a, x, z) \wedge$   
 $\text{membership}(\#\text{Lset}(i), x, mx) \wedge \text{pred\_set}(\#\text{Lset}(i), A, a, r, par) \wedge$   
 $\text{order\_isomorphism}(\#\text{Lset}(i), par, r, x, mx, g))]"$   
 $\langle proof \rangle$

```

lemma omap_replacement:
  "⟦L(A); L(r)⟧
  ⇒ strong_replacement(L,
    λa z. ∃x[L]. ∃g[L]. ∃mx[L]. ∃par[L].
    ordinal(L,x) ∧ pair(L,a,x,z) ∧ membership(L,x,mx) ∧
    pred_set(L,A,a,r,par) ∧ order_isomorphism(L,par,r,x,mx,g))"
⟨proof⟩

```

## 17.2 Instantiating the locale $M\_ordertype$

Separation (and Strong Replacement) for basic set-theoretic constructions such as intersection, Cartesian Product and image.

```

lemma M_ordertype_axioms_L: "M_ordertype_axioms(L)"
⟨proof⟩

```

```

theorem M_ordertype_L: "M_ordertype(L)"
⟨proof⟩

```

## 17.3 The Locale $M\_wfrank$

### 17.3.1 Separation for $wfrank$

```

lemma wfrank_Reflects:
  "REFLECTS[λx. ∀rplus[L]. tran_closure(L,r,rplus) →
    ¬ (∃f[L]. M_is_recfun(L, λx f y. is_range(L,f,y), rplus,
x, f)),
    λi x. ∀rplus ∈ Lset(i). tran_closure(##Lset(i),r,rplus) →
    ¬ (∃f ∈ Lset(i).
      M_is_recfun(##Lset(i), λx f y. is_range(##Lset(i),f,y),
        rplus, x, f))]"]"
⟨proof⟩

```

```

lemma wfrank_separation:
  "L(r) ⇒
    separation (L, λx. ∀rplus[L]. tran_closure(L,r,rplus) →
    ¬ (∃f[L]. M_is_recfun(L, λx f y. is_range(L,f,y), rplus, x,
f)))"
⟨proof⟩

```

### 17.3.2 Replacement for $wfrank$

```

lemma wfrank_replacement_Reflects:
  "REFLECTS[λz. ∃x[L]. x ∈ A ∧
    (∀rplus[L]. tran_closure(L,r,rplus) →
    (∃y[L]. ∃f[L]. pair(L,x,y,z) ∧
      M_is_recfun(L, λx f y. is_range(L,f,y), rplus,
x, f) ∧
      is_range(L,f,y))),

```

```

λi z. ∃ x ∈ Lset(i). x ∈ A ∧
  (∀ rplus ∈ Lset(i). tran_closure(##Lset(i), r, rplus) →
    (∃ y ∈ Lset(i). ∃ f ∈ Lset(i). pair(##Lset(i), x, y, z) ∧
      M_is_recfun(##Lset(i), λx f y. is_range(##Lset(i), f, y), rplus,
x, f) ∧
      is_range(##Lset(i), f, y))))] "
⟨proof⟩

```

```

lemma wfrank_strong_replacement:
  "L(r) ⇒
    strong_replacement(L, λx z.
      ∀ rplus[L]. tran_closure(L, r, rplus) →
      (∃ y[L]. ∃ f[L]. pair(L, x, y, z) ∧
        M_is_recfun(L, λx f y. is_range(L, f, y), rplus,
x, f) ∧
        is_range(L, f, y)))"
⟨proof⟩

```

### 17.3.3 Separation for Proving Ord\_wfrank\_range

```

lemma Ord_wfrank_Reflects:
  "REFLECTS[λx. ∀ rplus[L]. tran_closure(L, r, rplus) →
    ¬ (∀ f[L]. ∀ rangef[L].
      is_range(L, f, rangef) →
      M_is_recfun(L, λx f y. is_range(L, f, y), rplus, x, f) →
      ordinal(L, rangef)),
    λi x. ∀ rplus ∈ Lset(i). tran_closure(##Lset(i), r, rplus) →
    ¬ (∀ f ∈ Lset(i). ∀ rangef ∈ Lset(i).
      is_range(##Lset(i), f, rangef) →
      M_is_recfun(##Lset(i), λx f y. is_range(##Lset(i), f, y),
        rplus, x, f) →
      ordinal(##Lset(i), rangef))]"
⟨proof⟩

```

```

lemma Ord_wfrank_separation:
  "L(r) ⇒
    separation (L, λx.
      ∀ rplus[L]. tran_closure(L, r, rplus) →
      ¬ (∀ f[L]. ∀ rangef[L].
        is_range(L, f, rangef) →
        M_is_recfun(L, λx f y. is_range(L, f, y), rplus, x, f) →
        ordinal(L, rangef)))"
⟨proof⟩

```

### 17.3.4 Instantiating the locale M\_wfrank

```

lemma M_wfrank_axioms_L: "M_wfrank_axioms(L)"
  ⟨proof⟩

```

```

theorem M_wfrank_L: "M_wfrank(L)"

```



```

    <proof>

lemmas exists_wfrank = M_wfrank.exists_wfrank [OF M_wfrank_L]
  and M_wellfoundedrank = M_wfrank.M_wellfoundedrank [OF M_wfrank_L]
  and Ord_wfrank_range = M_wfrank.Ord_wfrank_range [OF M_wfrank_L]
  and Ord_range_wellfoundedrank = M_wfrank.Ord_range_wellfoundedrank [OF
M_wfrank_L]
  and function_wellfoundedrank = M_wfrank.function_wellfoundedrank [OF
M_wfrank_L]
  and domain_wellfoundedrank = M_wfrank.domain_wellfoundedrank [OF M_wfrank_L]
  and wellfoundedrank_type = M_wfrank.wellfoundedrank_type [OF M_wfrank_L]
  and Ord_wellfoundedrank = M_wfrank.Ord_wellfoundedrank [OF M_wfrank_L]
  and wellfoundedrank_eq = M_wfrank.wellfoundedrank_eq [OF M_wfrank_L]
  and wellfoundedrank_lt = M_wfrank.wellfoundedrank_lt [OF M_wfrank_L]
  and wellfounded_imp_subset_rvimage = M_wfrank.wellfounded_imp_subset_rvimage
[OF M_wfrank_L]
  and wellfounded_imp_wf = M_wfrank.wellfounded_imp_wf [OF M_wfrank_L]
  and wellfounded_on_imp_wf_on = M_wfrank.wellfounded_on_imp_wf_on [OF
M_wfrank_L]
  and wf_abs = M_wfrank.wf_abs [OF M_wfrank_L]
  and wf_on_abs = M_wfrank.wf_on_abs [OF M_wfrank_L]

end

```

## References

- [1] Kurt Gödel. The consistency of the axiom of choice and of the generalized continuum hypothesis with the axioms of set theory. In S. Feferman et al., editors, *Kurt Gödel: Collected Works*, volume II. Oxford University Press, 1990.
- [2] Kenneth Kunen. *Set Theory: An Introduction to Independence Proofs*. North-Holland, 1980.
- [3] Lawrence C. Paulson. The reflection theorem: A study in meta-theoretic reasoning. In Andrei Voronkov, editor, *Automated Deduction — CADE-18 International Conference*, LNAI 2392, pages 377–391. Springer, 2002.
- [4] Lawrence C. Paulson. The relative consistency of the axiom of choice — mechanized using Isabelle/ZF. *LMS Journal of Computation and Mathematics*, 6:198–248, 2003. <http://www.lms.ac.uk/jcm/6/lms2003-001/>.