Transitive Models of Fragments of ZF

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

March 17, 2025

Abstract

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

Contents

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

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

1 Introduction

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

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 For this reason, some comments of the original text as well as outdated **apply** commands appear profusely in the following theories.

A repeated pattern that appears is that the relativized result can be proved *mutatis mutandis*, with remaining proof obligations that the objects constructed actually belong to the model M. Another aspect was that the management of higher order constructs always posed some extra problems, already noted by Paulson [3, Sect. 7.3].

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

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

1. From *ZF*:
   (a) `Univ`. Here we decided to relativize only the term `Vfrom` that constructs the cumulative hierarchy up to some ordinal length and starting from an arbitrary set.
   (b) `Cardinal`. There are two targets for this source, `Least` and `Cardinal_Relative`. Both require some fair amount of preparation, trying to take advantage of absolute concepts. It is not

straightforward to compare source and targets in a line-by-line fashion at this point.

(c) `CardinalArith`. The hardest part was to formalize the cardinal successor function and ordertypes. We also disregarded the part treating finite cardinals since it is an absolute concept. Apart from that, the relative version closely parallels the source.

(d) `Cardinal_AC`. After some boilerplate, porting was rather straightforward, excepting cardinal arithmetic involving the higher-order union operator.

2. From *ZF-Constructible*:

   (a) `Normal`. The target here is `Aleph_Relative` since that is the only concept that we ported. Instead of porting all the machinery of normal functions (since it involved higher-order variables), we particularized the results for the Aleph function. We also used an alternative definition of the latter that worked better with our relativization discipline.

3. From *Delta_System_Lemma*:

   (a) `ZF_Library`. The target includes a big section of auxiliary lemmas and commands that aid the relativization. We needed to make explicit the witnesses (mainly functions) in some of the existential results proved in the source, since only in that way we would be able to show that they belonged to the model.

   (b) `Cardinal_Library`. Porting was relatively straightforward; most of the extra work laid in adjusting locale assumptions to obtain an appropriate context to state and prove the theorems.

   (c) `Delta_System`. Same comments as in the case of `Cardinal_Library` apply here.

4. From *Forcing*:

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

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

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

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

This theory encapsulates some ML utilities

⟨*ML*⟩

end

2 Auxiliary results on arithmetic

theory *Nat_Miscellanea*
imports
Delta_System_Lemma.ZF_Library
begin

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

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

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

lemma *nat_succD* : $m \in nat \Longrightarrow succ(n) \in succ(m) \Longrightarrow n \in m$
⟨*proof*⟩

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

lemma *in_n_in_nat* : $m \in nat \Longrightarrow n \in m \Longrightarrow n \in nat$
⟨*proof*⟩

lemma *ltI_neg* : $x \in nat \Longrightarrow j \leq x \Longrightarrow j \neq x \Longrightarrow j < x$
⟨*proof*⟩

lemma *succ_pred_eq* : $m \in nat \Longrightarrow m \neq 0 \Longrightarrow succ(pred(m)) = m$
⟨*proof*⟩

lemma *succ_ltI* : $succ(j) < n \Longrightarrow j < n$
⟨*proof*⟩

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

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

⟨*proof*⟩

lemma *succpred_n0* : $succ(n) \in p \Longrightarrow p \neq 0$
⟨*proof*⟩

lemmas $natEin = natE\ [OF\ lt\_nat\_in\_nat]$

lemmas $Un\_least\_lt\_iffn = Un\_least\_lt\_iff\ [OF\ nat\_into\_Ord\ nat\_into\_Ord]$

lemma *pred_type* : $m \in nat \Longrightarrow n \leq m \Longrightarrow n \in nat$
⟨*proof*⟩

lemma *pred_le* : $m \in nat \Longrightarrow n \leq succ(m) \Longrightarrow pred(n) \leq m$
⟨*proof*⟩

lemma *pred_le2* : $n \in nat \Longrightarrow m \in nat \Longrightarrow pred(n) \leq m \Longrightarrow n \leq succ(m)$
⟨*proof*⟩

lemma *Un_leD1* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow Ord(k) \Longrightarrow i \cup j \leq k \Longrightarrow i \leq k$
⟨*proof*⟩

lemma *Un_leD2* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow Ord(k) \Longrightarrow i \cup j \leq k \Longrightarrow j \leq k$
⟨*proof*⟩

lemma *gt1* : $n \in nat \Longrightarrow i \in n \Longrightarrow i \neq 0 \Longrightarrow i \neq 1 \Longrightarrow 1 < i$
⟨*proof*⟩

lemma *pred_mono* : $m \in nat \Longrightarrow n \leq m \Longrightarrow pred(n) \leq pred(m)$
⟨*proof*⟩

lemma *succ_mono* : $m \in nat \Longrightarrow n \leq m \Longrightarrow succ(n) \leq succ(m)$
⟨*proof*⟩

lemma *union_abs1* :
$[\![\ i \leq j\ ]\!] \Longrightarrow i \cup j = j$
⟨*proof*⟩

lemma *union_abs2* :
$[\![\ i \leq j\ ]\!] \Longrightarrow j \cup i = j$
⟨*proof*⟩

lemma *ord_un_max* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow i \cup j = max(i,j)$
⟨*proof*⟩

lemma *ord_max_ty* : $Ord(i) \Longrightarrow Ord(j) \Longrightarrow Ord(max(i,j))$
⟨*proof*⟩

lemmas $ord\_simp\_union = ord\_un\_max\ ord\_max\_ty\ max\_def$

lemma *le_succ* : $x{\in}nat \Longrightarrow x{\leq}succ(x)$ ⟨*proof*⟩

lemma *le_pred* : $x{\in}nat \Longrightarrow pred(x){\leq}x$
⟨*proof*⟩

lemma *not_le_anti_sym* : $x{\in}nat \Longrightarrow y \in nat \Longrightarrow \neg\ x{\leq}y \Longrightarrow \neg y{\leq}x \Longrightarrow y{=}x$
⟨*proof*⟩

lemma *Un_le_compat* : $o \leq p \Longrightarrow q \leq r \Longrightarrow Ord(o) \Longrightarrow Ord(p) \Longrightarrow Ord(q) \Longrightarrow Ord(r) \Longrightarrow o \cup q \leq p \cup r$
⟨*proof*⟩

lemma *Un_le* : $p \leq r \Longrightarrow q \leq r \Longrightarrow Ord(p) \Longrightarrow Ord(q) \Longrightarrow Ord(r) \Longrightarrow p \cup q \leq r$
⟨*proof*⟩

lemma *Un_leI3* : $o \leq r \Longrightarrow p \leq r \Longrightarrow q \leq r \Longrightarrow Ord(o) \Longrightarrow Ord(p) \Longrightarrow Ord(q) \Longrightarrow Ord(r) \Longrightarrow o \cup p \cup q \leq r$
⟨*proof*⟩

lemma *diff_mono* :
assumes $m \in nat$ $n{\in}nat$ $p \in nat$ $m < n$ $p{\leq}m$
shows $m\#{-}p < n\#{-}p$
⟨*proof*⟩

lemma *pred_Un*:
$x \in nat \Longrightarrow y \in nat \Longrightarrow pred(succ(x) \cup y) = x \cup pred(y)$
$x \in nat \Longrightarrow y \in nat \Longrightarrow pred(x \cup succ(y)) = pred(x) \cup y$
⟨*proof*⟩

lemma *le_natI* : $j \leq n \Longrightarrow n \in nat \Longrightarrow j{\in}nat$
⟨*proof*⟩

lemma *le_natE* : $n{\in}nat \Longrightarrow j < n \Longrightarrow\ j{\in}n$
⟨*proof*⟩

lemma *leD* : **assumes** $n{\in}nat$ $j \leq n$
shows $j < n \mid j = n$
⟨*proof*⟩

lemma *pred_nat_eq* :
assumes $n{\in}nat$
shows $pred(n) = \bigcup\ n$
⟨*proof*⟩

2.1 Some results in ordinal arithmetic

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

lemma *max_cong* :
assumes $x \leq y$ $Ord(y)$ $Ord(z)$
shows $max(x,y) \leq max(y,z)$
⟨*proof*⟩

lemma *max_commutes* :
assumes $Ord(x)$ $Ord(y)$
shows $max(x,y) = max(y,x)$
⟨*proof*⟩

lemma *max_cong2* :
assumes $x \leq y$ $Ord(y)$ $Ord(z)$ $Ord(x)$
shows $max(x,z) \leq max(y,z)$
⟨*proof*⟩

lemma *max_D1* :
assumes $x = y$ $w < z$ $Ord(x)$ $Ord(w)$ $Ord(z)$ $max(x,w) = max(y,z)$
shows $z \leq y$
⟨*proof*⟩

lemma *max_D2* :
assumes $w = y \vee w = z$ $x < y$ $Ord(x)$ $Ord(w)$ $Ord(y)$ $Ord(z)$ $max(x,w) = max(y,z)$
shows $x < w$
⟨*proof*⟩

lemma *oadd_lt_mono2* :
assumes $Ord(n)$ $Ord(\alpha)$ $Ord(\beta)$ $\alpha < \beta$ $x < n$ $y < n$ $0 < n$
shows $n ** \alpha + x < n ** \beta + y$
⟨*proof*⟩
end

3 Various results missing from ZF.

theory *ZF_Miscellanea*
imports
ZF
Nat_Miscellanea
begin

lemma *rex_mono* :
assumes $\exists\ d \in A\ .\ P(d)$ $A \subseteq B$
shows $\exists\ d \in B.\ P(d)$
⟨*proof*⟩

lemma *function_subset*:
function(f) $\Longrightarrow$ $g \subseteq f$ $\Longrightarrow$ *function*(g)
⟨*proof*⟩

lemma *converse_refl* : *refl*(A,r) $\Longrightarrow$ *refl*(A,*converse*(r))
⟨*proof*⟩

lemma *Ord_lt_subset* : *Ord*(b) $\Longrightarrow$ $a<b$ $\Longrightarrow$ $a \subseteq b$
⟨*proof*⟩

lemma *funcI* : $f \in A \to B \Longrightarrow a \in A \Longrightarrow b = f\ `\ a \Longrightarrow \langle a,\ b\rangle \in f$
⟨*proof*⟩

lemma *vimage_fun_sing*:
assumes $f \in A \to B$ $b \in B$
shows $\{a \in A\ .\ f`a = b\} = f\text{-}``\{b\}$
⟨*proof*⟩

lemma *image_fun_subset*: $S \in A \to B \Longrightarrow C \subseteq A \Longrightarrow \{S\ `\ x\ .\ x \in\ C\} = S``C$
⟨*proof*⟩

lemma *inj_range_Diff*:
assumes $f \in$ *inj*(A,A')
shows $f``A$ - $f``T = f``(A$ - $T)$
⟨*proof*⟩

lemma *subset_Diff_Un*: $X \subseteq A \Longrightarrow A = (A$ - $X) \cup X$ ⟨*proof*⟩

lemma *Diff_bij*:
assumes $\forall\, A \in F.\ X \subseteq A$ **shows** $(\lambda A \in F.\ A\text{-}X) \in$ *bij*(F, $\{A\text{-}X.\ A \in F\}$)
⟨*proof*⟩

lemma *function_space_nonempty*:
assumes $b \in B$
shows $(\lambda x \in A.\ b) : A \to B$
⟨*proof*⟩

lemma *lam_constant_eq_cartprod*: $(\lambda\_ \in A.\ y) = A \times \{y\}$
⟨*proof*⟩

lemma *vimage_lam*: $(\lambda x \in A.\ f(x))$ -`` $B = \{\ x \in A\ .\ f(x) \in B\ \}$
⟨*proof*⟩

lemma *range_fun_subset_codomain*:
assumes h:$B \to C$
shows *range*(h) $\subseteq C$
⟨*proof*⟩

lemma *Pi_rangeD*:
assumes $f{\in}Pi(A,B)$ $b \in range(f)$
shows $\exists\, a{\in}A.\ f`a = b$
⟨*proof*⟩

lemma *Pi_range_eq*: $f \in Pi(A,B) \Longrightarrow range(f) = \{f\ `\ x\ .\ x \in A\}$
⟨*proof*⟩

lemma *Pi_vimage_subset* : $f \in Pi(A,B) \Longrightarrow f\text{-}``C \subseteq A$
⟨*proof*⟩

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

lemma *minimum_in'*: $minimum(r,B) \in B \cup \{0\}$
⟨*proof*⟩

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

lemma *well_ord_surj_imp_inj_inverse*:
assumes $well\_ord(A,r)$ $h \in surj(A,B)$
shows $(\lambda b{\in}B.\ minimum(r,\ \{a{\in}A.\ h`a{=}b\})) \in inj(B,A)$
⟨*proof*⟩

lemma *well_ord_surj_imp_lepoll*:
assumes $well\_ord(A,r)$ $h \in surj(A,B)$
shows $B \lesssim A$
⟨*proof*⟩
lemma *surj_imp_well_ord*:
assumes $well\_ord(A,r)$ $h \in surj(A,B)$
shows $\exists\, s.\ well\_ord(B,s)$
⟨*proof*⟩

lemma *Pow_sing* : $Pow(\{a\}) = \{0,\{a\}\}$
⟨*proof*⟩

lemma *Pow_cons*:
shows $Pow(cons(a,A)) = Pow(A) \cup \{\{a\} \cup X\ .\ X{:}\ Pow(A)\}$
⟨*proof*⟩

lemma *app_nm* :
assumes $n{\in}nat$ $m{\in}nat$ $f{\in}n{\to}m$ $x \in nat$
shows $f`x \in nat$
⟨*proof*⟩

lemma *Upair_eq_cons*: $Upair(a,b) = \{a,b\}$
⟨*proof*⟩

lemma *converse_apply_eq* : *converse*(f) ‘ $x = \bigcup(f$ -‘‘ $\{x\})$
⟨*proof*⟩

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

lemma *Finite_imp_lesspoll_nat*:
assumes *Finite*(A)
shows $A \prec nat$
⟨*proof*⟩

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

lemma *curry_type* :
assumes $f \in A \times B \to C$
shows *curry*(A,B,f) $\in A \to (B \to C)$
⟨*proof*⟩

end

4 Renaming of variables in internalized formulas

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

4.1 Renaming of free variables

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

lemma *union_fun_type*:
assumes $f \in m \to n$
$g \in p \to q$
shows *union_fun*(f,g,m,p) $\in m \cup p \to n \cup q$
⟨*proof*⟩

lemma *union_fun_action* :
assumes
env $\in$ *list*(M)
env′ $\in$ *list*(M)
length(*env*) $= m \cup p$
$\forall\ i$. $i \in m \longrightarrow$ *nth*(f‘i,*env*′) = *nth*(i,*env*)
$\forall\ j$. $j \in p \longrightarrow$ *nth*(g‘j,*env*′) = *nth*(j,*env*)
shows $\forall\ i$. $i \in m \cup p \longrightarrow$

$nth(i,env) = nth(union\_fun(f,g,m,p)`i,env')$
⟨*proof*⟩

lemma *id_fn_type* :
assumes $n \in nat$
shows $id(n) \in n \to n$
⟨*proof*⟩

lemma *id_fn_action*:
assumes $n \in nat\ env \in list(M)$
shows $\bigwedge j\ .\ j < n \Longrightarrow nth(j,env) = nth(id(n)`j,env)$
⟨*proof*⟩

definition
$rsum :: [i,i,i,i,i] \Rightarrow i$ **where**
$rsum(f,g,m,n,p) \equiv \lambda j \in m+_\omega p\ \ .\ if\ j<m\ then\ f`j\ else\ (g`(j\#-m))+_\omega n$

lemma *sum_inl*:
assumes $m \in nat\ n \in nat$
$f \in m \to n\ x \in m$
shows $rsum(f,g,m,n,p)`x = f`x$
⟨*proof*⟩

lemma *sum_inr*:
assumes $m \in nat\ n \in nat\ p \in nat$
$g \in p \to q\ m \le x\ x < m+_\omega p$
shows $rsum(f,g,m,n,p)`x = g`(x\#-m)+_\omega n$
⟨*proof*⟩

lemma *sum_action* :
assumes $m \in nat\ n \in nat\ p \in nat\ q \in nat$
$f \in m \to n\ g \in p \to q$
$env \in list(M)$
$env' \in list(M)$
$env1 \in list(M)$
$env2 \in list(M)$
$length(env) = m$
$length(env1) = p$
$length(env') = n$
$\bigwedge i\ .\ i < m \Longrightarrow nth(i,env) = nth(f`i,env')$
$\bigwedge j.\ j < p \Longrightarrow nth(j,env1) = nth(g`j,env2)$
shows $\forall\ i\ .\ i < m+_\omega p \longrightarrow$
$nth(i,env@env1) = nth(rsum(f,g,m,n,p)`i,env'@env2)$
⟨*proof*⟩

lemma *sum_type* :

assumes $m \in nat\ n \in nat\ p \in nat\ q \in nat$
$f \in m \to n\ g \in p \to q$
shows $rsum(f,g,m,n,p) \in (m +_\omega p) \to (n +_\omega q)$
⟨*proof*⟩

lemma *sum_type_id* :
assumes
$f \in length(env) \to length(env')$
$env \in list(M)$
$env' \in list(M)$
$env1 \in list(M)$
shows
$rsum(f,id(length(env1)),length(env),length(env'),length(env1)) \in$
$(length(env) +_\omega length(env1)) \to (length(env') +_\omega length(env1))$
⟨*proof*⟩

lemma *sum_type_id_aux2* :
assumes
$f \in m \to n$
$m \in nat\ n \in nat$
$env1 \in list(M)$
shows
$rsum(f,id(length(env1)),m,n,length(env1)) \in$
$(m +_\omega length(env1)) \to (n +_\omega length(env1))$
⟨*proof*⟩

lemma *sum_action_id* :
assumes
$env \in list(M)$
$env' \in list(M)$
$f \in length(env) \to length(env')$
$env1 \in list(M)$
$\bigwedge i\ .\ i < length(env) \Longrightarrow nth(i,env) = nth(f`i,env')$
shows $\bigwedge i\ .\ i < length(env) +_\omega length(env1) \Longrightarrow$
$nth(i,env@env1) = nth(rsum(f,id(length(env1)),length(env),length(env'),length(env1))`i,env'@env1)$
⟨*proof*⟩

lemma *sum_action_id_aux* :
assumes
$f \in m \to n$
$env \in list(M)$
$env' \in list(M)$
$env1 \in list(M)$
$length(env) = m$
$length(env') = n$
$length(env1) = p$
$\bigwedge i\ .\ i < m \Longrightarrow nth(i,env) = nth(f`i,env')$
shows $\bigwedge i\ .\ i < m +_\omega length(env1) \Longrightarrow$
$nth(i,env@env1) = nth(rsum(f,id(length(env1)),m,n,length(env1))`i,env'@env1)$

⟨*proof*⟩

definition
sum_id :: [*i*,*i*] ⇒ *i* **where**
sum_id(*m*,*f*) ≡ *rsum*(λ*x*∈*1*.*x*,*f*,*1*,*1*,*m*)

lemma *sum_id0* : *m*∈*nat*⟹*sum_id*(*m*,*f*)‘*0* = *0*
⟨*proof*⟩

lemma *sum_idS* : *p*∈*nat* ⟹ *q*∈*nat* ⟹ *f*∈*p*→*q* ⟹ *x* ∈ *p* ⟹ *sum_id*(*p*,*f*)‘(*succ*(*x*))
= *succ*(*f*‘*x*)
⟨*proof*⟩

lemma *sum_id_tc_aux* :
p ∈ *nat* ⟹ *q* ∈ *nat* ⟹ *f* ∈ *p* → *q* ⟹ *sum_id*(*p*,*f*) ∈ *1*+$_\omega$*p* → *1*+$_\omega$*q*
⟨*proof*⟩

lemma *sum_id_tc* :
n ∈ *nat* ⟹ *m* ∈ *nat* ⟹ *f* ∈ *n* → *m* ⟹ *sum_id*(*n*,*f*) ∈ *succ*(*n*) → *succ*(*m*)
⟨*proof*⟩

4.2 Renaming of formulas

consts *ren* :: *i*⇒*i*
primrec
ren(*Member*(*x*,*y*)) =
(λ *n* ∈ *nat* . λ *m* ∈ *nat*. λ*f* ∈ *n* → *m*. *Member* (*f*‘*x*, *f*‘*y*))

ren(*Equal*(*x*,*y*)) =
(λ *n* ∈ *nat* . λ *m* ∈ *nat*. λ*f* ∈ *n* → *m*. *Equal* (*f*‘*x*, *f*‘*y*))

ren(*Nand*(*p*,*q*)) =
(λ *n* ∈ *nat* . λ *m* ∈ *nat*. λ*f* ∈ *n* → *m*. *Nand* (*ren*(*p*)‘*n*‘*m*‘*f*, *ren*(*q*)‘*n*‘*m*‘*f*))

ren(*Forall*(*p*)) =
(λ *n* ∈ *nat* . λ *m* ∈ *nat*. λ*f* ∈ *n* → *m*. *Forall* (*ren*(*p*)‘*succ*(*n*)‘*succ*(*m*)‘*sum_id*(*n*,*f*)))

lemma *arity_meml* : *l* ∈ *nat* ⟹ *Member*(*x*,*y*) ∈ *formula* ⟹ *arity*(*Member*(*x*,*y*))
≤ *l* ⟹ *x* ∈ *l*
⟨*proof*⟩
lemma *arity_memr* : *l* ∈ *nat* ⟹ *Member*(*x*,*y*) ∈ *formula* ⟹ *arity*(*Member*(*x*,*y*))
≤ *l* ⟹ *y* ∈ *l*
⟨*proof*⟩
lemma *arity_eql* : *l* ∈ *nat* ⟹ *Equal*(*x*,*y*) ∈ *formula* ⟹ *arity*(*Equal*(*x*,*y*)) ≤ *l*
⟹ *x* ∈ *l*
⟨*proof*⟩
lemma *arity_eqr* : *l* ∈ *nat* ⟹ *Equal*(*x*,*y*) ∈ *formula* ⟹ *arity*(*Equal*(*x*,*y*)) ≤ *l*
⟹ *y* ∈ *l*

⟨*proof*⟩
lemma *nand_ar1* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(p) \leq arity(Nand(p,q))$
⟨*proof*⟩
lemma *nand_ar2* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(q) \leq arity(Nand(p,q))$
⟨*proof*⟩

lemma *nand_ar1D* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(Nand(p,q)) \leq n \Longrightarrow arity(p) \leq n$
⟨*proof*⟩
lemma *nand_ar2D* : $p \in formula \Longrightarrow q \in formula \Longrightarrow arity(Nand(p,q)) \leq n \Longrightarrow arity(q) \leq n$
⟨*proof*⟩

lemma *ren_tc* : $p \in formula \Longrightarrow$
$(\bigwedge n\ m\ f\ .\ n \in nat \Longrightarrow m \in nat \Longrightarrow f \in n \to m \Longrightarrow ren(p)`n`m`f \in formula)$
⟨*proof*⟩

lemma *arity_ren* :
fixes p
assumes $p \in formula$
shows $\bigwedge n\ m\ f\ .\ n \in nat \Longrightarrow m \in nat \Longrightarrow f \in n \to m \Longrightarrow arity(p) \leq n \Longrightarrow arity(ren(p)`n`m`f) \leq m$
⟨*proof*⟩

lemma *arity_forallE* : $p \in formula \Longrightarrow m \in nat \Longrightarrow arity(Forall(p)) \leq m \Longrightarrow arity(p) \leq succ(m)$
⟨*proof*⟩

lemma *env_coincidence_sum_id* :
assumes $m \in nat\ n \in nat$
$\varrho \in list(A)\ \varrho' \in list(A)$
$f \in n \to m$
$\bigwedge i\ .\ i < n \Longrightarrow nth(i,\varrho) = nth(f`i,\varrho')$
$a \in A\ j \in succ(n)$
shows $nth(j,Cons(a,\varrho)) = nth(sum\_id(n,f)`j,Cons(a,\varrho'))$
⟨*proof*⟩

lemma *sats_iff_sats_ren* :
assumes $\varphi \in formula$
shows $[\![\ n \in nat\ ;\ m \in nat\ ;\ \varrho \in list(M)\ ;\ \varrho' \in list(M)\ ;\ f \in n \to m\ ;$
$arity(\varphi) \leq n\ ;$
$\bigwedge i\ .\ i < n \Longrightarrow nth(i,\varrho) = nth(f`i,\varrho')\ ]\!] \Longrightarrow$
$sats(M,\varphi,\varrho) \longleftrightarrow sats(M,ren(\varphi)`n`m`f,\varrho')$
⟨*proof*⟩

end
theory *Renaming_Auto*

imports
Utils
Renaming
keywords
rename :: *thy_decl* % *ML*
and
simple_rename :: *thy_decl* % *ML*
and
src
and
tgt
abbrevs
simple_rename =

begin

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

lemmas *nat_succI* = *nat_succ_iff*[*THEN iffD2*]
⟨*ML*⟩

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

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

locale *M_basic_no_repl* = *M_trivial* +
assumes *Inter_separation*:
M(*A*) ==> *separation*(*M*, λ*x*. ∀ *y*[*M*]. *y*∈*A* ⟶ *x*∈*y*)
and *Diff_separation*:
M(*B*) ==> *separation*(*M*, λ*x*. *x* ∉ *B*)
and *cartprod_separation*:
[| *M*(*A*); *M*(*B*) |]
==> *separation*(*M*, λ*z*. ∃ *x*[*M*]. *x*∈*A* & (∃ *y*[*M*]. *y*∈*B* & *pair*(*M*,*x*,*y*,*z*)))
and *image_separation*:
[| *M*(*A*); *M*(*r*) |]
==> *separation*(*M*, λ*y*. ∃ *p*[*M*]. *p*∈*r* & (∃ *x*[*M*]. *x*∈*A* & *pair*(*M*,*x*,*y*,*p*)))
and *converse_separation*:
M(*r*) ==> *separation*(*M*,
λ*z*. ∃ *p*[*M*]. *p*∈*r* & (∃ *x*[*M*]. ∃ *y*[*M*]. *pair*(*M*,*x*,*y*,*p*) & *pair*(*M*,*y*,*x*,*z*)))
and *restrict_separation*:
M(*A*) ==> *separation*(*M*, λ*z*. ∃ *x*[*M*]. *x*∈*A* & (∃ *y*[*M*]. *pair*(*M*,*x*,*y*,*z*)))
and *comp_separation*:
[| *M*(*r*); *M*(*s*) |]
==> *separation*(*M*, λ*xz*. ∃ *x*[*M*]. ∃ *y*[*M*]. ∃ *z*[*M*]. ∃ *xy*[*M*]. ∃ *yz*[*M*].
pair(*M*,*x*,*z*,*xz*) & *pair*(*M*,*x*,*y*,*xy*) & *pair*(*M*,*y*,*z*,*yz*) &
xy∈*s* & *yz*∈*r*)
and *pred_separation*:

[| $M(r)$; $M(x)$ |] ==> $separation(M, \lambda y.\ \exists\, p[M].\ p{\in}r\ \&\ pair(M,y,x,p))$
and *Memrel_separation*:
$separation(M, \lambda z.\ \exists\, x[M].\ \exists\, y[M].\ pair(M,x,y,z)\ \&\ x \in y)$
and *is_recfun_separation*:
— for well-founded recursion: used to prove *is_recfun_equal*
[| $M(r)$; $M(f)$; $M(g)$; $M(a)$; $M(b)$ |]
==> $separation(M,$
$\lambda x.\ \exists\, xa[M].\ \exists\, xb[M].$
$pair(M,x,a,xa)\ \&\ xa \in r\ \&\ pair(M,x,b,xb)\ \&\ xb \in r\ \&$
$(\exists\, fx[M].\ \exists\, gx[M].\ fun\_apply(M,f,x,fx)\ \&\ fun\_apply(M,g,x,gx)\ \&$
$fx \neq gx))$
and *power_ax*: $power\_ax(M)$

lemma (**in** *M_basic_no_repl*) *cartprod_iff*:
[| $M(A)$; $M(B)$; $M(C)$ |]
==> $cartprod(M,A,B,C) \longleftrightarrow$
$(\exists\, p1[M].\ \exists\, p2[M].\ powerset(M,A \cup B,p1)\ \&\ powerset(M,p1,p2)\ \&$
$C = \{z \in p2.\ \exists\, x{\in}A.\ \exists\, y{\in}B.\ z = <x,y>\})$
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *cartprod_closed_lemma*:
[| $M(A)$; $M(B)$ |] ==> $\exists\, C[M].\ cartprod(M,A,B,C)$
⟨*proof*⟩

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

lemma (**in** *M_basic_no_repl*) *cartprod_closed* [*intro*,*simp*]:
[| $M(A)$; $M(B)$ |] ==> $M(A{*}B)$
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *sum_closed* [*intro*,*simp*]:
[| $M(A)$; $M(B)$ |] ==> $M(A{+}B)$
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *sum_abs* [*simp*]:
[| $M(A)$; $M(B)$; $M(Z)$ |] ==> $is\_sum(M,A,B,Z) \longleftrightarrow (Z = A{+}B)$
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *M_converse_iff*:
$M(r)$ ==>
$converse(r) =$
$\{z \in \bigcup(\bigcup(r)) * \bigcup(\bigcup(r)).$
$\exists\, p{\in}r.\ \exists\, x[M].\ \exists\, y[M].\ p = \langle x,y\rangle\ \&\ z = \langle y,x\rangle\}$
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *converse_closed* [*intro*,*simp*]:
$M(r)$ ==> $M(converse(r))$
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *converse_abs* [*simp*]:
[| $M(r)$; $M(z)$ |] ==> *is_converse*(M,r,z) $\longleftrightarrow$ z = *converse*(r)
⟨*proof*⟩

4.2.1 image, preimage, domain, range

lemma (**in** *M_basic_no_repl*) *image_closed* [*intro*,*simp*]:
[| $M(A)$; $M(r)$ |] ==> $M(r$‘‘$A)$
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *vimage_abs* [*simp*]:
[| $M(r)$; $M(A)$; $M(z)$ |] ==> *pre_image*(M,r,A,z) $\longleftrightarrow$ z = r-‘‘A
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *vimage_closed* [*intro*,*simp*]:
[| $M(A)$; $M(r)$ |] ==> $M(r$-‘‘$A)$
⟨*proof*⟩

4.2.2 Domain, range and field

lemma (**in** *M_basic_no_repl*) *domain_closed* [*intro*,*simp*]:
$M(r)$ ==> M(*domain*(r))
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *range_closed* [*intro*,*simp*]:
$M(r)$ ==> M(*range*(r))
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *field_abs* [*simp*]:
[| $M(r)$; $M(z)$ |] ==> *is_field*(M,r,z) $\longleftrightarrow$ z = *field*(r)
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *field_closed* [*intro*,*simp*]:
$M(r)$ ==> M(*field*(r))
⟨*proof*⟩

4.2.3 Relations, functions and application

lemma (**in** *M_basic_no_repl*) *apply_closed* [*intro*,*simp*]:
[|$M(f)$; $M(a)$|] ==> $M(f$‘$a)$
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *apply_abs* [*simp*]:
[| $M(f)$; $M(x)$; $M(y)$ |] ==> *fun_apply*(M,f,x,y) $\longleftrightarrow$ f‘x = y
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *injection_abs* [*simp*]:
[| $M(A)$; $M(f)$ |] ==> *injection*(M,A,B,f) $\longleftrightarrow$ $f \in$ *inj*(A,B)
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *surjection_abs* [*simp*]:
[| *M*(*A*); *M*(*B*); *M*(*f*) |] ==> *surjection*(*M*,*A*,*B*,*f*) ⟷ *f* ∈ *surj*(*A*,*B*)
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *bijection_abs* [*simp*]:
[| *M*(*A*); *M*(*B*); *M*(*f*) |] ==> *bijection*(*M*,*A*,*B*,*f*) ⟷ *f* ∈ *bij*(*A*,*B*)
⟨*proof*⟩

4.2.4 Composition of relations

lemma (**in** *M_basic_no_repl*) *M_comp_iff*:
[| *M*(*r*); *M*(*s*) |]
==> *r* *O* *s* =
{*xz* ∈ *domain*(*s*) ∗ *range*(*r*).
∃ *x*[*M*]. ∃ *y*[*M*]. ∃ *z*[*M*]. *xz* = ⟨*x*,*z*⟩ & ⟨*x*,*y*⟩ ∈ *s* & ⟨*y*,*z*⟩ ∈ *r*}
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *comp_closed* [*intro*,*simp*]:
[| *M*(*r*); *M*(*s*) |] ==> *M*(*r* *O* *s*)
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *composition_abs* [*simp*]:
[| *M*(*r*); *M*(*s*); *M*(*t*) |] ==> *composition*(*M*,*r*,*s*,*t*) ⟷ *t* = *r* *O* *s*
⟨*proof*⟩

no longer needed

lemma (**in** *M_basic_no_repl*) *restriction_is_function*:
[| *restriction*(*M*,*f*,*A*,*z*); *function*(*f*); *M*(*f*); *M*(*A*); *M*(*z*) |]
==> *function*(*z*)
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *restrict_closed* [*intro*,*simp*]:
[| *M*(*A*); *M*(*r*) |] ==> *M*(*restrict*(*r*,*A*))
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *Inter_closed* [*intro*,*simp*]:
M(*A*) ==> *M*(⋂(*A*))
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *Int_closed* [*intro*,*simp*]:
[| *M*(*A*); *M*(*B*) |] ==> *M*(*A* ∩ *B*)
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *Diff_closed* [*intro*,*simp*]:
[|*M*(*A*); *M*(*B*)|] ==> *M*(*A*-*B*)
⟨*proof*⟩

4.2.5 Some Facts About Separation Axioms

lemma (**in** *M_basic_no_repl*) *separation_conj*:

[|*separation*(*M*,*P*); *separation*(*M*,*Q*)|] ==> *separation*(*M*, λ*z*. *P*(*z*) & *Q*(*z*))
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *separation_disj*:
[|*separation*(*M*,*P*); *separation*(*M*,*Q*)|] ==> *separation*(*M*, λ*z*. *P*(*z*) | *Q*(*z*))
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *separation_neg*:
separation(*M*,*P*) ==> *separation*(*M*, λ*z*. ~*P*(*z*))
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *separation_imp*:
[|*separation*(*M*,*P*); *separation*(*M*,*Q*)|]
==> *separation*(*M*, λ*z*. *P*(*z*) ⟶ *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_no_repl*) *separation_rall*:
[|*M*(*Y*); ∀ *y*[*M*]. *separation*(*M*, λ*x*. *P*(*x*,*y*));
∀ *z*[*M*]. *strong_replacement*(*M*, λ*x y*. *y* = {*u* ∈ *z* . *P*(*u*,*x*)})|]
==> *separation*(*M*, λ*x*. ∀ *y*[*M*]. *y*∈*Y* ⟶ *P*(*x*,*y*))
⟨*proof*⟩

4.2.6 Functions and function space

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

lemma (**in** *M_basic_no_repl*) *list_case'_closed* [*intro*,*simp*]:
[|*M*(*k*); *M*(*a*); ∀ *x*[*M*]. ∀ *y*[*M*]. *M*(*b*(*x*,*y*))|] ==> *M*(*list_case'*(*a*,*b*,*k*))
⟨*proof*⟩

lemma (**in** *M_basic_no_repl*) *tl'_closed*: *M*(*x*) ==> *M*(*tl'*(*x*))
⟨*proof*⟩

sublocale *M_basic* ⊆ *mbnr*:*M_basic_no_repl*
⟨*proof*⟩

end

5 Absoluteness Properties for Recursive Datatypes

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

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

definition
directed :: *i*=>*o* **where**
directed(*A*) == *A*≠*0* & (∀ *x*∈*A*. ∀ *y*∈*A*. *x* ∪ *y* ∈ *A*)

definition
contin :: (*i*=>*i*) => *o* **where**
contin(*h*) == (∀ *A*. *directed*(*A*) ⟶ *h*(⋃ *A*) = (⋃ *X*∈*A*. *h*(*X*)))

lemma *bnd_mono_iterates_subset*: [|*bnd_mono*(*D*, *h*); *n* ∈ *nat*|] ==> *h*^*n* (*0*) ⊆ *D*
⟨*proof*⟩

lemma *bnd_mono_increasing* [*rule_format*]:
[|*i* ∈ *nat*; *j* ∈ *nat*; *bnd_mono*(*D*,*h*)|] ==> *i* ≤ *j* ⟶ *h*^*i*(*0*) ⊆ *h*^*j*(*0*)
⟨*proof*⟩

lemma *directed_iterates*: *bnd_mono*(*D*,*h*) ==> *directed*({*h*^*n* (*0*). *n*∈*nat*})
⟨*proof*⟩

lemma *contin_iterates_eq*:
[|*bnd_mono*(*D*, *h*); *contin*(*h*)|]
==> *h*(⋃ *n*∈*nat*. *h*^*n* (*0*)) = (⋃ *n*∈*nat*. *h*^*n* (*0*))
⟨*proof*⟩

lemma *lfp_subset_Union*:
[|*bnd_mono*(*D*, *h*); *contin*(*h*)|] ==> *lfp*(*D*,*h*) ⊆ (⋃ *n*∈*nat*. *h*^*n*(*0*))
⟨*proof*⟩

lemma *Union_subset_lfp*:
bnd_mono(*D*,*h*) ==> (⋃ *n*∈*nat*. *h*^*n*(*0*)) ⊆ *lfp*(*D*,*h*)
⟨*proof*⟩

lemma *lfp_eq_Union*:
[|*bnd_mono*(*D*, *h*); *contin*(*h*)|] ==> *lfp*(*D*,*h*) = (⋃ *n*∈*nat*. *h*^*n*(*0*))
⟨*proof*⟩

5.1.1 Some Standard Datatype Constructions Preserve Continuity

lemma *contin_imp_mono*: [|*X*⊆*Y*; *contin*(*F*)|] ==> *F*(*X*) ⊆ *F*(*Y*)
⟨*proof*⟩

lemma *sum_contin*: [|*contin*(*F*); *contin*(*G*)|] ==> *contin*(λ*X*. *F*(*X*) + *G*(*X*))
⟨*proof*⟩

lemma *prod_contin*: [|*contin*(*F*); *contin*(*G*)|] ==> *contin*(λ*X*. *F*(*X*) * *G*(*X*))

⟨*proof*⟩

lemma *const_contin*: *contin*(λ*X*. *A*)
⟨*proof*⟩

lemma *id_contin*: *contin*(λ*X*. *X*)
⟨*proof*⟩

5.2 Absoluteness for "Iterates"

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

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

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

lemma (**in** *M_basic*) *iterates_MH_abs*:
[| *relation1*(*M*,*isF*,*F*); *M*(*n*); *M*(*g*); *M*(*z*) |]
==> *iterates_MH*(*M*,*isF*,*v*,*n*,*g*,*z*) ⟷ *z* = *nat_case*(*v*, λ*m*. *F*(*g*‘*m*), *n*)
⟨*proof*⟩

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

theorem (**in** *M_trancl*) *iterates_abs*:
[| *iterates_replacement*(*M*,*isF*,*v*); *relation1*(*M*,*isF*,*F*);
n ∈ *nat*; *M*(*v*); *M*(*z*); ∀ *x*[*M*]. *M*(*F*(*x*)) |]
==> *is_iterates*(*M*,*isF*,*v*,*n*,*z*) ⟷ *z* = *iterates*(*F*,*n*,*v*)
⟨*proof*⟩

lemma (**in** *M_trancl*) *iterates_closed* [*intro*,*simp*]:
[| *iterates_replacement*(*M*,*isF*,*v*); *relation1*(*M*,*isF*,*F*);
n ∈ *nat*; *M*(*v*); ∀ *x*[*M*]. *M*(*F*(*x*)) |]

$==>$ $M(iterates(F,n,v))$
⟨*proof*⟩

5.3 lists without univ

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

lemma *list_fun_bnd_mono*: $bnd\_mono(univ(A), \lambda X.\ \{0\} + A*X)$
⟨*proof*⟩

lemma *list_fun_contin*: $contin(\lambda X.\ \{0\} + A*X)$
⟨*proof*⟩

Re-expresses lists using sum and product

lemma *list_eq_lfp2*: $list(A) = lfp(univ(A), \lambda X.\ \{0\} + A*X)$
⟨*proof*⟩

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

lemma *list_eq_Union*:
$list(A) = (\bigcup n{\in}nat.\ (\lambda X.\ \{0\} + A*X)\ ^\frown n\ (0))$
⟨*proof*⟩

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

lemma (**in** *M_basic*) *list_functor_abs* [*simp*]:
$[|\ M(A);\ M(X);\ M(Z)\ |] ==> is\_list\_functor(M,A,X,Z) \longleftrightarrow (Z = \{0\} + A*X)$
⟨*proof*⟩

5.4 formulas without univ

lemma *formula_fun_bnd_mono*:
$bnd\_mono(univ(0), \lambda X.\ ((nat*nat) + (nat*nat)) + (X*X + X))$
⟨*proof*⟩

lemma *formula_fun_contin*:
$contin(\lambda X.\ ((nat*nat) + (nat*nat)) + (X*X + X))$
⟨*proof*⟩

Re-expresses formulas using sum and product

lemma *formula_eq_lfp2*:
$formula = lfp(univ(0), \lambda X.\ ((nat*nat) + (nat*nat)) + (X*X + X))$
⟨*proof*⟩

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

lemma *formula_eq_Union*:
$$formula = (\bigcup n{\in}nat.\ (\lambda X.\ ((nat{*}nat) + (nat{*}nat)) + (X{*}X + X))\ \widehat{}\ n\ (0))$$
⟨*proof*⟩

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

lemma (**in** *M_trancl*) *formula_functor_abs* [*simp*]:
[| *M*(*X*); *M*(*Z*) |]
==> *is_formula_functor*(*M*,*X*,*Z*) ⟷
$Z = ((nat{*}nat) + (nat{*}nat)) + (X{*}X + X)$
⟨*proof*⟩

5.5 *M* Contains the List and Formula Datatypes

definition
list_N :: [*i*,*i*] => *i* **where**
list_N(*A*,*n*) == $(\lambda X.\ \{0\} + A * X)\widehat{}n\ (0)$

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

lemma *Cons_in_list_N* [*simp*]:
Cons(*a*,*l*) ∈ *list_N*(*A*,*succ*(*n*)) ⟷ *a*∈*A* & *l* ∈ *list_N*(*A*,*n*)
⟨*proof*⟩

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

lemma *list_N_0*: *list_N*(*A*,0) = 0
⟨*proof*⟩

lemma *list_N_succ*: *list_N*(*A*,*succ*(*n*)) = {0} + *A* * (*list_N*(*A*,*n*))
⟨*proof*⟩

lemma *list_N_imp_list*:
[| *l* ∈ *list_N*(*A*,*n*); *n* ∈ *nat* |] ==> *l* ∈ *list*(*A*)
⟨*proof*⟩

lemma *list_N_imp_length_lt* [*rule_format*]:
n ∈ *nat* ==> ∀ *l* ∈ *list_N*(*A*,*n*). *length*(*l*) < *n*

⟨*proof*⟩

lemma *list_imp_list_N* [*rule_format*]:
    $l \in list(A) ==> \forall\, n{\in}nat.\ length(l) < n \longrightarrow l \in list\_N(A,\ n)$
⟨*proof*⟩

lemma *list_N_imp_eq_length*:
    $[|n \in nat;\ l \notin list\_N(A,\ n);\ l \in list\_N(A,\ succ(n))|]$
     $==> n = length(l)$
⟨*proof*⟩

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

lemma (**in** *M_trivial*) *list_rec_eq*:
  $l \in list(A) ==>$
   $list\_rec(a,g,l) =$
   $transrec\ (succ(length(l)),$
     $\lambda x\ h.\ Lambda\ (list(A),$
                $list\_case'\ (a,$
                     $\lambda a\ l.\ g(a,\ l,\ h\ `\ succ(length(l))\ `\ l))))\ `\ l$
⟨*proof*⟩

definition
  $is\_list\_N :: [i=>o,i,i,i] => o$ **where**
    $is\_list\_N(M,A,n,Z) ==$
      $\exists\, zero[M].\ empty(M,zero)\ \&$
            $is\_iterates(M,\ is\_list\_functor(M,A),\ zero,\ n,\ Z)$

definition
  $mem\_list :: [i=>o,i,i] => o$ **where**
    $mem\_list(M,A,l) ==$
      $\exists\, n[M].\ \exists\, listn[M].$
       $finite\_ordinal(M,n)\ \&\ is\_list\_N(M,A,n,listn)\ \&\ l \in listn$

definition
  $is\_list :: [i=>o,i,i] => o$ **where**
    $is\_list(M,A,Z) == \forall\, l[M].\ l \in Z \longleftrightarrow mem\_list(M,A,l)$

5.5.1 Towards Absoluteness of *formula_rec*

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

lemma *depth_type* [*TC*]: $p \in formula ==> depth(p) \in nat$
⟨*proof*⟩

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

lemma *Member_in_formula_N* [*simp*]:
Member(*x*,*y*) ∈ *formula_N*(*succ*(*n*)) ⟷ *x* ∈ *nat* & *y* ∈ *nat*
⟨*proof*⟩

lemma *Equal_in_formula_N* [*simp*]:
Equal(*x*,*y*) ∈ *formula_N*(*succ*(*n*)) ⟷ *x* ∈ *nat* & *y* ∈ *nat*
⟨*proof*⟩

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

lemma *Forall_in_formula_N* [*simp*]:
Forall(*x*) ∈ *formula_N*(*succ*(*n*)) ⟷ *x* ∈ *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*:
[| *p* ∈ *formula_N*(*n*); *n* ∈ *nat* |] ==> *p* ∈ *formula*
⟨*proof*⟩

lemma *formula_N_imp_depth_lt* [*rule_format*]:
n ∈ *nat* ==> ∀ *p* ∈ *formula_N*(*n*). *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*]:
[| $m \in nat$; $n \in nat$ |] ==> $m \leq n \longrightarrow formula\_N(m) \subseteq formula\_N(n)$
⟨*proof*⟩

lemma *formula_N_distrib*:
[| $m \in nat$; $n \in nat$ |] ==> $formula\_N(m \cup n) = formula\_N(m) \cup formula\_N(n)$
⟨*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*)

5.5.2 Absoluteness of the List Construction

5.5.3 Absoluteness of Formulas

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

Re-expresses eclose using "iterates"

lemma *eclose_eq_Union*:
eclose(*A*) = (⋃ *n*∈*nat*. *Union*^*n* (*A*))
⟨*proof*⟩

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

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

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

locale *M_eclose* = *M_trancl* +
 assumes *eclose_replacement1*:
  $M(A) \Longrightarrow$ *iterates_replacement*(M, *big_union*(M), A)
 and *eclose_replacement2*:
  $M(A) \Longrightarrow$ *strong_replacement*(M,
      $\lambda n\ y.\ n{\in}nat\ \&$ *is_iterates*(M, *big_union*(M), A, n, y))

lemma (**in** *M_eclose*) *eclose_replacement2'*:
 $M(A) \Longrightarrow$ *strong_replacement*(M, $\lambda n\ y.\ n{\in}nat\ \&\ y = Union\hat{}n\ (A)$)
⟨*proof*⟩

lemma (**in** *M_eclose*) *eclose_closed* [*intro*,*simp*]:
   $M(A) \Longrightarrow M(eclose(A))$
⟨*proof*⟩

lemma (**in** *M_eclose*) *is_eclose_n_abs* [*simp*]:
   $[|M(A);\ n{\in}nat;\ M(Z)|] \Longrightarrow$ *is_eclose_n*(M,A,n,Z) $\longleftrightarrow$ $Z = Union\hat{}n\ (A)$
⟨*proof*⟩

lemma (**in** *M_eclose*) *mem_eclose_abs* [*simp*]:
   $M(A) \Longrightarrow$ *mem_eclose*(M,A,l) $\longleftrightarrow$ $l \in eclose(A)$
⟨*proof*⟩

lemma (**in** *M_eclose*) *eclose_abs* [*simp*]:
   $[|M(A);\ M(Z)|] \Longrightarrow$ *is_eclose*(M,A,Z) $\longleftrightarrow$ $Z = eclose(A)$
⟨*proof*⟩

5.7 Absoluteness for *transrec*

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

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

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

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

theorem (**in** *M_eclose*) *transrec_abs*:

[|*transrec_replacement*(*M*,*MH*,*i*); *relation2*(*M*,*MH*,*H*);
Ord(*i*); *M*(*i*); *M*(*z*);
$\forall\, x[M].\ \forall\, g[M].\ \mathit{function}(g) \longrightarrow M(H(x,g))|]$
==> *is_transrec*(*M*,*MH*,*i*,*z*) $\longleftrightarrow$ *z* = *transrec*(*i*,*H*)
⟨*proof*⟩

theorem (**in** *M_eclose*) *transrec_closed*:
[|*transrec_replacement*(*M*,*MH*,*i*); *relation2*(*M*,*MH*,*H*);
Ord(*i*); *M*(*i*);
$\forall\, x[M].\ \forall\, g[M].\ \mathit{function}(g) \longrightarrow M(H(x,g))|]$
==> *M*(*transrec*(*i*,*H*))
⟨*proof*⟩

Helps to prove instances of *transrec_replacement*

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

5.8 Absoluteness for the List Operator *length*

But it is never used.

definition
is_length :: [*i*=>*o*,*i*,*i*,*i*] => *o* **where**
is_length(*M*,*A*,*l*,*n*) ==
$\exists\, sn[M].\ \exists\, list\_n[M].\ \exists\, list\_sn[M].$
is_list_N(*M*,*A*,*n*,*list_n*) & *l* $\notin$ *list_n* &
successor(*M*,*n*,*sn*) & *is_list_N*(*M*,*A*,*sn*,*list_sn*) & *l* $\in$ *list_sn*

Proof is trivial since *length* returns natural numbers.

lemma (**in** *M_trivial*) *length_closed* [*intro*,*simp*]:
l $\in$ *list*(*A*) ==> *M*(*length*(*l*))
⟨*proof*⟩

5.9 Absoluteness for the List Operator *nth*

lemma *nth_eq_hd_iterates_tl* [*rule_format*]:
xs $\in$ *list*(*A*) ==> $\forall\, n \in nat.$ *nth*(*n*,*xs*) = *hd*′ (*tl*′^*n* (*xs*))
⟨*proof*⟩

lemma (**in** *M_basic*) *iterates_tl′_closed*:
[|*n* $\in$ *nat*; *M*(*x*)|] ==> *M*(*tl*′^*n* (*x*))
⟨*proof*⟩

Immediate by type-checking

lemma (**in** *M_trancl*) *nth_closed* [*intro*,*simp*]:
    [|$xs \in list(A)$; $n \in nat$; $M(A)$|] ==> $M(nth(n,xs))$
⟨*proof*⟩

5.10 Relativization and Absoluteness for the *formula* Constructors

definition
  is_Member :: [*i*=>*o*,*i*,*i*,*i*] => *o* **where**
    — because $Member(x, y) \equiv Inl(Inl(\langle x, y\rangle))$
   is_Member(*M*,*x*,*y*,*Z*) ==
      ∃ *p*[*M*]. ∃ *u*[*M*]. *pair*(*M*,*x*,*y*,*p*) & *is_Inl*(*M*,*p*,*u*) & *is_Inl*(*M*,*u*,*Z*)

lemma (**in** *M_trivial*) *Member_abs* [*simp*]:
    [|$M(x)$; $M(y)$; $M(Z)$|] ==> $is\_Member(M,x,y,Z) \longleftrightarrow (Z = Member(x,y))$
⟨*proof*⟩

lemma (**in** *M_trivial*) *Member_in_M_iff* [*iff*]:
    $M(Member(x,y)) \longleftrightarrow M(x)$ & $M(y)$
⟨*proof*⟩

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

lemma (**in** *M_trivial*) *Equal_abs* [*simp*]:
    [|$M(x)$; $M(y)$; $M(Z)$|] ==> $is\_Equal(M,x,y,Z) \longleftrightarrow (Z = Equal(x,y))$
⟨*proof*⟩

lemma (**in** *M_trivial*) *Equal_in_M_iff* [*iff*]: $M(Equal(x,y)) \longleftrightarrow M(x)$ & $M(y)$
⟨*proof*⟩

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

lemma (**in** *M_trivial*) *Nand_abs* [*simp*]:
    [|$M(x)$; $M(y)$; $M(Z)$|] ==> $is\_Nand(M,x,y,Z) \longleftrightarrow (Z = Nand(x,y))$
⟨*proof*⟩

lemma (**in** *M_trivial*) *Nand_in_M_iff* [*iff*]: $M(Nand(x,y)) \longleftrightarrow M(x)$ & $M(y)$
⟨*proof*⟩

definition
  is_Forall :: [*i*=>*o*,*i*,*i*] => *o* **where**

— because $Forall(x) \equiv Inr(Inr(p))$
$is\_Forall(M,p,Z) == \exists\, u[M].\ is\_Inr(M,p,u)\ \&\ is\_Inr(M,u,Z)$

lemma (**in** *M_trivial*) *Forall_abs* [*simp*]:
$[|M(x);\ M(Z)|] ==> is\_Forall(M,x,Z) \longleftrightarrow (Z = Forall(x))$
⟨*proof*⟩

lemma (**in** *M_trivial*) *Forall_in_M_iff* [*iff*]: $M(Forall(x)) \longleftrightarrow M(x)$
⟨*proof*⟩

5.11 Absoluteness for *formula_rec*

definition
$formula\_rec\_case :: [[i,i]=>i,\ [i,i]=>i,\ [i,i,i,i]=>i,\ [i,i]=>i,\ i,\ i] => i$ **where**
— the instance of *formula_case* in *formula_rec*
$formula\_rec\_case(a,b,c,d,h) ==$
$formula\_case\ (a,\ b,$
$\lambda u\ v.\ c(u,\ v,\ h\ `\ succ(depth(u))\ `\ u,$
$h\ `\ succ(depth(v))\ `\ v),$
$\lambda u.\ d(u,\ h\ `\ succ(depth(u))\ `\ u))$

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

lemma (**in** *M_trivial*) *formula_rec_eq*:
$p \in formula ==>$
$formula\_rec(a,b,c,d,p) =$
$transrec\ (succ(depth(p)),$
$\lambda x\ h.\ Lambda\ (formula,\ formula\_rec\_case(a,b,c,d,h)))\ `\ p$
⟨*proof*⟩

5.11.1 Absoluteness for the Formula Operator *depth*

definition
$is\_depth :: [i=>o,i,i] => o$ **where**
$is\_depth(M,p,n) ==$
$\exists\, sn[M].\ \exists\, formula\_n[M].\ \exists\, formula\_sn[M].$
$is\_formula\_N(M,n,formula\_n)\ \&\ p \notin formula\_n\ \&$
$successor(M,n,sn)\ \&\ is\_formula\_N(M,sn,formula\_sn)\ \&\ p \in formula\_sn$

Proof is trivial since *depth* returns natural numbers.

lemma (**in** *M_trivial*) *depth_closed* [*intro*,*simp*]:
$p \in formula ==> M(depth(p))$
⟨*proof*⟩

end

6 Some enhanced theorems on recursion

theory *Recursion_Thms*
imports *Eclose_Absolute*

begin

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

— Removing arities from inherited simpset
declare *arity_And* [*simp del*] *arity_Or*[*simp del*] *arity_Implies*[*simp del*] *arity_Exists*[*simp del*] *arity_Iff*[*simp del*] *arity_subset_fm* [*simp del*] *arity_ordinal_fm*[*simp del*] *arity_transset_fm*[*simp del*]

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

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

lemma *fld_restrict_mono* : $relation(r) \Longrightarrow A \subseteq B \Longrightarrow r \cap A\times A \subseteq r \cap B\times B$
⟨*proof*⟩

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

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

lemma *tr_downD* : $x \in tr\_down(r,a) \Longrightarrow \langle x,a\rangle \in r^+$
⟨*proof*⟩

lemma *pred_down* : $relation(r) \Longrightarrow r\text{-}``\{a\} \subseteq tr\_down(r,a)$
⟨*proof*⟩

lemma *tr_down_mono* : $relation(r) \Longrightarrow x \in r\text{-}``\{a\} \Longrightarrow tr\_down(r,x) \subseteq tr\_down(r,a)$
⟨*proof*⟩

lemma *rest_eq* :
assumes $relation(r)$ **and** $r\text{-}``\{a\} \subseteq B$ **and** $a \in B$
shows $r\text{-}``\{a\} = (r \cap B\times B)\text{-}``\{a\}$
⟨*proof*⟩

lemma *wfrec_restr_eq* : $r' = r \cap A\times A \Longrightarrow wfrec[A](r,a,H) = wfrec(r',a,H)$
⟨*proof*⟩

lemma *wfrec_restr* :

assumes *rr*: *relation*(r) **and** *wfr*:*wf*(r)
shows $a \in A \Longrightarrow tr\_down(r,a) \subseteq A \Longrightarrow wfrec(r,a,H) = wfrec[A](r,a,H)$
⟨*proof*⟩

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

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

lemma *field_trancl* : *field*(r^+) = *field*(r)
⟨*proof*⟩

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

lemma *RrelI* : $x \in A \Longrightarrow y \in A \Longrightarrow R(x,y) \Longrightarrow \langle x,y\rangle \in Rrel(R,A)$
⟨*proof*⟩

lemma *Rrel_mem*: *Rrel*(*mem*,x) = *Memrel*(x)
⟨*proof*⟩

lemma *relation_Rrel*: *relation*(*Rrel*(R,d))
⟨*proof*⟩

lemma *field_Rrel*: *field*(*Rrel*(R,d)) $\subseteq$ d
⟨*proof*⟩

lemma *Rrel_mono* : $A \subseteq B \Longrightarrow Rrel(R,A) \subseteq Rrel(R,B)$
⟨*proof*⟩

lemma *Rrel_restr_eq* : *Rrel*(R,A) $\cap$ $B \times B$ = *Rrel*(R,$A \cap B$)
⟨*proof*⟩
lemma *field_Memrel* : *field*(*Memrel*(A)) $\subseteq$ A
⟨*proof*⟩

lemma *restrict_trancl_Rrel*:
assumes $R(w,y)$
shows *restrict*(f,*Rrel*(R,d)-‘‘$\{y\}$)‘w
= *restrict*(f,(*Rrel*(R,d)$^+$)-‘‘$\{y\}$)‘w
⟨*proof*⟩

lemma *restrict_trans_eq*:
assumes $w \in y$
shows *restrict*(f,*Memrel*(*eclose*($\{x\}$))-‘‘$\{y\}$)‘w
= *restrict*(f,(*Memrel*(*eclose*($\{x\}$))$^+$)-‘‘$\{y\}$)‘w

⟨*proof*⟩

lemma *wf_eq_trancl*:
assumes ⋀ *f y* . *H*(*y*,*restrict*(*f*,*R*-‘‘{*y*})) = *H*(*y*,*restrict*(*f*,*R*^+-‘‘{*y*}))
shows *wfrec*(*R*, *x*, *H*) = *wfrec*(*R*^+, *x*, *H*) (**is** *wfrec*(*?r*,_,_) = *wfrec*(*?r′*,_,_))
⟨*proof*⟩

lemma *transrec_equal_on_Ord*:
assumes
⋀*x f* . *Ord*(*x*) ⟹ *foo*(*x*,*f*) = *bar*(*x*,*f*)
Ord(*α*)
shows
transrec(*α*, *foo*) = *transrec*(*α*, *bar*)
⟨*proof*⟩
lemma (**in** *M_eclose*) *transrec_equal_on_M*:
assumes
⋀*x f* . *M*(*x*) ⟹ *M*(*f*) ⟹ *foo*(*x*,*f*) = *bar*(*x*,*f*)
⋀*β*. *M*(*β*) ⟹ *transrec_replacement*(*M*,*is_foo*,*β*) *relation2*(*M*,*is_foo*,*foo*)
strong_replacement(*M*, *λx y*. *y* = ⟨*x*, *transrec*(*x*, *foo*)⟩)
∀ *x*[*M*]. ∀ *g*[*M*]. *function*(*g*) ⟶ *M*(*foo*(*x*,*g*))
M(*α*) *Ord*(*α*)
shows
transrec(*α*, *foo*) = *transrec*(*α*, *bar*)
⟨*proof*⟩

lemma *ordermap_restr_eq*:
assumes *well_ord*(*X*,*r*)
shows *ordermap*(*X*, *r*) = *ordermap*(*X*, *r* ∩ *X*×*X*)
⟨*proof*⟩

end

7 Absoluteness Properties for Recursive Datatypes

theory *Datatype_absolute* **imports** *Eclose_Absolute* **begin**

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

$M(l)$ ==> *iterates_replacement*(M, %l t. *is_tl*(M,l,t), l)

7.0.1 Absoluteness of the List Construction

lemma (**in** *M_datatypes*) *list_replacement2'*:
$M(A)$ ==> *strong_replacement*(M, λn y. $n{\in}nat$ & $y = (\lambda X.\ \{0\} + A * X)\hat{}n$ (0))
⟨*proof*⟩

lemma (**in** *M_datatypes*) *list_closed* [*intro*,*simp*]:
$M(A)$ ==> $M(list(A))$
⟨*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*]:
[|$M(A)$; $n{\in}nat$; $M(Z)$|]
==> *is_list_N*(M,A,n,Z) ⟷ Z = *list_N*(A,n)
⟨*proof*⟩

lemma (**in** *M_datatypes*) *list_N_closed* [*intro*,*simp*]:
[|$M(A)$; $n{\in}nat$|] ==> M(*list_N*(A,n))
⟨*proof*⟩

lemma (**in** *M_datatypes*) *mem_list_abs* [*simp*]:
$M(A)$ ==> *mem_list*(M,A,l) ⟷ $l \in list(A)$
⟨*proof*⟩

lemma (**in** *M_datatypes*) *list_abs* [*simp*]:
[|$M(A)$; $M(Z)$|] ==> *is_list*(M,A,Z) ⟷ Z = *list*(A)
⟨*proof*⟩

7.0.2 Absoluteness of Formulas

lemma (**in** *M_datatypes*) *formula_replacement2'*:
strong_replacement(M, λn y. $n{\in}nat$ & $y = (\lambda X.\ ((nat{*}nat) + (nat{*}nat)) + (X{*}X + X))\hat{}n$ (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*]:
[|$n{\in}nat$; $M(Z)$|]
==> *is_formula_N*(M,n,Z) ⟷ Z = *formula_N*(n)
⟨*proof*⟩

lemma (**in** *M_datatypes*) *formula_N_closed* [*intro*,*simp*]:
    $n \in nat \Longrightarrow M(formula\_N(n))$
⟨*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*]:
    $[|M(Z)|] \Longrightarrow is\_formula(M,Z) \longleftrightarrow Z = formula$
⟨*proof*⟩

lemma (**in** *M_datatypes*) *length_abs* [*simp*]:
    $[|M(A);\ l \in list(A);\ n \in nat|] \Longrightarrow is\_length(M,A,l,n) \longleftrightarrow n = length(l)$
⟨*proof*⟩

definition
  $is\_nth :: [i{=>}o,i,i,i] => o$ **where**
    $is\_nth(M,n,l,Z) ==$
      $\exists X[M].\ is\_iterates(M,\ is\_tl(M),\ l,\ n,\ X)\ \&\ is\_hd(M,X,Z)$

lemma (**in** *M_datatypes*) *nth_abs* [*simp*]:
    $[|M(A);\ n \in nat;\ l \in list(A);\ M(Z)|]$
     $\Longrightarrow is\_nth(M,n,l,Z) \longleftrightarrow Z = nth(n,l)$
⟨*proof*⟩

lemma (**in** *M_datatypes*) *depth_abs* [*simp*]:
    $[|p \in formula;\ n \in nat|] \Longrightarrow is\_depth(M,p,n) \longleftrightarrow n = depth(p)$
⟨*proof*⟩

7.0.3 *is_formula_case*: relativization of *formula_case*

definition
 $is\_formula\_case ::$
    $[i{=>}o,\ [i,i,i]{=>}o,\ [i,i,i]{=>}o,\ [i,i,i]{=>}o,\ [i,i]{=>}o,\ i,\ i] => o$ **where**
  — no constraint on non-formulas
  $is\_formula\_case(M,\ is\_a,\ is\_b,\ is\_c,\ is\_d,\ p,\ z) ==$
      $(\forall x[M].\ \forall y[M].\ finite\_ordinal(M,x) \longrightarrow finite\_ordinal(M,y) \longrightarrow$
                $is\_Member(M,x,y,p) \longrightarrow is\_a(x,y,z))\ \&$
      $(\forall x[M].\ \forall y[M].\ finite\_ordinal(M,x) \longrightarrow finite\_ordinal(M,y) \longrightarrow$
                $is\_Equal(M,x,y,p) \longrightarrow is\_b(x,y,z))\ \&$
      $(\forall x[M].\ \forall y[M].\ mem\_formula(M,x) \longrightarrow mem\_formula(M,y) \longrightarrow$

$$is\_Nand(M,x,y,p) \longrightarrow is\_c(x,y,z)) \;\&$$
$$(\forall\, x[M].\ mem\_formula(M,x) \longrightarrow is\_Forall(M,x,p) \longrightarrow is\_d(x,z))$$

lemma (**in** *M_datatypes*) *formula_case_abs* [*simp*]:
$[|\ Relation2(M,nat,nat,is\_a,a);\ Relation2(M,nat,nat,is\_b,b);$
$Relation2(M,formula,formula,is\_c,c);\ Relation1(M,formula,is\_d,d);$
$p \in formula;\ M(z)\ |]$
$\Longrightarrow is\_formula\_case(M,is\_a,is\_b,is\_c,is\_d,p,z) \longleftrightarrow$
$z = formula\_case(a,b,c,d,p)$
⟨*proof*⟩

lemma (**in** *M_datatypes*) *formula_case_closed* [*intro*,*simp*]:
$[|p \in formula;$
$\forall\, x[M].\ \forall\, y[M].\ x{\in}nat \longrightarrow y{\in}nat \longrightarrow M(a(x,y));$
$\forall\, x[M].\ \forall\, y[M].\ x{\in}nat \longrightarrow y{\in}nat \longrightarrow M(b(x,y));$
$\forall\, x[M].\ \forall\, y[M].\ x{\in}formula \longrightarrow y{\in}formula \longrightarrow M(c(x,y));$
$\forall\, x[M].\ x{\in}formula \longrightarrow M(d(x))|] \Longrightarrow M(formula\_case(a,b,c,d,p))$
⟨*proof*⟩

7.0.4 Absoluteness for *formula_rec*: Final Results

definition
$is\_formula\_rec :: [i{=>}o,\ [i,i,i]{=>}o,\ i,\ i] => o$ **where**
— predicate to relativize the functional *formula_rec*
$is\_formula\_rec(M,MH,p,z) \;==$
$\exists\, dp[M].\ \exists\, i[M].\ \exists\, f[M].\ finite\_ordinal(M,dp)\ \&\ is\_depth(M,p,dp)\ \&$
$successor(M,dp,i)\ \&\ fun\_apply(M,f,p,z)\ \&\ is\_transrec(M,MH,i,f)$

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

lemma (**in** *M_datatypes*) *Relation1_formula_rec_case*:
$[|Relation2(M,\ nat,\ nat,\ is\_a,\ a);$
$Relation2(M,\ nat,\ nat,\ is\_b,\ b);$
$Relation2\ (M,\ formula,\ formula,$
$is\_c,\ \lambda u\ v.\ c(u,\ v,\ h`succ(depth(u))`u,\ h`succ(depth(v))`v));$
$Relation1(M,\ formula,$
$is\_d,\ \lambda u.\ d(u,\ h\ `\ succ(depth(u))\ `\ u));$
$M(h)\ |]$
$\Longrightarrow Relation1(M,\ formula,$
$is\_formula\_case\ (M,\ is\_a,\ is\_b,\ is\_c,\ is\_d),$
$formula\_rec\_case(a,\ b,\ c,\ d,\ h))$
⟨*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* + *M_datatypes* +
fixes *a* **and** *is_a* **and** *b* **and** *is_b* **and** *c* **and** *is_c* **and** *d* **and** *is_d* **and** *MH*
defines
$MH(u::i,f,z) ==$

$\forall fml[M].\ is\_formula(M,fml) \longrightarrow$
is_lambda
$(M,\ fml,\ is\_formula\_case\ (M,\ is\_a,\ is\_b,\ is\_c(f),\ is\_d(f)),\ z)$

assumes *a_closed*: $[|x{\in}nat;\ y{\in}nat|] ==> M(a(x,y))$
and *a_rel*: $Relation2(M,\ nat,\ nat,\ is\_a,\ a)$
and *b_closed*: $[|x{\in}nat;\ y{\in}nat|] ==> M(b(x,y))$
and *b_rel*: $Relation2(M,\ nat,\ nat,\ is\_b,\ b)$
and *c_closed*: $[|x \in formula;\ y \in formula;\ M(gx);\ M(gy)|]$
$==> M(c(x,\ y,\ gx,\ gy))$
and *c_rel*:
$M(f) ==>$
$Relation2\ (M,\ formula,\ formula,\ is\_c(f),$
$\lambda u\ v.\ c(u,\ v,\ f\ `\ succ(depth(u))\ `\ u,\ f\ `\ succ(depth(v))\ `\ v))$
and *d_closed*: $[|x \in formula;\ M(gx)|] ==> M(d(x,\ gx))$
and *d_rel*:
$M(f) ==>$
$Relation1(M,\ formula,\ is\_d(f),\ \lambda u.\ d(u,\ f\ `\ succ(depth(u))\ `\ u))$
and *fr_replace*: $n \in nat ==> transrec\_replacement(M,MH,n)$
and *fr_lam_replace*:
$M(g) ==>$
$strong\_replacement$
$(M,\ \lambda x\ y.\ x \in formula\ \&$
$y = \langle x,\ formula\_rec\_case(a,b,c,d,g,x)\rangle)$

lemma (**in** *Formula_Rec*) *formula_rec_case_closed*:
$[|M(g);\ p \in formula|] ==> M(formula\_rec\_case(a,\ b,\ c,\ d,\ g,\ p))$
⟨*proof*⟩

lemma (**in** *Formula_Rec*) *formula_rec_lam_closed*:
$M(g) ==> M(Lambda\ (formula,\ formula\_rec\_case(a,b,c,d,g)))$
⟨*proof*⟩

lemma (**in** *Formula_Rec*) *MH_rel2*:
$relation2\ (M,\ MH,$
$\lambda x\ h.\ Lambda\ (formula,\ formula\_rec\_case(a,b,c,d,h)))$
⟨*proof*⟩

lemma (**in** *Formula_Rec*) *fr_transrec_closed*:
$n \in nat$
$==> M(transrec$
$(n,\ \lambda x\ h.\ Lambda(formula,\ formula\_rec\_case(a,\ b,\ c,\ d,\ h))))$
⟨*proof*⟩

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

theorem (**in** *Formula_Rec*) *formula_rec_closed*:
$p \in formula ==> M(formula\_rec(a,b,c,d,p))$
⟨*proof*⟩

theorem (**in** *Formula_Rec*) *formula_rec_abs*:
[| $p \in formula$; $M(z)$|]
==> $is\_formula\_rec(M,MH,p,z) \longleftrightarrow z = formula\_rec(a,b,c,d,p)$
⟨*proof*⟩

end

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

7.1 Internalized Forms of Data Structuring Operators

7.1.1 The Formula *is_Inl*, Internalized

definition
$Inl\_fm :: [i,i]=>i$ **where**
$Inl\_fm(a,z) == Exists(And(empty\_fm(0), pair\_fm(0,succ(a),succ(z))))$

lemma *Inl_type* [*TC*]:
[| $x \in nat$; $z \in nat$ |] ==> $Inl\_fm(x,z) \in formula$
⟨*proof*⟩

lemma *sats_Inl_fm* [*simp*]:
[| $x \in nat$; $z \in nat$; $env \in list(A)$|]
==> $sats(A, Inl\_fm(x,z), env) \longleftrightarrow is\_Inl(##A, nth(x,env), nth(z,env))$
⟨*proof*⟩

lemma *Inl_iff_sats*:
[| $nth(i,env) = x$; $nth(k,env) = z$;
$i \in nat$; $k \in nat$; $env \in list(A)$|]
==> $is\_Inl(##A, x, z) \longleftrightarrow sats(A, Inl\_fm(i,k), env)$
⟨*proof*⟩

theorem *Inl_reflection*:
$REFLECTS[\lambda x.\ is\_Inl(L,f(x),h(x)),$
$\lambda i\ x.\ is\_Inl(##Lset(i),f(x),h(x))]$
⟨*proof*⟩

7.1.2 The Formula *is_Inr*, Internalized

definition
$Inr\_fm :: [i,i]=>i$ **where**
$Inr\_fm(a,z) == Exists(And(number1\_fm(0), pair\_fm(0,succ(a),succ(z))))$

lemma *Inr_type* [*TC*]:
[| $x \in nat$; $z \in nat$ |] ==> $Inr\_fm(x,z) \in formula$
⟨*proof*⟩

lemma *sats_Inr_fm* [*simp*]:

[| $x \in nat$; $z \in nat$; $env \in list(A)$|]
==> $sats(A, Inr\_fm(x,z), env) \longleftrightarrow is\_Inr(\#\#A, nth(x,env), nth(z,env))$
⟨*proof*⟩

lemma *Inr_iff_sats*:
[| $nth(i,env) = x$; $nth(k,env) = z$;
$i \in nat$; $k \in nat$; $env \in list(A)$|]
==> $is\_Inr(\#\#A, x, z) \longleftrightarrow sats(A, Inr\_fm(i,k), env)$
⟨*proof*⟩

theorem *Inr_reflection*:
$REFLECTS[\lambda x.\ is\_Inr(L,f(x),h(x)),$
$\lambda i\ x.\ is\_Inr(\#\#Lset(i),f(x),h(x))]$
⟨*proof*⟩

7.1.3 The Formula *is_Nil*, Internalized

definition
$Nil\_fm :: i=>i$ **where**
$Nil\_fm(x) == Exists(And(empty\_fm(0), Inl\_fm(0,succ(x))))$

lemma *Nil_type* [*TC*]: $x \in nat$ ==> $Nil\_fm(x) \in formula$
⟨*proof*⟩

lemma *sats_Nil_fm* [*simp*]:
[| $x \in nat$; $env \in list(A)$|]
==> $sats(A, Nil\_fm(x), env) \longleftrightarrow is\_Nil(\#\#A, nth(x,env))$
⟨*proof*⟩

lemma *Nil_iff_sats*:
[| $nth(i,env) = x$; $i \in nat$; $env \in list(A)$|]
==> $is\_Nil(\#\#A, x) \longleftrightarrow sats(A, Nil\_fm(i), env)$
⟨*proof*⟩

theorem *Nil_reflection*:
$REFLECTS[\lambda x.\ is\_Nil(L,f(x)),$
$\lambda i\ x.\ is\_Nil(\#\#Lset(i),f(x))]$
⟨*proof*⟩

7.1.4 The Formula *is_Cons*, Internalized

definition
$Cons\_fm :: [i,i,i]=>i$ **where**
$Cons\_fm(a,l,Z) ==$
$Exists(And(pair\_fm(succ(a),succ(l),0), Inr\_fm(0,succ(Z))))$

lemma *Cons_type* [*TC*]:
[| $x \in nat$; $y \in nat$; $z \in nat$ |] ==> $Cons\_fm(x,y,z) \in formula$
⟨*proof*⟩

lemma *sats_Cons_fm* [*simp*]:
[| $x \in nat$; $y \in nat$; $z \in nat$; $env \in list(A)$|]
==> $sats(A,\ Cons\_fm(x,y,z),\ env) \longleftrightarrow$
$is\_Cons(\#\#A,\ nth(x,env),\ nth(y,env),\ nth(z,env))$
⟨*proof*⟩

lemma *Cons_iff_sats*:
[| $nth(i,env) = x$; $nth(j,env) = y$; $nth(k,env) = z$;
$i \in nat$; $j \in nat$; $k \in nat$; $env \in list(A)$|]
==>$is\_Cons(\#\#A,\ x,\ y,\ z) \longleftrightarrow sats(A,\ Cons\_fm(i,j,k),\ env)$
⟨*proof*⟩

theorem *Cons_reflection*:
$REFLECTS[\lambda x.\ is\_Cons(L,f(x),g(x),h(x)),$
$\lambda i\ x.\ is\_Cons(\#\#Lset(i),f(x),g(x),h(x))]$
⟨*proof*⟩

7.1.5 The Formula *is_quasilist*, Internalized

definition
quasilist_fm :: i=>i **where**
$quasilist\_fm(x) ==$
$Or(Nil\_fm(x),\ Exists(Exists(Cons\_fm(1,0,succ(succ(x))))))$

lemma *quasilist_type* [*TC*]: $x \in nat$ ==> $quasilist\_fm(x) \in formula$
⟨*proof*⟩

lemma *sats_quasilist_fm* [*simp*]:
[| $x \in nat$; $env \in list(A)$|]
==> $sats(A,\ quasilist\_fm(x),\ env) \longleftrightarrow is\_quasilist(\#\#A,\ nth(x,env))$
⟨*proof*⟩

lemma *quasilist_iff_sats*:
[| $nth(i,env) = x$; $i \in nat$; $env \in list(A)$|]
==> $is\_quasilist(\#\#A,\ x) \longleftrightarrow sats(A,\ quasilist\_fm(i),\ env)$
⟨*proof*⟩

theorem *quasilist_reflection*:
$REFLECTS[\lambda x.\ is\_quasilist(L,f(x)),$
$\lambda i\ x.\ is\_quasilist(\#\#Lset(i),f(x))]$
⟨*proof*⟩

7.2 Absoluteness for the Function *nth*

7.2.1 The Formula *is_hd*, Internalized

definition
hd_fm :: [i,i]=>i **where**
$hd\_fm(xs,H) ==$
$And(Implies(Nil\_fm(xs),\ empty\_fm(H)),$

```
        And(Forall(Forall(Or(Neg(Cons_fm(1,0,xs#+2)), Equal(H#+2,1)))),
            Or(quasilist_fm(xs), empty_fm(H))))
```

lemma *hd_type* [*TC*]:
```
    [| x ∈ nat; y ∈ nat |] ==> hd_fm(x,y) ∈ formula
```
⟨*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*⟩

7.2.2 The Formula *is_tl*, Internalized

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

lemma *tl_type* [*TC*]:
```
    [| x ∈ nat; y ∈ nat |] ==> tl_fm(x,y) ∈ formula
```
⟨*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)),
```

$\lambda i\ x.\ is\_tl(\#\#Lset(i),f(x),g(x))]$
⟨*proof*⟩

7.2.3 The Operator *is_bool_of_o*

The formula p has no free variables.

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

lemma *is_bool_of_o_type* [*TC*]:
[| $p \in formula$; $z \in nat$ |] ==> $bool\_of\_o\_fm(p,z) \in formula$
⟨*proof*⟩

lemma *sats_bool_of_o_fm*:
assumes *p_iff_sats*: $P \longleftrightarrow sats(A, p, env)$
shows
[|$z \in nat$; $env \in list(A)$|]
==> $sats(A, bool\_of\_o\_fm(p,z), env) \longleftrightarrow$
$is\_bool\_of\_o(\#\#A, P, nth(z,env))$
⟨*proof*⟩

lemma *is_bool_of_o_iff_sats*:
[| $P \longleftrightarrow sats(A, p, env)$; $nth(k,env) = z$; $k \in nat$; $env \in list(A)$|]
==> $is\_bool\_of\_o(\#\#A, P, z) \longleftrightarrow sats(A, bool\_of\_o\_fm(p,k), env)$
⟨*proof*⟩

theorem *bool_of_o_reflection*:
$REFLECTS\ [P(L), \lambda i.\ P(\#\#Lset(i))]$ ==>
$REFLECTS[\lambda x.\ is\_bool\_of\_o(L, P(L,x), f(x)),$
$\lambda i\ x.\ is\_bool\_of\_o(\#\#Lset(i), P(\#\#Lset(i),x), f(x))]$
⟨*proof*⟩

7.3 More Internalizations

7.3.1 The Operator *is_lambda*

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

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

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

lemma *is_lambda_type* [*TC*]:
$[|\ p \in formula;\ x \in nat;\ y \in nat\ |]$
$==>\ lambda\_fm(p,x,y) \in formula$
⟨*proof*⟩

lemma *sats_lambda_fm*:
assumes *is_b_iff_sats*:
$!!a0\ a1\ a2.$
$[|a0{\in}A;\ a1{\in}A;\ a2{\in}A|]$
$==>\ is\_b(a1,\ a0) \longleftrightarrow sats(A,\ p,\ Cons(a0,Cons(a1,Cons(a2,env))))$
shows
$[|x \in nat;\ y \in nat;\ env \in list(A)|]$
$==>\ sats(A,\ lambda\_fm(p,x,y),\ env) \longleftrightarrow$
$is\_lambda(\#\#A,\ nth(x,env),\ is\_b,\ nth(y,env))$
⟨*proof*⟩

theorem *is_lambda_reflection*:
assumes *is_b_reflection*:
$!!f\ g\ h.\ REFLECTS[\lambda x.\ is\_b(L,\ f(x),\ g(x),\ h(x)),$
$\lambda i\ x.\ is\_b(\#\#Lset(i),\ f(x),\ g(x),\ h(x))]$
shows $REFLECTS[\lambda x.\ is\_lambda(L,\ A(x),\ is\_b(L,x),\ f(x)),$
$\lambda i\ x.\ is\_lambda(\#\#Lset(i),\ A(x),\ is\_b(\#\#Lset(i),x),\ f(x))]$
⟨*proof*⟩

7.3.2 The Operator *is_Member*, Internalized

definition
$Member\_fm :: [i,i,i]{=>}i$ **where**
$Member\_fm(x,y,Z) ==$
$Exists(Exists(And(pair\_fm(x\#{+}2,y\#{+}2,1),$
$And(Inl\_fm(1,0),\ Inl\_fm(0,Z\#{+}2)))))$

lemma *is_Member_type* [*TC*]:
$[|\ x \in nat;\ y \in nat;\ z \in nat\ |] ==>\ Member\_fm(x,y,z) \in formula$
⟨*proof*⟩

lemma *sats_Member_fm* [*simp*]:
$[|\ x \in nat;\ y \in nat;\ z \in nat;\ env \in list(A)|]$
$==>\ sats(A,\ Member\_fm(x,y,z),\ env) \longleftrightarrow$
$is\_Member(\#\#A,\ nth(x,env),\ nth(y,env),\ nth(z,env))$
⟨*proof*⟩

lemma *Member_iff_sats*:
$[|\ nth(i,env) = x;\ nth(j,env) = y;\ nth(k,env) = z;$
$i \in nat;\ j \in nat;\ k \in nat;\ env \in list(A)|]$
$==>\ is\_Member(\#\#A,\ x,\ y,\ z) \longleftrightarrow sats(A,\ Member\_fm(i,j,k),\ env)$
⟨*proof*⟩

theorem *Member_reflection*:
REFLECTS[$\lambda x.$ *is_Member*(*L*,*f*(*x*),*g*(*x*),*h*(*x*)),
$\lambda i\ x.$ *is_Member*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*))]
⟨*proof*⟩

7.3.3 The Operator *is_Equal*, Internalized

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

lemma *is_Equal_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *Equal_fm*(*x*,*y*,*z*) ∈ *formula*
⟨*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*)|]
==> *is_Equal*(##*A*, *x*, *y*, *z*) ⟷ *sats*(*A*, *Equal_fm*(*i*,*j*,*k*), *env*)
⟨*proof*⟩

theorem *Equal_reflection*:
REFLECTS[$\lambda x.$ *is_Equal*(*L*,*f*(*x*),*g*(*x*),*h*(*x*)),
$\lambda i\ x.$ *is_Equal*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*))]
⟨*proof*⟩

7.3.4 The Operator *is_Nand*, Internalized

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

lemma *is_Nand_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *Nand_fm*(*x*,*y*,*z*) ∈ *formula*
⟨*proof*⟩

lemma *sats_Nand_fm* [*simp*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *Nand_fm*(*x*,*y*,*z*), *env*) ⟷

$is\_Nand(\#\#A,\ nth(x,env),\ nth(y,env),\ nth(z,env))$
⟨*proof*⟩

lemma *Nand_iff_sats*:
$[|\ nth(i,env) = x;\ nth(j,env) = y;\ nth(k,env) = z;$
$i \in nat;\ j \in nat;\ k \in nat;\ env \in list(A)|]$
$==>\ is\_Nand(\#\#A,\ x,\ y,\ z) \longleftrightarrow sats(A,\ Nand\_fm(i,j,k),\ env)$
⟨*proof*⟩

theorem *Nand_reflection*:
$REFLECTS[\lambda x.\ is\_Nand(L,f(x),g(x),h(x)),$
$\lambda i\ x.\ is\_Nand(\#\#Lset(i),f(x),g(x),h(x))]$
⟨*proof*⟩

7.3.5 The Operator *is_Forall*, Internalized

definition
$Forall\_fm :: [i,i]=>i$ **where**
$Forall\_fm(x,Z) ==$
$Exists(And(Inr\_fm(succ(x),0),\ Inr\_fm(0,succ(Z))))$

lemma *is_Forall_type* [*TC*]:
$[|\ x \in nat;\ y \in nat\ |] ==>\ Forall\_fm(x,y) \in formula$
⟨*proof*⟩

lemma *sats_Forall_fm* [*simp*]:
$[|\ x \in nat;\ y \in nat;\ env \in list(A)|]$
$==>\ sats(A,\ Forall\_fm(x,y),\ env) \longleftrightarrow$
$is\_Forall(\#\#A,\ nth(x,env),\ nth(y,env))$
⟨*proof*⟩

lemma *Forall_iff_sats*:
$[|\ nth(i,env) = x;\ nth(j,env) = y;$
$i \in nat;\ j \in nat;\ env \in list(A)|]$
$==>\ is\_Forall(\#\#A,\ x,\ y) \longleftrightarrow sats(A,\ Forall\_fm(i,j),\ env)$
⟨*proof*⟩

theorem *Forall_reflection*:
$REFLECTS[\lambda x.\ is\_Forall(L,f(x),g(x)),$
$\lambda i\ x.\ is\_Forall(\#\#Lset(i),f(x),g(x))]$
⟨*proof*⟩

7.3.6 The Operator *is_and*, Internalized

definition
$and\_fm :: [i,i,i]=>i$ **where**
$and\_fm(a,b,z) ==$
$Or(And(number1\_fm(a),\ Equal(z,b)),$
$And(Neg(number1\_fm(a)),empty\_fm(z)))$

lemma *is_and_type* [*TC*]:
[| $x \in nat$; $y \in nat$; $z \in nat$ |] ==> $and\_fm(x,y,z) \in formula$
⟨*proof*⟩

lemma *sats_and_fm* [*simp*]:
[| $x \in nat$; $y \in nat$; $z \in nat$; $env \in list(A)$|]
==> $sats(A, and\_fm(x,y,z), env) \longleftrightarrow$
$is\_and(\#\#A, nth(x,env), nth(y,env), nth(z,env))$
⟨*proof*⟩

lemma *is_and_iff_sats*:
[| $nth(i,env) = x$; $nth(j,env) = y$; $nth(k,env) = z$;
$i \in nat$; $j \in nat$; $k \in nat$; $env \in list(A)$|]
==> $is\_and(\#\#A, x, y, z) \longleftrightarrow sats(A, and\_fm(i,j,k), env)$
⟨*proof*⟩

theorem *is_and_reflection*:
$REFLECTS[\lambda x.\ is\_and(L,f(x),g(x),h(x)),$
$\lambda i\ x.\ is\_and(\#\#Lset(i),f(x),g(x),h(x))]$
⟨*proof*⟩

7.3.7 The Operator *is_or*, Internalized

definition
or_fm :: [*i*,*i*,*i*]=>*i* **where**
$or\_fm(a,b,z) ==$
$Or(And(number1\_fm(a), number1\_fm(z)),$
$And(Neg(number1\_fm(a)), Equal(z,b)))$

lemma *is_or_type* [*TC*]:
[| $x \in nat$; $y \in nat$; $z \in nat$ |] ==> $or\_fm(x,y,z) \in formula$
⟨*proof*⟩

lemma *sats_or_fm* [*simp*]:
[| $x \in nat$; $y \in nat$; $z \in nat$; $env \in list(A)$|]
==> $sats(A, or\_fm(x,y,z), env) \longleftrightarrow$
$is\_or(\#\#A, nth(x,env), nth(y,env), nth(z,env))$
⟨*proof*⟩

lemma *is_or_iff_sats*:
[| $nth(i,env) = x$; $nth(j,env) = y$; $nth(k,env) = z$;
$i \in nat$; $j \in nat$; $k \in nat$; $env \in list(A)$|]
==> $is\_or(\#\#A, x, y, z) \longleftrightarrow sats(A, or\_fm(i,j,k), env)$
⟨*proof*⟩

theorem *is_or_reflection*:
$REFLECTS[\lambda x.\ is\_or(L,f(x),g(x),h(x)),$
$\lambda i\ x.\ is\_or(\#\#Lset(i),f(x),g(x),h(x))]$
⟨*proof*⟩

7.3.8 The Operator *is_not*, Internalized

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

lemma *is_not_type* [*TC*]:
[| *x* ∈ *nat*; *z* ∈ *nat* |] ==> *not_fm*(*x*,*z*) ∈ *formula*
⟨*proof*⟩

lemma *sats_is_not_fm* [*simp*]:
[| *x* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *not_fm*(*x*,*z*), *env*) ⟷ *is_not*(##*A*, *nth*(*x*,*env*), *nth*(*z*,*env*))
⟨*proof*⟩

lemma *is_not_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*k*,*env*) = *z*;
i ∈ *nat*; *k* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_not*(##*A*, *x*, *z*) ⟷ *sats*(*A*, *not_fm*(*i*,*k*), *env*)
⟨*proof*⟩

theorem *is_not_reflection*:
REFLECTS[λ*x*. *is_not*(*L*,*f*(*x*),*g*(*x*)),
λ*i* *x*. *is_not*(##*Lset*(*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

7.4 Well-Founded Recursion!

7.4.1 The Operator *M_is_recfun*

Alternative definition, minimizing nesting of quantifiers around MH

lemma *M_is_recfun_iff*:
M_is_recfun(*M*,*MH*,*r*,*a*,*f*) ⟷
(∀ *z*[*M*]. *z* ∈ *f* ⟷
(∃ *x*[*M*]. ∃ *f_r_sx*[*M*]. ∃ *y*[*M*].
MH(*x*, *f_r_sx*, *y*) & *pair*(*M*,*x*,*y*,*z*) &
(∃ *xa*[*M*]. ∃ *sx*[*M*]. ∃ *r_sx*[*M*].
pair(*M*,*x*,*a*,*xa*) & *upair*(*M*,*x*,*x*,*sx*) &
pre_image(*M*,*r*,*sx*,*r_sx*) & *restriction*(*M*,*f*,*r_sx*,*f_r_sx*) &
xa ∈ *r*)))
⟨*proof*⟩

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

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

lemma *is_recfun_type* [*TC*]:
[| *p* ∈ *formula*; *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |]
==> *is_recfun_fm*(*p*,*x*,*y*,*z*) ∈ *formula*
⟨*proof*⟩

lemma *sats_is_recfun_fm*:
assumes *MH_iff_sats*:
!!*a0 a1 a2 a3*.
[|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*; *a3*∈*A*|]
==> *MH*(*a2*, *a1*, *a0*) ⟷ *sats*(*A*, *p*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*Cons*(*a3*,*env*))))))
shows
[|*x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *is_recfun_fm*(*p*,*x*,*y*,*z*), *env*) ⟷
M_is_recfun(##*A*, *MH*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
⟨*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*[λ*x*. *MH*(*L*, f'(*x*), *f*(*x*), *g*(*x*), *h*(*x*)),
λ*i x*. *MH*(##*Lset*(*i*), f'(*x*), *f*(*x*), *g*(*x*), *h*(*x*))]

shows *REFLECTS*[$\lambda x.$ *M_is_recfun*(*L*, *MH*(*L*,*x*), *f*(*x*), *g*(*x*), *h*(*x*)),
$\lambda i\ x.$ *M_is_recfun*(##*Lset*(*i*), *MH*(##*Lset*(*i*),*x*), *f*(*x*), *g*(*x*), *h*(*x*))]
⟨*proof*⟩

7.4.2 The Operator *is_wfrec*

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

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

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

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

lemma *is_wfrec_type* [*TC*]:
[| *p* ∈ *formula*; *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |]
==> *is_wfrec_fm*(*p*,*x*,*y*,*z*) ∈ *formula*
⟨*proof*⟩

lemma *sats_is_wfrec_fm*:
assumes *MH_iff_sats*:
!!*a0 a1 a2 a3 a4*.
[|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*; *a3*∈*A*; *a4*∈*A*|]
==> *MH*(*a2*, *a1*, *a0*) ⟷ *sats*(*A*, *p*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*Cons*(*a3*,*Cons*(*a4*,*env*))))))
shows
[|*x* ∈ *nat*; *y* < *length*(*env*); *z* < *length*(*env*); *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *is_wfrec_fm*(*p*,*x*,*y*,*z*), *env*) ⟷
is_wfrec(##*A*, *MH*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
⟨*proof*⟩

lemma *is_wfrec_iff_sats*:
assumes *MH_iff_sats*:
!!*a0 a1 a2 a3 a4*.
[|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*; *a3*∈*A*; *a4*∈*A*|]
==> *MH*(*a2*, *a1*, *a0*) ⟷ *sats*(*A*, *p*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*Cons*(*a3*,*Cons*(*a4*,*env*))))))
shows
[|*nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
i ∈ *nat*; *j* < *length*(*env*); *k* < *length*(*env*); *env* ∈ *list*(*A*)|]
==> *is_wfrec*(##*A*, *MH*, *x*, *y*, *z*) ⟷ *sats*(*A*, *is_wfrec_fm*(*p*,*i*,*j*,*k*), *env*)
⟨*proof*⟩

theorem *is_wfrec_reflection*:
assumes *MH_reflection*:

!!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*⟩

7.5 For Datatypes

7.5.1 Binary Products, Internalized

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

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

lemma *cartprod_type* [*TC*]:
[| $x \in nat$; $y \in nat$; $z \in nat$ |] ==> *cartprod_fm*(*x*,*y*,*z*) ∈ *formula*
⟨*proof*⟩

lemma *sats_cartprod_fm* [*simp*]:
[| $x \in nat$; $y \in nat$; $z \in nat$; $env \in list(A)$|]
==> *sats*(*A*, *cartprod_fm*(*x*,*y*,*z*), *env*) ⟷
cartprod(*##A*, *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
⟨*proof*⟩

lemma *cartprod_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
$i \in nat$; $j \in nat$; $k \in nat$; $env \in list(A)$|]
==> *cartprod*(*##A*, *x*, *y*, *z*) ⟷ *sats*(*A*, *cartprod_fm*(*i*,*j*,*k*), *env*)
⟨*proof*⟩

theorem *cartprod_reflection*:
REFLECTS[λ*x*. *cartprod*(*L*,*f*(*x*),*g*(*x*),*h*(*x*)),
λ*i* *x*. *cartprod*(*##Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*))]
⟨*proof*⟩

7.5.2 Binary Sums, Internalized

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

lemma *sum_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *sum_fm*(*x*,*y*,*z*) ∈ *formula*
⟨*proof*⟩

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

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

theorem *sum_reflection*:
REFLECTS[λ*x*. *is_sum*(*L*,*f*(*x*),*g*(*x*),*h*(*x*)),
λ*i* *x*. *is_sum*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*))]
⟨*proof*⟩

7.5.3 The Operator *quasinat*

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

lemma *quasinat_type* [*TC*]:
x ∈ *nat* ==> *quasinat_fm*(*x*) ∈ *formula*
⟨*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[λ*x*. *is_quasinat*(*L*,*f*(*x*)),
λ*i* *x*. *is_quasinat*(##*Lset*(*i*),*f*(*x*))]
⟨*proof*⟩

7.5.4 The Operator *is_nat_case*

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

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

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

lemma *is_nat_case_type* [*TC*]:
[| *is_b* ∈ *formula*;
x ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |]
==> *is_nat_case_fm*(*x*,*is_b*,*y*,*z*) ∈ *formula*
⟨*proof*⟩

lemma *sats_is_nat_case_fm*:
assumes *is_b_iff_sats*:
!!*a*. *a* ∈ *A* ==> *is_b*(*a*,*nth*(*z*, *env*)) ⟷
sats(*A*, *p*, *Cons*(*nth*(*z*,*env*), *Cons*(*a*, *env*)))
shows
[|*x* ∈ *nat*; *y* ∈ *nat*; *z* < *length*(*env*); *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *is_nat_case_fm*(*x*,*p*,*y*,*z*), *env*) ⟷
is_nat_case(##*A*, *nth*(*x*,*env*), *is_b*, *nth*(*y*,*env*), *nth*(*z*,*env*))
⟨*proof*⟩

lemma *is_nat_case_iff_sats*:
[| (!!*a*. *a* ∈ *A* ==> *is_b*(*a*,*z*) ⟷
sats(*A*, *p*, *Cons*(*z*, *Cons*(*a*,*env*))));
nth(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*; *nth*(*k*,*env*) = *z*;
i ∈ *nat*; *j* ∈ *nat*; *k* < *length*(*env*); *env* ∈ *list*(*A*)|]
==> *is_nat_case*(##*A*, *x*, *is_b*, *y*, *z*) ⟷ *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*:
!!*h f g*. *REFLECTS*[λ*x*. *is_b*(*L*, *h*(*x*), *f*(*x*), *g*(*x*)),
λ*i x*. *is_b*(##*Lset*(*i*), *h*(*x*), *f*(*x*), *g*(*x*))]
shows *REFLECTS*[λ*x*. *is_nat_case*(*L*, *f*(*x*), *is_b*(*L*,*x*), *g*(*x*), *h*(*x*)),
λ*i x*. *is_nat_case*(##*Lset*(*i*), *f*(*x*), *is_b*(##*Lset*(*i*), *x*), *g*(*x*), *h*(*x*))]

⟨*proof*⟩

7.6 The Operator *iterates_MH*, Needed for Iteration

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

lemma *iterates_MH_type* [*TC*]:
[| *p* ∈ *formula*;
v ∈ *nat*; *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |]
==> *iterates_MH_fm*(*p*,*v*,*x*,*y*,*z*) ∈ *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*[λ*x*. *p*(*L*, *h*(*x*), *f*(*x*), *g*(*x*)),
λ*i x*. *p*(##*Lset*(*i*), *h*(*x*), *f*(*x*), *g*(*x*))]
shows *REFLECTS*[λ*x*. *iterates_MH*(*L*, *p*(*L*,*x*), *e*(*x*), *f*(*x*), *g*(*x*), *h*(*x*)),

$\lambda i\ x.\ iterates\_MH(\#\#Lset(i),\ p(\#\#Lset(i),x),\ e(x),\ f(x),\ g(x),\ h(x))]$

⟨*proof*⟩

7.6.1 The Operator *is_iterates*

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

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

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

lemma *is_iterates_type* [*TC*]:
[| $p \in formula$; $x \in nat$; $y \in nat$; $z \in nat$ |]
==> $is\_iterates\_fm(p,x,y,z) \in formula$

⟨*proof*⟩

lemma *sats_is_iterates_fm*:
assumes *is_F_iff_sats*:
!!a b c d e f g h i j k.
[| $a \in A$; $b \in A$; $c \in A$; $d \in A$; $e \in A$; $f \in A$;
$g \in A$; $h \in A$; $i \in A$; $j \in A$; $k \in A$|]
==> $is\_F(a,b) \longleftrightarrow$
$sats(A,\ p,\ Cons(b,\ Cons(a,\ Cons(c,\ Cons(d,\ Cons(e,\ Cons(f,$
$Cons(g,\ Cons(h,\ Cons(i,\ Cons(j,\ Cons(k,\ env))))))))))))$
shows
[|$x \in nat$; $y < length(env)$; $z < length(env)$; $env \in list(A)$|]
==> $sats(A,\ is\_iterates\_fm(p,x,y,z),\ env) \longleftrightarrow$
$is\_iterates(\#\#A,\ is\_F,\ nth(x,env),\ nth(y,env),\ nth(z,env))$

⟨*proof*⟩

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

$==>$ *is_iterates*(*##A*, *is_F*, *x*, *y*, *z*) $\longleftrightarrow$
sats(*A*, *is_iterates_fm*(*p*,*i*,*j*,*k*), *env*)
⟨*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 *is_iterates_reflection*:
assumes *p_reflection*:
!!*f g h*. *REFLECTS*[λx. *p*(*L*, *h*(*x*), *f*(*x*), *g*(*x*)),
$\lambda i\ x$. *p*(*##Lset*(*i*), *h*(*x*), *f*(*x*), *g*(*x*))]
shows *REFLECTS*[λx. *is_iterates*(*L*, *p*(*L*,*x*), *f*(*x*), *g*(*x*), *h*(*x*)),
$\lambda i\ x$. *is_iterates*(*##Lset*(*i*), *p*(*##Lset*(*i*),*x*), *f*(*x*), *g*(*x*), *h*(*x*))]
⟨*proof*⟩

7.6.2 The Formula *is_eclose_n*, Internalized

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

lemma *eclose_n_fm_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *eclose_n_fm*(*x*,*y*,*z*) ∈ *formula*
⟨*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*) $\longleftrightarrow$
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*) $\longleftrightarrow$ *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*)),
$\lambda i\ x$. *is_eclose_n*(*##Lset*(*i*), *f*(*x*), *g*(*x*), *h*(*x*))]
⟨*proof*⟩

7.6.3 Membership in *eclose*(*A*)

definition
mem_eclose_fm :: [*i*,*i*]=>*i* **where**
mem_eclose_fm(*x*,*y*) ==
Exists(*Exists*(
And(*finite_ordinal_fm*(*1*),

And(eclose_n_fm(x#+2,1,0), Member(y#+2,0)))))

lemma *mem_eclose_type* [*TC*]:
[| x ∈ nat; y ∈ nat |] ==> mem_eclose_fm(x,y) ∈ formula
⟨*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 ∈ nat; j ∈ nat; env ∈ list(A)|]
==> mem_eclose(##A, x, y) ⟷ sats(A, mem_eclose_fm(i,j), env)
⟨*proof*⟩

theorem *mem_eclose_reflection*:
REFLECTS[λx. mem_eclose(L,f(x),g(x)),
λi x. mem_eclose(##Lset(i),f(x),g(x))]
⟨*proof*⟩

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

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

lemma *is_eclose_type* [*TC*]:
[| x ∈ nat; y ∈ nat |] ==> is_eclose_fm(x,y) ∈ formula
⟨*proof*⟩

lemma *sats_is_eclose_fm* [*simp*]:
[| x ∈ nat; y ∈ nat; env ∈ list(A)|]
==> sats(A, is_eclose_fm(x,y), env) ⟷ is_eclose(##A, nth(x,env), nth(y,env))
⟨*proof*⟩

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

theorem *is_eclose_reflection*:
REFLECTS[λx. is_eclose(L,f(x),g(x)),
λi x. is_eclose(##Lset(i),f(x),g(x))]
⟨*proof*⟩

7.6.5 The List Functor, Internalized

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

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

lemma *list_functor_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *list_functor_fm*(*x*,*y*,*z*) ∈ *formula*
⟨*proof*⟩

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[λ*x*. *is_list_functor*(*L*,*f*(*x*),*g*(*x*),*h*(*x*)),
λ*i* *x*. *is_list_functor*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*))]
⟨*proof*⟩

7.6.6 The Formula *is_list_N*, Internalized

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

lemma *list_N_fm_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *list_N_fm*(*x*,*y*,*z*) ∈ *formula*
⟨*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[λ*x*. *is_list_N*(*L*, *f*(*x*), *g*(*x*), *h*(*x*)),
λ*i* *x*. *is_list_N*(##*Lset*(*i*), *f*(*x*), *g*(*x*), *h*(*x*))]
⟨*proof*⟩

7.6.7 The Predicate "Is A List"

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

lemma *mem_list_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat* |] ==> *mem_list_fm*(*x*,*y*) ∈ *formula*
⟨*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[λ*x*. *mem_list*(*L*,*f*(*x*),*g*(*x*)),
λ*i* *x*. *mem_list*(##*Lset*(*i*),*f*(*x*),*g*(*x*))]
⟨*proof*⟩

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

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

lemma *is_list_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat* |] ==> *is_list_fm*(*x*,*y*) ∈ *formula*
⟨*proof*⟩

lemma *sats_is_list_fm* [*simp*]:
[| $x \in nat$; $y \in nat$; $env \in list(A)$|]
==> $sats(A, is\_list\_fm(x,y), env) \longleftrightarrow is\_list(\#\#A, nth(x,env), nth(y,env))$
⟨*proof*⟩

lemma *is_list_iff_sats*:
[| $nth(i,env) = x$; $nth(j,env) = y$;
$i \in nat$; $j \in nat$; $env \in list(A)$|]
==> $is\_list(\#\#A, x, y) \longleftrightarrow sats(A, is\_list\_fm(i,j), env)$
⟨*proof*⟩

theorem *is_list_reflection*:
$REFLECTS[\lambda x.\ is\_list(L,f(x),g(x)),$
$\lambda i\ x.\ is\_list(\#\#Lset(i),f(x),g(x))]$
⟨*proof*⟩

7.6.9 The Formula Functor, Internalized

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

$formula\_functor\_fm(X,Z) ==$
$Exists(Exists(Exists(Exists(Exists($
$And(omega\_fm(4),$
$And(cartprod\_fm(4,4,3),$
$And(sum\_fm(3,3,2),$
$And(cartprod\_fm(X\#+5,X\#+5,1),$
$And(sum\_fm(1,X\#+5,0), sum\_fm(2,0,Z\#+5)))))))))))$

lemma *formula_functor_type* [*TC*]:
[| $x \in nat$; $y \in nat$ |] ==> $formula\_functor\_fm(x,y) \in formula$
⟨*proof*⟩

lemma *sats_formula_functor_fm* [*simp*]:
[| $x \in nat$; $y \in nat$; $env \in list(A)$|]
==> $sats(A, formula\_functor\_fm(x,y), env) \longleftrightarrow$
$is\_formula\_functor(\#\#A, nth(x,env), nth(y,env))$
⟨*proof*⟩

lemma *formula_functor_iff_sats*:
[| $nth(i,env) = x$; $nth(j,env) = y$;
$i \in nat$; $j \in nat$; $env \in list(A)$|]
==> $is\_formula\_functor(\#\#A, x, y) \longleftrightarrow sats(A, formula\_functor\_fm(i,j), env)$
⟨*proof*⟩

theorem *formula_functor_reflection*:
$REFLECTS[\lambda x.\ is\_formula\_functor(L,f(x),g(x)),$
$\lambda i\ x.\ is\_formula\_functor(\#\#Lset(i),f(x),g(x))]$

⟨*proof*⟩

7.6.10 The Formula *is_formula_N*, Internalized

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

lemma *formula_N_fm_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat* |] ==> *formula_N_fm*(*x*,*y*) ∈ *formula*
⟨*proof*⟩

lemma *sats_formula_N_fm* [*simp*]:
[| *x* < *length*(*env*); *y* < *length*(*env*); *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *formula_N_fm*(*x*,*y*), *env*) ⟷
is_formula_N(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
⟨*proof*⟩

lemma *formula_N_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*;
i < *length*(*env*); *j* < *length*(*env*); *env* ∈ *list*(*A*)|]
==> *is_formula_N*(##*A*, *x*, *y*) ⟷ *sats*(*A*, *formula_N_fm*(*i*,*j*), *env*)
⟨*proof*⟩

theorem *formula_N_reflection*:
REFLECTS[λ*x*. *is_formula_N*(*L*, *f*(*x*), *g*(*x*)),
λ*i* *x*. *is_formula_N*(##*Lset*(*i*), *f*(*x*), *g*(*x*))]
⟨*proof*⟩

7.6.11 The Predicate "Is A Formula"

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

lemma *mem_formula_type* [*TC*]:
x ∈ *nat* ==> *mem_formula_fm*(*x*) ∈ *formula*
⟨*proof*⟩

lemma *sats_mem_formula_fm* [*simp*]:
[| *x* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *mem_formula_fm*(*x*), *env*) ⟷ *mem_formula*(##*A*, *nth*(*x*,*env*))
⟨*proof*⟩

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

theorem *mem_formula_reflection*:
REFLECTS[λ*x*. *mem_formula*(*L*,*f*(*x*)),
λ*i* *x*. *mem_formula*(##*Lset*(*i*),*f*(*x*))]
⟨*proof*⟩

7.6.12 The Predicate "Is *formula*"

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

lemma *is_formula_type* [*TC*]:
x ∈ *nat* ==> *is_formula_fm*(*x*) ∈ *formula*
⟨*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[λ*x*. *is_formula*(*L*,*f*(*x*)),
λ*i* *x*. *is_formula*(##*Lset*(*i*),*f*(*x*))]
⟨*proof*⟩

7.6.13 The Operator *is_transrec*

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

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

lemma *is_transrec_type* [*TC*]:
[| *p* ∈ *formula*; *x* ∈ *nat*; *z* ∈ *nat* |]
==> *is_transrec_fm*(*p*,*x*,*z*) ∈ *formula*
⟨*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*[λ*x*. *MH*(*L*, *f'*(*x*), *f*(*x*), *g*(*x*), *h*(*x*)),
λ*i x*. *MH*(##*Lset*(*i*), *f'*(*x*), *f*(*x*), *g*(*x*), *h*(*x*))]
shows *REFLECTS*[λ*x*. *is_transrec*(*L*, *MH*(*L*,*x*), *f*(*x*), *h*(*x*)),
λ*i x*. *is_transrec*(##*Lset*(*i*), *MH*(##*Lset*(*i*),*x*), *f*(*x*), *h*(*x*))]
⟨*proof*⟩

end

8 Separation for Facts About Recursion

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

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

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

⟨*proof*⟩

8.1 The Locale *M_trancl*

8.1.1 Separation for Reflexive/Transitive Closure

First, The Defining Formula

definition
rtran_closure_mem_fm :: [*i*,*i*,*i*]=>*i* **where**
rtran_closure_mem_fm(*A*,*r*,*p*) ==
Exists(*Exists*(*Exists*(
And(*omega_fm*(*2*),
And(*Member*(*1*,*2*),
And(*succ_fm*(*1*,*0*),
Exists(*And*(*typed_function_fm*(*1*, *A#+4*, *0*),
And(*Exists*(*Exists*(*Exists*(
And(*pair_fm*(*2*,*1*,*p#+7*),
And(*empty_fm*(*0*),
And(*fun_apply_fm*(*3*,*0*,*2*), *fun_apply_fm*(*3*,*5*,*1*)))))))),
Forall(*Implies*(*Member*(*0*,*3*),
Exists(*Exists*(*Exists*(*Exists*(
And(*fun_apply_fm*(*5*,*4*,*3*),
And(*succ_fm*(*4*,*2*),
And(*fun_apply_fm*(*5*,*2*,*1*),
And(*pair_fm*(*3*,*1*,*0*), *Member*(*0*,*r#+9*)))))))))))))))))))))

lemma *rtran_closure_mem_type* [*TC*]:
[| *x* ∈ *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[λ*x*. *rtran_closure_mem*(*L*,*f*(*x*),*g*(*x*),*h*(*x*)),
λ*i* *x*. *rtran_closure_mem*(*##Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*))]
⟨*proof*⟩

Separation for r^*.

lemma *rtrancl_separation*:
[| *L*(*r*); *L*(*A*) |] ==> *separation* (*L*, *rtran_closure_mem*(*L*,*A*,*r*))
⟨*proof*⟩

8.1.2 Reflexive/Transitive Closure, Internalized

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

lemma *rtran_closure_type* [*TC*]:
[| *x* ∈ *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[λ*x*. *rtran_closure*(*L*,*f*(*x*),*g*(*x*)),
λ*i* *x*. *rtran_closure*(##*Lset*(*i*),*f*(*x*),*g*(*x*))]
⟨*proof*⟩

8.1.3 Transitive Closure of a Relation, Internalized

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

lemma *tran_closure_type* [*TC*]:
[| *x* ∈ *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*:
$[|\ nth(i,env) = x;\ nth(j,env) = y;$
$i \in nat;\ j \in nat;\ env \in list(A)|]$
$==>\ tran\_closure(\#\#A,\ x,\ y) \longleftrightarrow sats(A,\ tran\_closure\_fm(i,j),\ env)$
⟨*proof*⟩

theorem *tran_closure_reflection*:
$REFLECTS[\lambda x.\ tran\_closure(L,f(x),g(x)),$
$\lambda i\ x.\ tran\_closure(\#\#Lset(i),f(x),g(x))]$
⟨*proof*⟩

8.1.4 Separation for the Proof of *wellfounded_on_trancl*

lemma *wellfounded_trancl_reflects*:
$REFLECTS[\lambda x.\ \exists\, w[L].\ \exists\, wx[L].\ \exists\, rp[L].$
$w \in Z\ \&\ pair(L,w,x,wx)\ \&\ tran\_closure(L,r,rp)\ \&\ wx \in rp,$
$\lambda i\ x.\ \exists\, w \in Lset(i).\ \exists\, wx \in Lset(i).\ \exists\, rp \in Lset(i).$
$w \in Z\ \&\ pair(\#\#Lset(i),w,x,wx)\ \&\ tran\_closure(\#\#Lset(i),r,rp)\ \&$
$wx \in rp]$
⟨*proof*⟩

lemma *wellfounded_trancl_separation*:
$[|\ L(r);\ L(Z)\ |] ==>$
$separation\ (L,\ \lambda x.$
$\exists\, w[L].\ \exists\, wx[L].\ \exists\, rp[L].$
$w \in Z\ \&\ pair(L,w,x,wx)\ \&\ tran\_closure(L,r,rp)\ \&\ wx \in rp)$
⟨*proof*⟩

8.1.5 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*⟩

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

8.2.1 Instances of Replacement for Lists

lemma *list_replacement1_Reflects*:
$REFLECTS$
$[\lambda x.\ \exists\, u[L].\ u \in B \wedge (\exists\, y[L].\ pair(L,u,y,x) \wedge$
$is\_wfrec(L,\ iterates\_MH(L,\ is\_list\_functor(L,A),\ 0),\ memsn,\ u,\ y)),$
$\lambda i\ x.\ \exists\, u \in Lset(i).\ u \in B \wedge (\exists\, y \in Lset(i).\ pair(\#\#Lset(i),\ u,\ y,\ x) \wedge$

is_wfrec(##*Lset*(*i*),
iterates_MH(##*Lset*(*i*),
is_list_functor(##*Lset*(*i*), *A*), *0*), *memsn*, *u*, *y*))]
⟨*proof*⟩

lemma *list_replacement1*:
L(*A*) ==> *iterates_replacement*(*L*, *is_list_functor*(*L*,*A*), *0*)
⟨*proof*⟩

lemma *list_replacement2_Reflects*:
REFLECTS
[λ*x*. ∃ *u*[*L*]. *u* ∈ *B* & *u* ∈ *nat* &
is_iterates(*L*, *is_list_functor*(*L*, *A*), *0*, *u*, *x*),
λ*i x*. ∃ *u* ∈ *Lset*(*i*). *u* ∈ *B* & *u* ∈ *nat* &
is_iterates(##*Lset*(*i*), *is_list_functor*(##*Lset*(*i*), *A*), *0*, *u*, *x*)]
⟨*proof*⟩

lemma *list_replacement2*:
L(*A*) ==> *strong_replacement*(*L*,
λ*n y*. *n*∈*nat* & *is_iterates*(*L*, *is_list_functor*(*L*,*A*), *0*, *n*, *y*))
⟨*proof*⟩

8.3 *L* is Closed Under the Operator *formula*

8.3.1 Instances of Replacement for Formulas

lemma *formula_replacement1_Reflects*:
REFLECTS
[λ*x*. ∃ *u*[*L*]. *u* ∈ *B* & (∃ *y*[*L*]. *pair*(*L*,*u*,*y*,*x*) &
is_wfrec(*L*, *iterates_MH*(*L*, *is_formula_functor*(*L*), *0*), *memsn*, *u*, *y*)),
λ*i x*. ∃ *u* ∈ *Lset*(*i*). *u* ∈ *B* & (∃ *y* ∈ *Lset*(*i*). *pair*(##*Lset*(*i*), *u*, *y*, *x*) &
is_wfrec(##*Lset*(*i*),
iterates_MH(##*Lset*(*i*),
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
[λ*x*. ∃ *u*[*L*]. *u* ∈ *B* & *u* ∈ *nat* &
is_iterates(*L*, *is_formula_functor*(*L*), *0*, *u*, *x*),
λ*i x*. ∃ *u* ∈ *Lset*(*i*). *u* ∈ *B* & *u* ∈ *nat* &
is_iterates(##*Lset*(*i*), *is_formula_functor*(##*Lset*(*i*)), *0*, *u*, *x*)]
⟨*proof*⟩

lemma *formula_replacement2*:
strong_replacement(*L*,
λ*n 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!

8.3.2 The Formula *is_nth*, Internalized

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

lemma *nth_fm_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *nth_fm*(*x*,*y*,*z*) ∈ *formula*
⟨*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[λ*x*. *is_nth*(*L*, *f*(*x*), *g*(*x*), *h*(*x*)),
λ*i x*. *is_nth*(##*Lset*(*i*), *f*(*x*), *g*(*x*), *h*(*x*))]
⟨*proof*⟩

8.3.3 An Instance of Replacement for *nth*

lemma *nth_replacement_Reflects*:
REFLECTS
[λ*x*. ∃ *u*[*L*]. *u* ∈ *B* & (∃ *y*[*L*]. *pair*(*L*,*u*,*y*,*x*) &
is_wfrec(*L*, *iterates_MH*(*L*, *is_tl*(*L*), *z*), *memsn*, *u*, *y*)),
λ*i x*. ∃ *u* ∈ *Lset*(*i*). *u* ∈ *B* & (∃ *y* ∈ *Lset*(*i*). *pair*(##*Lset*(*i*), *u*, *y*, *x*) &
is_wfrec(##*Lset*(*i*),
iterates_MH(##*Lset*(*i*),
is_tl(##*Lset*(*i*)), *z*), *memsn*, *u*, *y*))]
⟨*proof*⟩

lemma *nth_replacement*:
$L(w) ==> iterates\_replacement(L,\ is\_tl(L),\ w)$
⟨*proof*⟩

8.3.4 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*⟩

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

8.4.1 Instances of Replacement for *eclose*

lemma *eclose_replacement1_Reflects*:
REFLECTS
$[\lambda x.\ \exists\, u[L].\ u \in B\ \&\ (\exists\, y[L].\ pair(L,u,y,x)\ \&$
$\quad is\_wfrec(L,\ iterates\_MH(L,\ big\_union(L),\ A),\ memsn,\ u,\ y)),$
$\ \lambda i\ x.\ \exists\, u \in Lset(i).\ u \in B\ \&\ (\exists\, y \in Lset(i).\ pair(\#\#Lset(i),\ u,\ y,\ x)\ \&$
$\quad is\_wfrec(\#\#Lset(i),$
$\qquad iterates\_MH(\#\#Lset(i),\ big\_union(\#\#Lset(i)),\ A),$
$\qquad memsn,\ u,\ y))]$
⟨*proof*⟩

lemma *eclose_replacement1*:
$L(A) ==> iterates\_replacement(L,\ big\_union(L),\ A)$
⟨*proof*⟩

lemma *eclose_replacement2_Reflects*:
REFLECTS
$[\lambda x.\ \exists\, u[L].\ u \in B\ \&\ u \in nat\ \&$
$\qquad is\_iterates(L,\ big\_union(L),\ A,\ u,\ x),$
$\ \lambda i\ x.\ \exists\, u \in Lset(i).\ u \in B\ \&\ u \in nat\ \&$
$\qquad is\_iterates(\#\#Lset(i),\ big\_union(\#\#Lset(i)),\ A,\ u,\ x)]$
⟨*proof*⟩

lemma *eclose_replacement2*:
$L(A) ==> strong\_replacement(L,$
$\quad \lambda n\ y.\ n{\in}nat\ \&\ is\_iterates(L,\ big\_union(L),\ A,\ n,\ y))$
⟨*proof*⟩

8.4.2 Instantiating the locale *M_eclose*

lemma *M_eclose_axioms_L*: $M\_eclose\_axioms(L)$
⟨*proof*⟩

theorem *M_eclose_L*: *M_eclose*(*L*)
⟨*proof*⟩

interpretation *L*: *M_eclose L* ⟨*proof*⟩

end

9 Absoluteness for the Satisfies Relation on Formulas

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

9.1 More Internalization

9.1.1 The Formula *is_depth*, Internalized

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

lemma *depth_fm_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat* |] ==> *depth_fm*(*x*,*y*) ∈ *formula*
⟨*proof*⟩

lemma *sats_depth_fm* [*simp*]:
[| *x* ∈ *nat*; *y* < *length*(*env*); *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *depth_fm*(*x*,*y*), *env*) ⟷
is_depth(##*A*, *nth*(*x*,*env*), *nth*(*y*,*env*))
⟨*proof*⟩

lemma *depth_iff_sats*:
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*;
i ∈ *nat*; *j* < *length*(*env*); *env* ∈ *list*(*A*)|]
==> *is_depth*(##*A*, *x*, *y*) ⟷ *sats*(*A*, *depth_fm*(*i*,*j*), *env*)
⟨*proof*⟩

theorem *depth_reflection*:
REFLECTS[λ*x*. *is_depth*(*L*, *f*(*x*), *g*(*x*)),
λ*i* *x*. *is_depth*(##*Lset*(*i*), *f*(*x*), *g*(*x*))]
⟨*proof*⟩

9.1.2 The Operator *is_formula_case*

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

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

lemma *is_formula_case_type* [*TC*]:
[| *is_a* ∈ *formula*; *is_b* ∈ *formula*; *is_c* ∈ *formula*; *is_d* ∈ *formula*;
x ∈ *nat*; *y* ∈ *nat* |]
==> *formula_case_fm*(*is_a*, *is_b*, *is_c*, *is_d*, *x*, *y*) ∈ *formula*
⟨*proof*⟩

lemma *sats_formula_case_fm*:
assumes *is_a_iff_sats*:
!!*a0 a1 a2*.
[|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*|]
==> *ISA*(*a2*, *a1*, *a0*) ⟷ *sats*(*A*, *is_a*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*env*))))
and *is_b_iff_sats*:
!!*a0 a1 a2*.
[|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*|]
==> *ISB*(*a2*, *a1*, *a0*) ⟷ *sats*(*A*, *is_b*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*env*))))
and *is_c_iff_sats*:
!!*a0 a1 a2*.
[|*a0*∈*A*; *a1*∈*A*; *a2*∈*A*|]
==> *ISC*(*a2*, *a1*, *a0*) ⟷ *sats*(*A*, *is_c*, *Cons*(*a0*,*Cons*(*a1*,*Cons*(*a2*,*env*))))
and *is_d_iff_sats*:
!!*a0 a1*.
[|*a0*∈*A*; *a1*∈*A*|]
==> *ISD*(*a1*, *a0*) ⟷ *sats*(*A*, *is_d*, *Cons*(*a0*,*Cons*(*a1*,*env*)))
shows
[|*x* ∈ *nat*; *y* < *length*(*env*); *env* ∈ *list*(*A*)|]

$==>$ $sats(A,\ formula\_case\_fm(is\_a,is\_b,is\_c,is\_d,x,y),\ env) \longleftrightarrow$
$is\_formula\_case(\#\#A,\ ISA,\ ISB,\ ISC,\ ISD,\ nth(x,env),\ nth(y,env))$
⟨*proof*⟩

lemma *formula_case_iff_sats*:
assumes *is_a_iff_sats*:
$!!a0\ a1\ a2.$
$[|a0{\in}A;\ a1{\in}A;\ a2{\in}A|]$
$==>$ $ISA(a2,\ a1,\ a0) \longleftrightarrow sats(A,\ is\_a,\ Cons(a0,Cons(a1,Cons(a2,env))))$
and *is_b_iff_sats*:
$!!a0\ a1\ a2.$
$[|a0{\in}A;\ a1{\in}A;\ a2{\in}A|]$
$==>$ $ISB(a2,\ a1,\ a0) \longleftrightarrow sats(A,\ is\_b,\ Cons(a0,Cons(a1,Cons(a2,env))))$
and *is_c_iff_sats*:
$!!a0\ a1\ a2.$
$[|a0{\in}A;\ a1{\in}A;\ a2{\in}A|]$
$==>$ $ISC(a2,\ a1,\ a0) \longleftrightarrow sats(A,\ is\_c,\ Cons(a0,Cons(a1,Cons(a2,env))))$
and *is_d_iff_sats*:
$!!a0\ a1.$
$[|a0{\in}A;\ a1{\in}A|]$
$==>$ $ISD(a1,\ a0) \longleftrightarrow sats(A,\ is\_d,\ Cons(a0,Cons(a1,env)))$
shows
$[|nth(i,env) = x;\ nth(j,env) = y;$
$i \in nat;\ j < length(env);\ env \in list(A)|]$
$==>$ $is\_formula\_case(\#\#A,\ ISA,\ ISB,\ ISC,\ ISD,\ x,\ y) \longleftrightarrow$
$sats(A,\ formula\_case\_fm(is\_a,is\_b,is\_c,is\_d,i,j),\ env)$
⟨*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*:
$!!h\ f\ g\ g'.\ REFLECTS[\lambda x.\ is\_a(L,\ h(x),\ f(x),\ g(x),\ g'(x)),$
$\lambda i\ x.\ is\_a(\#\#Lset(i),\ h(x),\ f(x),\ g(x),\ g'(x))]$
and *is_b_reflection*:
$!!h\ f\ g\ g'.\ REFLECTS[\lambda x.\ is\_b(L,\ h(x),\ f(x),\ g(x),\ g'(x)),$
$\lambda i\ x.\ is\_b(\#\#Lset(i),\ h(x),\ f(x),\ g(x),\ g'(x))]$
and *is_c_reflection*:
$!!h\ f\ g\ g'.\ REFLECTS[\lambda x.\ is\_c(L,\ h(x),\ f(x),\ g(x),\ g'(x)),$
$\lambda i\ x.\ is\_c(\#\#Lset(i),\ h(x),\ f(x),\ g(x),\ g'(x))]$
and *is_d_reflection*:
$!!h\ f\ g\ g'.\ REFLECTS[\lambda x.\ is\_d(L,\ h(x),\ f(x),\ g(x)),$
$\lambda i\ x.\ is\_d(\#\#Lset(i),\ h(x),\ f(x),\ g(x))]$
shows $REFLECTS[\lambda x.\ is\_formula\_case(L,\ is\_a(L,x),\ is\_b(L,x),\ is\_c(L,x),$
$is\_d(L,x),\ g(x),\ h(x)),$
$\lambda i\ x.\ is\_formula\_case(\#\#Lset(i),\ is\_a(\#\#Lset(i),\ x),\ is\_b(\#\#Lset(i),$
$x),\ is\_c(\#\#Lset(i),\ x),\ is\_d(\#\#Lset(i),\ x),\ g(x),\ h(x))]$
⟨*proof*⟩

9.2 Absoluteness for the Function *satisfies*

definition
is_depth_apply :: [*i*=>*o*,*i*,*i*,*i*] => *o* **where**
— Merely a useful abbreviation for the sequel.
is_depth_apply(*M*,*h*,*p*,*z*) ==
∃ *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*) ==
λ*x* *y*. λ*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*) ==
λ*x* *y* *zz*. ∀ *lA*[*M*]. *is_list*(*M*,*A*,*lA*) ⟶
is_lambda(*M*, *lA*,
λ*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*) ==
λ*x* *y*. λ*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*) ==
λ*x* *y* *zz*. ∀ *lA*[*M*]. *is_list*(*M*,*A*,*lA*) ⟶
is_lambda(*M*, *lA*,
λ*env* *z*. *is_bool_of_o*(*M*,
∃ *nx*[*M*]. *is_nth*(*M*,*x*,*env*,*nx*) & *is_nth*(*M*,*y*,*env*,*nx*), *z*),
zz)

definition
satisfies_c :: [*i*,*i*,*i*,*i*,*i*]=>*i* **where**
satisfies_c(*A*) == λ*p q rp rq*. λ*env* ∈ *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*) ==
λ*p q zz*. ∀ *lA*[*M*]. *is_list*(*M*,*A*,*lA*) ⟶
is_lambda(*M*, *lA*, λ*env z*. ∃ *hp*[*M*]. ∃ *hq*[*M*].
(∃ *rp*[*M*]. *is_depth_apply*(*M*,*h*,*p*,*rp*) & *fun_apply*(*M*,*rp*,*env*,*hp*)) &
(∃ *rq*[*M*]. *is_depth_apply*(*M*,*h*,*q*,*rq*) & *fun_apply*(*M*,*rq*,*env*,*hq*)) &
(∃ *pq*[*M*]. *is_and*(*M*,*hp*,*hq*,*pq*) & *is_not*(*M*,*pq*,*z*)),
zz)

definition
satisfies_d :: [*i*,*i*,*i*]=>*i* **where**
satisfies_d(*A*)
== λ*p rp*. λ*env* ∈ *list*(*A*). *bool_of_o* (∀ *x*∈*A*. *rp* ‘ (*Cons*(*x*,*env*)) = *1*)

definition
satisfies_is_d :: [*i*=>*o*,*i*,*i*,*i*,*i*]=>*o* **where**
satisfies_is_d(*M*,*A*,*h*) ==
λ*p zz*. ∀ *lA*[*M*]. *is_list*(*M*,*A*,*lA*) ⟶
is_lambda(*M*, *lA*,
λ*env z*. ∃ *rp*[*M*]. *is_depth_apply*(*M*,*h*,*p*,*rp*) &
is_bool_of_o(*M*,
∀ *x*[*M*]. ∀ *xenv*[*M*]. ∀ *hp*[*M*].
x∈*A* ⟶ *is_Cons*(*M*,*x*,*env*,*xenv*) ⟶
fun_apply(*M*,*rp*,*xenv*,*hp*) ⟶ *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 ==
λ*M 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*]=>*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* + *M_datatypes* +
assumes
Member_replacement:
[|*M*(*A*); *x* ∈ *nat*; *y* ∈ *nat*|]
==> *strong_replacement*
(*M*, λ*env 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*, λ*env z*. ∃ *bo*[*M*]. ∃ *nx*[*M*]. ∃ *ny*[*M*].
env ∈ *list*(*A*) & *is_nth*(*M*,*x*,*env*,*nx*) & *is_nth*(*M*,*y*,*env*,*ny*) &
is_bool_of_o(*M*, *nx* = *ny*, *bo*) &
pair(*M*, *env*, *bo*, *z*))
and
Nand_replacement:
[|*M*(*A*); *M*(*rp*); *M*(*rq*)|]
==> *strong_replacement*
(*M*, λ*env z*. ∃ *rpe*[*M*]. ∃ *rqe*[*M*]. ∃ *andpq*[*M*]. ∃ *notpq*[*M*].
fun_apply(*M*,*rp*,*env*,*rpe*) & *fun_apply*(*M*,*rq*,*env*,*rqe*) &
is_and(*M*,*rpe*,*rqe*,*andpq*) & *is_not*(*M*,*andpq*,*notpq*) &
env ∈ *list*(*A*) & *pair*(*M*, *env*, *notpq*, *z*))
and
Forall_replacement:
[|*M*(*A*); *M*(*rp*)|]
==> *strong_replacement*
(*M*, λ*env z*. ∃ *bo*[*M*].
env ∈ *list*(*A*) &
is_bool_of_o (*M*,
∀ *a*[*M*]. ∀ *co*[*M*]. ∀ *rpco*[*M*].
a∈*A* ⟶ *is_Cons*(*M*,*a*,*env*,*co*) ⟶
fun_apply(*M*,*rp*,*co*,*rpco*) ⟶ *number1*(*M*, *rpco*),
bo) &
pair(*M*,*env*,*bo*,*z*))
and

formula_rec_replacement:
— For the *transrec*
[|$n \in nat$; $M(A)$|] ==> *transrec_replacement*(M, *satisfies_MH*(M,A), n)
and
formula_rec_lambda_replacement:
— For the λ-*abstraction* in the *transrec* body
[|$M(g)$; $M(A)$|] ==>
strong_replacement (M,
$\lambda x\ y$. *mem_formula*(M,x) &
($\exists\, c[M]$. *is_formula_case*(M, *satisfies_is_a*(M,A),
satisfies_is_b(M,A),
satisfies_is_c(M,A,g),
satisfies_is_d(M,A,g), x, c) &
pair(M, x, c, y)))

lemma (**in** *M_satisfies*) *Member_replacement'*:
[|$M(A)$; $x \in nat$; $y \in nat$|]
==> *strong_replacement*
(M, λ*env z*. *env* $\in$ *list*(A) &
$z = \langle env,\ bool\_of\_o(nth(x,\ env) \in nth(y,\ env))\rangle$)
⟨*proof*⟩

lemma (**in** *M_satisfies*) *Equal_replacement'*:
[|$M(A)$; $x \in nat$; $y \in nat$|]
==> *strong_replacement*
(M, λ*env z*. *env* $\in$ *list*(A) &
$z = \langle env,\ bool\_of\_o(nth(x,\ env) = nth(y,\ env))\rangle$)
⟨*proof*⟩

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

lemma (**in** *M_satisfies*) *Forall_replacement'*:
[|$M(A)$; $M(rp)$|]
==> *strong_replacement*
(M, λ*env z*.
env $\in$ *list*(A) &
$z = \langle env,\ bool\_of\_o\ (\forall\, a{\in}A.\ rp\ `\ Cons(a,env) = 1)\rangle$)
⟨*proof*⟩

lemma (**in** *M_satisfies*) *a_closed*:
[|$M(A)$; $x{\in}nat$; $y{\in}nat$|] ==> 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*:
$[|M(A);\ x{\in}nat;\ y{\in}nat|] ==> M(satisfies\_b(A,x,y))$
⟨*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*:
$[|M(A);\ x \in formula;\ y \in formula;\ M(rx);\ M(ry)|]$
$==> M(satisfies\_c(A,x,y,rx,ry))$
⟨*proof*⟩

lemma (**in** *M_satisfies*) *c_rel*:
$[|M(A);\ M(f)|] ==>$
$Relation2\ (M,\ formula,\ formula,$
$satisfies\_is\_c(M,A,f),$
$\lambda u\ v.\ satisfies\_c(A,\ u,\ v,\ f\ `\ succ(depth(u))\ `\ u,$
$f\ `\ succ(depth(v))\ `\ v))$
⟨*proof*⟩

lemma (**in** *M_satisfies*) *d_closed*:
$[|M(A);\ x \in formula;\ M(rx)|] ==> M(satisfies\_d(A,x,rx))$
⟨*proof*⟩

lemma (**in** *M_satisfies*) *d_rel*:
$[|M(A);\ M(f)|] ==>$
$Relation1(M,\ formula,\ satisfies\_is\_d(M,A,f),$
$\lambda u.\ satisfies\_d(A,\ u,\ f\ `\ succ(depth(u))\ `\ u))$
⟨*proof*⟩

lemma (**in** *M_satisfies*) *fr_replace*:
$[|n \in nat;\ M(A)|] ==> transrec\_replacement(M,satisfies\_MH(M,A),n)$
⟨*proof*⟩

lemma (**in** *M_satisfies*) *formula_case_satisfies_closed*:
$[|M(g);\ M(A);\ x \in formula|] ==>$
$M(formula\_case\ (satisfies\_a(A),\ satisfies\_b(A),$
$\lambda u\ v.\ satisfies\_c(A,\ u,\ v,$
$g\ `\ succ(depth(u))\ `\ u,\ g\ `\ succ(depth(v))\ `\ v),$
$\lambda u.\ satisfies\_d\ (A,\ u,\ g\ `\ succ(depth(u))\ `\ u),$
$x))$
⟨*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⟩
```

9.3 Internalizations Needed to Instantiate *M_satisfies*

9.3.1 The Operator *is_depth_apply*, Internalized

```
definition
  depth_apply_fm :: [i,i,i]=>i where
    depth_apply_fm(h,p,z) ==
       Exists(Exists(Exists(
        And(finite_ordinal_fm(2),
```

```
        And(depth_fm(p#+3,2),
         And(succ_fm(2,1),
          And(fun_apply_fm(h#+3,1,0), fun_apply_fm(0,p#+3,z#+3)))))))
```

lemma *depth_apply_type* [*TC*]:
```
    [| x ∈ nat; y ∈ nat; z ∈ nat |] ==> depth_apply_fm(x,y,z) ∈ formula
```
⟨*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*⟩

9.3.2 The Operator *satisfies_is_a*, Internalized

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

lemma *satisfies_is_a_type* [*TC*]:
```
     [| A ∈ nat; x ∈ nat; y ∈ nat; z ∈ nat |]
      ==> satisfies_is_a_fm(A,x,y,z) ∈ formula
```
⟨*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*:
[| $nth(u,env) = nu$; $nth(x,env) = nx$; $nth(y,env) = ny$; $nth(z,env) = nz$;
$u \in nat$; $x < length(env)$; $y < length(env)$; $z \in nat$; $env \in list(A)$|]
==> $satisfies\_is\_a(\#\#A,nu,nx,ny,nz) \longleftrightarrow$
$sats(A, satisfies\_is\_a\_fm(u,x,y,z), env)$
⟨*proof*⟩

theorem *satisfies_is_a_reflection*:
$REFLECTS[\lambda x.\ satisfies\_is\_a(L,f(x),g(x),h(x),g'(x)),$
$\lambda i\ x.\ satisfies\_is\_a(\#\#Lset(i),f(x),g(x),h(x),g'(x))]$
⟨*proof*⟩

9.3.3 The Operator *satisfies_is_b*, Internalized

definition
satisfies_is_b_fm :: $[i,i,i,i]=>i$ **where**
$satisfies\_is\_b\_fm(A,x,y,z) ==$
$Forall($
$Implies(is\_list\_fm(succ(A),0),$
$lambda\_fm($
$bool\_of\_o\_fm(Exists(And(nth\_fm(x\#+5,2,0),\ nth\_fm(y\#+5,2,0))),\ 0),$
$0,\ succ(z))))$

lemma *satisfies_is_b_type* [*TC*]:
[| $A \in nat$; $x \in nat$; $y \in nat$; $z \in nat$ |]
==> $satisfies\_is\_b\_fm(A,x,y,z) \in formula$
⟨*proof*⟩

lemma *sats_satisfies_is_b_fm* [*simp*]:
[| $u \in nat$; $x < length(env)$; $y < length(env)$; $z \in nat$; $env \in list(A)$|]
==> $sats(A, satisfies\_is\_b\_fm(u,x,y,z), env) \longleftrightarrow$
$satisfies\_is\_b(\#\#A, nth(u,env), nth(x,env), nth(y,env), nth(z,env))$
⟨*proof*⟩

lemma *satisfies_is_b_iff_sats*:
[| $nth(u,env) = nu$; $nth(x,env) = nx$; $nth(y,env) = ny$; $nth(z,env) = nz$;
$u \in nat$; $x < length(env)$; $y < length(env)$; $z \in nat$; $env \in list(A)$|]
==> $satisfies\_is\_b(\#\#A,nu,nx,ny,nz) \longleftrightarrow$
$sats(A, satisfies\_is\_b\_fm(u,x,y,z), env)$
⟨*proof*⟩

theorem *satisfies_is_b_reflection*:
$REFLECTS[\lambda x.\ satisfies\_is\_b(L,f(x),g(x),h(x),g'(x)),$
$\lambda i\ x.\ satisfies\_is\_b(\#\#Lset(i),f(x),g(x),h(x),g'(x))]$
⟨*proof*⟩

9.3.4 The Operator *satisfies_is_c*, Internalized

definition

```
 satisfies_is_c_fm :: [i,i,i,i,i]=>i where
satisfies_is_c_fm(A,h,p,q,zz) ==
  Forall(
    Implies(is_list_fm(succ(A),0),
      lambda_fm(
        Exists(Exists(
       And(Exists(And(depth_apply_fm(h#+7,p#+7,0), fun_apply_fm(0,4,2))),
       And(Exists(And(depth_apply_fm(h#+7,q#+7,0), fun_apply_fm(0,4,1))),
            Exists(And(and_fm(2,1,0), not_fm(0,3))))))),
        0, succ(zz))))

lemma satisfies_is_c_type [TC]:
    [| A ∈ nat; h ∈ nat; x ∈ nat; y ∈ nat; z ∈ nat |]
     ==> satisfies_is_c_fm(A,h,x,y,z) ∈ formula
⟨proof⟩

lemma sats_satisfies_is_c_fm [simp]:
   [| u ∈ nat; v ∈ nat; x ∈ nat; y ∈ nat; z ∈ nat; env ∈ list(A)|]
    ==> sats(A, satisfies_is_c_fm(u,v,x,y,z), env) ⟷
        satisfies_is_c(##A, nth(u,env), nth(v,env), nth(x,env),
                       nth(y,env), nth(z,env))
⟨proof⟩

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⟩
```

9.3.5 The Operator *satisfies_is_d*, Internalized

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

Implies(*Cons_fm*(*2*,*5*,*1*),
Implies(*fun_apply_fm*(*3*,*1*,*0*), *number1_fm*(*0*))))))), *1*))),
0, *succ*(*zz*))))

lemma *satisfies_is_d_type* [*TC*]:
[| *A* ∈ *nat*; *h* ∈ *nat*; *x* ∈ *nat*; *z* ∈ *nat* |]
==> *satisfies_is_d_fm*(*A*,*h*,*x*,*z*) ∈ *formula*
⟨*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[λ*x*. *satisfies_is_d*(*L*,*f*(*x*),*g*(*x*),*h*(*x*),*g*′(*x*)),
λ*i* *x*. *satisfies_is_d*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*),*g*′(*x*))]
⟨*proof*⟩

9.3.6 The Operator *satisfies_MH*, Internalized

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

lemma *satisfies_MH_type* [*TC*]:
[| *A* ∈ *nat*; *u* ∈ *nat*; *x* ∈ *nat*; *z* ∈ *nat* |]
==> *satisfies_MH_fm*(*A*,*u*,*x*,*z*) ∈ *formula*
⟨*proof*⟩

lemma *sats_satisfies_MH_fm* [*simp*]:
[| *u* ∈ *nat*; *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat*; *env* ∈ *list*(*A*)|]

$==>$ *sats*(*A*, *satisfies_MH_fm*(*u*,*x*,*y*,*z*), *env*) $\longleftrightarrow$
satisfies_MH(##*A*, *nth*(*u*,*env*), *nth*(*x*,*env*), *nth*(*y*,*env*), *nth*(*z*,*env*))
⟨*proof*⟩

lemma *satisfies_MH_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_MH*(##*A*,*nu*,*nx*,*ny*,*nz*) $\longleftrightarrow$
sats(*A*, *satisfies_MH_fm*(*u*,*x*,*y*,*z*), *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[λ*x*. *satisfies_MH*(*L*,*f*(*x*),*g*(*x*),*h*(*x*),*g*′(*x*)),
λ*i x*. *satisfies_MH*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*),*g*′(*x*))]
⟨*proof*⟩

9.4 Lemmas for Instantiating the Locale *M_satisfies*

9.4.1 The *Member* Case

lemma *Member_Reflects*:
REFLECTS[λ*u*. ∃ *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*)),
λ*i 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*, λ*env 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*⟩

9.4.2 The *Equal* Case

lemma *Equal_Reflects*:
REFLECTS[λ*u*. ∃ *v*[*L*]. *v* ∈ *B* ∧ (∃ *bo*[*L*]. ∃ *nx*[*L*]. ∃ *ny*[*L*].

$v \in lstA \wedge is\_nth(L,\ x,\ v,\ nx) \wedge is\_nth(L,\ y,\ v,\ ny) \wedge$
$is\_bool\_of\_o(L,\ nx = ny,\ bo) \wedge pair(L,\ v,\ bo,\ u)),$
$\lambda i\ u.\ \exists\, v \in Lset(i).\ v \in B \wedge (\exists\, bo \in Lset(i).\ \exists\, nx \in Lset(i).\ \exists\, ny \in Lset(i).$
$v \in lstA \wedge is\_nth(\#\#Lset(i),\ x,\ v,\ nx) \wedge$
$is\_nth(\#\#Lset(i),\ y,\ v,\ ny) \wedge$
$is\_bool\_of\_o(\#\#Lset(i),\ nx = ny,\ bo) \wedge pair(\#\#Lset(i),\ v,\ bo,\ u))]$
⟨*proof*⟩

lemma *Equal_replacement*:
$[|L(A);\ x \in nat;\ y \in nat|]$
$==>$ *strong_replacement*
$(L,\ \lambda env\ z.\ \exists\, bo[L].\ \exists\, nx[L].\ \exists\, ny[L].$
$env \in list(A)\ \&\ is\_nth(L,x,env,nx)\ \&\ is\_nth(L,y,env,ny)\ \&$
$is\_bool\_of\_o(L,\ nx = ny,\ bo)\ \&$
$pair(L,\ env,\ bo,\ z))$
⟨*proof*⟩

9.4.3 The *Nand* Case

lemma *Nand_Reflects*:
$REFLECTS\ [\lambda x.\ \exists\, u[L].\ u \in B \wedge$
$(\exists\, rpe[L].\ \exists\, rqe[L].\ \exists\, andpq[L].\ \exists\, notpq[L].$
$fun\_apply(L,\ rp,\ u,\ rpe) \wedge fun\_apply(L,\ rq,\ u,\ rqe) \wedge$
$is\_and(L,\ rpe,\ rqe,\ andpq) \wedge is\_not(L,\ andpq,\ notpq) \wedge$
$u \in list(A) \wedge pair(L,\ u,\ notpq,\ x)),$
$\lambda i\ x.\ \exists\, u \in Lset(i).\ u \in B \wedge$
$(\exists\, rpe \in Lset(i).\ \exists\, rqe \in Lset(i).\ \exists\, andpq \in Lset(i).\ \exists\, notpq \in Lset(i).$
$fun\_apply(\#\#Lset(i),\ rp,\ u,\ rpe) \wedge fun\_apply(\#\#Lset(i),\ rq,\ u,\ rqe) \wedge$
$is\_and(\#\#Lset(i),\ rpe,\ rqe,\ andpq) \wedge is\_not(\#\#Lset(i),\ andpq,\ notpq) \wedge$
$u \in list(A) \wedge pair(\#\#Lset(i),\ u,\ notpq,\ x))]$
⟨*proof*⟩

lemma *Nand_replacement*:
$[|L(A);\ L(rp);\ L(rq)|]$
$==>$ *strong_replacement*
$(L,\ \lambda env\ z.\ \exists\, rpe[L].\ \exists\, rqe[L].\ \exists\, andpq[L].\ \exists\, notpq[L].$
$fun\_apply(L,rp,env,rpe)\ \&\ fun\_apply(L,rq,env,rqe)\ \&$
$is\_and(L,rpe,rqe,andpq)\ \&\ is\_not(L,andpq,notpq)\ \&$
$env \in list(A)\ \&\ pair(L,\ env,\ notpq,\ z))$
⟨*proof*⟩

9.4.4 The *Forall* Case

lemma *Forall_Reflects*:
$REFLECTS\ [\lambda x.\ \exists\, u[L].\ u \in B \wedge (\exists\, bo[L].\ u \in list(A) \wedge$
$is\_bool\_of\_o\ (L,$
$\forall\, a[L].\ \forall\, co[L].\ \forall\, rpco[L].\ a \in A \longrightarrow$
$is\_Cons(L,a,u,co) \longrightarrow fun\_apply(L,rp,co,rpco) \longrightarrow$
$number1(L,rpco),$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad bo) \wedge pair(L,u,bo,x)),\\
\lambda i\ x.\ \exists\, u \in Lset(i).\ u \in B \wedge (\exists\, bo \in Lset(i).\ u \in list(A) \wedge\\
\qquad is\_bool\_of\_o\ (\#\#Lset(i),\\
\forall\, a \in Lset(i).\ \forall\, co \in Lset(i).\ \forall\, rpco \in Lset(i).\ a \in A \longrightarrow\\
\qquad\quad is\_Cons(\#\#Lset(i),a,u,co) \longrightarrow fun\_apply(\#\#Lset(i),rp,co,rpco) \longrightarrow\\
\qquad\quad number1(\#\#Lset(i),rpco),\\
\qquad\qquad\qquad bo) \wedge pair(\#\#Lset(i),u,bo,x))]
\end{array}
$$

⟨*proof*⟩

lemma *Forall_replacement*:

$$
\begin{array}{l}
\quad [|L(A);\ L(rp)|]\\
\quad ==>\ strong\_replacement\\
\qquad (L,\ \lambda env\ z.\ \exists\, bo[L].\\
\qquad\quad env \in list(A)\ \&\\
\qquad\quad is\_bool\_of\_o\ (L,\\
\qquad\qquad\qquad \forall\, a[L].\ \forall\, co[L].\ \forall\, rpco[L].\\
\qquad\qquad\qquad\quad a{\in}A \longrightarrow is\_Cons(L,a,env,co) \longrightarrow\\
\qquad\qquad\qquad\quad fun\_apply(L,rp,co,rpco) \longrightarrow number1(L,\ rpco),\\
\qquad\qquad\qquad bo)\ \&\\
\qquad\quad pair(L,env,bo,z))
\end{array}
$$

⟨*proof*⟩

9.4.5 The *transrec_replacement* Case

lemma *formula_rec_replacement_Reflects*:

$$
\begin{array}{l}
REFLECTS\ [\lambda x.\ \exists\, u[L].\ u \in B \wedge (\exists\, y[L].\ pair(L,\ u,\ y,\ x) \wedge\\
\qquad is\_wfrec\ (L,\ satisfies\_MH(L,A),\ mesa,\ u,\ y)),\\
\quad \lambda i\ x.\ \exists\, u \in Lset(i).\ u \in B \wedge (\exists\, y \in Lset(i).\ pair(\#\#Lset(i),\ u,\ y,\ x) \wedge\\
\qquad is\_wfrec\ (\#\#Lset(i),\ satisfies\_MH(\#\#Lset(i),A),\ mesa,\ u,\ y))]
\end{array}
$$

⟨*proof*⟩

lemma *formula_rec_replacement*:

— For the *transrec*

$$
[|n \in nat;\ L(A)|] ==>\ transrec\_replacement(L,\ satisfies\_MH(L,A),\ n)
$$

⟨*proof*⟩

9.4.6 The Lambda Replacement Case

lemma *formula_rec_lambda_replacement_Reflects*:

$$
\begin{array}{l}
REFLECTS\ [\lambda x.\ \exists\, u[L].\ u \in B\ \&\\
\qquad mem\_formula(L,u)\ \&\\
\qquad (\exists\, c[L].\\
\qquad\quad is\_formula\_case\\
\qquad\quad\ (L,\ satisfies\_is\_a(L,A),\ satisfies\_is\_b(L,A),\\
\qquad\quad\ satisfies\_is\_c(L,A,g),\ satisfies\_is\_d(L,A,g),\\
\qquad\quad\ u,\ c)\ \&\\
\qquad\quad pair(L,u,c,x)),\\
\quad \lambda i\ x.\ \exists\, u \in Lset(i).\ u \in B\ \&\ mem\_formula(\#\#Lset(i),u)\ \&\\
\qquad (\exists\, c \in Lset(i).\\
\qquad\quad is\_formula\_case
\end{array}
$$

(##*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*,
λ*x 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*⟩

9.5 Instantiating *M_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

10 Absoluteness for the Definable Powerset Function

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

10.1 Preliminary Internalizations

10.1.1 The Operator *is_formula_rec*

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

definition
formula_rec_fm :: [*i*, *i*, *i*]=>*i* **where**
formula_rec_fm(*mh*,*p*,*z*) ==
Exists(*Exists*(*Exists*(
And(*finite_ordinal_fm*(*2*),

$And(depth\_fm(p\#+3,2),$
$And(succ\_fm(2,1),$
$And(fun\_apply\_fm(0,p\#+3,z\#+3),\ is\_transrec\_fm(mh,1,0))))))))$

lemma *is_formula_rec_type* [*TC*]:
$[|\ p \in formula;\ x \in nat;\ z \in nat\ |]$
$==>\ formula\_rec\_fm(p,x,z) \in formula$
⟨*proof*⟩

lemma *sats_formula_rec_fm*:
assumes *MH_iff_sats*:
$!!a0\ a1\ a2\ a3\ a4\ a5\ a6\ a7\ a8\ a9\ a10.$
$[|a0{\in}A;\ a1{\in}A;\ a2{\in}A;\ a3{\in}A;\ a4{\in}A;\ a5{\in}A;\ a6{\in}A;\ a7{\in}A;\ a8{\in}A;\ a9{\in}A;$
$a10{\in}A|]$
$==>\ MH(a2,\ a1,\ a0) \longleftrightarrow$
$sats(A,\ p,\ Cons(a0,Cons(a1,Cons(a2,Cons(a3,$
$Cons(a4,Cons(a5,Cons(a6,Cons(a7,$
$Cons(a8,Cons(a9,Cons(a10,env))))))))))))$
shows
$[|x \in nat;\ z \in nat;\ env \in list(A)|]$
$==>\ sats(A,\ formula\_rec\_fm(p,x,z),\ env) \longleftrightarrow$
$is\_formula\_rec(\#\#A,\ MH,\ nth(x,env),\ nth(z,env))$
⟨*proof*⟩

lemma *formula_rec_iff_sats*:
assumes *MH_iff_sats*:
$!!a0\ a1\ a2\ a3\ a4\ a5\ a6\ a7\ a8\ a9\ a10.$
$[|a0{\in}A;\ a1{\in}A;\ a2{\in}A;\ a3{\in}A;\ a4{\in}A;\ a5{\in}A;\ a6{\in}A;\ a7{\in}A;\ a8{\in}A;\ a9{\in}A;$
$a10{\in}A|]$
$==>\ MH(a2,\ a1,\ a0) \longleftrightarrow$
$sats(A,\ p,\ Cons(a0,Cons(a1,Cons(a2,Cons(a3,$
$Cons(a4,Cons(a5,Cons(a6,Cons(a7,$
$Cons(a8,Cons(a9,Cons(a10,env))))))))))))$
shows
$[|nth(i,env) = x;\ nth(k,env) = z;$
$i \in nat;\ k \in nat;\ env \in list(A)|]$
$==>\ is\_formula\_rec(\#\#A,\ MH,\ x,\ z) \longleftrightarrow\ sats(A,\ formula\_rec\_fm(p,i,k),$
$env)$
⟨*proof*⟩

theorem *formula_rec_reflection*:
assumes *MH_reflection*:
$!!f'\ f\ g\ h.\ REFLECTS[\lambda x.\ MH(L,\ f'(x),\ f(x),\ g(x),\ h(x)),$
$\lambda i\ x.\ MH(\#\#Lset(i),\ f'(x),\ f(x),\ g(x),\ h(x))]$
shows $REFLECTS[\lambda x.\ is\_formula\_rec(L,\ MH(L,x),\ f(x),\ h(x)),$
$\lambda i\ x.\ is\_formula\_rec(\#\#Lset(i),\ MH(\#\#Lset(i),x),\ f(x),\ h(x))]$
⟨*proof*⟩

10.1.2 The Operator *is_satisfies*

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

lemma *is_satisfies_type* [*TC*]:
[| *x* ∈ *nat*; *y* ∈ *nat*; *z* ∈ *nat* |] ==> *satisfies_fm*(*x*,*y*,*z*) ∈ *formula*
⟨*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[λ*x*. *is_satisfies*(*L*,*f*(*x*),*g*(*x*),*h*(*x*)),
λ*i* *x*. *is_satisfies*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*))]
⟨*proof*⟩

10.2 Relativization of the Operator *DPow′*

lemma *DPow′_eq*:
DPow′(*A*) = {*z* . *ep* ∈ *list*(*A*) ∗ *formula*,
∃ *env* ∈ *list*(*A*). ∃ *p* ∈ *formula*.
ep = <*env*,*p*> & *z* = {*x*∈*A*. *sats*(*A*, *p*, *Cons*(*x*,*env*))}}
⟨*proof*⟩

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

definition
is_DPow_sats :: [*i*=>*o*,*i*,*i*,*i*] => *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*:

[| $M(A)$; $env \in list(A)$; $p \in formula$ |]
==> $Collect(A,\ is\_DPow\_sats(M,A,env,p)) =$
$\{x \in A.\ sats(A,\ p,\ Cons(x,env))\}$
⟨*proof*⟩

10.2.1 The Operator *is_DPow_sats*, Internalized

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

lemma *is_DPow_sats_type* [*TC*]:
[| $A \in nat$; $x \in nat$; $y \in nat$; $z \in nat$ |]
==> *DPow_sats_fm*(*A*,*x*,*y*,*z*) ∈ *formula*
⟨*proof*⟩

lemma *sats_DPow_sats_fm* [*simp*]:
[| $u \in nat$; $x \in nat$; $y \in nat$; $z \in nat$; $env \in list(A)$|]
==> $sats(A,\ DPow\_sats\_fm(u,x,y,z),\ env) \longleftrightarrow$
$is\_DPow\_sats(\#\#A,\ nth(u,env),\ nth(x,env),\ nth(y,env),\ nth(z,env))$
⟨*proof*⟩

lemma *DPow_sats_iff_sats*:
[| $nth(u,env) = nu$; $nth(x,env) = nx$; $nth(y,env) = ny$; $nth(z,env) = nz$;
$u \in nat$; $x \in nat$; $y \in nat$; $z \in nat$; $env \in list(A)$|]
==> $is\_DPow\_sats(\#\#A,nu,nx,ny,nz) \longleftrightarrow$
$sats(A,\ DPow\_sats\_fm(u,x,y,z),\ env)$
⟨*proof*⟩

theorem *DPow_sats_reflection*:
REFLECTS[λ*x*. *is_DPow_sats*(*L*,*f*(*x*),*g*(*x*),*h*(*x*),*g*′(*x*)),
λ*i* *x*. *is_DPow_sats*(##*Lset*(*i*),*f*(*x*),*g*(*x*),*h*(*x*),*g*′(*x*))]
⟨*proof*⟩

10.3 A Locale for Relativizing the Operator *DPow*′

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

lemma (**in** *M_DPow*) *sep′*:
[| *M*(*A*); *env* ∈ *list*(*A*); *p* ∈ *formula* |]
==> *separation*(*M*, λ*x*. *sats*(*A*, *p*, *Cons*(*x*,*env*)))
⟨*proof*⟩

lemma (**in** *M_DPow*) *rep′*:
M(*A*)
==> *strong_replacement* (*M*,
λ*ep z*. ∃ *env*∈*list*(*A*). ∃ *p*∈*formula*.
ep = <*env*,*p*> & *z* = {*x* ∈ *A* . *sats*(*A*, *p*, *Cons*(*x*, *env*))})
⟨*proof*⟩

lemma *univalent_pair_eq*:
univalent (*M*, *A*, λ*xy 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*⟩

10.4 Instantiating the Locale *M_DPow*

10.4.1 The Instance of Separation

lemma *DPow_separation*:
[| *L*(*A*); *env* ∈ *list*(*A*); *p* ∈ *formula* |]
==> *separation*(*L*, λ*x*. *is_DPow_sats*(*L*,*A*,*env*,*p*,*x*))
⟨*proof*⟩

10.4.2 The Instance of Replacement

lemma *DPow_replacement_Reflects*:
REFLECTS [λ*x*. ∃ *u*[*L*]. *u* ∈ *B* &
(∃ *env*[*L*]. ∃ *p*[*L*].
mem_formula(*L*,*p*) & *mem_list*(*L*,*A*,*env*) & *pair*(*L*,*env*,*p*,*u*) &
is_Collect (*L*, *A*, *is_DPow_sats*(*L*,*A*,*env*,*p*), *x*)),

$\lambda i\ x.\ \exists\, u \in Lset(i).\ u \in B\ \&$
$\quad (\exists\, env \in Lset(i).\ \exists\, p \in Lset(i).$
$\quad\ mem\_formula(\#\#Lset(i),p)\ \&\ mem\_list(\#\#Lset(i),A,env)\ \&$
$\quad\ pair(\#\#Lset(i),env,p,u)\ \&$
$\quad\ is\_Collect\ (\#\#Lset(i),\ A,\ is\_DPow\_sats(\#\#Lset(i),A,env,p),\ x))]$
⟨*proof*⟩

lemma *DPow_replacement*:
$L(A)$
$==>$ *strong_replacement* $(L,$
$\lambda ep\ z.\ \exists\, env[L].\ \exists\, p[L].\ mem\_formula(L,p)\ \&\ mem\_list(L,A,env)\ \&$
$pair(L,env,p,ep)\ \&$
$is\_Collect(L,\ A,\ \lambda x.\ is\_DPow\_sats(L,A,env,p,x),\ z))$
⟨*proof*⟩

10.4.3 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*]

10.4.4 The Operator *is_Collect*

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

definition
Collect_fm :: $[i,\ i,\ i]=>i$ **where**
Collect_fm$(A,is\_P,z) ==$
$Forall(Iff(Member(0,succ(z)),$
$And(Member(0,succ(A)),\ is\_P)))$

lemma *is_Collect_type* [*TC*]:
$[|\ is\_P \in formula;\ x \in nat;\ y \in nat\ |]$
$==>\ Collect\_fm(x,is\_P,y) \in formula$
⟨*proof*⟩

lemma *sats_Collect_fm*:
assumes *is_P_iff_sats*:
$!!a.\ a \in A ==> is\_P(a) \longleftrightarrow sats(A,\ p,\ Cons(a,\ env))$
shows
$[|x \in nat;\ y \in nat;\ env \in list(A)|]$
$==>\ sats(A,\ Collect\_fm(x,p,y),\ env) \longleftrightarrow$
$is\_Collect(\#\#A,\ nth(x,env),\ is\_P,\ nth(y,env))$
⟨*proof*⟩

lemma *Collect_iff_sats*:
assumes *is_P_iff_sats*:
!!*a*. *a* ∈ *A* ==> *is_P*(*a*) ⟷ *sats*(*A*, *p*, *Cons*(*a*, *env*))
shows
[| *nth*(*i*,*env*) = *x*; *nth*(*j*,*env*) = *y*;
i ∈ *nat*; *j* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *is_Collect*(##*A*, *x*, *is_P*, *y*) ⟷ *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*:
!!*h f g*. *REFLECTS*[λ*x*. *is_P*(*L*, *f*(*x*), *g*(*x*)),
λ*i x*. *is_P*(##*Lset*(*i*), *f*(*x*), *g*(*x*))]
shows *REFLECTS*[λ*x*. *is_Collect*(*L*, *f*(*x*), *is_P*(*L*,*x*), *g*(*x*)),
λ*i x*. *is_Collect*(##*Lset*(*i*), *f*(*x*), *is_P*(##*Lset*(*i*), *x*), *g*(*x*))]
⟨*proof*⟩

10.4.5 The Operator *is_Replace*

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

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

lemma *is_Replace_type* [*TC*]:
[| *is_P* ∈ *formula*; *x* ∈ *nat*; *y* ∈ *nat* |]
==> *Replace_fm*(*x*,*is_P*,*y*) ∈ *formula*
⟨*proof*⟩

lemma *sats_Replace_fm*:
assumes *is_P_iff_sats*:
!!*a b*. [|*a* ∈ *A*; *b* ∈ *A*|]
==> *is_P*(*a*,*b*) ⟷ *sats*(*A*, *p*, *Cons*(*a*,*Cons*(*b*,*env*)))
shows
[|*x* ∈ *nat*; *y* ∈ *nat*; *env* ∈ *list*(*A*)|]
==> *sats*(*A*, *Replace_fm*(*x*,*p*,*y*), *env*) ⟷
is_Replace(##*A*, *nth*(*x*,*env*), *is_P*, *nth*(*y*,*env*))
⟨*proof*⟩

lemma *Replace_iff_sats*:
assumes *is_P_iff_sats*:
!!*a b*. [|*a* ∈ *A*; *b* ∈ *A*|]

$$\Longrightarrow is\_P(a,b) \longleftrightarrow sats(A,\ p,\ Cons(a,Cons(b,env)))$$

shows

$$[|\ nth(i,env) = x;\ nth(j,env) = y;\ i \in nat;\ j \in nat;\ env \in list(A)|] \Longrightarrow is\_Replace(\#\#A,\ x,\ is\_P,\ y) \longleftrightarrow sats(A,\ Replace\_fm(i,p,j),\ env)$$

⟨*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*:

$$!!h\ f\ g.\ REFLECTS[\lambda x.\ is\_P(L,\ f(x),\ g(x),\ h(x)),\ \lambda i\ x.\ is\_P(\#\#Lset(i),\ f(x),\ g(x),\ h(x))]$$

shows $REFLECTS[\lambda x.\ is\_Replace(L,\ f(x),\ is\_P(L,x),\ g(x)),\ \lambda i\ x.\ is\_Replace(\#\#Lset(i),\ f(x),\ is\_P(\#\#Lset(i),\ x),\ g(x))]$

⟨*proof*⟩

10.4.6 The Operator *is_DPow′*, Internalized

definition

DPow′_fm :: $[i,i]\Rightarrow i$ **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*]:

$$[|\ x \in nat;\ y \in nat\ |] \Longrightarrow DPow'\_fm(x,y) \in formula$$

⟨*proof*⟩

lemma *sats_DPow′_fm* [*simp*]:

$$[|\ x \in nat;\ y \in nat;\ env \in list(A)|] \Longrightarrow sats(A,\ DPow'\_fm(x,y),\ env) \longleftrightarrow is\_DPow'(\#\#A,\ nth(x,env),\ nth(y,env))$$

⟨*proof*⟩

lemma *DPow′_iff_sats*:

$$[|\ nth(i,env) = x;\ nth(j,env) = y;\ i \in nat;\ j \in nat;\ env \in list(A)|] \Longrightarrow is\_DPow'(\#\#A,\ x,\ y) \longleftrightarrow sats(A,\ DPow'\_fm(i,j),\ env)$$

⟨*proof*⟩

theorem *DPow′_reflection*:

$$REFLECTS[\lambda x.\ is\_DPow'(L,f(x),g(x)),$$

$\lambda i\ x.\ is\_DPow'(\#\#Lset(i),f(x),g(x))]$
⟨*proof*⟩

10.5 A Locale for Relativizing the Operator *Lset*

definition
transrec_body :: $[i=>o,i,i,i,i] => o$ **where**
transrec_body$(M,g,x) ==$
$\lambda y\ z.\ \exists\, gy[M].\ y \in x\ \&\ fun\_apply(M,g,y,gy)\ \&\ is\_DPow'(M,gy,z)$

lemma (**in** *M_DPow*) *transrec_body_abs*:
$[|M(x);\ M(g);\ M(z)|]$
$==>$ *transrec_body*$(M,g,x,y,z) \longleftrightarrow y \in x\ \&\ z = DPow'(g`y)$
⟨*proof*⟩

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

lemma (**in** *M_Lset*) *strong_rep'*:
$[|M(x);\ M(g)|]$
$==>$ *strong_replacement*$(M,\ \lambda y\ z.\ y \in x\ \&\ z = DPow'(g`y))$
⟨*proof*⟩

lemma (**in** *M_Lset*) *DPow_apply_closed*:
$[|M(f);\ M(x);\ y \in x|] ==> M(DPow'(f`y))$
⟨*proof*⟩

lemma (**in** *M_Lset*) *RepFun_DPow_apply_closed*:
$[|M(f);\ M(x)|] ==> M(\{DPow'(f`y).\ y \in x\})$
⟨*proof*⟩

lemma (**in** *M_Lset*) *RepFun_DPow_abs*:
$[|M(x);\ M(f);\ M(r)\ |]$
$==>$ *is_Replace*$(M,\ x,\ \lambda y\ z.$ *transrec_body*$(M,f,x,y,z),\ r) \longleftrightarrow$
$r = \{DPow'(f`y).\ y \in x\}$
⟨*proof*⟩

lemma (**in** *M_Lset*) *transrec_rep'*:
$M(i) ==>$ *transrec_replacement*$(M,\ \lambda x\ f\ u.\ u = (\bigcup y \in x.\ DPow'(f\ `\ y)),\ i)$
⟨*proof*⟩

Relativization of the Operator *Lset*

definition

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

lemma (**in** *M_Lset*) *Lset_abs*:
[|*Ord*(*i*); *M*(*i*); *M*(*z*)|]
==> *is_Lset*(*M*,*i*,*z*) ⟷ *z* = *Lset*(*i*)
⟨*proof*⟩

lemma (**in** *M_Lset*) *Lset_closed*:
[|*Ord*(*i*); *M*(*i*)|] ==> *M*(*Lset*(*i*))
⟨*proof*⟩

10.6 Instantiating the Locale *M_Lset*

10.6.1 The First Instance of Replacement

lemma *strong_rep_Reflects*:
REFLECTS [λ*u*. ∃ *v*[*L*]. *v* ∈ *B* & (∃ *gy*[*L*].
v ∈ *x* & *fun_apply*(*L*,*g*,*v*,*gy*) & *is_DPow'*(*L*,*gy*,*u*)),
λ*i u*. ∃ *v* ∈ *Lset*(*i*). *v* ∈ *B* & (∃ *gy* ∈ *Lset*(*i*).
v ∈ *x* & *fun_apply*(##*Lset*(*i*),*g*,*v*,*gy*) & *is_DPow'*(##*Lset*(*i*),*gy*,*u*))]
⟨*proof*⟩

lemma *strong_rep*:
[|*L*(*x*); *L*(*g*)|] ==> *strong_replacement*(*L*, λ*y z*. *transrec_body*(*L*,*g*,*x*,*y*,*z*))
⟨*proof*⟩

10.6.2 The Second Instance of Replacement

lemma *transrec_rep_Reflects*:
REFLECTS [λ*x*. ∃ *v*[*L*]. *v* ∈ *B* &
(∃ *y*[*L*]. *pair*(*L*,*v*,*y*,*x*) &
is_wfrec (*L*, λ*x f u*. ∃ *r*[*L*].
is_Replace (*L*, *x*, λ*y z*.
∃ *gy*[*L*]. *y* ∈ *x* & *fun_apply*(*L*,*f*,*y*,*gy*) &
is_DPow'(*L*,*gy*,*z*), *r*) & *big_union*(*L*,*r*,*u*), *mr*, *v*, *y*)),
λ*i x*. ∃ *v* ∈ *Lset*(*i*). *v* ∈ *B* &
(∃ *y* ∈ *Lset*(*i*). *pair*(##*Lset*(*i*),*v*,*y*,*x*) &
is_wfrec (##*Lset*(*i*), λ*x f u*. ∃ *r* ∈ *Lset*(*i*).
is_Replace (##*Lset*(*i*), *x*, λ*y z*.
∃ *gy* ∈ *Lset*(*i*). *y* ∈ *x* & *fun_apply*(##*Lset*(*i*),*f*,*y*,*gy*) &
is_DPow'(##*Lset*(*i*),*gy*,*z*), *r*) &
big_union(##*Lset*(*i*),*r*,*u*), *mr*, *v*, *y*))]
⟨*proof*⟩

lemma *transrec_rep*:
[|*L*(*j*)|]

$==>$ *transrec_replacement*(L, $\lambda x\ f\ u$.
$\exists\, r[L]$. *is_Replace*(L, x, *transrec_body*(L,f,x), r) &
big_union(L, r, u), j)
⟨*proof*⟩

10.6.3 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*]

10.7 The Notion of Constructible Set

definition
constructible :: [i=>o,i] => o **where**
constructible(M,x) ==
$\exists\, i[M]$. $\exists\, Li[M]$. *ordinal*(M,i) & *is_Lset*(M,i,Li) & $x \in Li$

theorem *V_equals_L_in_L*:
$L(x) \longleftrightarrow$ *constructible*(L,x)
⟨*proof*⟩

end

11 Automatic synthesis of formulas

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

and
for
and
from_definition
and
assuming
and
intermediate

begin

named_theorems *fm_definitions Definitions of synthetized formulas.*

named_theorems *iff_sats Theorems for synthetising formulas.*

named_theorems *arity Theorems for arity of formulas.*

named_theorems *arity_aux Auxiliary theorems for calculating arities.*

⟨*ML*⟩

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

end

12 Aids to internalize formulas

theory *Internalizations*
imports
DPow_absolute
Synthetic_Definition
Nat_Miscellanea
begin

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

definition
infinity_ax :: $(i \Rightarrow o) \Rightarrow o$ **where**
infinity_ax$(M) \equiv$
$(\exists\, I[M].\ (\exists\, z[M].\ empty(M,z) \wedge z{\in}I) \wedge (\forall\, y[M].\ y{\in}I \longrightarrow (\exists\, sy[M].\ successor(M,y,sy) \wedge sy{\in}I)))$

definition
wellfounded_trancl :: $[i{=>}o,i,i,i] => o$ **where**
wellfounded_trancl$(M,Z,r,p) \equiv$
$\exists\, w[M].\ \exists\, wx[M].\ \exists\, rp[M].$
$w \in Z\ \&\ pair(M,w,p,wx)\ \&\ tran\_closure(M,r,rp)\ \&\ wx \in rp$

lemma *empty_intf* :

infinity_ax(*M*) ⟹
(∃ *z*[*M*]. *empty*(*M*,*z*))
⟨*proof*⟩

lemma *Transset_intf* :
Transset(*M*) ⟹ *y*∈*x* ⟹ *x* ∈ *M* ⟹ *y* ∈ *M*
⟨*proof*⟩

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

lemma (**in** *M_basic*) *choice_ax_abs* :
choice_ax(*M*) ⟷ (∀ *x*[*M*]. ∃ *a*[*M*]. ∃ *f*[*M*]. *Ord*(*a*) ∧ *f* ∈ *surj*(*a*,*x*))
⟨*proof*⟩

Setting up notation for internalized formulas

abbreviation
dec10 :: *i* (‹*10*›) **where** *10* ≡ *succ*(*9*)
abbreviation
dec11 :: *i* (‹*11*›) **where** *11* ≡ *succ*(*10*)
abbreviation
dec12 :: *i* (‹*12*›) **where** *12* ≡ *succ*(*11*)
abbreviation
dec13 :: *i* (‹*13*›) **where** *13* ≡ *succ*(*12*)
abbreviation
dec14 :: *i* (‹*14*›) **where** *14* ≡ *succ*(*13*)
abbreviation
dec15 :: *i* (‹*15*›) **where** *15* ≡ *succ*(*14*)
abbreviation
dec16 :: *i* (‹*16*›) **where** *16* ≡ *succ*(*15*)
abbreviation
dec17 :: *i* (‹*17*›) **where** *17* ≡ *succ*(*16*)
abbreviation
dec18 :: *i* (‹*18*›) **where** *18* ≡ *succ*(*17*)
abbreviation
dec19 :: *i* (‹*19*›) **where** *19* ≡ *succ*(*18*)
abbreviation
dec20 :: *i* (‹*20*›) **where** *20* ≡ *succ*(*19*)
abbreviation
dec21 :: *i* (‹*21*›) **where** *21* ≡ *succ*(*20*)
abbreviation
dec22 :: *i* (‹*22*›) **where** *22* ≡ *succ*(*21*)
abbreviation
dec23 :: *i* (‹*23*›) **where** *23* ≡ *succ*(*22*)
abbreviation
dec24 :: *i* (‹*24*›) **where** *24* ≡ *succ*(*23*)
abbreviation
dec25 :: *i* (‹*25*›) **where** *25* ≡ *succ*(*24*)

abbreviation
  dec26 :: *i* (‹*26*›) **where** *26* ≡ *succ*(*25*)
abbreviation
  dec27 :: *i* (‹*27*›) **where** *27* ≡ *succ*(*26*)
abbreviation
  dec28 :: *i* (‹*28*›) **where** *28* ≡ *succ*(*27*)
abbreviation
  dec29 :: *i* (‹*29*›) **where** *29* ≡ *succ*(*28*)

notation *Member* (‹·_ ∈/ _·›)
notation *Equal* (‹·_ =/ _·›)
notation *Nand* (‹·¬′(_ ∧/ _′)·›)
notation *And* (‹·_ ∧/ _·›)
notation *Or* (‹·_ ∨/ _·›)
notation *Iff* (‹·_ ↔/ _·›)
notation *Implies* (‹·_ →/ _·›)
notation *Neg* (‹·¬_·›)
notation *Forall* (‹′(·∀(/_)·′)›)
notation *Exists* (‹′(·∃(/_)·′)›)

notation *subset_fm* (‹·_ ⊆/ _·›)
notation *succ_fm* (‹·*succ*′(_′) *is* _·›)
notation *empty_fm* (‹·_ *is empty*·›)
notation *fun_apply_fm* (‹·_‘_ *is* _·›)
notation *big_union_fm* (‹·⋃_ *is* _·›)
notation *upair_fm* (‹·{_,_} *is* _ ·›)
notation *ordinal_fm* (‹·_ *is ordinal*·›)

notation *pair_fm* (‹·⟨_,_⟩ *is* _ ·›)
notation *composition_fm* (‹·_ ∘ _ *is* _ ·›)
notation *domain_fm* (‹·*dom*′(_′) *is* _ ·›)
notation *range_fm* (‹·*ran*′(_′) *is* _ ·›)
notation *union_fm* (‹·_ ∪ _ *is* _ ·›)
notation *image_fm* (‹·_ ‘‘ _ *is* _ ·›)
notation *pre_image_fm* (‹·_ -‘‘ _ *is* _ ·›)
notation *field_fm* (‹·*fld*′(_′) *is* _ ·›)
notation *cons_fm* (‹·*cons*′(_,_′) *is* _ ·›)
notation *number1_fm* (‹·_ *is the number one*·›)
notation *function_fm* (‹·_ *is funct*·›)
notation *relation_fm* (‹·_ *is relat*·›)
notation *restriction_fm* (‹·_ ↾ _ *is* _ ·›)
notation *transset_fm* (‹·_ *is transitive*·›)
notation *limit_ordinal_fm* (‹·_ *is limit*·›)
notation *finite_ordinal_fm* (‹·_ *is finite ord*·›)
notation *omega_fm* (‹·_ *is* ω·›)
notation *cartprod_fm* (‹·_ × _ *is* _·›)
notation *Memrel_fm* (‹·*Memrel*′(_′) *is* _·›)
notation *quasinat_fm* (‹·_ *is qnat*·›)

notation *Inl_fm* (‹·*Inl′*(_′) *is* _ ·›)
notation *Inr_fm* (‹·*Inr′*(_′) *is* _ ·›)
notation *pred_set_fm* (‹·_-*predecessors of* _ *are* _·›)

abbreviation
fm_typedfun :: [*i*,*i*,*i*] ⇒ *i* (‹·_ : _ → _·›) **where**
fm_typedfun(*f*,*A*,*B*) ≡ *typed_function_fm*(*A*,*B*,*f*)

abbreviation
fm_surjection :: [*i*,*i*,*i*] ⇒ *i* (‹·_ *surjects* _ *to* _·›) **where**
fm_surjection(*f*,*A*,*B*) ≡ *surjection_fm*(*A*,*B*,*f*)

abbreviation
fm_injection :: [*i*,*i*,*i*] ⇒ *i* (‹·_ *injects* _ *to* _·›) **where**
fm_injection(*f*,*A*,*B*) ≡ *injection_fm*(*A*,*B*,*f*)

abbreviation
fm_bijection :: [*i*,*i*,*i*] ⇒ *i* (‹·_ *bijects* _ *to* _·›) **where**
fm_bijection(*f*,*A*,*B*) ≡ *bijection_fm*(*A*,*B*,*f*)

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

lemma *nth_closed* :
assumes *env*∈*list*(*A*) *0*∈*A*
shows *nth*(*n*,*env*)∈*A*
⟨*proof*⟩

lemma *conj_setclass_model_iff_sats* [*iff_sats*]:
[| *0* ∈ *A*; *nth*(*i*,*env*) = *x*; *env* ∈ *list*(*A*);
P ⟷ *sats*(*A*,*p*,*env*); *env* ∈ *list*(*A*) |]
==> (*P* ∧ (##*A*)(*x*)) ⟷ *sats*(*A*, *p*, *env*)
[| *0* ∈ *A*; *nth*(*i*,*env*) = *x*; *env* ∈ *list*(*A*);
P ⟷ *sats*(*A*,*p*,*env*); *env* ∈ *list*(*A*) |]
==> ((##*A*)(*x*) ∧ *P*) ⟷ *sats*(*A*, *p*, *env*)
⟨*proof*⟩

lemma *conj_mem_model_iff_sats* [*iff_sats*]:
[| *0* ∈ *A*; *nth*(*i*,*env*) = *x*; *env* ∈ *list*(*A*);
P ⟷ *sats*(*A*,*p*,*env*); *env* ∈ *list*(*A*) |]
==> (*P* ∧ *x* ∈ *A*) ⟷ *sats*(*A*, *p*, *env*)
[| *0* ∈ *A*; *nth*(*i*,*env*) = *x*; *env* ∈ *list*(*A*);
P ⟷ *sats*(*A*,*p*,*env*); *env* ∈ *list*(*A*) |]
==> (*x* ∈ *A* ∧ *P*) ⟷ *sats*(*A*, *p*, *env*)
⟨*proof*⟩

lemma *mem_model_iff_sats* [*iff_sats*]:
[| $0 \in A$; $nth(i,env) = x$; $env \in list(A)$|]
==> $(x \in A) \longleftrightarrow sats(A, Exists(Equal(0,0)), env)$
⟨*proof*⟩

lemma *subset_iff_sats*[*iff_sats*]:
$nth(i, env) = x \Longrightarrow nth(j, env) = y \Longrightarrow i \in nat \Longrightarrow j \in nat \Longrightarrow$
$env \in list(A) \Longrightarrow subset(\#\#A, x, y) \longleftrightarrow sats(A, subset\_fm(i, j), env)$
⟨*proof*⟩

lemma *not_mem_model_iff_sats* [*iff_sats*]:
[| $0 \in A$; $nth(i,env) = x$; $env \in list(A)$|]
==> $(\forall\ x\ .\ x \notin A) \longleftrightarrow sats(A, Neg(Exists(Equal(0,0))), env)$
⟨*proof*⟩

lemma *top_iff_sats* [*iff_sats*]:
$env \in list(A) \Longrightarrow 0 \in A \Longrightarrow sats(A, Exists(Equal(0,0)), env)$
⟨*proof*⟩

lemma *prefix1_iff_sats*[*iff_sats*]:
assumes
$x \in nat\ env \in list(A)\ 0 \in A\ a \in A$
shows
$a = nth(x,env) \longleftrightarrow sats(A, Equal(0,x+_\omega 1), Cons(a,env))$
$nth(x,env) = a \longleftrightarrow sats(A, Equal(x+_\omega 1,0), Cons(a,env))$
$a \in nth(x,env) \longleftrightarrow sats(A, Member(0,x+_\omega 1), Cons(a,env))$
$nth(x,env) \in a \longleftrightarrow sats(A, Member(x+_\omega 1,0), Cons(a,env))$
⟨*proof*⟩

lemma *prefix2_iff_sats*[*iff_sats*]:
assumes
$x \in nat\ env \in list(A)\ 0 \in A\ a \in A\ b \in A$
shows
$b = nth(x,env) \longleftrightarrow sats(A, Equal(1,x+_\omega 2), Cons(a,Cons(b,env)))$
$nth(x,env) = b \longleftrightarrow sats(A, Equal(x+_\omega 2,1), Cons(a,Cons(b,env)))$
$b \in nth(x,env) \longleftrightarrow sats(A, Member(1,x+_\omega 2), Cons(a,Cons(b,env)))$
$nth(x,env) \in b \longleftrightarrow sats(A, Member(x+_\omega 2,1), Cons(a,Cons(b,env)))$
⟨*proof*⟩

lemma *prefix3_iff_sats*[*iff_sats*]:
assumes
$x \in nat\ env \in list(A)\ 0 \in A\ a \in A\ b \in A\ c \in A$
shows
$c = nth(x,env) \longleftrightarrow sats(A, Equal(2,x+_\omega 3), Cons(a,Cons(b,Cons(c,env))))$
$nth(x,env) = c \longleftrightarrow sats(A, Equal(x+_\omega 3,2), Cons(a,Cons(b,Cons(c,env))))$
$c \in nth(x,env) \longleftrightarrow sats(A, Member(2,x+_\omega 3), Cons(a,Cons(b,Cons(c,env))))$
$nth(x,env) \in c \longleftrightarrow sats(A, Member(x+_\omega 3,2), Cons(a,Cons(b,Cons(c,env))))$
⟨*proof*⟩

lemmas *FOL_sats_iff = sats_Nand_iff sats_Forall_iff sats_Neg_iff sats_And_iff sats_Or_iff sats_Implies_iff sats_Iff_iff sats_Exists_iff*

lemma *nth_ConsI*: $[\![nth(n,l) = x;\ n \in nat]\!] \Longrightarrow nth(succ(n),\ Cons(a,l)) = x$
⟨*proof*⟩

lemmas *nth_rules = nth_0 nth_ConsI nat_0I nat_succI*
lemmas *sep_rules = nth_0 nth_ConsI FOL_iff_sats function_iff_sats fun_plus_iff_sats successor_iff_sats omega_iff_sats FOL_sats_iff Replace_iff_sats*

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

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

lemmas *formulas_def [fm_definitions] = fm_defs*
is_iterates_fm_def iterates_MH_fm_def is_wfrec_fm_def is_recfun_fm_def is_transrec_fm_def
is_nat_case_fm_def quasinat_fm_def number1_fm_def ordinal_fm_def finite_ordinal_fm_def cartprod_fm_def sum_fm_def Inr_fm_def Inl_fm_def
formula_functor_fm_def
Memrel_fm_def transset_fm_def subset_fm_def pre_image_fm_def restriction_fm_def
list_functor_fm_def tl_fm_def quasilist_fm_def Cons_fm_def Nil_fm_def

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

lemmas *more_iff_sats [iff_sats] = rtran_closure_iff_sats tran_closure_iff_sats is_eclose_iff_sats Inl_iff_sats Inr_iff_sats fun_apply_iff_sats cartprod_iff_sats Collect_iff_sats*

end

13 The binder *Least*

theory *Least*
imports
Internalizations

begin

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

lemma *Least_Ord*: $(\mu\ \alpha.\ R(\alpha)) = (\mu\ \alpha.\ Ord(\alpha) \wedge R(\alpha))$

⟨*proof*⟩

lemma *Ord_Least_cong*:
assumes $\bigwedge y.\ Ord(y) \Longrightarrow R(y) \longleftrightarrow Q(y)$
shows $(\mu\ \alpha.\ R(\alpha)) = (\mu\ \alpha.\ Q(\alpha))$
⟨*proof*⟩

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

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

lemma *least_fm_type*[*TC*] :$i \in nat \Longrightarrow q∈formula \Longrightarrow least\_fm(q,i) \in formula$
⟨*proof*⟩

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

lemma *sats_least_fm* :
assumes *p_iff_sats*:
$\bigwedge a.\ a \in A \Longrightarrow P(a) \longleftrightarrow sats(A,\ p,\ Cons(a,\ env))$
shows
$[\![y \in nat;\ env \in list(A)\ ;\ 0∈A]\!]$
$\Longrightarrow sats(A,\ least\_fm(p,y),\ env) \longleftrightarrow$
$least(\#\#A,\ P,\ nth(y,env))$
⟨*proof*⟩

lemma *least_iff_sats* [*iff_sats*]:
assumes *is_Q_iff_sats*:
$\bigwedge a.\ a \in A \Longrightarrow is\_Q(a) \longleftrightarrow sats(A,\ q,\ Cons(a,env))$
shows
$[\![nth(j,env) = y;\ j \in nat;\ env \in list(A);\ 0∈A]\!]$
$\Longrightarrow least(\#\#A,\ is\_Q,\ y) \longleftrightarrow sats(A,\ least\_fm(q,j),\ env)$
⟨*proof*⟩

lemma *least_conj*: $a∈M \Longrightarrow least(\#\#M,\ \lambda x.\ x∈M \wedge Q(x),a) \longleftrightarrow least(\#\#M,Q,a)$
⟨*proof*⟩

context *M_trivial*
begin

13.1 Uniqueness, absoluteness and closure under *Least*

lemma *unique_least*:
assumes $M(a)$ $M(b)$ $least(M,Q,a)$ $least(M,Q,b)$
shows $a=b$
⟨*proof*⟩

lemma *least_abs*:
assumes $\bigwedge x.\ Q(x) \Longrightarrow Ord(x) \Longrightarrow \exists\, y[M].\ Q(y) \wedge Ord(y)\ M(a)$
shows $least(M,Q,a) \longleftrightarrow a = (\mu\ x.\ Q(x))$
⟨*proof*⟩

lemma *Least_closed*:
assumes $\bigwedge x.\ Q(x) \Longrightarrow Ord(x) \Longrightarrow \exists\, y[M].\ Q(y) \wedge Ord(y)$
shows $M(\mu\ x.\ Q(x))$
⟨*proof*⟩

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

lemma *least_abs'*:
assumes $\bigwedge x.\ Q(x) \Longrightarrow M(x)\ M(a)$
shows $least(M,Q,a) \longleftrightarrow a = (\mu\ x.\ Q(x))$
⟨*proof*⟩

lemma *Least_closed'*:
assumes $\bigwedge x.\ Q(x) \Longrightarrow M(x)$
shows $M(\mu\ x.\ Q(x))$
⟨*proof*⟩

end — *M_trivial*

end

14 Fully relational versions of higher order constructs

theory *Higher_Order_Constructs*
imports
Recursion_Thms
Least
begin

syntax
_sats :: $[i,\ i,\ i] \Rightarrow o$ $(\langle(\_,\ \_ \models \_)\rangle\ [36,36,36]\ 25)$
syntax_consts
_sats $\rightleftharpoons$ *sats*
translations
$(M,env \models \varphi) \rightleftharpoons CONST\ sats(M,\varphi,env)$

definition

is_If :: [*i*⇒*o*,*o*,*i*,*i*,*i*] ⇒ *o* **where**
is_If(*M*,*b*,*t*,*f*,*r*) ≡ (*b* ⟶ *r*=*t*) ∧ (¬*b* ⟶ *r*=*f*)

lemma (**in** *M_trans*) *If_abs*:
is_If(*M*,*b*,*t*,*f*,*r*) ⟷ *r* = *If*(*b*,*t*,*f*)
⟨*proof*⟩

definition
is_If_fm :: [*i*,*i*,*i*,*i*] ⇒ *i* **where**
is_If_fm(*φ*,*t*,*f*,*r*) ≡ *Or*(*And*(*φ*,*Equal*(*t*,*r*)),*And*(*Neg*(*φ*),*Equal*(*f*,*r*)))

lemma *is_If_fm_type* [*TC*]: *φ* ∈ *formula* ⟹ *t* ∈ *nat* ⟹ *f* ∈ *nat* ⟹ *r* ∈ *nat* ⟹
is_If_fm(*φ*,*t*,*f*,*r*) ∈ *formula*
⟨*proof*⟩

lemma *sats_is_If_fm*:
assumes *Qsats*: *Q* ⟷ *A*, *env* ⊨ *φ* *env* ∈ *list*(*A*)
shows *is_If*(##*A*, *Q*, *nth*(*t*, *env*), *nth*(*f*, *env*), *nth*(*r*, *env*)) ⟷ *A*, *env* ⊨ *is_If_fm*(*φ*,*t*,*f*,*r*)
⟨*proof*⟩

lemma *is_If_fm_iff_sats* [*iff_sats*]:
assumes *Qsats*: *Q* ⟷ *A*, *env* ⊨ *φ* **and**
nth(*t*, *env*) = *ta* *nth*(*f*, *env*) = *fa* *nth*(*r*, *env*) = *ra*
t ∈ *nat* *f* ∈ *nat* *r* ∈ *nat* *env* ∈ *list*(*A*)
shows *is_If*(##*A*,*Q*,*ta*,*fa*,*ra*) ⟷ *A*, *env* ⊨ *is_If_fm*(*φ*,*t*,*f*,*r*)
⟨*proof*⟩

lemma *arity_is_If_fm* [*arity*]:
φ ∈ *formula* ⟹ *t* ∈ *nat* ⟹ *f* ∈ *nat* ⟹ *r* ∈ *nat* ⟹
arity(*is_If_fm*(*φ*, *t*, *f*, *r*)) = *arity*(*φ*) ∪ *succ*(*t*) ∪ *succ*(*r*) ∪ *succ*(*f*)
⟨*proof*⟩

definition
is_The :: [*i*⇒*o*,*i*⇒*o*,*i*] ⇒ *o* **where**
is_The(*M*,*Q*,*i*) ≡ (*Q*(*i*) ∧ (∃ *x*[*M*]. *Q*(*x*) ∧ (∀ *y*[*M*]. *Q*(*y*) ⟶ *y* = *x*))) ∨
(¬(∃ *x*[*M*]. *Q*(*x*) ∧ (∀ *y*[*M*]. *Q*(*y*) ⟶ *y* = *x*))) ∧ *empty*(*M*,*i*)

lemma (**in** *M_trans*) *The_abs*:
assumes ⋀*x*. *Q*(*x*) ⟹ *M*(*x*) *M*(*a*)
shows *is_The*(*M*,*Q*,*a*) ⟷ *a* = (*THE x*. *Q*(*x*))
⟨*proof*⟩

definition
is_recursor :: [*i*⇒*o*,*i*,[*i*,*i*,*i*]⇒*o*,*i*,*i*] ⇒*o* **where**
is_recursor(*M*,*a*,*is_b*,*k*,*r*) ≡ *is_transrec*(*M*, λ*n f ntc*. *is_nat_case*(*M*,*a*,
λ*m bmfm*.
∃*fm*[*M*]. *fun_apply*(*M*,*f*,*m*,*fm*) ∧ *is_b*(*m*,*fm*,*bmfm*),*n*,*ntc*),*k*,*r*)

lemma (**in** *M_eclose*) *recursor_abs*:
assumes *Ord*(*k*) **and**
types: *M*(*a*) *M*(*k*) *M*(*r*) **and**
b_iff: ⋀*m f bmf*. *M*(*m*) ⟹ *M*(*f*) ⟹ *M*(*bmf*) ⟹ *is_b*(*m*,*f*,*bmf*) ⟷ *bmf* = *b*(*m*,*f*) **and**
b_closed: ⋀*m f bmf*. *M*(*m*) ⟹ *M*(*f*) ⟹ *M*(*b*(*m*,*f*)) **and**
repl: *transrec_replacement*(*M*, λ*n f ntc*. *is_nat_case*(*M*, *a*,
λ*m bmfm*. ∃*fm*[*M*]. *fun_apply*(*M*, *f*, *m*, *fm*) ∧ *is_b*(*m*, *fm*, *bmfm*), *n*, *ntc*), *k*)
shows
is_recursor(*M*,*a*,*is_b*,*k*,*r*) ⟷ *r* = *recursor*(*a*,*b*,*k*)
⟨*proof*⟩

definition
is_wfrec_on :: [*i*=>*o*,[*i*,*i*,*i*]=>*o*,*i*,*i*,*i*, *i*] => *o* **where**
is_wfrec_on(*M*,*MH*,*A*,*r*,*a*,*z*) == *is_wfrec*(*M*,*MH*,*r*,*a*,*z*)

lemma (**in** *M_trancl*) *trans_wfrec_on_abs*:
[|*wf*(*r*); *trans*(*r*); *relation*(*r*); *M*(*r*); *M*(*a*); *M*(*z*);
wfrec_replacement(*M*,*MH*,*r*); *relation2*(*M*,*MH*,*H*);
∀*x*[*M*]. ∀*g*[*M*]. *function*(*g*) ⟶ *M*(*H*(*x*,*g*));
r-‘‘{*a*}⊆*A*; *a* ∈ *A*|]
==> *is_wfrec_on*(*M*,*MH*,*A*,*r*,*a*,*z*) ⟷ *z*=*wfrec*[*A*](*r*,*a*,*H*)
⟨*proof*⟩

end

15 Automatic relativization of terms and formulas

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

theory *Relativization*
imports
Eclose_Absolute
Higher_Order_Constructs
keywords
relativize :: *thy_decl* % *ML*
and
relativize_tm :: *thy_decl* % *ML*
and
reldb_add :: *thy_decl* % *ML*

and
reldb_rem :: *thy_decl* % *ML*
and
relationalize :: *thy_decl* % *ML*
and
rel_closed :: *thy_goal_stmt* % *ML*
and
is_iff_rel :: *thy_goal_stmt* % *ML*
and
univalent :: *thy_goal_stmt* % *ML*
and
absolute
and
functional
and
relational
and
external
and
for

begin

⟨*ML*⟩

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

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

M_trivial.nat_case_abs

M_trivial.omega_abs
M_basic.sum_abs
M_trivial.Inl_abs
M_trivial.Inr_abs
M_basic.converse_abs
M_basic.vimage_abs

M_trans.domain_abs
M_trans.range_abs
M_basic.field_abs

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

M_trivial.list_case_abs
M_trivial.hd_abs
M_trivial.tl_abs
M_trivial.least_abs'
M_eclose.transrec_abs
M_trans.If_abs
M_trans.The_abs
M_eclose.recursor_abs
M_trancl.trans_wfrec_abs
M_trancl.trans_wfrec_on_abs

lemmas *datatype_abs* =
M_eclose.is_eclose_n_abs
M_eclose.eclose_abs
M_trivial.Member_abs
M_trivial.Equal_abs
M_trivial.Nand_abs
M_trivial.Forall_abs

declare *relative_abs*[*absolut*]

⟨*ML*⟩

declare *relative_abs*[*Rel*]

declare *datatype_abs*[*Rel*]

⟨*ML*⟩

end
theory *Discipline_Base*
imports
ZF-Constructible.Rank
M_Basic_No_Repl

Relativization
ZF_Miscellanea

begin

hide_const (**open**) *Order.pred*
declare [[*syntax_ambiguity_warning* = *false*]]

15.1 Discipline of relativization of basic concepts

definition
is_singleton :: [*i*⇒*o*,*i*,*i*] ⇒ *o* **where**
is_singleton(*A*,*x*,*z*) ≡ ∃ *c*[*A*]. *empty*(*A*,*c*) ∧ *is_cons*(*A*,*x*,*c*,*z*)

lemma (**in** *M_trivial*) *singleton_abs*[*simp*] :
⟦ *M*(*x*) ; *M*(*s*) ⟧ ⟹ *is_singleton*(*M*,*x*,*s*) ⟷ *s* = {*x*}
⟨*proof*⟩

⟨*ML*⟩
notation *singleton_fm* (‹·{_} *is* _·›)

lemmas (**in** *M_trivial*) *singleton_closed* = *singleton_in_MI*

lemma (**in** *M_trivial*) *Upair_closed*[*simp*]: *M*(*a*) ⟹ *M*(*b*) ⟹ *M*(*Upair*(*a*,*b*))
⟨*proof*⟩

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

named_theorems *trans_closed*

definition
is_hcomp :: [*i*⇒*o*,*i*⇒*i*⇒*o*,*i*⇒*i*⇒*o*,*i*,*i*] ⇒ *o* **where**
is_hcomp(*M*,*is_f*,*is_g*,*a*,*w*) ≡ ∃ *z*[*M*]. *is_g*(*a*,*z*) ∧ *is_f*(*z*,*w*)

lemma (**in** *M_trivial*) *is_hcomp_abs*:
assumes
is_f_abs:⋀*a z*. *M*(*a*) ⟹ *M*(*z*) ⟹ *is_f*(*a*,*z*) ⟷ *z* = *f*(*a*) **and**
is_g_abs:⋀*a z*. *M*(*a*) ⟹ *M*(*z*) ⟹ *is_g*(*a*,*z*) ⟷ *z* = *g*(*a*) **and**
g_closed:⋀*a*. *M*(*a*) ⟹ *M*(*g*(*a*))
M(*a*) *M*(*w*)
shows
is_hcomp(*M*,*is_f*,*is_g*,*a*,*w*) ⟷ *w* = *f*(*g*(*a*))
⟨*proof*⟩

definition
hcomp_fm :: [*i*⇒*i*⇒*i*,*i*⇒*i*⇒*i*,*i*,*i*] ⇒ *i* **where**
hcomp_fm(*pf*,*pg*,*a*,*w*) ≡ *Exists*(*And*(*pg*(*succ*(*a*),0),*pf*(0,*succ*(*w*))))

lemma *sats_hcomp_fm*:

assumes
f_iff_sats:$\bigwedge a\ b\ z.\ a{\in}nat \Longrightarrow b{\in}nat \Longrightarrow z{\in}M \Longrightarrow$
$is\_f(nth(a,Cons(z,env)),nth(b,Cons(z,env))) \longleftrightarrow sats(M,pf(a,b),Cons(z,env))$
and
g_iff_sats:$\bigwedge a\ b\ z.\ a{\in}nat \Longrightarrow b{\in}nat \Longrightarrow z{\in}M \Longrightarrow$
$is\_g(nth(a,Cons(z,env)),nth(b,Cons(z,env))) \longleftrightarrow sats(M,pg(a,b),Cons(z,env))$
and
$a{\in}nat\ w{\in}nat\ env{\in}list(M)$
shows
$sats(M,hcomp\_fm(pf,pg,a,w),env) \longleftrightarrow is\_hcomp(\#\#M,is\_f,is\_g,nth(a,env),nth(w,env))$
⟨*proof*⟩

definition
$hcomp\_r :: [i{\Rightarrow}o,[i{\Rightarrow}o,i,i]{\Rightarrow}o,[i{\Rightarrow}o,i,i]{\Rightarrow}o,i,i] \Rightarrow o$ **where**
$hcomp\_r(M,is\_f,is\_g,a,w) \equiv \exists z[M].\ is\_g(M,a,z) \wedge is\_f(M,z,w)$

definition
$is\_hcomp2\_2 :: [i{\Rightarrow}o,[i{\Rightarrow}o,i,i,i]{\Rightarrow}o,[i{\Rightarrow}o,i,i,i]{\Rightarrow}o,[i{\Rightarrow}o,i,i,i]{\Rightarrow}o,i,i,i] \Rightarrow o$ **where**
$is\_hcomp2\_2(M,is\_f,is\_g1,is\_g2,a,b,w) \equiv \exists\, g1ab[M].\ \exists\, g2ab[M].$
$is\_g1(M,a,b,g1ab) \wedge is\_g2(M,a,b,g2ab) \wedge is\_f(M,g1ab,g2ab,w)$

lemma (**in** *M_trivial*) *hcomp_abs*:
assumes
is_f_abs:$\bigwedge a\ z.\ M(a) \Longrightarrow M(z) \Longrightarrow is\_f(M,a,z) \longleftrightarrow z = f(a)$ **and**
is_g_abs:$\bigwedge a\ z.\ M(a) \Longrightarrow M(z) \Longrightarrow is\_g(M,a,z) \longleftrightarrow z = g(a)$ **and**
g_closed:$\bigwedge a.\ M(a) \Longrightarrow M(g(a))$
$M(a)\ M(w)$
shows
$hcomp\_r(M,is\_f,is\_g,a,w) \longleftrightarrow w = f(g(a))$
⟨*proof*⟩

lemma *hcomp_uniqueness*:
assumes
uniq_is_f:
$\bigwedge r\ d\ d'.\ M(r) \Longrightarrow M(d) \Longrightarrow M(d') \Longrightarrow is\_f(M,\ r,\ d) \Longrightarrow is\_f(M,\ r,\ d') \Longrightarrow$
$d = d'$
and
uniq_is_g:
$\bigwedge r\ d\ d'.\ M(r) \Longrightarrow M(d) \Longrightarrow M(d') \Longrightarrow is\_g(M,\ r,\ d) \Longrightarrow is\_g(M,\ r,\ d') \Longrightarrow$
$d = d'$
and
$M(a)\ M(w)\ M(w')$
$hcomp\_r(M,is\_f,is\_g,a,w)$
$hcomp\_r(M,is\_f,is\_g,a,w')$
shows
$w{=}w'$
⟨*proof*⟩

lemma *hcomp_witness*:
assumes
wit_is_f: $\bigwedge r.\ M(r) \Longrightarrow \exists\, d[M].\ is\_f(M,r,d)$ **and**
wit_is_g: $\bigwedge r.\ M(r) \Longrightarrow \exists\, d[M].\ is\_g(M,r,d)$ **and**
$M(a)$
shows
$\exists\, w[M].\ hcomp\_r(M,is\_f,is\_g,a,w)$
⟨*proof*⟩
lemma (**in** *M_trivial*) *hcomp2_2_abs*:
assumes
is_f_abs:$\bigwedge r1\ r2\ z.\ M(r1) \Longrightarrow M(r2) \Longrightarrow\ M(z) \Longrightarrow is\_f(M,r1,r2,z) \longleftrightarrow z = f(r1,r2)$ **and**
is_g1_abs:$\bigwedge r1\ r2\ z.\ M(r1) \Longrightarrow M(r2) \Longrightarrow\ M(z) \Longrightarrow is\_g1(M,r1,r2,z) \longleftrightarrow z = g1(r1,r2)$ **and**
is_g2_abs:$\bigwedge r1\ r2\ z.\ M(r1) \Longrightarrow M(r2) \Longrightarrow\ M(z) \Longrightarrow is\_g2(M,r1,r2,z) \longleftrightarrow z = g2(r1,r2)$ **and**
types: $M(a)\ M(b)\ M(w)\ M(g1(a,b))\ M(g2(a,b))$
shows
$is\_hcomp2\_2(M,is\_f,is\_g1,is\_g2,a,b,w) \longleftrightarrow w = f(g1(a,b),g2(a,b))$
⟨*proof*⟩

lemma *hcomp2_2_uniqueness*:
assumes
uniq_is_f:
$\bigwedge r1\ r2\ d\ d'.\ M(r1) \Longrightarrow M(r2) \Longrightarrow M(d) \Longrightarrow M(d') \Longrightarrow$
$is\_f(M,\ r1,\ r2\ ,\ d) \Longrightarrow is\_f(M,\ r1,\ r2,\ d') \Longrightarrow\ d = d'$
and
uniq_is_g1:
$\bigwedge r1\ r2\ d\ d'.\ M(r1) \Longrightarrow M(r2) \Longrightarrow M(d) \Longrightarrow M(d') \Longrightarrow is\_g1(M,\ r1,r2,\ d) \Longrightarrow is\_g1(M,\ r1,r2,\ d') \Longrightarrow$
$d = d'$
and
uniq_is_g2:
$\bigwedge r1\ r2\ d\ d'.\ M(r1) \Longrightarrow M(r2) \Longrightarrow M(d) \Longrightarrow M(d') \Longrightarrow is\_g2(M,\ r1,r2,\ d) \Longrightarrow is\_g2(M,\ r1,r2,\ d') \Longrightarrow$
$d = d'$
and
$M(a)\ M(b)\ M(w)\ M(w')$
$is\_hcomp2\_2(M,is\_f,is\_g1,is\_g2,a,b,w)$
$is\_hcomp2\_2(M,is\_f,is\_g1,is\_g2,a,b,w')$
shows
$w{=}w'$
⟨*proof*⟩

lemma *hcomp2_2_witness*:
assumes
wit_is_f: $\bigwedge r1\ r2.\ M(r1) \Longrightarrow M(r2) \Longrightarrow \exists\, d[M].\ is\_f(M,r1,r2,d)$ **and**
wit_is_g1: $\bigwedge r1\ r2.\ M(r1) \Longrightarrow M(r2) \Longrightarrow \exists\, d[M].\ is\_g1(M,r1,r2,d)$ **and**
wit_is_g2: $\bigwedge r1\ r2.\ M(r1) \Longrightarrow M(r2) \Longrightarrow \exists\, d[M].\ is\_g2(M,r1,r2,d)$ **and**

$M(a)$ $M(b)$
shows
$\exists\, w[M].\ is\_hcomp2\_2(M,is\_f,is\_g1,is\_g2,a,b,w)$
⟨*proof*⟩

lemma (**in** *M_trivial*) *extensionality_trans*:
assumes
$M(d) \wedge (\forall\, x[M].\ x{\in}d \longleftrightarrow P(x))$
$M(d') \wedge (\forall\, x[M].\ x{\in}d' \longleftrightarrow P(x))$
shows
$d{=}d'$
⟨*proof*⟩

definition
$lt\_rel :: [i{\Rightarrow}o,i,i] \Rightarrow o$ **where**
$lt\_rel(M,a,b) \equiv a{\in}b \wedge ordinal(M,b)$

lemma (**in** *M_trans*) *lt_abs*[*absolut*]: $M(a) \Longrightarrow M(b) \Longrightarrow lt\_rel(M,a,b) \longleftrightarrow a{<}b$
⟨*proof*⟩

definition
$le\_rel :: [i{\Rightarrow}o,i,i] \Rightarrow o$ **where**
$le\_rel(M,a,b) \equiv \exists\, sb[M].\ successor(M,b,sb) \wedge lt\_rel(M,a,sb)$

lemma (**in** *M_trivial*) *le_abs*[*absolut*]: $M(a) \Longrightarrow M(b) \Longrightarrow le\_rel(M,a,b) \longleftrightarrow a{\leq}b$
⟨*proof*⟩

15.2 Discipline for *Pow*

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

definition
$Pow\_rel :: [i{\Rightarrow}o,i] \Rightarrow i$ (‹$Pow{-}'(\_')$›) **where**
$Pow\_rel(M,r) \equiv THE\ d.\ is\_Pow(M,r,d)$

abbreviation
$Pow\_r\_set ::\ [i,i] \Rightarrow i$ (‹$Pow{-}'(\_')$›) **where**
$Pow\_r\_set(M) \equiv Pow\_rel(\#\#M)$

context *M_basic_no_repl*
begin

lemma *is_Pow_uniqueness*:
assumes
$M(r)$
$is\_Pow(M,r,d)$ $is\_Pow(M,r,d')$

shows
$d{=}d'$
⟨*proof*⟩

lemma *is_Pow_witness*: $M(r) \Longrightarrow \exists\, d[M].\ is\_Pow(M,r,d)$
⟨*proof*⟩

lemma *is_Pow_closed* : $[\![\ M(r);is\_Pow(M,r,d)\ ]\!] \Longrightarrow M(d)$
⟨*proof*⟩

lemma *Pow_rel_closed*[*intro*,*simp*]: $M(r) \Longrightarrow M(Pow\_rel(M,r))$
⟨*proof*⟩

lemmas *trans_Pow_rel_closed*[*trans_closed*] = *transM*[*OF _ Pow_rel_closed*]

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

lemma *Pow_rel_iff*:
assumes $M(r)$ $M(d)$
shows $is\_Pow(M,r,d) \longleftrightarrow d = Pow\_rel(M,r)$
⟨*proof*⟩

The next "def_" result really corresponds to *Pow_iff*

lemma *def_Pow_rel*: $M(A) \Longrightarrow M(r) \Longrightarrow A{\in}Pow\_rel(M,r) \longleftrightarrow A \subseteq r$
⟨*proof*⟩

lemma *Pow_rel_char*: $M(r) \Longrightarrow Pow\_rel(M,r) = \{A{\in}Pow(r).\ M(A)\}$
⟨*proof*⟩

lemma *mem_Pow_rel_abs*: $M(a) \Longrightarrow M(r) \Longrightarrow a \in Pow\_rel(M,r) \longleftrightarrow a \in Pow(r)$
⟨*proof*⟩

end — *M_basic_no_repl*

15.3 Discipline for *PiP*

definition
PiP_rel:: $[i{\Rightarrow}o,i,i]{\Rightarrow}o$ **where**
$PiP\_rel(M,A,f) \equiv \exists\, df[M].\ is\_domain(M,f,df) \wedge subset(M,A,df) \wedge is\_function(M,f)$

context *M_basic*
begin

lemma *def_PiP_rel*:
assumes
$M(A)$ $M(f)$
shows

PiP_rel(*M*,*A*,*f*) ⟷ *A* ⊆ *domain*(*f*) ∧ *function*(*f*)
⟨*proof*⟩

end — *M_basic*

definition
Sigfun :: [*i*,*i*⇒*i*]⇒*i* **where**
Sigfun(*x*,*B*) ≡ ⋃ *y*∈*B*(*x*). {⟨*x*,*y*⟩}

lemma *SigFun_char* : *Sigfun*(*x*,*B*) = {*x*}×*B*(*x*)
⟨*proof*⟩

lemma *Sigma_Sigfun*: *Sigma*(*A*,*B*) = ⋃ {*Sigfun*(*x*,*B*) . *x*∈*A*}
⟨*proof*⟩

definition — Note that *B* below is a higher order argument
is_Sigfun :: [*i*⇒*o*,*i*,*i*⇒*i*,*i*]⇒*o* **where**
is_Sigfun(*M*,*x*,*B*,*Sd*) ≡ *M*(*Sd*) ∧ (∃ *RB*[*M*]. *is_Replace*(*M*,*B*(*x*),λ*y z*. *z*={⟨*x*,*y*⟩},*RB*)
∧ *big_union*(*M*,*RB*,*Sd*))

context *M_trivial*
begin

lemma *is_Sigfun_abs*:
assumes
strong_replacement(*M*,λ*y z*. *z*={⟨*x*,*y*⟩})
M(*x*) *M*(*B*(*x*)) *M*(*Sd*)
shows
is_Sigfun(*M*,*x*,*B*,*Sd*) ⟷ *Sd* = *Sigfun*(*x*,*B*)
⟨*proof*⟩

lemma *Sigfun_closed*:
assumes
strong_replacement(*M*, λ*y z*. *y* ∈ *B*(*x*) ∧ *z* = {⟨*x*, *y*⟩})
M(*x*) *M*(*B*(*x*))
shows
M(*Sigfun*(*x*,*B*))
⟨*proof*⟩

lemmas *trans_Sigfun_closed*[*trans_closed*] = *transM*[*OF* _ *Sigfun_closed*]

end — *M_trivial*

definition
is_Sigma :: [*i*⇒*o*,*i*,*i*⇒*i*,*i*]⇒*o* **where**
is_Sigma(*M*,*A*,*B*,*S*) ≡ *M*(*S*) ∧ (∃ *RSf*[*M*].
is_Replace(*M*,*A*,λ*x z*. *z*=*Sigfun*(*x*,*B*),*RSf*) ∧ *big_union*(*M*,*RSf*,*S*))

locale *M_Pi* = *M_basic* +

assumes
Pi_separation: $M(A) \Longrightarrow separation(M,\ PiP\_rel(M,A))$
and
Pi_replacement:
$M(x) \Longrightarrow M(y) \Longrightarrow$
$strong\_replacement(M,\ \lambda ya\ z.\ ya \in y \wedge z = \{\langle x,\ ya\rangle\})$
$M(y) \Longrightarrow$
$strong\_replacement(M,\ \lambda x\ z.\ z = (\bigcup xa{\in}y.\ \{\langle x,\ xa\rangle\}))$

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

lemma *Sigma_abs*[*simp*]:
assumes
$M(S)$
shows
$is\_Sigma(M,A,B,S) \longleftrightarrow S = Sigma(A,B)$
⟨*proof*⟩

lemma *Sigma_closed*[*intro*,*simp*]: $M(Sigma(A,B))$
⟨*proof*⟩

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

end — *M_Pi_assumptions*

15.4 Discipline for *Pi*

definition — completely relational
$is\_Pi :: [i{\Rightarrow}o,i,i{\Rightarrow}i,i]{\Rightarrow}o$ **where**
$is\_Pi(M,A,B,I) \equiv M(I) \wedge (\exists\, S[M].\ \exists\, PS[M].\ is\_Sigma(M,A,B,S) \wedge$
$is\_Pow(M,S,PS) \wedge$
$is\_Collect(M,PS,PiP\_rel(M,A),I))$

definition
$Pi\_rel :: [i{\Rightarrow}o,i,i{\Rightarrow}i] \Rightarrow i$ $(\langle Pi{-}'(\_,\_')\rangle)$ **where**
$Pi\_rel(M,A,B) \equiv THE\ d.\ is\_Pi(M,A,B,d)$

abbreviation
$Pi\_r\_set :: [i,i,i{\Rightarrow}i] \Rightarrow i$ $(\langle Pi{-}'(\_,\_')\rangle)$ **where**
$Pi\_r\_set(M,A,B) \equiv Pi\_rel(\#\#M,A,B)$

context *M_basic*
begin

lemmas *Pow_rel_iff* = *mbnr.Pow_rel_iff*
lemmas *Pow_rel_char* = *mbnr.Pow_rel_char*
lemmas *mem_Pow_rel_abs* = *mbnr.mem_Pow_rel_abs*
lemmas *Pow_rel_closed* = *mbnr.Pow_rel_closed*
lemmas *def_Pow_rel* = *mbnr.def_Pow_rel*
lemmas *trans_Pow_rel_closed* = *mbnr.trans_Pow_rel_closed*

end — *M_basic*

context *M_Pi_assumptions*
begin

lemma *is_Pi_uniqueness*:
  assumes
    is_Pi(*M*,*A*,*B*,*d*) *is_Pi*(*M*,*A*,*B*,*d'*)
  shows
    d=*d'*
  ⟨*proof*⟩

lemma *is_Pi_witness*: ∃ *d*[*M*]. *is_Pi*(*M*,*A*,*B*,*d*)
  ⟨*proof*⟩

lemma *is_Pi_closed* : *is_Pi*(*M*,*A*,*B*,*d*) ⟹ *M*(*d*)
  ⟨*proof*⟩

lemma *Pi_rel_closed*[*intro*,*simp*]: *M*(*Pi_rel*(*M*,*A*,*B*))
⟨*proof*⟩

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

lemma *Pi_rel_iff*:
  assumes *M*(*d*)
  shows *is_Pi*(*M*,*A*,*B*,*d*) ⟷ *d* = *Pi_rel*(*M*,*A*,*B*)
⟨*proof*⟩

lemma *def_Pi_rel*:
  Pi_rel(*M*,*A*,*B*) = {*f*∈*Pow_rel*(*M*,*Sigma*(*A*,*B*)). *A*⊆*domain*(*f*) ∧ *function*(*f*)}
⟨*proof*⟩

lemma *Pi_rel_char*: *Pi_rel*(*M*,*A*,*B*) = {*f*∈*Pi*(*A*,*B*). *M*(*f*)}
  ⟨*proof*⟩

lemma *mem_Pi_rel_abs*:
  assumes *M*(*f*)
  shows *f* ∈ *Pi_rel*(*M*,*A*,*B*) ⟷ *f* ∈ *Pi*(*A*,*B*)

⟨*proof*⟩

end — *M_Pi_assumptions*

The next locale (and similar ones below) are used to show the relationship between versions of simple (i.e. Σ_1^{ZF}, Π_1^{ZF}) concepts in two different transitive models.

locale *M_N_Pi_assumptions* = *M*:*M_Pi_assumptions* + *N*:*M_Pi_assumptions* *N* **for** *N* +
  assumes
    M_imp_N:$M(x) \Longrightarrow N(x)$
begin

lemma *Pi_rel_transfer*: $Pi^M(A,B) \subseteq Pi^N(A,B)$
  ⟨*proof*⟩

end — *M_N_Pi_assumptions*

locale *M_Pi_assumptions_0* = *M_Pi_assumptions* _ *0*
begin

This is used in the proof of *AC_Pi_rel*

lemma *Pi_rel_empty1*[*simp*]: $Pi^M(0,B) = \{0\}$
  ⟨*proof*⟩

end — *M_Pi_assumptions_0*

context *M_Pi_assumptions*
begin

15.5 Auxiliary ported results on *Pi_rel*, now unused

lemma *Pi_rel_iff'*:
  assumes *types*:$M(f)$
  shows
    $f \in Pi\_rel(M,A,B) \longleftrightarrow function(f) \wedge f \subseteq Sigma(A,B) \wedge A \subseteq domain(f)$
  ⟨*proof*⟩

lemma *lam_type_M*:
  assumes $M(A)$ $\bigwedge x.\ x{\in}A \Longrightarrow\ M(B(x))$
    $\bigwedge x.\ x \in A \Longrightarrow b(x){\in}B(x)$ *strong_replacement*$(M,\lambda x\ y.\ y{=}\langle x,\ b(x)\rangle)$
  shows $(\lambda x{\in}A.\ b(x)) \in Pi\_rel(M,A,B)$
⟨*proof*⟩

end — *M_Pi_assumptions*

locale *M_Pi_assumptions2* = *M_Pi_assumptions* +

PiC: *M_Pi_assumptions* _ _ *C* **for** *C*
begin

lemma *Pi_rel_type*:
assumes $f \in Pi^M(A,C)$ $\bigwedge x.\ x \in A \Longrightarrow f`x \in B(x)$
and *types*: $M(f)$
shows $f \in Pi^M(A,B)$
⟨*proof*⟩

lemma *Pi_rel_weaken_type*:
assumes $f \in Pi^M(A,B)$ $\bigwedge x.\ x \in A \Longrightarrow B(x) \subseteq C(x)$
and *types*: $M(f)$
shows $f \in Pi^M(A,C)$
⟨*proof*⟩

end — *M_Pi_assumptions2*

end

16 Arities of internalized formulas

theory *Arities*
imports
Discipline_Base
begin

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

declare *pred_Un_distrib*[*arity_aux*]

context
notes *FOL_arities*[*simp*]
begin

lemma *arity_upair_fm* [*arity*] : ⟦ *t1*∈*nat* ; *t2*∈*nat* ; *up*∈*nat* ⟧ ⟹
arity(*upair_fm*(*t1*,*t2*,*up*)) = ⋃ {*succ*(*t1*),*succ*(*t2*),*succ*(*up*)}
⟨*proof*⟩

end

lemma *Un_trasposition_aux1*: $r \cup s \cup r = r \cup s$ ⟨*proof*⟩

lemma *Un_trasposition_aux2*:
$r \cup (s \cup (r \cup u)) = r \cup (s \cup u)$
$r \cup (s \cup (t \cup (r \cup u))) = r \cup (s \cup (t \cup u))$ ⟨*proof*⟩

We use the previous lemmas to guide automatic arity calculations.

context

notes *Un_assoc*[*symmetric*,*simp*] *Un_trasposition_aux1*[*simp*]
begin

⟨*ML*⟩

end — context

context
notes *FOL_arities*[*simp*]
begin

lemma *arity_is_recfun_fm* [*arity*]:
⟦*p*∈*formula* ; *v*∈*nat* ; *n*∈*nat*; *Z*∈*nat*;*i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
arity(*is_recfun_fm*(*p*,*v*,*n*,*Z*)) = *succ*(*v*) ∪ *succ*(*n*) ∪ *succ*(*Z*) ∪ (*pred*^4(*i*))
⟨*proof*⟩

lemma *arity_is_wfrec_fm* [*arity*]:
⟦*p*∈*formula* ; *v*∈*nat* ; *n*∈*nat*; *Z*∈*nat* ; *i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
arity(*is_wfrec_fm*(*p*,*v*,*n*,*Z*)) = *succ*(*v*) ∪ *succ*(*n*) ∪ *succ*(*Z*) ∪ (*pred*^5(*i*))
⟨*proof*⟩

lemma *arity_is_nat_case_fm* [*arity*]:
⟦*p*∈*formula* ; *v*∈*nat* ; *n*∈*nat*; *Z*∈*nat*; *i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
arity(*is_nat_case_fm*(*v*,*p*,*n*,*Z*)) = *succ*(*v*) ∪ *succ*(*n*) ∪ *succ*(*Z*) ∪ *pred*(*pred*(*i*))
⟨*proof*⟩

lemma *arity_iterates_MH_fm* [*arity*]:
assumes *isF*∈*formula v*∈*nat n*∈*nat g*∈*nat z*∈*nat i*∈*nat*
arity(*isF*) = *i*
shows *arity*(*iterates_MH_fm*(*isF*,*v*,*n*,*g*,*z*)) =
succ(*v*) ∪ *succ*(*n*) ∪ *succ*(*g*) ∪ *succ*(*z*) ∪ (*pred*^4(*i*))
⟨*proof*⟩

lemma *arity_is_iterates_fm* [*arity*]:
assumes *p*∈*formula v*∈*nat n*∈*nat Z*∈*nat i*∈*nat*
arity(*p*) = *i*
shows *arity*(*is_iterates_fm*(*p*,*v*,*n*,*Z*)) = *succ*(*v*) ∪ *succ*(*n*) ∪ *succ*(*Z*) ∪
(*pred*^11(*i*))
⟨*proof*⟩

lemma *arity_eclose_n_fm* [*arity*]:
assumes *A*∈*nat x*∈*nat t*∈*nat*
shows *arity*(*eclose_n_fm*(*A*,*x*,*t*)) = *succ*(*A*) ∪ *succ*(*x*) ∪ *succ*(*t*)
⟨*proof*⟩

lemma *arity_mem_eclose_fm* [*arity*]:
assumes *x*∈*nat t*∈*nat*
shows *arity*(*mem_eclose_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
⟨*proof*⟩

lemma *arity_is_eclose_fm* [*arity*]:
  ⟦*x*∈*nat* ; *t*∈*nat*⟧ ⟹ *arity*(*is_eclose_fm*(*x*,*t*)) = *succ*(*x*) ∪ *succ*(*t*)
  ⟨*proof*⟩

lemma *arity_Collect_fm* [*arity*]:
  assumes *x* ∈ *nat y* ∈ *nat p*∈*formula*
  shows *arity*(*Collect_fm*(*x*,*p*,*y*)) = *succ*(*x*) ∪ *succ*(*y*) ∪ *pred*(*arity*(*p*))
  ⟨*proof*⟩

schematic_goal *arity_least_fm′*:
  assumes
    i ∈ *nat q* ∈ *formula*
  shows
    arity(*least_fm*(*q*,*i*)) ≡ *?ar*
  ⟨*proof*⟩

lemma *arity_least_fm* [*arity*]:
  assumes
    i ∈ *nat q* ∈ *formula*
  shows
    arity(*least_fm*(*q*,*i*)) = *succ*(*i*) ∪ *pred*(*arity*(*q*))
  ⟨*proof*⟩

lemma *arity_Replace_fm* [*arity*]:
  ⟦*p*∈*formula* ; *v*∈*nat* ; *n*∈*nat*; *i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
    arity(*Replace_fm*(*v*,*p*,*n*)) = *succ*(*n*) ∪ *succ*(*v*) ∪ *pred*(*pred*(*i*))
  ⟨*proof*⟩

lemma *arity_lambda_fm* [*arity*]:
  ⟦*p*∈*formula*; *v*∈*nat* ; *n*∈*nat*; *i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
    arity(*lambda_fm*(*p*,*v*,*n*)) = *succ*(*n*) ∪ (*succ*(*v*) ∪ (*pred*^*3*(*i*)))
  ⟨*proof*⟩

lemma *arity_transrec_fm* [*arity*]:
  ⟦*p*∈*formula* ; *v*∈*nat* ; *n*∈*nat*; *i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
     arity(*is_transrec_fm*(*p*,*v*,*n*)) = *succ*(*v*) ∪ *succ*(*n*) ∪ (*pred*^*8*(*i*))
  ⟨*proof*⟩

lemma *arity_wfrec_replacement_fm* :
  ⟦*p*∈*formula* ; *v*∈*nat* ; *n*∈*nat*; *Z*∈*nat* ; *i*∈*nat*⟧ ⟹ *arity*(*p*) = *i* ⟹
    arity(*Exists*(*And*(*pair_fm*(*1*,*0*,*2*),*is_wfrec_fm*(*p*,*v*,*n*,*Z*))))
   = *2* ∪ *v* ∪ *n* ∪ *Z* ∪ (*pred*^*6*(*i*))
  ⟨*proof*⟩

end — *FOL_arities*

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

context
  notes *Un_assoc*[*symmetric*,*simp*] *Un_trasposition_aux1*[*simp*]
begin
⟨*ML*⟩
end

context
  notes *Un_assoc*[*simp*] *Un_trasposition_aux2*[*simp*]
begin
⟨*ML*⟩
end

⟨*ML*⟩

end

17 Basic relativization of function spaces

theory *Discipline_Function*
  imports
    Arities
begin

17.1 Discipline for *fst* and *snd*

⟨*ML*⟩
definition
  is_fst :: (*i*⇒*o*)⇒*i*⇒*i*⇒*o* **where**
  is_fst(*M*,*x*,*t*) ≡ (∃ *z*[*M*]. *pair*(*M*,*t*,*z*,*x*)) ∨
                    (¬(∃ *z*[*M*]. ∃ *w*[*M*]. *pair*(*M*,*w*,*z*,*x*)) ∧ *empty*(*M*,*t*))
⟨*ML*⟩
notation *fst_fm* (‹·*fst*′(_′) *is* _·›)

⟨*ML*⟩

definition *fst_rel* :: [*i*⇒*o*,*i*] ⇒ *i* **where**
  fst_rel(*M*,*p*) ≡ *THE d*. *M*(*d*) ∧ *is_fst*(*M*,*p*,*d*)

⟨*ML*⟩

definition
  is_snd :: (*i*⇒*o*)⇒*i*⇒*i*⇒*o* **where**
  is_snd(*M*,*x*,*t*) ≡ (∃ *z*[*M*]. *pair*(*M*,*z*,*t*,*x*)) ∨
                    (¬(∃ *z*[*M*]. ∃ *w*[*M*]. *pair*(*M*,*z*,*w*,*x*)) ∧ *empty*(*M*,*t*))
⟨*ML*⟩
notation *snd_fm* (‹·*snd*′(_′) *is* _·›)
⟨*ML*⟩

definition *snd_rel* :: $[i \Rightarrow o, i] \Rightarrow i$ **where**
$snd\_rel(M,p) \equiv THE\ d.\ M(d) \wedge is\_snd(M,p,d)$

⟨*ML*⟩

context *M_trans*
begin

lemma *fst_snd_closed*:
assumes $M(p)$
shows $M(fst(p)) \wedge M(snd(p))$
⟨*proof*⟩

lemma *fst_closed*[*intro*,*simp*]: $M(x) \Longrightarrow M(fst(x))$
⟨*proof*⟩

lemma *snd_closed*[*intro*,*simp*]: $M(x) \Longrightarrow M(snd(x))$
⟨*proof*⟩

lemma *fst_abs* [*absolut*]:
$[\![M(p);\ M(x)\ ]\!] \Longrightarrow is\_fst(M,p,x) \longleftrightarrow x = fst(p)$
⟨*proof*⟩

lemma *snd_abs* [*absolut*]:
$[\![M(p);\ M(y)\ ]\!] \Longrightarrow is\_snd(M,p,y) \longleftrightarrow y = snd(p)$
⟨*proof*⟩

lemma *empty_rel_abs* : $M(x) \Longrightarrow M(0) \Longrightarrow x = 0 \longleftrightarrow x = (THE\ d.\ M(d) \wedge empty(M,\ d))$
⟨*proof*⟩

lemma *fst_rel_abs*:
assumes $M(p)$
shows $fst\_rel(M,p) = fst(p)$
⟨*proof*⟩

lemma *snd_rel_abs*:
assumes $M(p)$
shows $snd\_rel(M,p) = snd(p)$
⟨*proof*⟩

end — *M_trans*

17.2 Discipline for *minimum*

⟨*ML*⟩
context *M_trans*
begin

lemma *minimum_closed*[*simp*,*intro*]:
assumes $M(A)$
shows $M(minimum(r,A))$
⟨*proof*⟩

lemma *first_abs* :
assumes $M(B)$
shows $first(z,B,r) \longleftrightarrow first\_rel(M,z,B,r)$
⟨*proof*⟩

lemma *minimum_abs*:
assumes $M(B)$
shows $minimum(r,B) = minimum\_rel(M,r,B)$
⟨*proof*⟩

end — *M_trans*

17.3 Discipline for $\lambda A\ B.\ A \to B$

definition
is_function_space :: $[i⇒o,i,i,i] ⇒ o$ **where**
$is\_function\_space(M,A,B,fs) \equiv M(fs) \wedge is\_funspace(M,A,B,fs)$

definition
function_space_rel :: $[i⇒o,i,i] ⇒ i$ **where**
$function\_space\_rel(M,A,B) \equiv THE\ d.\ is\_function\_space(M,A,B,d)$

⟨*ML*⟩

abbreviation
function_space_r :: $[i,i⇒o,i] ⇒ i$ (‹_ →– _› [61,1,61] 60) **where**
$A \to^M B \equiv function\_space\_rel(M,A,B)$

abbreviation
function_space_r_set :: $[i,i,i] ⇒ i$ (‹_ →– _› [61,1,61] 60) **where**
$function\_space\_r\_set(A,M) \equiv function\_space\_rel(\#\#M,A)$

context *M_Pi*
begin

lemma *is_function_space_uniqueness*:
assumes
$M(r)\ M(B)$
$is\_function\_space(M,r,B,d)\ is\_function\_space(M,r,B,d')$
shows
$d=d'$
⟨*proof*⟩

lemma *is_function_space_witness*:

assumes $M(A)$ $M(B)$
shows $\exists\, d[M].$ *is_function_space*(M,A,B,d)
⟨*proof*⟩

lemma *is_function_space_closed* :
is_function_space$(M,A,B,d) \Longrightarrow M(d)$
⟨*proof*⟩
lemma *function_space_rel_closed*[*intro*,*simp*]:
assumes $M(x)$ $M(y)$
shows $M($*function_space_rel*$(M,x,y))$
⟨*proof*⟩

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

lemma *is_function_space_iff*:
assumes $M(x)$ $M(y)$ $M(d)$
shows *is_function_space*$(M,x,y,d) \longleftrightarrow d =$ *function_space_rel*(M,x,y)
⟨*proof*⟩

lemma *def_function_space_rel*:
assumes $M(A)$ $M(y)$
shows *function_space_rel*$(M,A,y) =$ *Pi_rel*$(M,A,\lambda\_.\ y)$
⟨*proof*⟩

lemma *function_space_rel_char*:
assumes $M(A)$ $M(y)$
shows *function_space_rel*$(M,A,y) = \{f \in A \to y.\ M(f)\}$
⟨*proof*⟩

lemma *mem_function_space_rel_abs*:
assumes $M(A)$ $M(y)$ $M(f)$
shows $f \in$ *function_space_rel*$(M,A,y) \longleftrightarrow\ f \in A \to y$
⟨*proof*⟩

end — *M_Pi*

locale *M_N_Pi* = *M*:*M_Pi* + *N*:*M_Pi N* **for** *N* +
assumes
M_imp_N:$M(x) \Longrightarrow N(x)$
begin

lemma *function_space_rel_transfer*: $M(A) \Longrightarrow M(B) \Longrightarrow$
function_space_rel$(M,A,B) \subseteq$ *function_space_rel*(N,A,B)
⟨*proof*⟩

end — *M_N_Pi*

abbreviation

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

17.4 Discipline for *Collect* terms.

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

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

⟨*ML*⟩

context *M_basic*
begin

— I'm undecided on keeping the relative quantifiers here. Same with *surjP* below. It might relieve from changing *exI allI* to *rexI rallI* in some proofs. I wonder if this escalates well. Assuming that all terms appearing in the "def_" theorem are in *M* and using *transM*, it might do.
lemma *def_injP_rel*:
assumes
M(*A*) *M*(*f*)
shows
injP_rel(*M*,*A*,*f*) ⟷ (∀ *w*[*M*]. ∀ *x*[*M*]. *w*∈*A* ∧ *x*∈*A* ∧ *f*‘*w*=*f*‘*x* ⟶ *w*=*x*)
⟨*proof*⟩

end — *M_basic*

17.5 Discipline for *inj*

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

declare *typed_function_iff_sats Collect_iff_sats* [*iff_sats*]

⟨*ML*⟩
notation *is_function_space_fm* (‹·_ → _ *is* _·›)

⟨*ML*⟩
notation *is_inj_fm* (‹·*inj*′(_,_′) *is* _·›)

⟨*ML*⟩

lemma *arity_is_inj_fm*[*arity*]:

$A \in nat \Longrightarrow$
$B \in nat \Longrightarrow I \in nat \Longrightarrow arity(is\_inj\_fm(A,\ B,\ I)) = succ(A) \cup succ(B) \cup succ(I)$
⟨*proof*⟩

definition
$inj\_rel :: [i \Rightarrow o,i,i] \Rightarrow i$ (‹*inj*–′(_,_′)›) **where**
$inj\_rel(M,A,B) \equiv THE\ d.\ is\_inj(M,A,B,d)$

abbreviation
$inj\_r\_set :: [i,i,i] \Rightarrow i$ (‹*inj*–′(_,_′)›) **where**
$inj\_r\_set(M) \equiv inj\_rel(\#\#M)$

locale $M\_inj = M\_Pi +$
assumes
injP_separation: $M(r) \Longrightarrow separation(M,injP\_rel(M,\ r))$
begin

lemma *is_inj_uniqueness*:
assumes
$M(r)\ M(B)$
$is\_inj(M,r,B,d)\ is\_inj(M,r,B,d')$
shows
$d=d'$
⟨*proof*⟩

lemma *is_inj_witness*: $M(r) \Longrightarrow M(B) \Longrightarrow \exists\, d[M].\ is\_inj(M,r,B,d)$
⟨*proof*⟩

lemma *is_inj_closed* :
$is\_inj(M,x,y,d) \Longrightarrow M(d)$
⟨*proof*⟩

lemma *inj_rel_closed*[*intro*,*simp*]:
assumes $M(x)\ M(y)$
shows $M(inj\_rel(M,x,y))$
⟨*proof*⟩

lemmas $trans\_inj\_rel\_closed[trans\_closed] = transM[OF\ \_\ inj\_rel\_closed]$

lemma *inj_rel_iff*:
assumes $M(x)\ M(y)\ M(d)$
shows $is\_inj(M,x,y,d) \longleftrightarrow d = inj\_rel(M,x,y)$
⟨*proof*⟩

lemma *def_inj_rel*:
assumes $M(A)\ M(B)$
shows $inj\_rel(M,A,B) =$
$\{f \in function\_space\_rel(M,A,B).\ \ \forall\, w[M].\ \forall\, x[M].\ w{\in}A \wedge x{\in}A \wedge f`w =$

$f`x \longrightarrow w{=}x\}$
(**is** _ = *Collect*(_,*?P*))
⟨*proof*⟩

lemma *inj_rel_char*:
assumes *M*(*A*) *M*(*B*)
shows *inj_rel*(*M*,*A*,*B*) = {*f* ∈ *inj*(*A*,*B*). *M*(*f*)}
⟨*proof*⟩

end — *M_inj*

locale *M_N_inj* = *M*:*M_inj* + *N*:*M_inj N* **for** *N* +
assumes
M_imp_N:*M*(*x*) ⟹ *N*(*x*)
begin

lemma *inj_rel_transfer*: *M*(*A*) ⟹ *M*(*B*) ⟹ *inj_rel*(*M*,*A*,*B*) ⊆ *inj_rel*(*N*,*A*,*B*)
⟨*proof*⟩

end — *M_N_inj*

17.6 Discipline for *surj*

definition
surjP_rel:: [*i*⇒*o*,*i*,*i*,*i*]⇒*o* **where**
surjP_rel(*M*,*A*,*B*,*f*) ≡
∀ *y*[*M*]. ∃ *x*[*M*]. ∃ *fx*[*M*]. *y*∈*B* ⟶ *x*∈*A* ∧ *is_apply*(*M*,*f*,*x*,*fx*) ∧ *fx*=*y*

⟨*ML*⟩

context *M_basic*
begin

lemma *def_surjP_rel*:
assumes
M(*A*) *M*(*B*) *M*(*f*)
shows
surjP_rel(*M*,*A*,*B*,*f*) ⟷ (∀ *y*[*M*]. ∃ *x*[*M*]. *y*∈*B* ⟶ *x*∈*A* ∧ *f`x*=*y*)
⟨*proof*⟩

end — *M_basic*

17.7 Discipline for *surj*

definition — completely relational
is_surj :: [*i*⇒*o*,*i*,*i*,*i*]⇒*o* **where**
is_surj(*M*,*A*,*B*,*I*) ≡ *M*(*I*) ∧ (∃ *F*[*M*]. *is_function_space*(*M*,*A*,*B*,*F*) ∧
is_Collect(*M*,*F*,*surjP_rel*(*M*,*A*,*B*),*I*))

⟨*ML*⟩
notation *is_surj_fm* (‹·*surj'*(_,_') *is* _·›)

definition
  surj_rel :: [*i*⇒*o*,*i*,*i*] ⇒ *i* (‹*surj*–'(_,_')›) **where**
  surj_rel(*M*,*A*,*B*) ≡ *THE d*. *is_surj*(*M*,*A*,*B*,*d*)

abbreviation
  surj_r_set :: [*i*,*i*,*i*] ⇒ *i* (‹*surj*–'(_,_')›) **where**
  surj_r_set(*M*) ≡ *surj_rel*(##*M*)

locale *M_surj* = *M_Pi* +
  assumes
    surjP_separation: *M*(*A*)⟹*M*(*B*)⟹*separation*(*M*,λ*x*. *surjP_rel*(*M*,*A*,*B*,*x*))
begin

lemma *is_surj_uniqueness*:
  assumes
    M(*r*) *M*(*B*)
    is_surj(*M*,*r*,*B*,*d*) *is_surj*(*M*,*r*,*B*,*d'*)
  shows
    d=*d'*
  ⟨*proof*⟩

lemma *is_surj_witness*: *M*(*r*) ⟹ *M*(*B*)⟹ ∃ *d*[*M*]. *is_surj*(*M*,*r*,*B*,*d*)
  ⟨*proof*⟩

lemma *is_surj_closed* :
  is_surj(*M*,*x*,*y*,*d*) ⟹ *M*(*d*)
  ⟨*proof*⟩

lemma *surj_rel_closed*[*intro*,*simp*]:
  assumes *M*(*x*) *M*(*y*)
  shows *M*(*surj_rel*(*M*,*x*,*y*))
⟨*proof*⟩

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

lemma *surj_rel_iff*:
  assumes *M*(*x*) *M*(*y*) *M*(*d*)
  shows *is_surj*(*M*,*x*,*y*,*d*) ⟷ *d* = *surj_rel*(*M*,*x*,*y*)
⟨*proof*⟩

lemma *def_surj_rel*:
  assumes *M*(*A*) *M*(*B*)
  shows *surj_rel*(*M*,*A*,*B*) =
        {*f* ∈ *function_space_rel*(*M*,*A*,*B*). ∀ *y*[*M*]. ∃ *x*[*M*]. *y*∈*B* ⟶ *x*∈*A* ∧ *f*‘*x*=*y* }
    (**is** _ = *Collect*(_,*?P*))
⟨*proof*⟩

lemma *surj_rel_char*:
  assumes $M(A)$ $M(B)$
  shows $surj\_rel(M,A,B) = \{f \in surj(A,B).\ M(f)\}$
⟨*proof*⟩

end — *M_surj*

locale $M\_N\_surj = M{:}M\_surj + N{:}M\_surj\ N$ **for** $N +$
  assumes
    $M\_imp\_N{:}M(x) \Longrightarrow N(x)$
begin

lemma *surj_rel_transfer*: $M(A) \Longrightarrow M(B) \Longrightarrow surj\_rel(M,A,B) \subseteq surj\_rel(N,A,B)$
  ⟨*proof*⟩

end — *M_N_surj*

17.8 Discipline for *Inter*

definition
  $is\_Int :: [i{\Rightarrow}o,i,i,i]{\Rightarrow}o$ **where**
  $is\_Int(M,A,B,I) \equiv M(I) \wedge (\forall x[M].\ x \in I \longleftrightarrow x \in A \wedge x \in B)$

⟨*ML*⟩
notation *is_Int_fm* (‹_ ∩ _ *is* _›)

context *M_basic*
begin

lemma *is_Int_closed* :
  $is\_Int(M,A,B,I) \Longrightarrow M(I)$
  ⟨*proof*⟩

lemma *is_Int_abs*:
  assumes
    $M(A)$ $M(B)$ $M(I)$
  shows
    $is\_Int(M,A,B,I) \longleftrightarrow I = A \cap B$
  ⟨*proof*⟩

lemma *is_Int_uniqueness*:
  assumes
    $M(r)$ $M(B)$
    $is\_Int(M,r,B,d)$ $is\_Int(M,r,B,d')$
  shows
    $d{=}d'$
⟨*proof*⟩

Note: *Int_closed* already in *ZF-Constructible.Relative*.

end — *M_basic*

17.9 Discipline for *bij*

⟨*ML*⟩
notation *is_bij_fm* (‹·*bij*′(_,_′) *is* _·›)

abbreviation
bij_r_class :: [*i*⇒*o*,*i*,*i*] ⇒ *i* (‹*bij*–′(_,_′)›) **where**
bij_r_class ≡ *bij_rel*

abbreviation
bij_r_set :: [*i*,*i*,*i*] ⇒ *i* (‹*bij*–′(_,_′)›) **where**
bij_r_set(*M*) ≡ *bij_rel*(##*M*)

locale *M_Perm* = *M_Pi* + *M_inj* + *M_surj*
begin

lemma *is_bij_closed* : *is_bij*(*M*,*f*,*y*,*d*) ⟹ *M*(*d*)
⟨*proof*⟩

lemma *bij_rel_closed*[*intro*,*simp*]:
assumes *M*(*x*) *M*(*y*)
shows *M*(*bij_rel*(*M*,*x*,*y*))
⟨*proof*⟩

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

lemma *bij_rel_iff*:
assumes *M*(*x*) *M*(*y*) *M*(*d*)
shows *is_bij*(*M*,*x*,*y*,*d*) ⟷ *d* = *bij_rel*(*M*,*x*,*y*)
⟨*proof*⟩

lemma *def_bij_rel*:
assumes *M*(*A*) *M*(*B*)
shows *bij_rel*(*M*,*A*,*B*) = *inj_rel*(*M*,*A*,*B*) ∩ *surj_rel*(*M*,*A*,*B*)
⟨*proof*⟩

lemma *bij_rel_char*:
assumes *M*(*A*) *M*(*B*)
shows *bij_rel*(*M*,*A*,*B*) = {*f* ∈ *bij*(*A*,*B*). *M*(*f*)}
⟨*proof*⟩

end — *M_Perm*

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

begin

lemma *bij_rel_transfer*: $M(A) \Longrightarrow M(B) \Longrightarrow bij\_rel(M,A,B) \subseteq bij\_rel(N,A,B)$
⟨*proof*⟩

end — *M_N_Perm*

context *M_Perm*
begin

lemma *mem_bij_rel*: $[\![f \in bij^M(A,B);\ M(A);\ M(B)]\!] \Longrightarrow f \in bij(A,B)$
⟨*proof*⟩

lemma *mem_inj_rel*: $[\![f \in inj^M(A,B);\ M(A);\ M(B)]\!] \Longrightarrow f \in inj(A,B)$
⟨*proof*⟩

lemma *mem_surj_rel*: $[\![f \in surj^M(A,B);\ M(A);\ M(B)]\!] \Longrightarrow f \in surj(A,B)$
⟨*proof*⟩

end — *M_Perm*

17.10 Discipline for $(\approx)$

⟨*ML*⟩
notation *is_eqpoll_fm* $(\langle\cdot\_ \approx \_\cdot\rangle)$

context *M_Perm* **begin**

⟨*ML*⟩
⟨*proof*⟩

end — *M_Perm*

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

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

context *M_Perm*
begin

lemma *def_eqpoll_rel*:
assumes
$M(A)\ M(B)$
shows
$eqpoll\_rel(M,A,B) \longleftrightarrow (\exists f[M].\ f \in bij\_rel(M,A,B))$
⟨*proof*⟩

end — *M_Perm*

context *M_N_Perm*
begin

The next lemma is not part of the discipline

lemma *eqpoll_rel_transfer*: **assumes** $A \approx^M B$ $M(A)$ $M(B)$
shows $A \approx^N B$
⟨*proof*⟩

end — *M_N_Perm*

17.11 Discipline for $(\lesssim)$

⟨*ML*⟩
notation *is_lepoll_fm* $(\langle\cdot\_ \lesssim \_\cdot\rangle)$
⟨*ML*⟩

context *M_inj* **begin**

⟨*ML*⟩
⟨*proof*⟩

end — *M_inj*

abbreviation
lepoll_r :: $[i,i \Rightarrow o,i] \Rightarrow o$ $(\langle\_ \lesssim^- \_\rangle$ $[51,1,51]$ $50)$ **where**
$A \lesssim^M B \equiv$ *lepoll_rel*(M,A,B)

abbreviation
lepoll_r_set :: $[i,i,i] \Rightarrow o$ $(\langle\_ \lesssim^- \_\rangle$ $[51,1,51]$ $50)$ **where**
lepoll_r_set$(A,M) \equiv$ *lepoll_rel*$(\#\#M,A)$

context *M_Perm*
begin

lemma *def_lepoll_rel*:
assumes
$M(A)$ $M(B)$
shows
lepoll_rel$(M,A,B) \longleftrightarrow (\exists f[M].\ f \in$ *inj_rel*$(M,A,B))$
⟨*proof*⟩

end — *M_Perm*

context *M_N_Perm*
begin

— This lemma is not part of the discipline.
lemma *lepoll_rel_transfer*: **assumes** $A \lesssim^M B$ $M(A)$ $M(B)$
  shows $A \lesssim^N B$
⟨*proof*⟩

end — *M_N_Perm*

17.12 Discipline for $(\prec)$

⟨*ML*⟩
notation *is_lesspoll_fm* (‹·_ ≺ _·›)
⟨*ML*⟩

context *M_Perm* **begin**

⟨*ML*⟩
  ⟨*proof*⟩

end — *M_Perm*

abbreviation
  lesspoll_r :: [*i*,*i*⇒*o*,*i*] => *o* (‹_ ≺← _› [*51*,*1*,*51*] *50*) **where**
  $A \prec^M B \equiv$ *lesspoll_rel*(*M*,*A*,*B*)

abbreviation
  lesspoll_r_set :: [*i*,*i*,*i*] ⇒ *o* (‹_ ≺← _› [*51*,*1*,*51*] *50*) **where**
  lesspoll_r_set(*A*,*M*) ≡ *lesspoll_rel*(*##M*,*A*)

Since *lesspoll_rel* is defined as a propositional combination of older terms, there is no need for a separate "def" theorem for it.

Note that *lesspoll_rel* is neither Σ_1^{ZF} nor Π_1^{ZF}, so there is no "transfer" theorem for it.

definition
  Powapply :: [*i*,*i*] ⇒ *i* **where**
  Powapply(*f*,*y*) ≡ *Pow*(*f*‘*y*)

⟨*ML*⟩

declare *Replace_iff_sats*[*iff_sats*]
⟨*ML*⟩

notation *Powapply_rel* (‹*Powapply*–'(_,_')›)

context *M_basic*
begin

⟨*ML*⟩
  ⟨*proof*⟩

⟨*ML*⟩
⟨*proof*⟩

end — *M_basic*

end

18 Replacements using Lambdas

theory *Lambda_Replacement*
imports
Discipline_Function
begin

In this theory we prove several instances of separation and replacement in *M_basic*. Moreover we introduce a new locale assuming two instances of separation and six instances of lambda replacements (ie, replacement of the form $\lambda xy.\ y = \langle x, f(x)\rangle$) and we prove a bunch of other instances.

definition
lam_replacement :: [*i*⇒*o*,*i*⇒*i*] ⇒ *o* **where**
lam_replacement(*M*,*b*) ≡ *strong_replacement*(*M*, λ*x y*. *y* = ⟨*x*, *b*(*x*)⟩)

lemma *separation_univ* :
shows *separation*(*M*,*M*)
⟨*proof*⟩

context *M_trivial*
begin

lemma *lam_replacement_iff_lam_closed*:
assumes ∀ *x*[*M*]. *M*(*b*(*x*))
shows *lam_replacement*(*M*, *b*) ⟷ (∀ *A*[*M*]. *M*(λ*x*∈*A*. *b*(*x*)))
⟨*proof*⟩

lemma *lam_replacement_imp_lam_closed*:
assumes *lam_replacement*(*M*, *b*) *M*(*A*) ∀ *x*∈*A*. *M*(*b*(*x*))
shows *M*(λ*x*∈*A*. *b*(*x*))
⟨*proof*⟩

lemma *lam_replacement_cong*:
assumes *lam_replacement*(*M*,*f*) ∀ *x*[*M*]. *f*(*x*) = *g*(*x*) ∀ *x*[*M*]. *M*(*f*(*x*))
shows *lam_replacement*(*M*,*g*)
⟨*proof*⟩

end — *M_trivial*

context *M_basic*

begin

lemma *separation_iff'*:
  assumes *separation*(M,λx . $P(x)$) *separation*(M,λx . $Q(x)$)
  shows *separation*(M,λx . $P(x) \longleftrightarrow Q(x)$)
  ⟨*proof*⟩

lemma *separation_in_constant* :
  assumes $M(a)$
  shows *separation*(M,λx . $x{\in}a$)
⟨*proof*⟩

lemma *separation_equal* :
  shows *separation*(M,λx . $x{=}a$)
⟨*proof*⟩

lemma (**in** *M_basic*) *separation_in_rev*:
  assumes $(M)(a)$
  shows *separation*(M,λx . $a{\in}x$)
⟨*proof*⟩
lemma *lam_replacement_imp_strong_replacement_aux*:
  assumes *lam_replacement*(M, b) $\forall\, x[M].\ M(b(x))$
  shows *strong_replacement*(M, $\lambda x\ y.\ y = b(x)$)
⟨*proof*⟩
lemma *strong_lam_replacement_imp_strong_replacement* :
 assumes *strong_replacement*(M,$\lambda\, x\, z$. $P(fst(x),snd(x)) \wedge z{=}\langle x,f(fst(x),snd(x))\rangle$)
    $\bigwedge A$. $M(A) \Longrightarrow \forall\, x{\in}A.\ P(X,x) \longrightarrow M(f(X,x))\ M(X)$
  shows *strong_replacement*(M,$\lambda\ x\ z$. $P(X,x) \wedge z{=}f(X,x)$)
  ⟨*proof*⟩

lemma *lam_replacement_imp_RepFun_Lam*:
  assumes *lam_replacement*(M, f) $M(A)$
  shows $M(\{y\ .\ x{\in}A\ ,\ M(y) \wedge y{=}\langle x,f(x)\rangle\})$
⟨*proof*⟩

lemma *lam_closed_imp_closed*:
  assumes $\forall\, A[M].\ M(\lambda x{\in}A.\ f(x))$
  shows $\forall\, x[M].\ M(f(x))$
⟨*proof*⟩

lemma *lam_replacement_if*:
  assumes *lam_replacement*(M,f) *lam_replacement*(M,g) *separation*(M,b)
    $\forall\, x[M].\ M(f(x))\ \forall\, x[M].\ M(g(x))$
  shows *lam_replacement*(M, $\lambda x.$ *if* $b(x)$ *then* $f(x)$ *else* $g(x)$)
⟨*proof*⟩

lemma *lam_replacement_constant*: $M(b) \Longrightarrow$ *lam_replacement*(M,$\lambda\_.\ b$)
  ⟨*proof*⟩

18.1 Replacement instances obtained through Powerset

The next few lemmas provide bounds for certain constructions.

lemma *not_functional_Replace_0*:
assumes $\neg(\forall\, y\ y'.\ P(y) \wedge P(y') \longrightarrow y{=}y')$
shows $\{y\ .\ x \in A,\ P(y)\} = 0$
⟨*proof*⟩

lemma *Replace_in_Pow_rel*:
assumes $\bigwedge x\ b.\ x \in A \Longrightarrow P(x,b) \Longrightarrow b \in U\ \forall\, x{\in}A.\ \forall\, y\ y'.\ P(x,y) \wedge P(x,y') \longrightarrow y{=}y'$
separation$(M,\ \lambda y.\ \exists\, x[M].\ x \in A \wedge P(x,\ y))$
$M(U)$ $M(A)$
shows $\{y\ .\ x \in A,\ P(x,\ y)\} \in Pow^M(U)$
⟨*proof*⟩

lemma *Replace_sing_0_in_Pow_rel*:
assumes $\bigwedge b.\ P(b) \Longrightarrow b \in U$
separation$(M,\ \lambda y.\ P(y))$ $M(U)$
shows $\{y\ .\ x \in \{0\},\ P(y)\} \in Pow^M(U)$
⟨*proof*⟩

lemma *The_in_Pow_rel_Union*:
assumes $\bigwedge b.\ P(b) \Longrightarrow b \in U$ *separation*$(M,\ \lambda y.\ P(y))$ $M(U)$
shows $(THE\ i.\ P(i)) \in Pow^M(\bigcup U)$
⟨*proof*⟩

lemma *separation_least*: *separation*$(M,\ \lambda y.\ Ord(y) \wedge P(y) \wedge (\forall\, j.\ j < y \longrightarrow \neg P(j)))$
⟨*proof*⟩

lemma *Least_in_Pow_rel_Union*:
assumes $\bigwedge b.\ P(b) \Longrightarrow b \in U$
$M(U)$
shows $(\mu\ i.\ P(i)) \in Pow^M(\bigcup U)$
⟨*proof*⟩

lemma *bounded_lam_replacement*:
fixes U
assumes $\forall\, X[M].\ \forall\, x{\in}X.\ f(x) \in U(X)$
and *separation_f*:$\forall\, A[M].$ *separation*$(M,\lambda y.\ \exists\, x[M].\ x{\in}A \wedge y = \langle x,\ f(x)\rangle)$
and *U_closed* [*intro*,*simp*]: $\bigwedge X.\ M(X) \Longrightarrow M(U(X))$
shows *lam_replacement*$(M,\ f)$
⟨*proof*⟩

lemma *fst_in_double_Union*:
assumes $x{\in}X$
shows $fst(x) \in \{0\} \cup \bigcup\bigcup X$
⟨*proof*⟩

lemma *snd_in_double_Union*:
assumes $x{\in}X$
shows $snd(x) \in \{0\} \cup \bigcup\bigcup X$
⟨*proof*⟩

lemma *bounded_lam_replacement_binary*:
fixes U
assumes $\forall\, X[M].\ \forall\, x{\in}X.\ \forall\, y{\in}X.\ f(x,y) \in U(X)$
and *separation_f*:$\forall\, A[M].\ separation(M,\lambda y.\ \exists\, x[M].\ x{\in}A \wedge y = \langle x, f(fst(x),snd(x))\rangle)$
and *U_closed* [*intro*,*simp*]: $\bigwedge X.\ M(X) \Longrightarrow M(U(X))$
shows $lam\_replacement(M,\ \lambda r\ .\ f(fst(r),snd(r)))$
⟨*proof*⟩
lemma *lam_replacement_fst'*:
assumes $\forall\, A[M].\ separation(M,\ \lambda y.\ \exists\, x[M].\ x{\in}A \wedge y = \langle x,\ fst(x)\rangle)$
shows $lam\_replacement(M,fst)$
⟨*proof*⟩

lemma *lam_replacement_snd'*:
assumes $\forall\, A[M].\ separation(M,\ \lambda y.\ \exists\, x[M].\ x{\in}A \wedge y = \langle x,\ snd(x)\rangle)$
shows $lam\_replacement(M,snd)$
⟨*proof*⟩
lemma *lam_replacement_restrict*:
assumes $\forall\, A[M].\ separation(M,\ \lambda y.\ \exists\, x[M].\ x{\in}A \wedge y = \langle x,\ restrict(x,B)\rangle)\ \ M(B)$
shows $lam\_replacement(M,\ \lambda r\ .\ restrict(r,B))$
⟨*proof*⟩

lemma *lam_replacement_Union'* :
assumes $\forall\, A[M].\ separation(M,\ \lambda y.\ \exists\, x[M].\ x \in A \wedge y = \langle x,\ \bigcup x\rangle)$
shows $lam\_replacement(M,Union)$
⟨*proof*⟩

lemma *Image_in_Pow_rel_Union3*:
assumes $x{\in}X\ y{\in}X\ M(X)$
shows $Image(x,y) \in Pow^M(\bigcup\bigcup\bigcup X)$
⟨*proof*⟩

lemma *lam_replacement_Image'*:
assumes
$\forall\, A[M].\ separation(M,\lambda y.\ \exists\, x[M].\ x{\in}A \wedge y = \langle x,\ Image(fst(x),snd(x))\rangle)$
shows
$lam\_replacement(M,\ \lambda r\ .\ Image(fst(r),snd(r)))$
⟨*proof*⟩

lemma *minimum_in_Union*:
assumes $x{\in}X\ y{\in}X$
shows $minimum(x,y) \in \{0\} \cup \bigcup X$
⟨*proof*⟩

lemma *lam_replacement_minimum'*:
assumes
$\forall\, A[M].\ separation(M,\lambda y.\ \exists\, x[M].\ x{\in}A \wedge y = \langle x,\ minimum(fst(x),snd(x))\rangle)$
shows
lam_replacement$(M,\ \lambda r\ .\ minimum(fst(r),snd(r)))$
⟨*proof*⟩

lemma *lam_replacement_Pow_rel*:
assumes $\forall\, A[M].\ separation(M,\ \lambda y.\ \exists\, x[M].\ x \in A \wedge y = \langle x,\ Pow\_rel(M,x)\rangle)$
shows *lam_replacement*$(M,Pow\_rel(M))$
⟨*proof*⟩

end — *M_basic*

definition
middle_del :: $i{\Rightarrow}i{\Rightarrow}i$ **where**
$middle\_del(x,y) \equiv \langle fst(x),snd(y)\rangle$

⟨*ML*⟩

context *M_basic*
begin

lemma *middle_del_in_cartprod*:
assumes $x{\in}X\ y{\in}X\ M(X)$
shows $middle\_del(x,y) \in (\{0\} \cup \bigcup\bigcup X) \times (\{0\} \cup \bigcup\bigcup X)$
⟨*proof*⟩

lemma *lam_replacement_middle_del'*:
assumes
$\forall\, A[M].\ separation(M,\lambda y.\ \exists\, x[M].\ x{\in}A \wedge y = \langle x,\ middle\_del(fst(x),snd(x))\rangle)$
shows
lam_replacement$(M,\ \lambda r\ .\ middle\_del(fst(r),snd(r)))$
⟨*proof*⟩

end — *M_basic*

definition
prodRepl :: $i{\Rightarrow}i{\Rightarrow}i$ **where**
$prodRepl(x,y) \equiv \langle snd(x),\langle fst(x),snd(y)\rangle\rangle$

⟨*ML*⟩

context *M_basic*
begin

lemma *prodRepl_in_cartprod*:
assumes $x{\in}X\ y{\in}X\ M(X)$
shows $prodRepl(x,y) \in (\{0\} \cup \bigcup\bigcup X) \times (\{0\} \cup \bigcup\bigcup X) \times (\{0\} \cup \bigcup\bigcup X)$

⟨*proof*⟩

lemma *lam_replacement_prodRepl'*:
assumes
∀ *A*[*M*]. *separation*(*M*,λ*y*. ∃ *x*[*M*]. *x*∈*A* ∧ *y* = ⟨*x*, *prodRepl*(*fst*(*x*),*snd*(*x*))⟩)
shows
lam_replacement(*M*, λ*r* . *prodRepl*(*fst*(*r*),*snd*(*r*)))
⟨*proof*⟩

end — *M_basic*

context *M_Perm*
begin

lemma *inj_rel_in_Pow_rel_Pow_rel*:
assumes *x*∈*X* *y*∈*X* *M*(*X*)
shows $inj^M(x,y) \in Pow^M(Pow^M(\bigcup X \times \bigcup X))$
⟨*proof*⟩

lemma *lam_replacement_inj_rel'*:
assumes
∀ *A*[*M*]. *separation*(*M*,λ*y*. ∃ *x*[*M*]. *x*∈*A* ∧ *y* = ⟨*x*, inj^M(*fst*(*x*),*snd*(*x*))⟩)
shows
lam_replacement(*M*, λ*r* . inj^M(*fst*(*r*),*snd*(*r*)))
⟨*proof*⟩

end — *M_Perm*

locale *M_replacement* = *M_basic* +
assumes
lam_replacement_fst: *lam_replacement*(*M*,*fst*)
and
lam_replacement_snd: *lam_replacement*(*M*,*snd*)
and
lam_replacement_Union: *lam_replacement*(*M*,*Union*)
and
lam_replacement_middle_del: *lam_replacement*(*M*, λ*r*. *middle_del*(*fst*(*r*),*snd*(*r*)))
and
lam_replacement_prodRepl: *lam_replacement*(*M*, λ*r*. *prodRepl*(*fst*(*r*),*snd*(*r*)))
and
lam_replacement_Image:*lam_replacement*(*M*, λ*p*. *fst*(*p*) `` *snd*(*p*))
and
middle_separation: *separation*(*M*, λ*x*. *snd*(*fst*(*x*))=*fst*(*snd*(*x*)))
and
separation_fst_in_snd: *separation*(*M*, λ*y*. *fst*(*snd*(*y*)) ∈ *snd*(*snd*(*y*)))
begin

— This lemma is similar to *strong_lam_replacement_imp_strong_replacement* and *lam_replacement_imp_strong_replacement_aux* but does not require *g* to be

closed under M.
lemma *lam_replacement_imp_strong_replacement*:
assumes *lam_replacement*(M, f)
shows *strong_replacement*(M, $\lambda x\ y.\ y = f(x)$)
⟨*proof*⟩

lemma *Collect_middle*: $\{p \in (\lambda x{\in}A.\ f(x)) \times (\lambda x{\in}\{f(x)\ .\ x{\in}A\}.\ g(x))\ .\ snd(fst(p))=fst(snd(p))\}$
$= \{\ \langle\langle x,f(x)\rangle,\langle f(x),g(f(x))\rangle\rangle\ .\ x{\in}A\ \}$
⟨*proof*⟩

lemma *RepFun_middle_del*: $\{\ \langle fst(fst(p)),snd(snd(p))\rangle\ .\ p \in \{\ \langle\langle x,f(x)\rangle,\langle f(x),g(f(x))\rangle\rangle$
$.\ x{\in}A\ \}\}$
$=\ \{\ \langle x,g(f(x))\rangle\ .\ x{\in}A\ \}$
⟨*proof*⟩

lemma *lam_replacement_imp_RepFun*:
assumes *lam_replacement*(M, f) $M(A)$
shows $M(\{y\ .\ x{\in}A\ ,\ M(y) \wedge y{=}f(x)\})$
⟨*proof*⟩

lemma *lam_replacement_product*:
assumes *lam_replacement*(M,f) *lam_replacement*(M,g)
shows *lam_replacement*(M, $\lambda x.\ \langle f(x),g(x)\rangle$)
⟨*proof*⟩

lemma *lam_replacement_hcomp*:
assumes *lam_replacement*(M,f) *lam_replacement*(M,g) $\forall\, x[M].\ M(f(x))$
shows *lam_replacement*(M, $\lambda x.\ g(f(x))$)
⟨*proof*⟩

lemma *lam_replacement_hcomp2*:
assumes *lam_replacement*(M,f) *lam_replacement*(M,g)
$\forall\, x[M].\ M(f(x))$ $\forall\, x[M].\ M(g(x))$
lam_replacement(M, $\lambda p.\ h(fst(p),snd(p))$)
shows *lam_replacement*(M, $\lambda x.\ h(f(x),g(x))$)
⟨*proof*⟩

lemma *lam_replacement_identity*: *lam_replacement*(M,$\lambda x.\ x$)
⟨*proof*⟩

lemma *strong_replacement_separation_aux* :
assumes *strong_replacement*(M,$\lambda\ x\ y\ .\ y{=}f(x)$) *separation*($M$,$P$)
shows *strong_replacement*(M, $\lambda x\ y\ .\ P(x) \wedge y{=}f(x)$)
⟨*proof*⟩

lemma *separation_in*:
assumes $\forall\, x[M].\ M(f(x))$ *lam_replacement*(M,f)
$\forall\, x[M].\ M(g(x))$ *lam_replacement*(M,g)
shows *separation*(M,$\lambda x\ .\ f(x){\in}g(x)$)

⟨*proof*⟩

lemma *lam_replacement_swap*: *lam_replacement*(*M*, λ*x*. ⟨*snd*(*x*),*fst*(*x*)⟩)
⟨*proof*⟩

lemma *relation_separation*: *separation*(*M*, λ*z*. ∃ *x y*. *z* = ⟨*x*, *y*⟩)
⟨*proof*⟩

lemma *separation_Pair*:
assumes *separation*(*M*, λ*y* . *P*(*fst*(*y*), *snd*(*y*)))
shows *separation*(*M*, λ*y*. ∃ *u v* . *y*=⟨*u*,*v*⟩ ∧ *P*(*u*,*v*))
⟨*proof*⟩

lemma *lam_replacement_separation* :
assumes *lam_replacement*(*M*,*f*) *separation*(*M*,*P*)
shows *strong_replacement*(*M*, λ*x y* . *P*(*x*) ∧ *y*=⟨*x*,*f*(*x*)⟩)
⟨*proof*⟩

lemmas *strong_replacement_separation* =
strong_replacement_separation_aux[*OF lam_replacement_imp_strong_replacement*]

lemma *id_closed*: *M*(*A*) ⟹ *M*(*id*(*A*))
⟨*proof*⟩

lemma *lam_replacement_Pair*:
shows *lam_replacement*(*M*, λ*x*. ⟨*fst*(*x*), *snd*(*x*)⟩)
⟨*proof*⟩

lemma *lam_replacement_Upair*: *lam_replacement*(*M*, λ*p*. *Upair*(*fst*(*p*), *snd*(*p*)))
⟨*proof*⟩

lemma *lam_replacement_Un*: *lam_replacement*(*M*, λ*p*. *fst*(*p*) ∪ *snd*(*p*))
⟨*proof*⟩

lemma *lam_replacement_cons*: *lam_replacement*(*M*, λ*p*. *cons*(*fst*(*p*),*snd*(*p*)))
⟨*proof*⟩

lemma *lam_replacement_sing_fun*:
assumes *lam_replacement*(*M*, *f*) ∀ *x*[*M*]. *M*(*f*(*x*))
shows *lam_replacement*(*M*, λ*x*. {*f*(*x*)})
⟨*proof*⟩

lemma *lam_replacement_sing*: *lam_replacement*(*M*, λ*x*. {*x*})
⟨*proof*⟩

lemma *lam_replacement_CartProd*:
assumes *lam_replacement*(*M*,*f*) *lam_replacement*(*M*,*g*)
∀ *x*[*M*]. *M*(*f*(*x*)) ∀ *x*[*M*]. *M*(*g*(*x*))

shows *lam_replacement*(M, $\lambda x.\ f(x) \times g(x)$)
⟨*proof*⟩

lemma *lam_replacement_RepFun* :
assumes *lam_replacement*(M,$\lambda x\ .\ f(fst(x),snd(x))$) *lam_replacement*($M$,$g$)
$\forall\, x[M].\forall\, y{\in}g(x).M(f(x,y))$ $\forall\, x[M].M(g(x))$
shows *lam_replacement*(M,$\lambda x\ .\ \{f(x,z)\ .\ z{\in}g(x)\}$)
⟨*proof*⟩

lemma *lam_replacement_Collect* :
assumes *lam_replacement*(M,f) $\forall\, x[M].M(f(x))$
separation(M,$\lambda x\ .\ F(fst(x),snd(x))$)
shows *lam_replacement*(M,$\lambda x.\ \{y{\in}f(x)\ .\ F(x,y)\}$)
⟨*proof*⟩

lemma *separation_eq*:
assumes $\forall\, x[M].\ M(f(x))$ *lam_replacement*(M,f)
$\forall\, x[M].\ M(g(x))$ *lam_replacement*(M,g)
shows *separation*(M,$\lambda x\ .\ f(x) = g(x)$)
⟨*proof*⟩
lemma *separation_comp* :
assumes *separation*(M,P) *lam_replacement*(M,f) $\forall\, x[M].\ M(f(x))$
shows *separation*(M,$\lambda x.\ P(f(x))$)
⟨*proof*⟩

lemma *lam_replacement_domain*: *lam_replacement*(M,*domain*)
⟨*proof*⟩

lemma *separation_in_domain* : $M(a) \Longrightarrow$ *separation*(M, $\lambda x.\ a{\in}domain(x)$)
⟨*proof*⟩

lemma *separation_all*:
assumes *separation*(M, $\lambda x\ .\ P(fst(fst(x)),snd(x))$)
lam_replacement(M,f) *lam_replacement*(M,g)
$\forall\, x[M].\ M(f(x))$ $\forall\, x[M].\ M(g(x))$
shows *separation*(M, $\lambda z.\ \forall\, x{\in}f(z).\ P(g(z),x)$)
⟨*proof*⟩

lemma *separation_ex*:
assumes *separation*(M, $\lambda x\ .\ P(fst(fst(x)),snd(x))$)
lam_replacement(M,f) *lam_replacement*(M,g)
$\forall\, x[M].\ M(f(x))$ $\forall\, x[M].\ M(g(x))$
shows *separation*(M, $\lambda z.\ \exists\, x{\in}f(z).\ P(g(z),x)$)
⟨*proof*⟩

lemma *lam_replacement_converse*: *lam_replacement*(M,*converse*)
⟨*proof*⟩

lemma *lam_replacement_Diff*: *lam_replacement*(M, λx . *fst*(x) - *snd*(x))
⟨*proof*⟩

lemma *lam_replacement_range* : *lam_replacement*(M,*range*)
⟨*proof*⟩

lemma *separation_in_range* : $M(a) \Longrightarrow$ *separation*(M, λx. a∈*range*(x))
⟨*proof*⟩

lemmas *tag_replacement* = *lam_replacement_constant*[*unfolded lam_replacement_def*]

lemma *tag_lam_replacement* : $M(X) \Longrightarrow$ *lam_replacement*(M,λx. <X,x>)
⟨*proof*⟩

lemma *strong_lam_replacement_imp_lam_replacement_RepFun* :
assumes *strong_replacement*(M,$\lambda\, x\, z$. P(*fst*(x),*snd*(x)) ∧ z=⟨x,f(*fst*(x),*snd*(x))⟩)
lam_replacement(M,g)
$\bigwedge A\ y$. $M(y) \Longrightarrow M(A) \Longrightarrow \forall\, x{\in}A$. $P(y,x) \longrightarrow M(f(y,x))$
$\forall\, x[M]$. $M(g(x))$
separation(M, λx. P(*fst*(x),*snd*(x)))
shows *lam_replacement*(M, λx. {y . $r{\in}$ $g(x)$, $P(x,r)$ ∧ y=$f(x,r)$})
⟨*proof*⟩

lemma *lam_replacement_Collect′* :
assumes $M(A)$ *separation*(M,λp . F(*fst*(p),*snd*(p)))
shows *lam_replacement*(M,λx. {$y{\in}A$. $F(x,y)$})
⟨*proof*⟩

lemma *lam_replacement_id2*: *lam_replacement*(M, λx. ⟨x, x⟩)
⟨*proof*⟩

lemmas *id_replacement* = *lam_replacement_id2*[*unfolded lam_replacement_def*]

lemma *lam_replacement_apply2*:*lam_replacement*(M, λp. *fst*(p) ‘ *snd*(p))
⟨*proof*⟩

lemma *lam_replacement_apply*:$M(S) \Longrightarrow$ *lam_replacement*(M, λx. S ‘ x)
⟨*proof*⟩

lemma *apply_replacement*:$M(S) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ y$. y = S ‘ x)
⟨*proof*⟩

lemma *lam_replacement_id_const*: $M(b) \Longrightarrow$ *lam_replacement*(M, λx. ⟨x, b⟩)
⟨*proof*⟩

lemmas *pospend_replacement* = *lam_replacement_id_const*[*unfolded lam_replacement_def*]

lemma *lam_replacement_const_id*: $M(b) \Longrightarrow$ *lam_replacement*(M, λz. ⟨b, z⟩)

⟨*proof*⟩

lemmas *prepend_replacement* = *lam_replacement_const_id*[*unfolded lam_replacement_def*]

lemma *lam_replacement_apply_const_id*: $M(f) \Longrightarrow M(z) \Longrightarrow$
lam_replacement(M, $\lambda x.\ f$ ‘ $\langle z, x\rangle$)
⟨*proof*⟩

lemmas *apply_replacement2* = *lam_replacement_apply_const_id*[*unfolded lam_replacement_def*]

lemma *lam_replacement_Inl*: *lam_replacement*(M, *Inl*)
⟨*proof*⟩

lemma *lam_replacement_Inr*: *lam_replacement*(M, *Inr*)
⟨*proof*⟩

lemmas *Inl_replacement1* = *lam_replacement_Inl*[*unfolded lam_replacement_def*]

lemma *lam_replacement_Diff′*: $M(X) \Longrightarrow$ *lam_replacement*(M, $\lambda x.\ x$ - X)
⟨*proof*⟩

lemmas *Pair_diff_replacement* = *lam_replacement_Diff′*[*unfolded lam_replacement_def*]

lemma *diff_Pair_replacement*: $M(p) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ y\ .\ y$=$\langle x$,x-{p}$\rangle$)
⟨*proof*⟩

lemma *swap_replacement*:*strong_replacement*(M, $\lambda x\ y.\ y = \langle x, (\lambda\langle x,y\rangle.\ \langle y, x\rangle)(x)\rangle$)
⟨*proof*⟩

lemma *lam_replacement_Un_const*:$M(b) \Longrightarrow$ *lam_replacement*(M, $\lambda x.\ x \cup b$)
⟨*proof*⟩

lemmas *tag_union_replacement* = *lam_replacement_Un_const*[*unfolded lam_replacement_def*]

lemma *lam_replacement_csquare*: *lam_replacement*(M,$\lambda p.\ \langle fst(p) \cup snd(p), fst(p),$
$snd(p)\rangle$)
⟨*proof*⟩

lemma *csquare_lam_replacement*:*strong_replacement*(M, $\lambda x\ y.\ y = \langle x, (\lambda\langle x,y\rangle.$
$\langle x \cup y, x, y\rangle)(x)\rangle$)
⟨*proof*⟩

lemma *lam_replacement_assoc*:*lam_replacement*(M,$\lambda x.\ \langle fst(fst(x)), snd(fst(x)),$
$snd(x)\rangle$)
⟨*proof*⟩

lemma *assoc_replacement*:*strong_replacement*(M, $\lambda x\ y.\ y = \langle x, (\lambda\langle\langle x,y\rangle,z\rangle.\ \langle x,$
$y, z\rangle)(x)\rangle$)
⟨*proof*⟩

lemma *lam_replacement_prod_fun*: $M(f) \Longrightarrow M(g) \Longrightarrow$ *lam_replacement*$(M,\lambda x.$
$\langle f\ `\ fst(x),\ g\ `\ snd(x)\rangle)$
⟨*proof*⟩

lemma *prod_fun_replacement*:$M(f) \Longrightarrow M(g) \Longrightarrow$
strong_replacement$(M,\ \lambda x\ y.\ y = \langle x,\ (\lambda\langle w,y\rangle.\ \langle f\ `\ w,\ g\ `\ y\rangle)(x)\rangle)$
⟨*proof*⟩

lemma *separation_subset*:
assumes $\forall\, x[M].\ M(f(x))$ *lam_replacement*(M,f)
$\forall\, x[M].\ M(g(x))$ *lam_replacement*(M,g)
shows *separation*$(M,\lambda x\ .\ f(x) \subseteq g(x))$
⟨*proof*⟩

lemma *lam_replacement_twist*: *lam_replacement*$(M,\lambda\langle\langle x,y\rangle,z\rangle.\ \langle x,y,z\rangle)$
⟨*proof*⟩

lemma *twist_closed*[*intro*,*simp*]: $M(x) \Longrightarrow M((\lambda\langle\langle x,y\rangle,z\rangle.\ \langle x,y,z\rangle)(x))$
⟨*proof*⟩

lemma *lam_replacement_vimage* :
shows *lam_replacement*$(M,\ \lambda x.\ fst(x)$-`` $snd(x))$
⟨*proof*⟩

lemma *lam_replacement_vimage_sing*: *lam_replacement*$(M,\ \lambda p.\ fst(p)$ -`` $\{snd(p)\})$
⟨*proof*⟩

lemma *lam_replacement_vimage_sing_fun*: $M(f) \Longrightarrow$ *lam_replacement*$(M,\ \lambda x.$
f -`` $\{x\})$
⟨*proof*⟩

lemma *lam_replacement_image_sing_fun*: $M(f) \Longrightarrow$ *lam_replacement*$(M,\ \lambda x.\ f$
`` $\{x\})$
⟨*proof*⟩

lemma *converse_apply_projs*: $\forall\, x[M].\ \bigcup\ (fst(x)$ -`` $\{snd(x)\}) = converse(fst(x))$
$`\ (snd(x))$
⟨*proof*⟩

lemma *lam_replacement_converse_app*: *lam_replacement*$(M,\ \lambda p.\ converse(fst(p))$
$`\ snd(p))$
⟨*proof*⟩

lemmas *cardinal_lib_assms4* = *lam_replacement_vimage_sing_fun*[*unfolded lam_replacement_def*]

lemma *lam_replacement_sing_const_id*:
$M(x) \Longrightarrow$ *lam_replacement*$(M,\ \lambda y.\ \{\langle x,\ y\rangle\})$
⟨*proof*⟩

lemma *tag_singleton_closed*: $M(x) \Longrightarrow M(z) \Longrightarrow M(\{\{\langle z,\, y\rangle\}\ .\ y \in x\})$
⟨*proof*⟩

lemma *separation_ball*:
assumes *separation*(M, $\lambda y.\ f(fst(y),snd(y)))\ M(X)$
shows *separation*(M, $\lambda y.\ \forall\, u{\in}X.\ f(y,u)$)
⟨*proof*⟩

lemma *separation_bex*:
assumes *separation*(M, $\lambda y.\ f(fst(y),snd(y)))\ M(X)$
shows *separation*(M, $\lambda y.\ \exists\, u{\in}X.\ f(y,u)$)
⟨*proof*⟩

lemma *lam_replacement_Lambda*:
assumes *lam_replacement*(M, $\lambda y.\ b(fst(y),\ snd(y))$)
$\forall\, w[M].\ \forall\, y[M].\ M(b(w,\ y))\ M(W)$
shows *lam_replacement*(M, $\lambda x.\ \lambda w{\in}W.\ b(x,\ w)$)
⟨*proof*⟩

lemma *lam_replacement_apply_Pair*:
assumes $M(y)$
shows *lam_replacement*(M, $\lambda x.\ y\ `\ \langle fst(x),\ snd(x)\rangle$)
⟨*proof*⟩

lemma *lam_replacement_apply_fst_snd*:
shows *lam_replacement*(M, $\lambda w.\ fst(w)\ `\ fst(snd(w))\ `\ snd(snd(w))$)
⟨*proof*⟩

lemma *lam_replacement_RepFun_apply* :
assumes $M(f)$
shows *lam_replacement*(M, $\lambda x\ .\ \{f`y\ .\ y \in x\}$)
⟨*proof*⟩

lemma *separation_snd_in_fst*: *separation*(M, $\lambda x.\ snd(x) \in fst(x)$)
⟨*proof*⟩

lemma *lam_replacement_if_mem*:
lam_replacement(M, $\lambda x.$ *if* $snd(x) \in fst(x)$ *then 1 else 0*)
⟨*proof*⟩

lemma *lam_replacement_Lambda_apply_fst_snd*:
assumes $M(X)$
shows *lam_replacement*(M, $\lambda x.\ \lambda w{\in}X.\ x\ `\ fst(w)\ `\ snd(w)$)
⟨*proof*⟩

lemma *lam_replacement_Lambda_apply_Pair*:
assumes $M(X)\ M(y)$
shows *lam_replacement*(M, $\lambda x.\ \lambda w{\in}X.\ y\ `\ \langle x,\ w\rangle$)

⟨*proof*⟩

lemma *lam_replacement_Lambda_if_mem*:
assumes $M(X)$
shows *lam_replacement*(M, $\lambda x.\ \lambda xa{\in}X.$ *if* $xa \in x$ *then 1 else 0*)
⟨*proof*⟩

lemma *case_closed* :
assumes $\forall\, x[M].\ M(f(x))\ \forall\, x[M].\ M(g(x))$
shows $\forall\, x[M].\ M(case(f,g,x))$
⟨*proof*⟩

lemma *separation_fst_equal* : $M(a) \Longrightarrow$ *separation*(M,λx . *fst*(x)=a)
⟨*proof*⟩

lemma *lam_replacement_case* :
assumes *lam_replacement*(M,f) *lam_replacement*(M,g)
$\forall\, x[M].\ M(f(x))\ \forall\, x[M].\ M(g(x))$
shows *lam_replacement*(M, λx . *case*(f,g,x))
⟨*proof*⟩

lemma *Pi_replacement1*: $M(x) \Longrightarrow M(y) \Longrightarrow$ *strong_replacement*(M, $\lambda ya\ z.\ ya$
$\in y \wedge z = \{\langle x,\ ya\rangle\}$)
⟨*proof*⟩

lemma *surj_imp_inj_replacement1*:
$M(f) \Longrightarrow M(x) \Longrightarrow$ *strong_replacement*(M, $\lambda y\ z.\ y \in f$ -`` $\{x\} \wedge z = \{\langle x,\ y\rangle\}$)
⟨*proof*⟩

lemmas *domain_replacement* = *lam_replacement_domain*[*unfolded lam_replacement_def*]

lemma *domain_replacement_simp*: *strong_replacement*(M, $\lambda x\ y.\ y$=*domain*(x))
⟨*proof*⟩

lemma *un_Pair_replacement*: $M(p) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ y$. $y = x \cup \{p\}$)
⟨*proof*⟩

lemma *diff_replacement*: $M(X) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ y.\ y = x$ - X)
⟨*proof*⟩

lemma *lam_replacement_succ*:
lam_replacement(M,λz . *succ*(z))
⟨*proof*⟩

lemma *lam_replacement_hcomp_Least*:
assumes *lam_replacement*(M, g) *lam_replacement*(M,$\lambda x.\ \mu\ i.\ x{\in}F(i,x)$)
$\forall\, x[M].\ M(g(x))\ \bigwedge x\ i.\ M(x) \Longrightarrow i \in F(i,\ x) \Longrightarrow M(i)$
shows *lam_replacement*(M,$\lambda x.\ \mu\ i.\ g(x){\in}F(i,g(x))$)
⟨*proof*⟩

lemma *domain_mem_separation*: $M(A) \Longrightarrow separation(M,\ \lambda x\ .\ domain(x) \in A)$
⟨*proof*⟩

lemma *domain_eq_separation*: $M(p) \Longrightarrow separation(M,\ \lambda x\ .\ domain(x) = p)$
⟨*proof*⟩

lemma *lam_replacement_Int*:
shows $lam\_replacement(M,\ \lambda x.\ fst(x) \cap snd(x))$
⟨*proof*⟩

lemma *restrict_eq_separation'*: $M(B) \Longrightarrow \forall\, A[M].\ separation(M,\ \lambda y.\ \exists\, x[M].\ x \in A \wedge\ y = \langle x,\ restrict(x,\ B)\rangle)$
⟨*proof*⟩

lemmas $lam\_replacement\_restrict' = lam\_replacement\_restrict[OF\ restrict\_eq\_separation']$

lemma *restrict_strong_replacement*: $M(A) \Longrightarrow strong\_replacement(M,\ \lambda x\ y.\ y = restrict(x,A))$
⟨*proof*⟩

lemma *restrict_eq_separation*: $M(r) \Longrightarrow M(p) \Longrightarrow separation(M,\ \lambda x\ .\ restrict(x,r) = p)$
⟨*proof*⟩

lemma *separation_equal_fst2* : $M(a) \Longrightarrow separation(M,\lambda x\ .\ fst(fst(x)) = a)$
⟨*proof*⟩

lemma *separation_equal_apply*: $M(f) \Longrightarrow M(a) \Longrightarrow separation(M,\lambda x.\ f`x = a)$
⟨*proof*⟩

lemma *lam_apply_replacement*: $M(A) \Longrightarrow M(f) \Longrightarrow lam\_replacement(M,\ \lambda x\ .\ \lambda n \in A.\ f\ `\ \langle x,\ n\rangle)$
⟨*proof*⟩
lemma *lam_replacement_comp* :
$lam\_replacement(M,\ \lambda x\ .\ fst(x)\ O\ snd(x))$
⟨*proof*⟩

lemma *lam_replacement_comp'*:
$M(f) \Longrightarrow M(g) \Longrightarrow lam\_replacement(M,\ \lambda x\ .\ f\ O\ x\ O\ g)$
⟨*proof*⟩

lemma *RepFun_SigFun_closed*: $M(x) \Longrightarrow M(z) \Longrightarrow M(\{\{\langle z,\ u\rangle\}\ .\ u \in x\})$
⟨*proof*⟩
lemma *separation_orthogonal*: $M(x) \Longrightarrow M(Q) \Longrightarrow separation(M,\ \lambda a\ .\ \ \forall\, s \in x.\ \langle s,\ a\rangle \in Q)$
⟨*proof*⟩

lemma *separation_Transset*: $separation(M, Transset)$
⟨*proof*⟩

lemma *separation_Ord*: *separation*(M,*Ord*)
⟨*proof*⟩

lemma *ord_iso_separation*: $M(A) \Longrightarrow M(r) \Longrightarrow M(s) \Longrightarrow$
separation(M, $\lambda f.\ \forall\, x{\in}A.\ \forall\, y{\in}A.\ \langle x,\ y\rangle \in r \longleftrightarrow \langle f\ `\ x,\ f\ `\ y\rangle \in s$)
⟨*proof*⟩

lemma *separation_dist*: *separation*(M, $\lambda\ x\ .\ \exists\, a.\ \exists\, b\ .\ x{=}\langle a,b\rangle \wedge a{\neq}b$)
⟨*proof*⟩

lemma *separation_imp_lam_closed*:
assumes $\forall\, x{\in}A.\ F(x) \in B$ *separation*(M, $\lambda\langle x,y\rangle.\ F(x) = y$)
$M(A)$ $M(B)$
shows $M(\lambda x{\in}A.\ F(x))$
⟨*proof*⟩

lemma *curry_closed* :
assumes $M(f)$ $M(A)$ $M(B)$
shows $M(curry(A,B,f))$
⟨*proof*⟩

end — *M_replacement*

definition
RepFun_cons :: $i{\Rightarrow}i{\Rightarrow}i$ **where**
RepFun_cons(x,y) ≡ *RepFun*(x, $\lambda z.\ \{\langle y,z\rangle\}$)

context *M_replacement*
begin

lemma *lam_replacement_RepFun_cons′*:
shows *lam_replacement*(M, $\lambda x\ .$ *RepFun_cons*(*fst*(x),*snd*(x)))
⟨*proof*⟩

lemma *RepFun_cons_replacement*: *lam_replacement*(M, $\lambda p.$ *RepFun*(*fst*(p), $\lambda x.$
$\{\langle snd(p),x\rangle\}$))
⟨*proof*⟩

lemma *lam_replacement_Sigfun*:
assumes *lam_replacement*(M,f) $\forall\, y[M].\ M(f(y))$
shows *lam_replacement*(M, $\lambda x.$ *Sigfun*(x,f))
⟨*proof*⟩

lemma *phrank_repl*:
assumes
$M(f)$
shows

strong_replacement(M, $\lambda x\ y.\ y = succ(f`x)$)
⟨*proof*⟩

lemma *lam_replacement_Powapply_rel*:
assumes $\forall\, A[M].\ separation(M,\ \lambda y.\ \exists\, x[M].\ x \in A \wedge y = \langle x,\ Pow\_rel(M,x)\rangle)$
$M(f)$
shows
lam_replacement(M, *Powapply_rel*(M,f))
⟨*proof*⟩

lemmas *Powapply_rel_replacement* = *lam_replacement_Powapply_rel*[*THEN*
lam_replacement_imp_strong_replacement]

lemma *surj_imp_inj_replacement2*:
$M(f) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ z.\ z = Sigfun(x,\ \lambda y.\ f\ \text{-``}\ \{y\})$)
⟨*proof*⟩

lemma *lam_replacement_Pi*: $M(y) \Longrightarrow$ *lam_replacement*(M, $\lambda x.\ \bigcup xa{\in}y.\ \{\langle x,$ $xa\rangle\}$)
⟨*proof*⟩

lemma *Pi_replacement2*: $M(y) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ z.\ z = (\bigcup xa{\in}y.$ $\{\langle x,\ xa\rangle\})$)
⟨*proof*⟩

lemma *if_then_Inj_replacement*:
shows $M(A) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ y.\ y = \langle x,\ if\ x \in A\ then\ Inl(x)\ else$ $Inr(x)\rangle$)
⟨*proof*⟩

lemma *lam_if_then_replacement*:
$M(b) \Longrightarrow$
$M(a) \Longrightarrow M(f) \Longrightarrow$ *strong_replacement*(M, $\lambda y\ ya.\ ya = \langle y,\ if\ y = a\ then\ b$ $else\ f\ `\ y\rangle$)
⟨*proof*⟩

lemma *if_then_replacement*:
$M(A) \Longrightarrow M(f) \Longrightarrow M(g) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ y.\ y = \langle x,\ if\ x \in A$ $then\ f\ `\ x\ else\ g\ `\ x\rangle$)
⟨*proof*⟩

lemma *ifx_replacement*:
$M(f) \Longrightarrow$
$M(b) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ y.\ y = \langle x,\ if\ x \in range(f)\ then\ converse(f)$ $`\ x\ else\ b\rangle$)
⟨*proof*⟩

lemma *if_then_range_replacement2*:
$M(A) \Longrightarrow M(C) \Longrightarrow$ *strong_replacement*(M, $\lambda x\ y.\ y = \langle x,\ if\ x = Inl(A)\ then\ C$

else x⟩)
⟨*proof*⟩

lemma *if_then_range_replacement*:
M(*u*) ⟹
M(*f*) ⟹
strong_replacement
(*M*,
λ*z y*. *y* = ⟨*z*, *if z* = *u then f* ‘ *0 else if z* ∈ *range*(*f*) *then f* ‘ *succ*(*converse*(*f*)
‘ *z*) *else z*⟩)
⟨*proof*⟩

lemma *Inl_replacement2*:
M(*A*) ⟹
strong_replacement(*M*, λ*x y*. *y* = ⟨*x*, *if fst*(*x*) = *A then Inl*(*snd*(*x*)) *else Inr*(*x*)⟩)
⟨*proof*⟩

lemma *case_replacement1*:
strong_replacement(*M*, λ*z y*. *y* = ⟨*z*, *case*(*Inr*, *Inl*, *z*)⟩)
⟨*proof*⟩

lemma *case_replacement2*:
strong_replacement(*M*, λ*z y*. *y* = ⟨*z*, *case*(*case*(*Inl*, λ*y*. *Inr*(*Inl*(*y*))), λ*y*. *Inr*(*Inr*(*y*)),
z)⟩)
⟨*proof*⟩

lemma *case_replacement4*:
M(*f*) ⟹ *M*(*g*) ⟹ *strong_replacement*(*M*, λ*z y*. *y* = ⟨*z*, *case*(λ*w*. *Inl*(*f* ‘ *w*),
λ*y*. *Inr*(*g* ‘ *y*), *z*)⟩)
⟨*proof*⟩

lemma *case_replacement5*:
strong_replacement(*M*, λ*x y*. *y* = ⟨*x*, (λ⟨*x*,*z*⟩. *case*(λ*y*. *Inl*(⟨*y*, *z*⟩), λ*y*. *Inr*(⟨*y*,
z⟩), *x*))(*x*)⟩)
⟨*proof*⟩

end — *M_replacement*

locale *M_Pi_replacement* = *M_Pi* + *M_replacement*

begin
lemma *curry_rel_exp* :
assumes *M*(*f*) *M*(*A*) *M*(*B*) *M*(*C*) *f* ∈ *A* × *B* → *C*
shows *curry*(*A*,*B*,*f*) ∈ *A* →^{M} (*B* →^{M} *C*)
⟨*proof*⟩

end — *M_Pi_replacement*

— To be used in the relativized treatment of Cohen posets

definition
— "domain collect F"
dC_F :: $i \Rightarrow i \Rightarrow i$ **where**
dC_F(A,d) ≡ {$p \in A$. *domain*(p) = d }

definition
— "domain restrict SepReplace Y"
drSR_Y :: $i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow i$ **where**
drSR_Y(B,D,A,x) ≡ {y . $r \in A$, *restrict*(r,B) = $x \wedge y$ = *domain*(r) $\wedge$ *domain*(r) $\in$ D}

lemma *drSR_Y_equality*: *drSR_Y*(B,D,A,x) = { $dr \in D$. ($\exists\, r \in A$. *restrict*(r,B) = $x \wedge$ dr=*domain*(r)) }
⟨*proof*⟩

context *M_replacement*
begin

lemma *separation_restrict_eq_dom_eq*:$\forall\, x[M]$.*separation*(M, $\lambda dr.\, \exists\, r \in A$. *restrict*(r,B) = $x \wedge$ dr=*domain*(r))
if $M(A)$ **and** $M(B)$ **for** A B
⟨*proof*⟩

lemma *separation_is_insnd_restrict_eq_dom* : *separation*(M, λp. ($\exists\, r \in A$. *restrict*(r, B) = *fst*(p) $\wedge$ *snd*(p) = *domain*(r)))
if $M(B)$ $M(D)$ $M(A)$ **for** A B D
⟨*proof*⟩

lemma *lam_replacement_drSR_Y*:
assumes $M(B)$ $M(D)$ $M(A)$
shows *lam_replacement*(M, *drSR_Y*(B,D,A))
⟨*proof*⟩

lemma *drSR_Y_closed*:
assumes $M(B)$ $M(D)$ $M(A)$ $M(f)$
shows M(*drSR_Y*(B,D,A,f))
⟨*proof*⟩

lemma *lam_if_then_apply_replacement*: $M(f) \Longrightarrow M(v) \Longrightarrow M(u) \Longrightarrow$
lam_replacement(M, λx. *if* f ‘ $x = v$ *then* f ‘ u *else* f ‘ x)
⟨*proof*⟩

lemma *lam_if_then_apply_replacement2*: $M(f) \Longrightarrow M(m) \Longrightarrow M(y) \Longrightarrow$
lam_replacement(M, λz . *if* f ‘ $z = m$ *then* y *else* f ‘ z)
⟨*proof*⟩

lemma *lam_if_then_replacement2*: $M(A) \Longrightarrow M(f) \Longrightarrow$
lam_replacement(M, λx . *if* $x \in A$ *then* f ‘ x *else* x)

⟨*proof*⟩

lemma *lam_if_then_replacement_apply*: $M(G) \Longrightarrow$ *lam_replacement*$(M, \lambda x.$ *if* $M(x)$ *then* G ‘ x *else* $0)$
⟨*proof*⟩

lemma *lam_replacement_dC_F*:
assumes $M(A)$
shows *lam_replacement*$(M,$ *dC_F*$(A))$
⟨*proof*⟩

lemma *dCF_closed*:
assumes $M(A)$ $M(f)$
shows $M($*dC_F*$(A,f))$
⟨*proof*⟩

lemma *lam_replacement_Collect_ball_Pair*:
assumes $M(G)$ $M(Q)$
shows *lam_replacement*$(M, \lambda x\ .\ \{a \in G\ .\ \forall\, s{\in}x.\ \langle s,\ a\rangle \in Q\})$
⟨*proof*⟩

lemma *surj_imp_inj_replacement3*:
assumes $M(G)$ $M(Q)$ $M(x)$
shows *strong_replacement*$(M, \lambda y\ z.\ y \in \{a \in G\ .\ \forall\, s{\in}x.\ \langle s,\ a\rangle \in Q\} \wedge z = \{\langle x,$ $y\rangle\})$
⟨*proof*⟩

lemmas *replacements* = *Pair_diff_replacement id_replacement tag_replacement*
pospend_replacement prepend_replacement
Inl_replacement1 diff_Pair_replacement
swap_replacement tag_union_replacement csquare_lam_replacement
assoc_replacement prod_fun_replacement
cardinal_lib_assms4 domain_replacement
apply_replacement
un_Pair_replacement restrict_strong_replacement diff_replacement
if_then_Inj_replacement lam_if_then_replacement if_then_replacement
ifx_replacement if_then_range_replacement2 if_then_range_replacement
Inl_replacement2
case_replacement1 case_replacement2 case_replacement4 case_replacement5

lemma *zermelo_separation*: $M(Q) \Longrightarrow M(f) \Longrightarrow$ *separation*$(M, \lambda X.\ Q \cup f$ ‘‘ $X \subseteq X)$
⟨*proof*⟩

end — *M_replacement*

18.2 Some basic replacement lemmas

lemma (**in** *M_trans*) *strong_replacement_conj*:

assumes $\bigwedge A.\ M(A) \Longrightarrow univalent(M,A,P)$ *strong_replacement*(M,P)
separation$(M,\ \lambda x.\ \exists\, b[M].\ Q(x,b) \wedge P(x,b))$
shows *strong_replacement*$(M,\ \lambda x\ z.\ Q(x,z) \wedge P(x,z))$
⟨*proof*⟩

lemma *strong_replacement_iff_bounded_M*:
strong_replacement$(M,P) \longleftrightarrow$ *strong_replacement*$(M,\lambda\ x\ z\ .\ M(z) \wedge M(x) \wedge P(x,z))$
⟨*proof*⟩

end

19 Basic relativization of cardinality

theory *Discipline_Cardinal*
imports
Discipline_Function
begin

⟨*ML*⟩

notation *is_cardinal_fm* (‹*cardinal*′(_′) *is* _›)

abbreviation
cardinal_r :: $[i,i\Rightarrow o] \Rightarrow i$ (‹|_|$^-$›) **where**
$|x|^M \equiv$ *cardinal_rel*(M,x)

abbreviation
cardinal_r_set :: $[i,i]\Rightarrow i$ (‹|_|$^-$›) **where**
$|x|^M \equiv$ *cardinal_rel*$(\#\#M,x)$

context *M_trivial*
begin
⟨*ML*⟩
⟨*proof*⟩
end — *M_trivial*

⟨*ML*⟩
⟨*proof*⟩

⟨*ML*⟩

lemma *arity_is_surj_fm* [*arity*] :
$A \in nat \Longrightarrow B \in nat \Longrightarrow I \in nat \Longrightarrow arity(is\_surj\_fm(A,\ B,\ I)) = succ(A) \cup succ(B) \cup succ(I)$
⟨*proof*⟩

⟨*ML*⟩

lemma *arity_is_inj_fm* [*arity*]:
A ∈ *nat* ⟹ *B* ∈ *nat* ⟹ *I* ∈ *nat* ⟹ *arity*(*is_inj_fm*(*A*, *B*, *I*)) = *succ*(*A*) ∪ *succ*(*B*) ∪ *succ*(*I*)
⟨*proof*⟩

⟨*ML*⟩

context *M_Perm*
begin
⟨*ML*⟩
⟨*proof*⟩
end — *M_Perm*

⟨*ML*⟩
notation *lt_rel_fm* (‹·_ < _·›)
⟨*ML*⟩

lemma *arity_lt_rel_fm*[*arity*]: *a* ∈ *nat* ⟹ *b* ∈ *nat* ⟹ *arity*(*lt_rel_fm*(*a*, *b*)) = *succ*(*a*) ∪ *succ*(*b*)
⟨*proof*⟩

⟨*ML*⟩
notation *is_Card_fm* (‹·*Card*′(_′)·›)
⟨*ML*⟩

notation *Card_rel* (‹*Card*–′(_′)›)

lemma (**in** *M_Perm*) *is_Card_iff*: *M*(*A*) ⟹ *is_Card*(*M*, *A*) ⟷ *Card*^{M}(*A*)
⟨*proof*⟩

abbreviation
Card_r_set :: [*i*,*i*]⇒*o* (‹*Card*–′(_′)›) **where**
Card^M(*i*) ≡ *Card_rel*(##*M*,*i*)

⟨*ML*⟩
notation *is_InfCard_fm* (‹·*InfCard*′(_′)·›)
⟨*ML*⟩

notation *InfCard_rel* (‹*InfCard*–′(_′)›)

abbreviation
InfCard_r_set :: [*i*,*i*]⇒*o* (‹*InfCard*–′(_′)›) **where**
InfCard^M(*i*) ≡ *InfCard_rel*(##*M*,*i*)

19.1 Disicpline for (⊕)

⟨*ML*⟩

abbreviation

cadd_r :: [*i*,*i*⇒*o*,*i*] ⇒ *i* (‹_ ⊕– _› [*66*,*1*,*66*] *65*) **where**
A ⊕^{M} *B* ≡ *cadd_rel*(*M*,*A*,*B*)

context *M_basic*
begin
⟨*ML*⟩
⟨*proof*⟩
end — *M_basic*

⟨*ML*⟩
⟨*proof*⟩
⟨*ML*⟩

context *M_Perm*
begin
⟨*ML*⟩
⟨*proof*⟩

end — *M_Perm*

19.2 Disicpline for (⊗)

⟨*ML*⟩

abbreviation
cmult_r :: [*i*,*i*⇒*o*,*i*] ⇒ *i* (‹_ ⊗– _› [*66*,*1*,*66*] *65*) **where**
A ⊗^{M} *B* ≡ *cmult_rel*(*M*,*A*,*B*)

⟨*ML*⟩

declare *cartprod_iff_sats* [*iff_sats*]

⟨*ML*⟩

context *M_Perm*
begin
⟨*ML*⟩
⟨*proof*⟩

⟨*ML*⟩
⟨*proof*⟩

end — *M_Perm*

end

20 Relativization of the cumulative hierarchy

theory *Univ_Relative*

imports
ZF-Constructible.Rank
ZF.Univ
Discipline_Cardinal

begin

declare *arity_ordinal_fm*[*arity*]

context *M_trivial*
begin
declare *powerset_abs*[*simp*]

lemma *family_union_closed*: ⟦*strong_replacement*(*M*, λ*x* *y*. *y* = *f*(*x*)); *M*(*A*); ∀ *x*∈*A*. *M*(*f*(*x*))⟧
⟹ *M*(⋃ *x*∈*A*. *f*(*x*))
⟨*proof*⟩

lemma *family_union_closed′*: ⟦*strong_replacement*(*M*, λ*x* *y*. *x*∈*A* ∧ *y* = *f*(*x*)); *M*(*A*); ∀ *x*∈*A*. *M*(*f*(*x*))⟧
⟹ *M*(⋃ *x*∈*A*. *f*(*x*))
⟨*proof*⟩

end — *M_trivial*

definition
HVfrom :: [*i*,*i*,*i*] ⇒ *i* **where**
HVfrom(*A*,*x*,*f*) ≡ *A* ∪ (⋃ *y*∈*x*. *Powapply*(*f*,*y*))

⟨*ML*⟩

lemma *arity_is_HVfrom_fm*:
A ∈ *nat* ⟹
x ∈ *nat* ⟹
f ∈ *nat* ⟹
d ∈ *nat* ⟹
arity(*is_HVfrom_fm*(*A*, *x*, *f*, *d*)) = *succ*(*A*) ∪ *succ*(*d*) ∪ (*succ*(*x*) ∪ *succ*(*f*))
⟨*proof*⟩

notation *HVfrom_rel* (‹*HVfrom*–′(_,_,_′)›)

locale *M_HVfrom* = *M_eclose* +
assumes
Powapply_replacement:
M(*f*) ⟹ *strong_replacement*(*M*,λ*y* *z*. *z* = *Powapply*^{M}(*f*,*y*))
begin

⟨*ML*⟩
⟨*proof*⟩

⟨*ML*⟩
⟨*proof*⟩

end — *M_HVfrom*

definition
Vfrom_rel :: [*i*⇒*o*,*i*,*i*] ⇒ *i* (‹*Vfrom*–′(_,_′)›) **where**
$Vfrom^M(A,i) = transrec(i,\ HVfrom\_rel(M,A))$

definition
is_Vfrom :: [*i*⇒*o*,*i*,*i*,*i*] ⇒ *o* **where**
is_Vfrom(*M*,*A*,*i*,*z*) ≡ *is_transrec*(*M*,*is_HVfrom*(*M*,*A*),*i*,*z*)

definition
Hrank :: [*i*,*i*] ⇒ *i* **where**
Hrank(*x*,*f*) ≡ $(\bigcup y{\in}x.\ succ(f`y))$

definition
rrank :: *i* ⇒ *i* **where**
rrank(*a*) ≡ *Memrel*(*eclose*({*a*}))^+

⟨*ML*⟩

lemma *arity_is_Hrank_fm* : *x* ∈ *nat* ⟹
f ∈ *nat* ⟹
d ∈ *nat* ⟹
arity(*is_Hrank_fm*(*x*, *f*, *d*)) =
succ(*d*) ∪ *succ*(*x*) ∪ *succ*(*f*)
⟨*proof*⟩

locale *M_Vfrom* = *M_HVfrom* +
assumes
trepl_HVfrom : ⟦ *M*(*A*);*M*(*i*) ⟧ ⟹ *transrec_replacement*(*M*,*is_HVfrom*(*M*,*A*),*i*)
and
Hrank_replacement : *M*(*f*) ⟹ *strong_replacement*(*M*,λ*x y* . *y* = *succ*(*f*`*x*))
and
is_Hrank_replacement : *M*(*x*) ⟹ *wfrec_replacement*(*M*,*is_Hrank*(*M*),*rrank*(*x*))
and
HVfrom_replacement : ⟦ *M*(*i*) ; *M*(*A*) ⟧ ⟹
transrec_replacement(*M*,*is_HVfrom*(*M*,*A*),*i*)

begin

lemma *Vfrom_rel_iff* :
assumes *M*(*A*) *M*(*i*) *M*(*z*) *Ord*(*i*)
shows *is_Vfrom*(*M*,*A*,*i*,*z*) ⟷ *z* = $Vfrom^M(A,i)$
⟨*proof*⟩

lemma *relation2_HVfrom*: $M(A) \Longrightarrow$ *relation2*(M,*is_HVfrom*(M,A),*HVfrom_rel*(M,A))
⟨*proof*⟩

lemma *HVfrom_closed* :
$M(A) \Longrightarrow \forall\, x[M].\ \forall\, g[M].$ *function*(g) $\longrightarrow$ M(*HVfrom_rel*(M,A,x,g))
⟨*proof*⟩

lemma *Vfrom_rel_closed*:
assumes $M(A)$ $M(i)$ *Ord*(i)
shows M(*transrec*(i, *HVfrom_rel*(M, A)))
⟨*proof*⟩

lemma *transrec_HVfrom*:
assumes $M(A)$
shows *Ord*(i) $\Longrightarrow$ $M(i) \Longrightarrow$ {x∈*Vfrom*(A,i). $M(x)$} = *transrec*(i,*HVfrom_rel*(M,A))
⟨*proof*⟩

lemma *Vfrom_abs*: ⟦ $M(A)$; $M(i)$; $M(V)$; *Ord*(i) ⟧ $\Longrightarrow$ *is_Vfrom*(M,A,i,V) $\longleftrightarrow$
V = {x∈*Vfrom*(A,i). $M(x)$}
⟨*proof*⟩

lemma *Vfrom_closed*: ⟦ $M(A)$; $M(i)$; *Ord*(i) ⟧ $\Longrightarrow$ M({x∈*Vfrom*(A,i). $M(x)$})
⟨*proof*⟩

end — *M_Vfrom*

20.1 Formula synthesis

context *M_Vfrom*
begin

⟨*ML*⟩
⟨*proof*⟩

⟨*ML*⟩
⟨*proof*⟩

lemma *relation2_Hrank* :
relation2(M,*is_Hrank*(M),*Hrank*)
⟨*proof*⟩

lemma *Hrank_closed* :
$\forall\, x[M].\ \forall\, g[M].$ *function*(g) $\longrightarrow$ M(*Hrank*(x,g))
⟨*proof*⟩

end — *M_basic*

context *M_eclose*
begin

lemma *wf_rrank* : $M(x) \Longrightarrow wf(rrank(x))$
⟨*proof*⟩

lemma *trans_rrank* : $M(x) \Longrightarrow trans(rrank(x))$
⟨*proof*⟩

lemma *relation_rrank* : $M(x) \Longrightarrow relation(rrank(x))$
⟨*proof*⟩

lemma *rrank_in_M* : $M(x) \Longrightarrow M(rrank(x))$
⟨*proof*⟩

end — *M_eclose*

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

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

definition
$Vset'$ :: $[i] \Rightarrow i$ **where**
$Vset'(A) \equiv Vfrom(0,A)$

⟨*ML*⟩

schematic_goal *sats_is_Vset_fm_auto*:
assumes
$i{\in}nat\ v{\in}nat\ env{\in}list(A)\ 0{\in}A$
$i < length(env)\ v < length(env)$
shows
$is\_Vset(\#\#A,nth(i,\ env),nth(v,\ env)) \longleftrightarrow sats(A,?ivs\_fm(i,v),env)$
⟨*proof*⟩

⟨*ML*⟩
context *M_Vfrom*
begin

lemma *Vset_abs*: $[\![\ M(i); M(V); Ord(i)\ ]\!] \Longrightarrow is\_Vset(M,i,V) \longleftrightarrow V = \{x{\in}Vset(i).\ M(x)\}$
⟨*proof*⟩

lemma *Vset_closed*: $[\![\ M(i);\ Ord(i)\ ]\!] \Longrightarrow M(\{x{\in}Vset(i).\ M(x)\})$
⟨*proof*⟩

lemma *rank_closed*: $M(a) \Longrightarrow M(rank(a))$
⟨*proof*⟩

lemma *M_into_Vset*:
  assumes $M(a)$
  shows $\exists\, i[M].\ \exists\, V[M].\ ordinal(M,i) \wedge is\_Vset(M,i,V) \wedge a \in V$
⟨*proof*⟩

end — *M_HVfrom*

end

21 Relative, Choice-less Cardinal Numbers

theory *Cardinal_Relative*
  imports
    Lambda_Replacement
    Univ_Relative
begin

The following command avoids that a commonly used one-letter variable be captured by the definition of the constructible universe L.

hide_const (**open**) L

We also return to the old notation for (+) to preserve the old Constructibility code.

no_notation *oadd* (**infixl** ‹+› *65*)
notation *sum* (**infixr** ‹+› *65*)

definition
  Finite_rel :: $[i \Rightarrow o, i] => o$ **where**
  $Finite\_rel(M,A) \equiv \exists\, om[M].\ \exists\, n[M].\ omega(M,om) \wedge n \in om \wedge eqpoll\_rel(M,A,n)$

definition
  banach_functor :: $[i,i,i,i,i] \Rightarrow i$ **where**
  $banach\_functor(X,Y,f,g,W) \equiv X$ - g‘‘$(Y$ - f‘‘$W)$

lemma *banach_functor_subset*: $banach\_functor(X,Y,f,g,W) \subseteq X$
  ⟨*proof*⟩

definition
  is_banach_functor :: $[i \Rightarrow o, i,i,i,i,i] \Rightarrow o$ **where**
  $is\_banach\_functor(M,X,Y,f,g,W,b) \equiv$
    $\exists\, fW[M].\ \exists\, YfW[M].\ \exists\, gYfW[M].\ image(M,f,W,fW) \wedge setdiff(M,Y,fW,YfW)$
$\wedge$
                $image(M,g,YfW,gYfW) \wedge setdiff(M,X,gYfW,b)$

lemma (**in** *M_basic*) *banach_functor_abs* :
  assumes $M(X)$ $M(Y)$ $M(f)$ $M(g)$
  shows $relation1(M, is\_banach\_functor(M,X,Y,f,g), banach\_functor(X,Y,f,g))$
  ⟨*proof*⟩

lemma (**in** *M_basic*) *banach_functor_closed*:
assumes $M(X)$ $M(Y)$ $M(f)$ $M(g)$
shows $\forall\, W[M].\ M(banach\_functor(X,Y,f,g,W))$
⟨*proof*⟩

context *M_trancl*
begin

lemma *iterates_banach_functor_closed*:
assumes $n{\in}\omega$ $M(X)$ $M(Y)$ $M(f)$ $M(g)$
shows $M(banach\_functor(X,Y,f,g)^{\frown}n(0))$
⟨*proof*⟩

lemma *banach_repl_iter′*:
assumes
$\bigwedge A.\ M(A) \Longrightarrow separation(M,\ \lambda b.\ \exists\, x{\in}A.\ x \in \omega \wedge b = banach\_functor(X,\ Y,\ f,\ g)^{\frown}x\ (0))$
$M(X)$ $M(Y)$ $M(f)$ $M(g)$
shows
$strong\_replacement(M,\ \lambda x\ y.\ x{\in}nat \wedge y = banach\_functor(X,\ Y,\ f,\ g)^{\frown}x\ (0))$
⟨*proof*⟩

end — *M_trancl*

context *M_Perm*
begin

lemma *mem_Pow_rel*: $M(r) \Longrightarrow a \in Pow\_rel(M,r) \Longrightarrow a \in Pow(r) \wedge M(a)$
⟨*proof*⟩

lemma *mem_bij_abs*: $[\![M(f);M(A);M(B)]\!] \Longrightarrow\ f \in bij^M(A,B) \longleftrightarrow f{\in}bij(A,B)$
⟨*proof*⟩

lemma *mem_inj_abs*: $[\![M(f);M(A);M(B)]\!] \Longrightarrow\ f \in inj^M(A,B) \longleftrightarrow f{\in}inj(A,B)$
⟨*proof*⟩

lemma *mem_surj_abs*: $[\![M(f);M(A);M(B)]\!] \Longrightarrow f \in surj^M(A,B) \longleftrightarrow f{\in}surj(A,B)$
⟨*proof*⟩

end — *M_Perm*

locale *M_cardinals* = *M_ordertype* + *M_trancl* + *M_Perm* + *M_replacement* +
assumes
radd_separation: $M(R) \Longrightarrow M(S) \Longrightarrow$
$separation(M,\ \lambda z.$
$(\exists\, x\ y.\ z = \langle Inl(x),\ Inr(y)\rangle) \vee$
$(\exists\, x'\ x.\ z = \langle Inl(x'),\ Inl(x)\rangle \wedge \langle x',\ x\rangle \in R) \vee$

$(\exists\, y'\ y.\ z = \langle Inr(y'),\ Inr(y)\rangle \wedge \langle y',\ y\rangle \in S))$
and
rmult_separation: $M(b) \Longrightarrow M(d) \Longrightarrow$ *separation*(M,
$\lambda z.\ \exists\, x'\ y'\ x\ y.\ z = \langle\langle x',\ y'\rangle,\ x,\ y\rangle \wedge ((\langle x',\ x\rangle \in b \vee x' = x \wedge \langle y',\ y\rangle \in d))$

begin

lemma *rvimage_separation*: $M(f) \Longrightarrow M(r) \Longrightarrow$
separation(M, $\lambda z.\ \exists\, x\ y.\ z = \langle x,\ y\rangle \wedge \langle f\ `\ x,\ f\ `\ y\rangle \in r$)
⟨*proof*⟩

lemma *radd_closed*[*intro*,*simp*]: $M(a) \Longrightarrow M(b) \Longrightarrow M(c) \Longrightarrow M(d) \Longrightarrow M(radd(a,b,c,d))$
⟨*proof*⟩

lemma *rmult_closed*[*intro*,*simp*]: $M(a) \Longrightarrow M(b) \Longrightarrow M(c) \Longrightarrow M(d) \Longrightarrow M(rmult(a,b,c,d))$
⟨*proof*⟩

end — *M_cardinals*

lemma (**in** *M_cardinals*) *is_cardinal_iff_Least*:
assumes $M(A)$ $M(\kappa)$
shows *is_cardinal*(M,A,κ) $\longleftrightarrow$ $\kappa = (\mu\ i.\ M(i) \wedge i \approx^M A)$
⟨*proof*⟩

21.1 The Schroeder-Bernstein Theorem

See Davey and Priestly, page 106

context *M_cardinals*
begin

21.2 Banach's Decomposition Theorem

lemma *bnd_mono_banach_functor*: *bnd_mono*(X, *banach_functor*(X,Y,f,g))
⟨*proof*⟩

lemma *inj_Inter*:
assumes $g \in inj(Y,X)$ $A \neq 0$ $\forall\, a \in A.\ a \subseteq Y$
shows $g\ ``(\bigcap A) = (\bigcap a \in A.\ g\ ``a)$
⟨*proof*⟩

lemma *contin_banach_functor*:
assumes $g \in inj(Y,X)$
shows *contin*(*banach_functor*(X,Y,f,g))
⟨*proof*⟩

lemma *lfp_banach_functor*:
assumes $g \in inj(Y,X)$
shows *lfp*(X, *banach_functor*(X,Y,f,g)) =
$(\bigcup n \in nat.\ banach\_functor(X,Y,f,g)\hat{}n\ (0))$

⟨*proof*⟩

lemma *lfp_banach_functor_closed*:
assumes $M(g)$ $M(X)$ $M(Y)$ $M(f)$ $g{\in}inj(Y,X)$
and *banach_repl_iter*: $M(X) \Longrightarrow M(Y) \Longrightarrow M(f) \Longrightarrow M(g) \Longrightarrow$ *strong_replacement*$(M, \lambda x\ y.\ x{\in}nat \wedge y = banach\_functor(X, Y, f, g)^{\frown}x\ (0))$
shows $M(lfp(X, banach\_functor(X,Y,f,g)))$
⟨*proof*⟩

lemma *banach_decomposition_rel*:
assumes
banach_repl_iter: $M(f) \Longrightarrow M(g) \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow$ *strong_replacement*$(M, \lambda x\ y.\ x{\in}nat \wedge y = banach\_functor(X, Y, f, g)^{\frown}x\ (0))$
shows
$[|\ M(f);\ M(g);\ M(X);\ M(Y);\ f \in X{->}Y;\ \ g \in inj(Y,X)\ |] ==>$
$\exists\, XA[M].\ \exists\, XB[M].\ \exists\, YA[M].\ \exists\, YB[M].$
$(XA \cap XB = 0)\ \&\ (XA \cup XB = X)\ \&$
$(YA \cap YB = 0)\ \&\ (YA \cup YB = Y)\ \&$
$f``XA{=}YA\ \&\ g``YB{=}XB$
⟨*proof*⟩

lemma *schroeder_bernstein_closed*:
assumes
banach_repl_iter: $M(X) \Longrightarrow M(Y) \Longrightarrow M(f) \Longrightarrow M(g) \Longrightarrow$ *strong_replacement*$(M, \lambda x\ y.\ x{\in}nat \wedge y = banach\_functor(X, Y, f, g)^{\frown}x\ (0))$
shows
$[|\ M(f);\ M(g);\ M(X);\ M(Y);\ f \in inj(X,Y);\ \ g \in inj(Y,X)\ |] ==> \exists\, h[M].\ h \in bij(X,Y)$
⟨*proof*⟩

The previous lemmas finish our original, direct relativization of the material involving the iterative proof (as appearing in *ZF.Cardinal*) of the Schröder-Bernstein theorem. Next, we formalize Zermelo's proof that replaces the recursive construction by a fixed point represented as an intersection [2, Exr. x4.27]. This allows to avoid at least one replacement assumption.

lemma *dedekind_zermelo*:
assumes
$A' \subseteq B\ B \subseteq A\ A \approx^M A'$
and *types*: $M(A')\ M(B)\ M(A)$
shows
$A \approx^M B$
⟨*proof*⟩

lemma *schroeder_bernstein_closed'*:
assumes $f \in inj^M(A,C)\ \ g \in inj^M(C,A)$
and *types*:$M(A)\ M(C)$

shows $A \approx^M C$
⟨*proof*⟩

Relative equipollence is an equivalence relation

declare *mem_bij_abs*[*simp*] *mem_inj_abs*[*simp*]

lemma *bij_imp_eqpoll_rel*:
assumes $f \in bij(A,B)$ $M(f)$ $M(A)$ $M(B)$
shows $A \approx^M B$
⟨*proof*⟩

lemma *eqpoll_rel_refl*: $M(A) \Longrightarrow A \approx^M A$
⟨*proof*⟩

lemma *eqpoll_rel_sym*: $X \approx^M Y \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow Y \approx^M X$
⟨*proof*⟩

lemma *eqpoll_rel_trans* [*trans*]:
$[|X \approx^M Y;\ Y \approx^M Z;\ M(X); M(Y)\ ; M(Z)\ |] ==> X \approx^M Z$
⟨*proof*⟩

Relative le-pollence is a preorder

lemma *subset_imp_lepoll_rel*: $X \subseteq Y \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow X \lesssim^M Y$
⟨*proof*⟩

lemmas *lepoll_rel_refl* = *subset_refl* [*THEN subset_imp_lepoll_rel*, *simp*]

lemmas *le_imp_lepoll_rel* = *le_imp_subset* [*THEN subset_imp_lepoll_rel*]

lemma *eqpoll_rel_imp_lepoll_rel*: $X \approx^M Y ==> M(X) \Longrightarrow M(Y) \Longrightarrow X \lesssim^M Y$
⟨*proof*⟩

lemma *lepoll_rel_trans* [*trans*]:
assumes
$X \lesssim^M Y$ $Y \lesssim^M Z$ $M(X)$ $M(Y)$ $M(Z)$
shows
$X \lesssim^M Z$
⟨*proof*⟩

lemma *eq_lepoll_rel_trans* [*trans*]:
assumes
$X \approx^M Y$ $Y \lesssim^M Z$ $M(X)$ $M(Y)$ $M(Z)$
shows
$X \lesssim^M Z$
⟨*proof*⟩

lemma *lepoll_rel_eq_trans* [*trans*]:
assumes $X \lesssim^M Y$ $Y \approx^M Z$ $M(X)$ $M(Y)$ $M(Z)$

shows $X \lesssim^M Z$
⟨*proof*⟩

lemma *eqpoll_relI*: $[\![X \lesssim^M Y;\ Y \lesssim^M X;\ M(X)\ ;\ M(Y)\ ]\!] \Longrightarrow X \approx^M Y$
⟨*proof*⟩

lemma *eqpoll_relE*:
$[\!| X \approx^M Y;\ [\!| X \lesssim^M Y;\ Y \lesssim^M X\ |\!] ==> P\ ;\ \ M(X)\ ;\ M(Y)\ |\!] ==> P$
⟨*proof*⟩

lemma *eqpoll_rel_iff*: $M(X) \Longrightarrow M(Y) \Longrightarrow X \approx^M Y \longleftrightarrow X \lesssim^M Y\ \&\ Y \lesssim^M X$
⟨*proof*⟩

lemma *lepoll_rel_0_is_0*: $A \lesssim^M 0 \Longrightarrow M(A) \Longrightarrow A = 0$
⟨*proof*⟩
lemmas *empty_lepoll_relI* = *empty_subsetI* [*THEN subset_imp_lepoll_rel*, *OF nonempty*]

lemma *lepoll_rel_0_iff*: $M(A) \Longrightarrow A \lesssim^M 0 \longleftrightarrow A=0$
⟨*proof*⟩

lemma *Un_lepoll_rel_Un*:
$[\!| A \lesssim^M B;\ C \lesssim^M D;\ B \cap D = 0;\ M(A);\ M(B);\ M(C);\ M(D)\ |\!] ==> A \cup C \lesssim^M B \cup D$
⟨*proof*⟩

lemma *eqpoll_rel_0_is_0*: $A \approx^M 0 \Longrightarrow M(A) \Longrightarrow A = 0$
⟨*proof*⟩

lemma *eqpoll_rel_0_iff*: $M(A) \Longrightarrow A \approx^M 0 \longleftrightarrow A=0$
⟨*proof*⟩

lemma *eqpoll_rel_disjoint_Un*:
$[\!| A \approx^M B;\ \ C \approx^M D;\ \ A \cap C = 0;\ \ B \cap D = 0;\ M(A);\ M(B);\ M(C)\ ;\ M(D)\ |\!]$
$\quad ==> A \cup C \approx^M B \cup D$
⟨*proof*⟩

21.3 lesspoll: contributions by Krzysztof Grabczewski

lemma *lesspoll_rel_not_refl*: $M(i) \Longrightarrow \sim (i \prec^M i)$
⟨*proof*⟩

lemma *lesspoll_rel_irrefl*: $i \prec^M i ==> M(i) \Longrightarrow P$
⟨*proof*⟩

lemma *lesspoll_rel_imp_lepoll_rel*: $[\![A \prec^M B;\ M(A);\ M(B)]\!] \Longrightarrow A \lesssim^M B$
⟨*proof*⟩

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

assumes
$M(A)$ $M(f)$ $M(r)$
shows
$M(rvimage(A,f,r))$
⟨*proof*⟩

lemma *lepoll_rel_well_ord*: [| $A \lesssim^M B$; *well_ord*(B,r); $M(A)$; $M(B)$; $M(r)$ |] ==> $\exists s[M].$ *well_ord*(A,s)
⟨*proof*⟩

lemma *lepoll_rel_iff_leqpoll_rel*: $[\![M(A);\ M(B)]\!] \Longrightarrow A \lesssim^M B \longleftrightarrow A \prec^M B \mid A \approx^M B$
⟨*proof*⟩

end — *M_cardinals*

context *M_cardinals*
begin

lemma *inj_rel_is_fun_M*: $f \in inj^M(A,B) \Longrightarrow M(f) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow f \in A \rightarrow^M B$
⟨*proof*⟩
lemma *inj_rel_not_surj_rel_succ*:
notes *mem_inj_abs*[*simp del*]
assumes *fi*: $f \in inj^M(A,\ succ(m))$ **and** *fns*: $f \notin surj^M(A,\ succ(m))$
and *types*: $M(f)$ $M(A)$ $M(m)$
shows $\exists f[M].\ f \in inj^M(A,m)$
⟨*proof*⟩

lemma *lesspoll_rel_trans* [*trans*]:
[| $X \prec^M Y$; $Y \prec^M Z$; $M(X)$; $M(Y)$; $M(Z)$ |] ==> $X \prec^M Z$
⟨*proof*⟩

lemma *lesspoll_rel_trans1* [*trans*]:
[| $X \lesssim^M Y$; $Y \prec^M Z$; $M(X)$; $M(Y)$; $M(Z)$ |] ==> $X \prec^M Z$
⟨*proof*⟩

lemma *lesspoll_rel_trans2* [*trans*]:
[| $X \prec^M Y$; $Y \lesssim^M Z$; $M(X)$; $M(Y)$; $M(Z)$|] ==> $X \prec^M Z$
⟨*proof*⟩

lemma *eq_lesspoll_rel_trans* [*trans*]:
[| $X \approx^M Y$; $Y \prec^M Z$; $M(X)$; $M(Y)$; $M(Z)$ |] ==> $X \prec^M Z$
⟨*proof*⟩

lemma *lesspoll_rel_eq_trans* [*trans*]:
[| $X \prec^M Y$; $Y \approx^M Z$; $M(X)$; $M(Y)$; $M(Z)$ |] ==> $X \prec^M Z$
⟨*proof*⟩

lemma *is_cardinal_cong*:
  assumes $X \approx^M Y\ M(X)\ M(Y)$
  shows $\exists \kappa[M].\ is\_cardinal(M,X,\kappa) \wedge is\_cardinal(M,Y,\kappa)$
⟨*proof*⟩
lemma *cardinal_rel_cong*: $X \approx^M Y \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow |X|^M = |Y|^M$
  ⟨*proof*⟩

lemma *well_ord_is_cardinal_eqpoll_rel*:
  assumes *well_ord*(A,r) **shows** *is_cardinal*$(M,A,\kappa) \Longrightarrow M(A) \Longrightarrow M(\kappa) \Longrightarrow M(r) \Longrightarrow \kappa \approx^M A$
⟨*proof*⟩

lemmas *Ord_is_cardinal_eqpoll_rel* = *well_ord_Memrel*[*THEN well_ord_is_cardinal_eqpoll_rel*]

22 Porting from *ZF.Cardinal*

The following results were ported more or less directly from *ZF.Cardinal*

lemma *well_ord_cardinal_rel_eqpoll_rel*:
  assumes r: *well_ord*(A,r) **and** $M(A)\ M(r)$ **shows** $|A|^M \approx^M A$
  ⟨*proof*⟩

lemmas *Ord_cardinal_rel_eqpoll_rel* = *well_ord_Memrel*[*THEN well_ord_cardinal_rel_eqpoll_rel*]

lemma *Ord_cardinal_rel_idem*: *Ord*$(A) \Longrightarrow M(A) \Longrightarrow ||A|^M|^M = |A|^M$
  ⟨*proof*⟩

lemma *well_ord_cardinal_rel_eqE*:
  assumes *woX*: *well_ord*(X,r) **and** *woY*: *well_ord*(Y,s) **and** *eq*: $|X|^M = |Y|^M$
    and *types*: $M(X)\ M(r)\ M(Y)\ M(s)$
  shows $X \approx^M Y$
⟨*proof*⟩

lemma *well_ord_cardinal_rel_eqpoll_rel_iff*:
  [| *well_ord*(X,r); *well_ord*(Y,s); $M(X)$; $M(r)$; $M(Y)$; $M(s)$ |] ==> $|X|^M = |Y|^M \longleftrightarrow X \approx^M Y$
  ⟨*proof*⟩

lemma *Ord_cardinal_rel_le*: *Ord*$(i) \Longrightarrow M(i)$ ==> $|i|^M \le i$
  ⟨*proof*⟩

lemma *Card_rel_cardinal_rel_eq*: *Card*$^M(K)$ ==> $M(K) \Longrightarrow |K|^M = K$
  ⟨*proof*⟩

lemma *Card_relI*: [| *Ord*(i); !!j. $j<i \Longrightarrow M(j)$ ==> $\sim(j \approx^M i)$; $M(i)$ |] ==> *Card*$^M(i)$
  ⟨*proof*⟩

lemma *Card_rel_is_Ord*: *Card*$^M(i)$ ==> $M(i) \Longrightarrow$ *Ord*(i)

⟨*proof*⟩

lemma *Card_rel_cardinal_rel_le*: $Card^M(K) ==> M(K) \Longrightarrow K \leq |K|^M$
⟨*proof*⟩

lemma *Ord_cardinal_rel* [*simp*,*intro*!]: $M(A) \Longrightarrow Ord(|A|^M)$
⟨*proof*⟩

lemma *Card_rel_iff_initial*: **assumes** *types*:$M(K)$
shows $Card^M(K) \longleftrightarrow Ord(K)$ & $(\forall j[M].\ j<K \longrightarrow \sim (j \approx^M K))$
⟨*proof*⟩

lemma *lt_Card_rel_imp_lesspoll_rel*: [| $Card^M(a)$; $i<a$; $M(a)$; $M(i)$ |] ==> $i \prec^M a$
⟨*proof*⟩

lemma *Card_rel_0*: $Card^M(0)$
⟨*proof*⟩

lemma *Card_rel_Un*: [| $Card^M(K)$; $Card^M(L)$; $M(K)$; $M(L)$ |] ==> $Card^M(K \cup L)$
⟨*proof*⟩

lemma *Card_rel_cardinal_rel* [*iff*]: **assumes** *types*:$M(A)$ **shows** $Card^M(|A|^M)$
⟨*proof*⟩

lemma *cardinal_rel_eq_lemma*:
assumes *i*:$|i|^M \leq j$ **and** *j*: $j \leq i$ **and** *types*: $M(i)$ $M(j)$
shows $|j|^M = |i|^M$
⟨*proof*⟩

lemma *cardinal_rel_mono*:
assumes *ij*: $i \leq j$ **and** *types*:$M(i)$ $M(j)$ **shows** $|i|^M \leq |j|^M$
⟨*proof*⟩

lemma *cardinal_rel_lt_imp_lt*: [| $|i|^M < |j|^M$; $Ord(i)$; $Ord(j)$; $M(i)$; $M(j)$ |] ==> $i < j$
⟨*proof*⟩

lemma *Card_rel_lt_imp_lt*: [| $|i|^M < K$; $Ord(i)$; $Card^M(K)$; $M(i)$; $M(K)$|] ==> $i < K$
⟨*proof*⟩

lemma *Card_rel_lt_iff*: [| $Ord(i)$; $Card^M(K)$; $M(i)$; $M(K)$ |] ==> $(|i|^M < K) \longleftrightarrow (i < K)$
⟨*proof*⟩

lemma *Card_rel_le_iff*: [| $Ord(i)$; $Card^M(K)$; $M(i)$; $M(K)$ |] ==> $(K \leq |i|^M) \longleftrightarrow (K \leq i)$

⟨*proof*⟩

lemma *well_ord_lepoll_rel_imp_cardinal_rel_le*:
assumes *wB*: *well_ord*(*B*,*r*) **and** *AB*: $A \lesssim^M B$
and
types: $M(B)$ $M(r)$ $M(A)$
shows $|A|^M \leq |B|^M$
⟨*proof*⟩

lemma *lepoll_rel_cardinal_rel_le*: [| $A \lesssim^M i$; *Ord*(*i*); $M(A)$; $M(i)$ |] ==> $|A|^M \leq i$
⟨*proof*⟩

lemma *lepoll_rel_Ord_imp_eqpoll_rel*: [| $A \lesssim^M i$; *Ord*(*i*); $M(A)$; $M(i)$ |] ==> $|A|^M \approx^M A$
⟨*proof*⟩

lemma *lesspoll_rel_imp_eqpoll_rel*: [| $A \prec^M i$; *Ord*(*i*); $M(A)$; $M(i)$ |] ==> $|A|^M \approx^M A$
⟨*proof*⟩

lemma *lesspoll_cardinal_lt_rel*:
shows [| $A \prec^M i$; *Ord*(*i*); $M(i)$; $M(A)$ |] ==> $|A|^M < i$
⟨*proof*⟩

lemma *cardinal_rel_subset_Ord*: [|*A*<=*i*; *Ord*(*i*); $M(A)$; $M(i)$|] ==> $|A|^M \subseteq i$
⟨*proof*⟩
lemma *cons_lepoll_rel_consD*:
[| *cons*(*u*,*A*) $\lesssim^M$ *cons*(*v*,*B*); $u \notin A$; $v \notin B$; $M(u)$; $M(A)$; $M(v)$; $M(B)$ |] ==> $A \lesssim^M B$
⟨*proof*⟩

lemma *cons_eqpoll_rel_consD*: [| *cons*(*u*,*A*) $\approx^M$ *cons*(*v*,*B*); $u \notin A$; $v \notin B$; $M(u)$; $M(A)$; $M(v)$; $M(B)$ |] ==> $A \approx^M B$
⟨*proof*⟩

lemma *succ_lepoll_rel_succD*: *succ*(*m*) $\lesssim^M$ *succ*(*n*) $\Longrightarrow$ $M(m)$ $\Longrightarrow$ $M(n)$ ==> $m \lesssim^M n$
⟨*proof*⟩

lemma *nat_lepoll_rel_imp_le*:
$m \in nat$ ==> $n \in nat$ $\Longrightarrow$ $m \lesssim^M n$ $\Longrightarrow$ $M(m)$ $\Longrightarrow$ $M(n)$ $\Longrightarrow$ $m \leq n$
⟨*proof*⟩

lemma *nat_eqpoll_rel_iff*: [| $m \in nat$; $n \in nat$; $M(m)$; $M(n)$ |] ==> $m \approx^M n \longleftrightarrow m = n$
⟨*proof*⟩

lemma *nat_into_Card_rel*:

assumes *n*: $n \in nat$ **and** *types*: $M(n)$ **shows** $Card^M(n)$
⟨*proof*⟩

lemmas *cardinal_rel_0* = *nat_0I* [*THEN nat_into_Card_rel*, *THEN Card_rel_cardinal_rel_eq*, *simplified*, *iff*]
lemmas *cardinal_rel_1* = *nat_1I* [*THEN nat_into_Card_rel*, *THEN Card_rel_cardinal_rel_eq*, *simplified*, *iff*]

lemma *succ_lepoll_rel_natE*: [| $succ(n) \lesssim^M n$; $n \in nat$ |] ==> *P*
⟨*proof*⟩

lemma *nat_lepoll_rel_imp_ex_eqpoll_rel_n*:
[| $n \in nat$; $nat \lesssim^M X$; $M(n)$; $M(X)$|] ==> $\exists Y[M]. Y \subseteq X \ \& \ n \approx^M Y$
⟨*proof*⟩

lemma *lepoll_rel_succ*: $M(i) \Longrightarrow i \lesssim^M succ(i)$
⟨*proof*⟩

lemma *lepoll_rel_imp_lesspoll_rel_succ*:
assumes *A*: $A \lesssim^M m$ **and** *m*: $m \in nat$
and *types*: $M(A)$ $M(m)$
shows $A \prec^M succ(m)$
⟨*proof*⟩

lemma *lesspoll_rel_succ_imp_lepoll_rel*:
[| $A \prec^M succ(m)$; $m \in nat$; $M(A)$; $M(m)$ |] ==> $A \lesssim^M m$
⟨*proof*⟩

lemma *lesspoll_rel_succ_iff*: $m \in nat \Longrightarrow M(A)$ ==> $A \prec^M succ(m) \longleftrightarrow A \lesssim^M m$
⟨*proof*⟩

lemma *lepoll_rel_succ_disj*: [| $A \lesssim^M succ(m)$; $m \in nat$; $M(A)$; $M(m)$|] ==> $A \lesssim^M m \mid A \approx^M succ(m)$
⟨*proof*⟩

lemma *lesspoll_rel_cardinal_rel_lt*: [| $A \prec^M i$; $Ord(i)$; $M(A)$; $M(i)$ |] ==> $|A|^M < i$
⟨*proof*⟩

lemma *lt_not_lepoll_rel*:
assumes *n*: $n<i$ $n \in nat$
and *types*:$M(n)$ $M(i)$ **shows** $\sim i \lesssim^M n$
⟨*proof*⟩

A slightly weaker version of *nat_eqpoll_rel_iff*

lemma *Ord_nat_eqpoll_rel_iff*:
assumes *i*: $Ord(i)$ **and** *n*: $n \in nat$
and *types*: $M(i)$ $M(n)$

shows $i \approx^M n \longleftrightarrow i{=}n$
⟨*proof*⟩

lemma *Card_rel_nat*: $Card^M(nat)$
⟨*proof*⟩

lemma *nat_le_cardinal_rel*: $nat \le i \Longrightarrow M(i) \Longrightarrow nat \le |i|^M$
⟨*proof*⟩

lemma *n_lesspoll_rel_nat*: $n \in nat \Longrightarrow n \prec^M nat$
⟨*proof*⟩

lemma *cons_lepoll_rel_cong*:
$[\!|\ A \lesssim^M B;\ \ b \notin B;\ M(A);\ M(B);\ M(b);\ M(a)\ |\!] \Longrightarrow cons(a,A) \lesssim^M cons(b,B)$
⟨*proof*⟩

lemma *cons_eqpoll_rel_cong*:
$[\!|\ A \approx^M B;\ \ a \notin A;\ \ b \notin B;\ \ M(A);\ M(B);\ M(a)\ ;\ M(b)\ |\!] \Longrightarrow cons(a,A) \approx^M cons(b,B)$
⟨*proof*⟩

lemma *cons_lepoll_rel_cons_iff*:
$[\!|\ a \notin A;\ \ b \notin B;\ M(a);\ M(A);\ M(b);\ M(B)\ |\!] \Longrightarrow cons(a,A) \lesssim^M cons(b,B) \longleftrightarrow\ A \lesssim^M B$
⟨*proof*⟩

lemma *cons_eqpoll_rel_cons_iff*:
$[\!|\ a \notin A;\ \ b \notin B;\ M(a);\ M(A);\ M(b);\ M(B)\ |\!] \Longrightarrow cons(a,A) \approx^M cons(b,B) \longleftrightarrow\ A \approx^M B$
⟨*proof*⟩

lemma *singleton_eqpoll_rel_1*: $M(a) \Longrightarrow \{a\} \approx^M 1$
⟨*proof*⟩

lemma *cardinal_rel_singleton*: $M(a) \Longrightarrow |\{a\}|^M = 1$
⟨*proof*⟩

lemma *not_0_is_lepoll_rel_1*: $A \neq 0 \Longrightarrow M(A) \Longrightarrow 1 \lesssim^M A$
⟨*proof*⟩

lemma *succ_eqpoll_rel_cong*: $A \approx^M B \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow succ(A) \approx^M succ(B)$
⟨*proof*⟩

The next result was not straightforward to port, and even a different statement was needed.

lemma *sum_bij_rel*:
$[\!|\ f \in bij^M(A,C);\ g \in bij^M(B,D);\ M(f);\ M(A);\ M(C);\ M(g);\ M(B);\ M(D)|\!]$
$\Longrightarrow (\lambda z{\in}A{+}B.\ case(\%x.\ Inl(f`x),\ \%y.\ Inr(g`y),\ z)) \in bij^M(A{+}B,\ C{+}D)$

⟨*proof*⟩

lemma *sum_bij_rel'*:
assumes $f \in bij^M(A,C)$ $g \in bij^M(B,D)$ $M(f)$
$M(A)$ $M(C)$ $M(g)$ $M(B)$ $M(D)$
shows
$(\lambda z{\in}A{+}B.\ case(\lambda x.\ Inl(f`x),\ \lambda y.\ Inr(g`y),\ z)) \in bij(A{+}B,\ C{+}D)$
$M(\lambda z{\in}A{+}B.\ case(\lambda x.\ Inl(f`x),\ \lambda y.\ Inr(g`y),\ z))$
⟨*proof*⟩

lemma *sum_eqpoll_rel_cong*:
assumes $A \approx^M C$ $B \approx^M D$ $M(A)$ $M(C)$ $M(B)$ $M(D)$
shows $A{+}B \approx^M C{+}D$
⟨*proof*⟩

lemma *prod_bij_rel'*:
assumes $f \in bij^M(A,C)$ $g \in bij^M(B,D)$ $M(f)$
$M(A)$ $M(C)$ $M(g)$ $M(B)$ $M(D)$
shows
$(\lambda{<}x,y{>}{\in}A{*}B.\ {<}f`x,\ g`y{>}) \in bij(A{*}B,\ C{*}D)$
$M(\lambda{<}x,y{>}{\in}A{*}B.\ {<}f`x,\ g`y{>})$
⟨*proof*⟩

lemma *prod_eqpoll_rel_cong*:
assumes $A \approx^M C$ $B \approx^M D$ $M(A)$ $M(C)$ $M(B)$ $M(D)$
shows $A{\times}B \approx^M C{\times}D$
⟨*proof*⟩

lemma *inj_rel_disjoint_eqpoll_rel*:
$[\!|\ f \in inj^M(A,B);\ \ A \cap B = 0; M(f);\ M(A); M(B)\ |\!] \Longrightarrow A \cup (B - range(f)) \approx^M B$
⟨*proof*⟩

lemma *Diff_sing_lepoll_rel*:
$[\!|\ a \in A;\ \ A \lesssim^M succ(n);\ M(a);\ M(A);\ M(n)\ |\!] \Longrightarrow A - \{a\} \lesssim^M n$
⟨*proof*⟩

lemma *lepoll_rel_Diff_sing*:
assumes *A*: $succ(n) \lesssim^M A$
and *types*: $M(n)$ $M(A)$ $M(a)$
shows $n \lesssim^M A - \{a\}$
⟨*proof*⟩

lemma *Diff_sing_eqpoll_rel*: $[\!|\ a \in A;\ A \approx^M succ(n);\ M(a);\ M(A);\ M(n)\ |\!] \Longrightarrow A - \{a\} \approx^M n$
⟨*proof*⟩

lemma *lepoll_rel_1_is_sing*: $[\!|\ A \lesssim^M 1;\ a \in A\ ; M(a);\ M(A)\ |\!] \Longrightarrow A = \{a\}$
⟨*proof*⟩

lemma *Un_lepoll_rel_sum*: $M(A) \Longrightarrow M(B) \Longrightarrow A \cup B \lesssim^M A+B$
⟨*proof*⟩

lemma *well_ord_Un_M*:
assumes *well_ord*(X,R) *well_ord*(Y,S)
and *types*: $M(X)$ $M(R)$ $M(Y)$ $M(S)$
shows $\exists\, T[M].$ *well_ord*($X \cup Y,\ T$)
⟨*proof*⟩

lemma *disj_Un_eqpoll_rel_sum*: $M(A) \Longrightarrow M(B) \Longrightarrow A \cap B = 0 \Longrightarrow A \cup B \approx^M A + B$
⟨*proof*⟩

lemma *eqpoll_rel_imp_Finite_rel_iff*: $A \approx^M B$ ==> $M(A) \Longrightarrow M(B) \Longrightarrow$ *Finite_rel*(M,A) $\longleftrightarrow$ *Finite_rel*(M,B)
⟨*proof*⟩

lemma *Finite_abs*[*simp*]:
assumes $M(A)$
shows *Finite_rel*(M,A) $\longleftrightarrow$ *Finite*(A)
⟨*proof*⟩

lemma *lepoll_rel_nat_imp_Finite_rel*:
assumes A: $A \lesssim^M n$ **and** n: $n \in nat$
and *types*: $M(A)$ $M(n)$
shows *Finite_rel*(M,A)
⟨*proof*⟩

lemma *lesspoll_rel_nat_is_Finite_rel*:
$A \prec^M nat \Longrightarrow M(A) \Longrightarrow$ *Finite_rel*(M,A)
⟨*proof*⟩

lemma *lepoll_rel_Finite_rel*:
assumes Y: $Y \lesssim^M X$ **and** X: *Finite_rel*(M,X)
and *types*:$M(Y)$ $M(X)$
shows *Finite_rel*(M,Y)
⟨*proof*⟩

lemma *succ_lepoll_rel_imp_not_empty*: *succ*(x) $\lesssim^M$ y ==> $M(x) \Longrightarrow M(y) \Longrightarrow y \neq 0$
⟨*proof*⟩

lemma *eqpoll_rel_succ_imp_not_empty*: $x \approx^M$ *succ*(n) ==> $M(x) \Longrightarrow M(n) \Longrightarrow x \neq 0$
⟨*proof*⟩

lemma *Finite_subset_closed*:
assumes *Finite*(B) $B \subseteq A$ $M(A)$

shows $M(B)$
⟨*proof*⟩

lemma *Finite_Pow_abs*:
assumes *Finite*(*A*) $M(A)$
shows *Pow*(*A*) = *Pow_rel*(*M*,*A*)
⟨*proof*⟩

lemma *Finite_Pow_rel*:
assumes *Finite*(*A*) $M(A)$
shows *Finite*(*Pow_rel*(*M*,*A*))
⟨*proof*⟩

lemma *Pow_rel_0* [*simp*]: *Pow_rel*(*M*,*0*) = {*0*}
⟨*proof*⟩

lemma *eqpoll_rel_imp_Finite*: $A \approx^M B \Longrightarrow$ *Finite*(*A*) $\Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow$ *Finite*(*B*)
⟨*proof*⟩

lemma *eqpoll_rel_imp_Finite_iff*: $A \approx^M B \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow$ *Finite*(*A*) $\longleftrightarrow$ *Finite*(*B*)
⟨*proof*⟩

end — *M_cardinals*

end

23 Relative, Choice-less Cardinal Arithmetic

theory *CardinalArith_Relative*
imports
Cardinal_Relative

begin

⟨*ML*⟩

definition
csquare_lam :: $i \Rightarrow i$ **where**
csquare_lam(*K*) ≡ $\lambda\langle x,y\rangle \in K \times K.\ \langle x \cup y,\ x,\ y\rangle$

— Can't do the next thing because split is a missing HOC

⟨*ML*⟩

definition
is_csquare_lam :: $[i \Rightarrow o, i, i] \Rightarrow o$ **where**
is_csquare_lam(*M*,*K*,*l*) ≡ $\exists$ *K2*[*M*]. *cartprod*(*M*,*K*,*K*,*K2*) ∧

is_lambda(*M*,*K2*,*is_csquare_lam_body*(*M*),*l*)

definition *jump_cardinal_body* :: [*i*,*i*] ⇒ *i* **where**
jump_cardinal_body(*U*,*X*) ≡
{*z* . *r* ∈ *U*, *well_ord*(*X*, *r*) ∧ *z* = *ordertype*(*X*, *r*)}

lemma *jump_cardinal_body_char* :
jump_cardinal_body(*U*,*X*) = {*ordertype*(*X*, *r*) . *r* ∈ {*r*∈*U* . *well_ord*(*X*, *r*)}}
⟨*proof*⟩
definition *jump_cardinal_body′* **where**
jump_cardinal_body′(*x*) ≡ *jump_cardinal_body*(*Pow*(*x*×*x*),*x*)

lemma (**in** *M_cardinals*) *csquare_lam_closed*[*intro*,*simp*]: *M*(*K*) ⟹ *M*(*csquare_lam*(*K*))
⟨*proof*⟩

locale *M_pre_cardinal_arith* = *M_cardinals* +
assumes
wfrec_pred_replacement:*M*(*A*) ⟹ *M*(*r*) ⟹
wfrec_replacement(*M*, λ*x f z*. *z* = *f* ‘‘ *Order.pred*(*A*, *x*, *r*), *r*)

⟨*ML*⟩

lemma (**in** *M_trivial*) *rmultP_abs* [*absolut*]: ⟦ *M*(*r*); *M*(*s*); *M*(*z*) ⟧ ⟹ *is_rmultP*(*M*,*s*,*r*,*z*)
⟷
(∃ *x′ y′ x y*. *z* = ⟨⟨*x′*, *y′*⟩, *x*, *y*⟩ ∧ (⟨*x′*, *x*⟩ ∈ *r* ∨ *x′* = *x* ∧ ⟨*y′*, *y*⟩ ∈ *s*))
⟨*proof*⟩

definition
is_csquare_rel :: [*i*⇒*o*,*i*,*i*]⇒*o* **where**
is_csquare_rel(*M*,*K*,*cs*) ≡ ∃ *K2*[*M*]. ∃ *la*[*M*]. ∃ *memK*[*M*].
∃ *rmKK*[*M*]. ∃ *rmKK2*[*M*].
cartprod(*M*,*K*,*K*,*K2*) ∧ *is_csquare_lam*(*M*,*K*,*la*) ∧
membership(*M*,*K*,*memK*) ∧ *is_rmult*(*M*,*K*,*memK*,*K*,*memK*,*rmKK*) ∧
is_rmult(*M*,*K*,*memK*,*K2*,*rmKK*,*rmKK2*) ∧ *is_rvimage*(*M*,*K2*,*la*,*rmKK2*,*cs*)

context *M_basic*
begin

lemma *rvimage_abs*[*absolut*]:
assumes *M*(*A*) *M*(*f*) *M*(*r*) *M*(*z*)
shows *is_rvimage*(*M*,*A*,*f*,*r*,*z*) ⟷ *z* = *rvimage*(*A*,*f*,*r*)
⟨*proof*⟩

lemma *rmult_abs* [*absolut*]: ⟦ *M*(*A*); *M*(*r*); *M*(*B*); *M*(*s*); *M*(*z*) ⟧ ⟹
is_rmult(*M*,*A*,*r*,*B*,*s*,*z*) ⟷ *z*=*rmult*(*A*,*r*,*B*,*s*)
⟨*proof*⟩

lemma *csquare_lam_body_abs*[*absolut*]: *M*(*x*) ⟹ *M*(*z*) ⟹
is_csquare_lam_body(*M*,*x*,*z*) ⟷ *z* = <*fst*(*x*) ∪ *snd*(*x*), *fst*(*x*), *snd*(*x*)>

⟨*proof*⟩

lemma *csquare_lam_abs*[*absolut*]: $M(K) \Longrightarrow M(l) \Longrightarrow$
is_csquare_lam(M,K,l) $\longleftrightarrow$ $l = (\lambda x{\in}K{\times}K.\ \langle fst(x) \cup snd(x),\ fst(x),\ snd(x)\rangle)$
⟨*proof*⟩

lemma *csquare_lam_eq_lam*:*csquare_lam*(K) $=$ $(\lambda z{\in}K{\times}K.\ <fst(z) \cup snd(z),$
$fst(z),\ snd(z)>)$
⟨*proof*⟩

end — *M_basic*

context *M_pre_cardinal_arith*
begin

lemma *csquare_rel_closed*[*intro*,*simp*]: $M(K) \Longrightarrow M(csquare\_rel(K))$
⟨*proof*⟩

lemma *csquare_rel_abs*[*absolut*]: $[\![\ M(K);\ M(cs)]\!] \Longrightarrow$
is_csquare_rel(M,K,cs) $\longleftrightarrow$ $cs = csquare\_rel(K)$
⟨*proof*⟩

end — *M_pre_cardinal_arith*

23.1 Discipline for *csucc*

⟨*ML*⟩

abbreviation
csucc_r :: $[i,i{\Rightarrow}o] \Rightarrow i$ (‹$'($_$^+')$–›) **where**
csucc_r(x,M) $\equiv$ *csucc_rel*(M,x)

abbreviation
csucc_r_set :: $[i,i] \Rightarrow i$ (‹$'($_$^+')$–›) **where**
csucc_r_set(x,M) $\equiv$ *csucc_rel*($\#\#M$,x)

context *M_Perm*
begin

⟨*ML*⟩
⟨*proof*⟩

⟨*ML*⟩
⟨*proof*⟩

end — *M_Perm*

notation *csucc_rel* (‹*csucc*–$'($_$')$›)

context *M_cardinals*
begin

lemma *Card_rel_Union* [*simp*,*intro*,*TC*]:
  assumes *A*: $\bigwedge x.\ x{\in}A \Longrightarrow Card^M(x)$ **and**
    types:$M(A)$
  shows $Card^M(\bigcup(A))$
⟨*proof*⟩

We are not relativizing directly the results involving unions of (ordinal) indexed families (namely, *Card_UN* and *Card_OUN*, respectively) because of their higher order nature.

lemma *in_Card_imp_lesspoll*: [| $Card^M(K)$; $b \in K$; $M(K)$; $M(b)$ |] ==> $b \prec^M K$
  ⟨*proof*⟩

23.2 Cardinal addition

Note (Paulson): Could omit proving the algebraic laws for cardinal addition and multiplication. On finite cardinals these operations coincide with addition and multiplication of natural numbers; on infinite cardinals they coincide with union (maximum). Either way we get most laws for free.

23.2.1 Cardinal addition is commutative

lemma *sum_commute_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow A{+}B \approx^M B{+}A$
⟨*proof*⟩

lemma *cadd_rel_commute*: $M(i) \Longrightarrow M(j) \Longrightarrow i \oplus^M j = j \oplus^M i$
  ⟨*proof*⟩

23.2.2 Cardinal addition is associative

lemma *sum_assoc_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (A{+}B){+}C \approx^M A{+}(B{+}C)$
  ⟨*proof*⟩

Unconditional version requires AC

lemma *well_ord_cadd_rel_assoc*:
  assumes *i*: *well_ord*(*i*,*ri*) **and** *j*: *well_ord*(*j*,*rj*) **and** *k*: *well_ord*(*k*,*rk*)
    and
    types: $M(i)$ $M(ri)$ $M(j)$ $M(rj)$ $M(k)$ $M(rk)$
  shows $(i \oplus^M j) \oplus^M k = i \oplus^M (j \oplus^M k)$
⟨*proof*⟩

23.2.3 0 is the identity for addition

lemma *case_id_eq*: $x{\in}sum(A,B) \Longrightarrow case(\lambda z\ .\ z,\ \lambda z.\ z\ ,x) = snd(x)$

⟨*proof*⟩

lemma *lam_case_id*: $(\lambda z{\in}0 + A.\ case(\lambda x.\ x,\ \lambda y.\ y,\ z)) = (\lambda z{\in}0 + A\ .\ snd(z))$
⟨*proof*⟩

lemma *sum_0_eqpoll_rel*: $M(A) \Longrightarrow 0{+}A \approx^M A$
⟨*proof*⟩

lemma *cadd_rel_0* [*simp*]: $Card^M(K) \Longrightarrow M(K) \Longrightarrow 0 \oplus^M K = K$
⟨*proof*⟩

23.2.4 Addition by another cardinal

lemma *sum_lepoll_rel_self*: $M(A) \Longrightarrow M(B) \Longrightarrow A \lesssim^M A{+}B$
⟨*proof*⟩

lemma *cadd_rel_le_self*:
assumes K: $Card^M(K)$ **and** L: $Ord(L)$ **and**
types:$M(K)$ $M(L)$
shows $K \le (K \oplus^M L)$
⟨*proof*⟩

23.2.5 Monotonicity of addition

lemma *sum_lepoll_rel_mono*:
$[\!|\ A \lesssim^M C;\ \ B \lesssim^M D;\ M(A);\ M(B);\ M(C);\ M(D)\ |\!] ==> A + B \lesssim^M C + D$
⟨*proof*⟩

lemma *cadd_rel_le_mono*:
$[\!|\ K' \le K;\ \ L' \le L; M(K'); M(K); M(L'); M(L)\ |\!] ==> (K' \oplus^M L') \le (K \oplus^M L)$
⟨*proof*⟩

23.2.6 Addition of finite cardinals is "ordinary" addition

lemma *sum_succ_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow succ(A){+}B \approx^M succ(A{+}B)$
⟨*proof*⟩

lemma *cadd_succ_lemma*:
assumes $Ord(m)$ $Ord(n)$ **and**
types: $M(m)$ $M(n)$
shows $succ(m) \oplus^M n = |succ(m \oplus^M n)|^M$
⟨*proof*⟩

lemma *nat_cadd_rel_eq_add*:
assumes m: $m \in nat$ **and** [*simp*]: $n \in nat$ **shows**$m \oplus^M n = m +_\omega n$
⟨*proof*⟩

23.3 Cardinal multiplication

23.3.1 Cardinal multiplication is commutative

lemma *prod_commute_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow A*B \approx^M B*A$
⟨*proof*⟩

lemma *cmult_rel_commute*: $M(i) \Longrightarrow M(j) \Longrightarrow i \otimes^M j = j \otimes^M i$
⟨*proof*⟩

23.3.2 Cardinal multiplication is associative

lemma *prod_assoc_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (A*B)*C \approx^M A*(B*C)$
⟨*proof*⟩

Unconditional version requires AC

lemma *well_ord_cmult_rel_assoc*:
assumes *i*: *well_ord*(*i*,*ri*) **and** *j*: *well_ord*(*j*,*rj*) **and** *k*: *well_ord*(*k*,*rk*)
and
types: $M(i)$ $M(ri)$ $M(j)$ $M(rj)$ $M(k)$ $M(rk)$
shows $(i \otimes^M j) \otimes^M k = i \otimes^M (j \otimes^M k)$
⟨*proof*⟩

23.3.3 Cardinal multiplication distributes over addition

lemma *sum_prod_distrib_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (A+B)*C \approx^M (A*C)+(B*C)$
⟨*proof*⟩

lemma *well_ord_cadd_cmult_distrib*:
assumes *i*: *well_ord*(*i*,*ri*) **and** *j*: *well_ord*(*j*,*rj*) **and** *k*: *well_ord*(*k*,*rk*)
and
types: $M(i)$ $M(ri)$ $M(j)$ $M(rj)$ $M(k)$ $M(rk)$
shows $(i \oplus^M j) \otimes^M k = (i \otimes^M k) \oplus^M (j \otimes^M k)$
⟨*proof*⟩

23.3.4 Multiplication by 0 yields 0

lemma *prod_0_eqpoll_rel*: $M(A) \Longrightarrow 0*A \approx^M 0$
⟨*proof*⟩

lemma *cmult_rel_0* [*simp*]: $M(i) \Longrightarrow 0 \otimes^M i = 0$
⟨*proof*⟩

23.3.5 1 is the identity for multiplication

lemma *prod_singleton_eqpoll_rel*: $M(x) \Longrightarrow M(A) \Longrightarrow \{x\}*A \approx^M A$
⟨*proof*⟩

lemma *cmult_rel_1* [*simp*]: $Card^M(K) \Longrightarrow M(K) \Longrightarrow 1 \otimes^M K = K$

⟨*proof*⟩

23.4 Some inequalities for multiplication

lemma *prod_square_lepoll_rel*: $M(A) \Longrightarrow A \lesssim^M A*A$
⟨*proof*⟩

lemma *cmult_rel_square_le*: $Card^M(K) \Longrightarrow M(K) \Longrightarrow K \leq K \otimes^M K$
⟨*proof*⟩

23.4.1 Multiplication by a non-zero cardinal

lemma *prod_lepoll_rel_self*: $b \in B \Longrightarrow M(b) \Longrightarrow M(B) \Longrightarrow M(A) \Longrightarrow A \lesssim^M A*B$
⟨*proof*⟩

lemma *cmult_rel_le_self*:
$[\!|\ Card^M(K);\ \ Ord(L);\ \ 0<L;\ M(K);M(L)\ |\!] ==> K \leq (K \otimes^M L)$
⟨*proof*⟩

23.4.2 Monotonicity of multiplication

lemma *prod_lepoll_rel_mono*:
$[\!|\ A \lesssim^M C;\ \ B \lesssim^M D;\ M(A);\ M(B);\ M(C);\ M(D)|\!] ==> A * B\ \lesssim^M\ \ C * D$
⟨*proof*⟩

lemma *cmult_rel_le_mono*:
$[\!|\ K' \leq K;\ \ L' \leq L;M(K');M(K);M(L');M(L)\ |\!] ==> (K' \otimes^M L') \leq (K \otimes^M L)$
⟨*proof*⟩

23.5 Multiplication of finite cardinals is "ordinary" multiplication

lemma *prod_succ_eqpoll_rel*: $M(A) \Longrightarrow M(B) \Longrightarrow succ(A)*B \approx^M B + A*B$
⟨*proof*⟩

lemma *cmult_rel_succ_lemma*:
$[\!|\ Ord(m);\ \ Ord(n)\ ;\ M(m);\ M(n)\ |\!] ==> succ(m) \otimes^M n = n \oplus^M (m \otimes^M n)$
⟨*proof*⟩

lemma *nat_cmult_rel_eq_mult*: $[\!|\ m \in nat;\ \ n \in nat\ |\!] ==> m \otimes^M n = m\#*n$
⟨*proof*⟩

lemma *cmult_rel_2*: $Card^M(n) \Longrightarrow M(n) \Longrightarrow 2 \otimes^M n = n \oplus^M n$
⟨*proof*⟩

lemma *sum_lepoll_rel_prod*:
assumes C: $2 \lesssim^M C$ **and**
types:$M(C)$ $M(B)$
shows $B+B \lesssim^M C*B$

⟨*proof*⟩

lemma *lepoll_imp_sum_lepoll_prod*: [| $A \lesssim^M B$; $2 \lesssim^M A$; $M(A)$;$M(B)$ |] ==> $A+B \lesssim^M A*B$
⟨*proof*⟩

end — *M_cardinals*

23.6 Infinite Cardinals are Limit Ordinals

context *M_pre_cardinal_arith*
begin

lemma *nat_cons_lepoll_rel*: $nat \lesssim^M A \Longrightarrow M(A) \Longrightarrow M(u) ==> cons(u,A) \lesssim^M A$
⟨*proof*⟩

lemma *nat_cons_eqpoll_rel*: $nat \lesssim^M A ==> M(A) \Longrightarrow M(u) \Longrightarrow cons(u,A) \approx^M A$
⟨*proof*⟩

lemma *nat_succ_eqpoll_rel*: $nat \subseteq A ==> M(A) \Longrightarrow succ(A) \approx^M A$
⟨*proof*⟩

lemma *InfCard_rel_nat*: $InfCard^M(nat)$
⟨*proof*⟩

lemma *InfCard_rel_is_Card_rel*: $M(K) \Longrightarrow InfCard^M(K) \Longrightarrow Card^M(K)$
⟨*proof*⟩

lemma *InfCard_rel_Un*:
[| $InfCard^M(K)$; $Card^M(L)$; $M(K)$; $M(L)$ |] ==> $InfCard^M(K \cup L)$
⟨*proof*⟩

lemma *InfCard_rel_is_Limit*: $InfCard^M(K) ==> M(K) \Longrightarrow Limit(K)$
⟨*proof*⟩

end — *M_pre_cardinal_arith*

lemma (**in** *M_ordertype*) *ordertype_abs*[*absolut*]:
assumes *wellordered*(M,A,r) $M(A)$ $M(r)$ $M(i)$
shows *otype*(M,A,r,i) $\longleftrightarrow$ i = *ordertype*(A,r)
⟨*proof*⟩

lemma (**in** *M_ordertype*) *ordertype_closed*[*intro*,*simp*]: ⟦ *wellordered*(M,A,r);$M(A)$;$M(r)$⟧ $\Longrightarrow$ M(*ordertype*(A,r))
⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_trivial*) *is_transitive_iff_transitive_rel*:
M(*A*)⟹ *M*(*r*) ⟹ *transitive_rel*(*M*, *A*, *r*) ⟷ *is_transitive*(*M*,*A*, *r*)
⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_trivial*) *is_linear_iff_linear_rel*:
M(*A*)⟹ *M*(*r*) ⟹ *is_linear*(*M*,*A*, *r*) ⟷ *linear_rel*(*M*, *A*, *r*)
⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_trivial*) *is_wellfounded_on_iff_wellfounded_on*:
M(*A*)⟹ *M*(*r*) ⟹ *is_wellfounded_on*(*M*,*A*, *r*) ⟷ *wellfounded_on*(*M*, *A*, *r*)
⟨*proof*⟩

definition
is_well_ord :: [*i*=>*o*,*i*,*i*]=>*o* **where**
— linear and wellfounded on *A*
is_well_ord(*M*,*A*,*r*) ==
is_transitive(*M*,*A*,*r*) ∧ *is_linear*(*M*,*A*,*r*) ∧ *is_wellfounded_on*(*M*,*A*,*r*)

lemma (**in** *M_trivial*) *is_well_ord_iff_wellordered*:
M(*A*)⟹ *M*(*r*) ⟹ *is_well_ord*(*M*,*A*, *r*) ⟷ *wellordered*(*M*, *A*, *r*)
⟨*proof*⟩

⟨*ML*⟩

context *M_pre_cardinal_arith*
begin

⟨*ML*⟩
⟨*proof*⟩

⟨*ML*⟩
⟨*proof*⟩

end — *M_pre_cardinal_arith*

⟨*ML*⟩

lemma *trans_on_iff_trans*: *trans*[*A*](*r*) ⟷ *trans*(*r* ∩ *A*×*A*)
⟨*proof*⟩

lemma *trans_on_subset*: *trans*[*A*](*r*) ⟹ *B* ⊆ *A* ⟹ *trans*[*B*](*r*)
⟨*proof*⟩

lemma *relation_Int*: *relation*(*r* ∩ *B*×*B*)

⟨*proof*⟩

Discipline for *ordermap*

⟨*ML*⟩

lemma *wfrec_on_pred_eq*:
assumes $r \in Pow(A \times A)$
shows $wfrec[A](r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r)) = wfrec(r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r))$
⟨*proof*⟩

context *M_pre_cardinal_arith*
begin

lemma *wfrec_on_pred_closed*:
assumes $wf[A](r)\ trans[A](r)\ r \in Pow^M(A \times A)\ M(A)\ x \in A$
shows $M(wfrec(r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r)))$
⟨*proof*⟩

lemma *wfrec_on_pred_closed′*:
assumes $wf[A](r)\ trans[A](r)\ r \in Pow^M(A \times A)\ M(A)\ x \in A$
shows $M(wfrec[A](r,\ x,\ \lambda x\ f.\ f\ ``\ Order.pred(A,\ x,\ r)))$
⟨*proof*⟩

lemma *ordermap_rel_closed*[*intro*,*simp*]:
assumes $wf[A](r)\ trans[A](r)\ r \in Pow^M(A \times A)\ M(A)$
shows $M(ordermap\_rel(M,\ A,\ r))$
⟨*proof*⟩

lemma *is_ordermap_iff*:
assumes $r \in Pow^M(A \times A)\ wf[A](r)\ trans[A](r)$
$M(A)\ M(res)$
shows $is\_ordermap(M,\ A,\ r,\ res) \longleftrightarrow res = ordermap\_rel(M,\ A,\ r)$
⟨*proof*⟩

end — *M_pre_cardinal_arith*

Discipline for *ordertype*

⟨*ML*⟩

definition *is_order_body*
where $is\_order\_body(M,p,z) \equiv \exists\, X[M].\exists\, r[M].\exists\, A[M].\ is\_fst(M,p,X) \wedge is\_snd(M,p,r) \wedge$
$cartprod(M,X,X,A) \wedge subset(M,r,A) \wedge M(z) \wedge M(r) \wedge is\_well\_ord(M,X,r) \wedge is\_ordertype(M,X,r,z)$

context *M_pre_cardinal_arith*
begin

lemma *is_ordertype_iff*:
assumes $r \in Pow^M(A\times A)$ *well_ord*(A,r)
shows $M(A) \Longrightarrow M(res) \Longrightarrow$ *is_ordertype*(M, A, r, res) $\longleftrightarrow$ res = *ordertype_rel*(M, A, r)
⟨*proof*⟩

lemma *ordertype_rel_abs*:
assumes *wellordered*(M,X,r) $M(X)$ $M(r)$
shows *ordertype_rel*(M,X,r) = *ordertype*(X,r)
⟨*proof*⟩

lemma (**in** *M_pre_cardinal_arith*) *is_order_body_abs* :
$M(Xr) \Longrightarrow M(z) \Longrightarrow$ *is_order_body*(M, Xr, z) $\longleftrightarrow$
snd(Xr)$\in Pow^M$(*fst*(Xr)$\times$*fst*(Xr)) $\wedge$ *well_ord*(*fst*(Xr), *snd*(Xr)) $\wedge$ z = *ordertype*(*fst*(Xr), *snd*(Xr))
⟨*proof*⟩

lemma *well_ord_restr*: *well_ord*(X, r) $\Longrightarrow$ *well_ord*(X, $r \cap X\times X$)
⟨*proof*⟩

lemma *ordertype_restr_eq* :
assumes *well_ord*(X,r)
shows *ordertype*(X, r) = *ordertype*(X, $r \cap X\times X$)
⟨*proof*⟩

lemma *ordertype_rel_closed*[*intro*,*simp*]:
assumes *well_ord*(A,r) $r \in Pow^M(A\times A)$ $M(A)$
shows M(*ordertype_rel*(M,A,r))
⟨*proof*⟩

end — *M_pre_cardinal_arith*

⟨*ML*⟩

Notice that this is not quite the same as *jump_cardinal*: observe $Pow(X \times X)$.

definition
jump_cardinal' :: $i\Rightarrow i$ **where**
jump_cardinal'(K) ≡
$\bigcup X\in Pow(K).\ \{z.\ r \in Pow(X*X),\ z = jump\_cardinal\_body(Pow(X*X),X)\}$

⟨*ML*⟩

lemma (**in** *M_ordertype*) *ordermap_closed*[*intro*,*simp*]:
assumes *wellordered*(M,A,r) **and** *types*:$M(A)$ $M(r)$
shows M(*ordermap*(A,r))
⟨*proof*⟩

context *M_pre_cardinal_arith*
begin

lemma *def_jump_cardinal_body*:
M(*X*) ⟹ *M*(*U*) ⟹
jump_cardinal_body(*U*,*X*) = {*z* . *r* ∈ *U*, *M*(*z*) ∧ *well_ord*(*X*, *r*) ∧ *z* = *ordertype*(*X*, *r*)}
⟨*proof*⟩

lemma *jump_cardinal_body_abs*:
shows *M*(*A*) ⟹ *jump_cardinal_body_rel*(*M*,*Pow*^{M}(*A*×*A*),*A*) = *jump_cardinal_body*(*Pow*^{M}(*A*×*A*),*A*)
⟨*proof*⟩

end — *M_pre_cardinal_arith*

locale *M_cardinal_arith* = *M_pre_cardinal_arith* +
assumes
is_ordertype_replacement :
strong_replacement(*M*,λ *x z* . ∃ *y*[*M*]. *is_order_body*(*M*,*x*,*y*) ∧ *z*=⟨*x*,*y*⟩)
and
pow_rel_separation : ∀ *A*[*M*]. *separation*(*M*, λ*y*. ∃ *x*[*M*]. *x* ∈ *A* ∧ *y* = ⟨*x*, *Pow*^{M}(*x*)⟩)
and
separation_is_well_ord : *separation*(*M*, λ*x* . *is_well_ord*(*M*,*fst*(*x*),*snd*(*x*)))
begin

lemmas *Pow_rel_replacement* = *lam_replacement_Pow_rel*[*OF pow_rel_separation*]

lemma *ordtype_replacement* :
strong_replacement(*M* , λ*x z* . (*snd*(*x*) ∈ *Pow*^{M}(*fst*(*x*)×*fst*(*x*)) ∧ *well_ord*(*fst*(*x*),*snd*(*x*))) ∧
z =⟨*x*, *ordertype*(*fst*(*x*),*snd*(*x*))⟩)
⟨*proof*⟩

lemma *separation_well_ord* : *separation*(*M*, λ*x* . *well_ord*(*fst*(*x*),*snd*(*x*)))
⟨*proof*⟩

lemma *jump_cardinal_body_lam_replacement* :
shows *lam_replacement*(*M*, λ*X* .*jump_cardinal_body*(*Pow*^{M}(*X* × *X*),*X*)) **and**
M(*X*) ⟹ *M*(*jump_cardinal_body*(*Pow*^{M}(*X* × *X*), *X*))
⟨*proof*⟩

lemmas *jump_cardinal_body_closed* = *jump_cardinal_body_lam_replacement*(*2*)

lemma *jump_cardinal_closed*:
assumes *M*(*K*)
shows *M*({*jump_cardinal_body*(*Pow*^{M}(*X* × *X*),*X*) . *X* ∈ *Pow*^{M}(*K*)})
⟨*proof*⟩

end — *M_cardinal_arith*

context *M_pre_cardinal_arith*
begin

lemma *ordermap_eqpoll_pred*:
[| *well_ord*(*A*,*r*); *x* ∈ *A* ; *M*(*A*);*M*(*r*);*M*(*x*)|] ==> *ordermap*(*A*,*r*)‘*x* ≈^{M} *Order.pred*(*A*,*x*,*r*)
⟨*proof*⟩

Kunen: ”each ⟨*x*, *y*⟩ ∈ *K* × *K* has no more than *z* × *z* predecessors...” (page 29)

lemma *ordermap_csquare_le*:
assumes *K*: *Limit*(*K*) **and** *x*: *x*<*K* **and** *y*: *y*<*K*
and *types*: *M*(*K*) *M*(*x*) *M*(*y*)
shows |*ordermap*(*K* × *K*, *csquare_rel*(*K*)) ‘ ⟨*x*,*y*⟩|^M ≤ |*succ*(*succ*(*x* ∪ *y*))|^M ⊗^{M} |*succ*(*succ*(*x* ∪ *y*))|^M
⟨*proof*⟩

Kunen: ”... so the order type is ≤ K”

lemma *ordertype_csquare_le_M*:
assumes *IK*: *InfCard*^{M}(*K*) **and** *eq*: ⋀*y*. *y*∈*K* ⟹ *InfCard*^{M}(*y*) ⟹ *M*(*y*) ⟹ *y* ⊗^{M} *y* = *y*
— Note the weakened hypothesis *eq*
and *types*: *M*(*K*)
shows *ordertype*(*K*∗*K*, *csquare_rel*(*K*)) ≤ *K*
⟨*proof*⟩

lemma *InfCard_rel_csquare_eq*:
assumes *IK*: *InfCard*^{M}(*K*) **and**
types: *M*(*K*)
shows *K* ⊗^{M} *K* = *K*
⟨*proof*⟩

lemma *well_ord_InfCard_rel_square_eq*:
assumes *r*: *well_ord*(*A*,*r*) **and** *I*: *InfCard*^{M}(|*A*|^M) **and**
types: *M*(*A*) *M*(*r*)
shows *A* × *A* ≈^{M} *A*
⟨*proof*⟩

lemma *InfCard_rel_square_eqpoll*:
assumes *InfCard*^{M}(*K*) **and** *types*:*M*(*K*) **shows** *K* × *K* ≈^{M} *K*
⟨*proof*⟩

lemma *Inf_Card_rel_is_InfCard_rel*: [| $Card^M(i)$; ~ *Finite_rel*(*M*,*i*) ; $M(i)$ |] ==> $InfCard^M(i)$
⟨*proof*⟩

23.6.1 Toward's Kunen's Corollary 10.13 (1)

lemma *InfCard_rel_le_cmult_rel_eq*: [| $InfCard^M(K)$; $L \leq K$; $0<L$; $M(K)$; $M(L)$ |] ==> $K \otimes^M L = K$
⟨*proof*⟩

lemma *InfCard_rel_cmult_rel_eq*: [| $InfCard^M(K)$; $InfCard^M(L)$; $M(K)$; $M(L)$ |] ==> $K \otimes^M L = K \cup L$
⟨*proof*⟩

lemma *InfCard_rel_cdouble_eq*: $InfCard^M(K) \Longrightarrow M(K) \Longrightarrow K \oplus^M K = K$
⟨*proof*⟩

lemma *InfCard_rel_le_cadd_rel_eq*: [| $InfCard^M(K)$; $L \leq K$; $M(K)$; $M(L)$|] ==> $K \oplus^M L = K$
⟨*proof*⟩

lemma *InfCard_rel_cadd_rel_eq*: [| $InfCard^M(K)$; $InfCard^M(L)$; $M(K)$; $M(L)$ |] ==> $K \oplus^M L = K \cup L$
⟨*proof*⟩

end — *M_pre_cardinal_arith*

23.7 For Every Cardinal Number There Exists A Greater One

This result is Kunen's Theorem 10.16, which would be trivial using AC

locale *M_cardinal_arith_jump* = *M_cardinal_arith* + *M_ordertype*
begin

lemma *def_jump_cardinal_rel_aux*:
$X \in Pow^M(K) \Longrightarrow M(K) \Longrightarrow$
$\{z \,.\, r \in Pow^M(X \times X),\ M(z) \wedge well\_ord(X,\ r) \wedge z = ordertype(X,\ r)\} =$
$\{z \,.\, r \in Pow^M(K \times K),\ M(z) \wedge well\_ord(X,\ r) \wedge z = ordertype(X,\ r)\}$
⟨*proof*⟩

lemma *def_jump_cardinal_rel_aux2*:
assumes $X \in Pow^M(K)$ $M(K)$
shows *jump_cardinal_body*($Pow^M(K\times K)$,*X*) = *jump_cardinal_body*($Pow^M(X\times X)$,*X*)
⟨*proof*⟩

lemma *def_jump_cardinal_rel*:
  assumes $M(K)$
  shows *jump_cardinal_rel*$(M,K) =$
        $(\bigcup\{$*jump_cardinal_body*$(Pow^M(K*K),X)$. $X\in Pow^M(K)\})$
⟨*proof*⟩

lemma *Ord_jump_cardinal_rel*: $M(K) \Longrightarrow$ *Ord*(*jump_cardinal_rel*(M,K))
  ⟨*proof*⟩

declare *conj_cong* [*cong del*]
  — incompatible with some of the proofs of the original theory

lemma *jump_cardinal_rel_iff_old*:
  $M(i) \Longrightarrow M(K) \Longrightarrow i \in$ *jump_cardinal_rel*$(M,K) \longleftrightarrow$
     $(\exists\, r[M].\ \exists\, X[M].\ r \subseteq K*K\ \&\ X \subseteq K\ \&\ $*well_ord*$(X,r)\ \&\ i =$ *ordertype*$(X,r))$
  ⟨*proof*⟩

lemma *K_lt_jump_cardinal_rel*: *Ord*(K) ==> $M(K) \Longrightarrow K <$ *jump_cardinal_rel*(M,K)
  ⟨*proof*⟩

lemma *def_jump_cardinal_rel'*:
  assumes $M(K)$
  shows *jump_cardinal_rel*$(M,K) =$
          $(\bigcup X\in Pow^M(K).\ \{z.\ r \in Pow^M(X\times X),\ $*well_ord*$(X,r) \wedge z =$ *ordertype*$(X,r)\})$
⟨*proof*⟩

⟨*ML*⟩
⟨*proof*⟩

lemma *Card_rel_jump_cardinal_rel_lemma*:
  [| *well_ord*(X,r); $r \subseteq K * K$; $X \subseteq K$;
        $f \in$ *bij*(*ordertype*(X,r), *jump_cardinal_rel*(M,K));
        $M(X)$; $M(r)$; $M(K)$; $M(f)$ |]
     ==> *jump_cardinal_rel*$(M,K) \in$ *jump_cardinal_rel*(M,K)
  ⟨*proof*⟩

lemma *Card_rel_jump_cardinal_rel*: $M(K) \Longrightarrow$ *Card_rel*$(M,$*jump_cardinal_rel*$(M,K))$
  ⟨*proof*⟩

23.8 Basic Properties of Successor Cardinals

lemma *csucc_rel_basic*: *Ord*(K) ==> $M(K) \Longrightarrow$ *Card_rel*$(M,$*csucc_rel*$(M,K))$
& $K <$ *csucc_rel*(M,K)
  ⟨*proof*⟩

lemmas *Card_rel_csucc_rel* = *csucc_rel_basic* [*THEN conjunct1*]

lemmas *lt_csucc_rel* = *csucc_rel_basic* [*THEN conjunct2*]

lemma *Ord_0_lt_csucc_rel*: *Ord*(*K*) ==> *M*(*K*) ⟹ 0 < *csucc_rel*(*M*,*K*)
⟨*proof*⟩

lemma *csucc_rel_le*: [| *Card_rel*(*M*,*L*); *K*<*L*; *M*(*K*); *M*(*L*) |] ==> *csucc_rel*(*M*,*K*)
≤ *L*
⟨*proof*⟩

lemma *lt_csucc_rel_iff*: [| *Ord*(*i*); *Card_rel*(*M*,*K*); *M*(*K*); *M*(*i*)|] ==> *i* <
csucc_rel(*M*,*K*) ⟷ |*i*|^*M* ≤ *K*
⟨*proof*⟩

lemma *Card_rel_lt_csucc_rel_iff*:
[| *Card_rel*(*M*,*K*′); *Card_rel*(*M*,*K*); *M*(*K*′); *M*(*K*) |] ==> *K*′ < *csucc_rel*(*M*,*K*)
⟷ *K*′ ≤ *K*
⟨*proof*⟩

lemma *InfCard_rel_csucc_rel*: *InfCard_rel*(*M*,*K*) ⟹ *M*(*K*) ==> *InfCard_rel*(*M*,*csucc_rel*(*M*,*K*))
⟨*proof*⟩

23.8.1 Theorems by Krzysztof Grabczewski, proofs by lcp

lemma *nat_sum_eqpoll_rel_sum*:
assumes *m*: *m* ∈ *nat* **and** *n*: *n* ∈ *nat* **shows** *m* + *n* ≈^*M* *m* +_ω *n*
⟨*proof*⟩

lemma *Ord_nat_subset_into_Card_rel*: [| *Ord*(*i*); *i* ⊆ *nat* |] ==> *Card*^*M*(*i*)
⟨*proof*⟩

end — *M_cardinal_arith_jump*

end
theory *Aleph_Relative*
imports
CardinalArith_Relative
begin

definition
HAleph :: [*i*,*i*] ⇒ *i* **where**
HAleph(*i*,*r*) ≡ *if*(¬(*Ord*(*i*)),*i*,*if*(*i*=0, *nat*, *if*(¬*Limit*(*i*) ∧ *i*≠0,
csucc(*r*‘(⋃ *i*)),
⋃*j*∈*i*. *r*‘*j*)))

⟨*ML*⟩

lemma *arity_is_HAleph_fm_aux*:

assumes
$i \in nat\ r \in nat$
— NOTE: assumptions are **not** used, but if omitted, next lemma fails!
shows
$arity(Replace\_fm(8 +_\omega i, \cdot 10 +_\omega r`0\ is\ 1\cdot, 3)) = 9 +_\omega i \cup pred(pred(11 +_\omega r))$
⟨*proof*⟩

lemma *arity_is_HAleph_fm*[*arity*]:
assumes
$i \in nat\ r \in nat\ l \in nat$
shows
$arity(is\_HAleph\_fm(i, r, l)) = succ(i) \cup succ(l) \cup succ(r)$
⟨*proof*⟩

definition
$Aleph' :: i => i$ **where**
$Aleph'(a) == transrec(a,\lambda i\ r.\ HAleph(i,r))$

⟨*ML*⟩

The extra assumptions $a < length(env)$ and $c < length(env)$ in this schematic goal (and the following results on synthesis that depend on it) are imposed by *is_transrec_iff_sats*.

schematic_goal *sats_is_Aleph_fm_auto*:
$a \in nat \Longrightarrow c \in nat \Longrightarrow env \in list(A) \Longrightarrow$
$a < length(env) \Longrightarrow c < length(env) \Longrightarrow 0 \in A \Longrightarrow$
$is\_Aleph(\#\#A, nth(a, env), nth(c, env)) \longleftrightarrow A, env \models ?fm(a, c)$
⟨*proof*⟩

⟨*ML*⟩

notation *is_Aleph_fm* ($\langle\cdot\aleph'(\_')\ is\ \_\cdot\rangle$)

lemma *is_Aleph_fm_type* [*TC*]: $a \in nat \Longrightarrow c \in nat \Longrightarrow is\_Aleph\_fm(a, c) \in formula$
⟨*proof*⟩

lemma *sats_is_Aleph_fm*:
assumes $f{\in}nat\ r{\in}nat\ env \in list(A)\ 0{\in}A\ f < length(env)\ r{<}\ length(env)$
shows $is\_Aleph(\#\#A, nth(f, env), nth(r, env)) \longleftrightarrow A, env \models is\_Aleph\_fm(f,r)$
⟨*proof*⟩

lemma *is_Aleph_iff_sats* [*iff_sats*]:
assumes
$nth(f, env) = fa\ nth(r, env) = ra\ f < length(env)\ r{<}\ length(env)$
$f \in nat\ r \in nat\ env \in list(A)\ 0{\in}A$
shows $is\_Aleph(\#\#A,fa,ra) \longleftrightarrow A, env \models is\_Aleph\_fm(f,r)$
⟨*proof*⟩

⟨*ML*⟩

lemma (**in** *M_cardinal_arith_jump*) *is_Limit_iff*:
  assumes *M*(*a*)
  shows *is_Limit*(*M*,*a*) ⟷ *Limit*(*a*)
  ⟨*proof*⟩

lemma *HAleph_eq_Aleph_recursive*:
  Ord(*i*) ⟹ *HAleph*(*i*,*r*) = (*if i* = *0 then nat*
            else if ∃*j*. *i* = *succ*(*j*) *then csucc*(*r* ‘ (*THE j*. *i* = *succ*(*j*))) *else* ⋃*j*<*i*.
r ‘ *j*)
⟨*proof*⟩

lemma *Aleph′_eq_Aleph*: *Ord*(*a*) ⟹ *Aleph′*(*a*) = *Aleph*(*a*)
  ⟨*proof*⟩

⟨*ML*⟩

abbreviation
  Aleph_r :: [*i*,*i*⇒*o*] ⇒ *i* (‹ℵ_—›) **where**
  Aleph_r(*a*,*M*) ≡ *Aleph_rel*(*M*,*a*)

abbreviation
  Aleph_r_set :: [*i*,*i*] ⇒ *i* (‹ℵ_—›) **where**
  Aleph_r_set(*a*,*M*) ≡ *Aleph_rel*(##*M*,*a*)

lemma *Aleph_rel_def′*: *Aleph_rel*(*M*,*a*) ≡ *transrec*(*a*, λ*i r*. *HAleph_rel*(*M*, *i*, *r*))
  ⟨*proof*⟩

lemma *succ_mem_Limit*: *Limit*(*j*) ⟹ *i* ∈ *j* ⟹ *succ*(*i*) ∈ *j*
  ⟨*proof*⟩

locale *M_pre_aleph* = *M_eclose* + *M_cardinal_arith_jump* +
  assumes
   haleph_transrec_replacement: *M*(*a*) ⟹ *transrec_replacement*(*M*,*is_HAleph*(*M*),*a*)

begin

lemma *aux_ex_Replace_funapply*:
  assumes *M*(*a*) *M*(*f*)
  shows ∃*x*[*M*]. *is_Replace*(*M*, *a*, λ*j y*. *f* ‘ *j* = *y*, *x*)
⟨*proof*⟩

lemma *is_HAleph_zero*:
  assumes *M*(*f*)
  shows *is_HAleph*(*M*,*0*,*f*,*res*) ⟷ *res* = *nat*
  ⟨*proof*⟩

lemma *is_HAleph_succ*:
  assumes $M(f)$ $M(x)$ $Ord(x)$ $M(res)$
  shows $is\_HAleph(M,succ(x),f,res) \longleftrightarrow res = csucc\_rel(M,f`x)$
  ⟨*proof*⟩

lemma *is_HAleph_limit*:
  assumes $M(f)$ $M(x)$ $Limit(x)$ $M(res)$
  shows $is\_HAleph(M,x,f,res) \longleftrightarrow res = (\bigcup \{y\ .\ i{\in}x\ ,M(i) \wedge M(y) \wedge y = f`i\})$
⟨*proof*⟩

lemma *is_HAleph_iff*:
  assumes $M(a)$ $M(f)$ $M(res)$
  shows $is\_HAleph(M,\ a,\ f,\ res) \longleftrightarrow res = HAleph\_rel(M,\ a,\ f)$
⟨*proof*⟩

lemma *HAleph_rel_closed* [*intro*,*simp*]:
  assumes $function(f)$ $M(a)$ $M(f)$
  shows $M(HAleph\_rel(M,a,f))$
  ⟨*proof*⟩

lemma *Aleph_rel_closed*[*intro*, *simp*]:
  assumes $Ord(a)$ $M(a)$
  shows $M(Aleph\_rel(M,a))$
⟨*proof*⟩

lemma *Aleph_rel_zero*: ${\aleph_0}^M = nat$
  ⟨*proof*⟩

lemma *Aleph_rel_succ*: $Ord(\alpha) \Longrightarrow M(\alpha) \Longrightarrow {\aleph_{succ(\alpha)}}^M = ({\aleph_\alpha}^{M+})^M$
  ⟨*proof*⟩

lemma *Aleph_rel_limit*:
  assumes $Limit(\alpha)$ $M(\alpha)$
  shows ${\aleph_\alpha}^M = \bigcup \{{\aleph_j}^M\ .\ j \in \alpha\}$
⟨*proof*⟩

lemma *is_Aleph_iff*:
  assumes $Ord(a)$ $M(a)$ $M(res)$
  shows $is\_Aleph(M,\ a,\ res) \longleftrightarrow res = {\aleph_a}^M$
⟨*proof*⟩

end — *M_pre_aleph*

locale *M_aleph* = *M_pre_aleph* +
  assumes
   aleph_rel_separation: $Ord(x) \Longrightarrow M(x) \Longrightarrow separation(M,\ \lambda y.\ \exists\, z{\in}x.\ y = {\aleph_z}^M)$
begin

lemma *Aleph_rel_cont*: $Limit(l) \Longrightarrow M(l) \Longrightarrow {\aleph_l}^M = (\bigcup i{<}l.\ {\aleph_i}^M)$

⟨*proof*⟩

lemma *Ord_Aleph_rel*:
assumes *Ord*(*a*)
shows *M*(*a*) ⟹ *Ord*(ℵ$_a{}^M$)
⟨*proof*⟩

lemma *Aleph_rel_increasing*:
assumes *a* < *b* **and** *types*: *M*(*a*) *M*(*b*)
shows ℵ$_a{}^M$ < ℵ$_b{}^M$
⟨*proof*⟩

lemma *Card_rel_Aleph_rel* [*simp*, *intro*]:
assumes *Ord*(*a*) **and** *types*: *M*(*a*) **shows** *Card*^{M}(ℵ$_a{}^M$)
⟨*proof*⟩

lemmas *nat_subset_Aleph_rel_1* =
Ord_lt_subset[*OF Ord_Aleph_rel*[*of 1*] *Aleph_rel_increasing*[*of 0 1*,*simplified*],*simplified*]

end — *M_aleph*

end

24 Relative, Cardinal Arithmetic Using AC

theory *Cardinal_AC_Relative*
imports
CardinalArith_Relative

begin

locale *M_AC* =
fixes *M*
assumes
choice_ax: *choice_ax*(*M*)

locale *M_cardinal_AC* = *M_cardinal_arith* + *M_AC* +
assumes
lam_replacement_minimum:*lam_replacement*(*M*, λ*p*. *minimum*(*fst*(*p*),*snd*(*p*)))
begin

lemma *lam_replacement_minimum_vimage*:
M(*f*) ⟹ *M*(*r*) ⟹ *lam_replacement*(*M*, λ*x*. *minimum*(*r*, *f* -‘‘ {*x*}))
⟨*proof*⟩

lemmas *surj_imp_inj_replacement4* = *lam_replacement_minimum_vimage*[*unfolded lam_replacement_def*]

lemmas *surj_imp_inj_replacement* =

surj_imp_inj_replacement1 surj_imp_inj_replacement2 surj_imp_inj_replacement4
lam_replacement_vimage_sing_fun[*THEN lam_replacement_imp_strong_replacement*]

lemma *well_ord_surj_imp_lepoll_rel*:
assumes *well_ord*(*A*,*r*) $h \in surj(A,B)$ **and**
types:$M(A)$ $M(r)$ $M(h)$ $M(B)$
shows $B \lesssim^M A$
⟨*proof*⟩

lemma *surj_imp_well_ord_M*:
assumes *wos*: *well_ord*(*A*,*r*) $h \in surj(A,B)$
and
types: $M(A)$ $M(r)$ $M(h)$ $M(B)$
shows $\exists\, s[M].$ *well_ord*(*B*,*s*)
⟨*proof*⟩

lemma *choice_ax_well_ord*: $M(S) \Longrightarrow \exists\, r[M].$ *well_ord*(*S*,*r*)
⟨*proof*⟩

lemma *Finite_cardinal_rel_Finite*:
assumes *Finite*($|i|^M$) $M(i)$
shows *Finite*(*i*)
⟨*proof*⟩

end — *M_cardinal_AC*

locale *M_Pi_assumptions_choice* = *M_Pi_assumptions* + *M_cardinal_AC* +
assumes
B_replacement: *strong_replacement*(*M*, $\lambda x\ y.\ y = B(x)$)
and
— The next one should be derivable from (some variant) of B_replacement. Proving both instances each time seems inconvenient.
minimum_replacement: $M(r) \Longrightarrow$ *strong_replacement*(*M*, $\lambda x\ y.\ y = \langle x,$ *minimum*$(r,\ B(x))\rangle$)
begin

lemma *AC_M*:
assumes $a \in A$ $\bigwedge x.\ x \in A \Longrightarrow \exists\, y.\ y \in B(x)$
shows $\exists\, z[M].\ z \in Pi^M(A,\ B)$
⟨*proof*⟩

lemma *AC_Pi_rel*: **assumes** $\bigwedge x.\ x \in A \Longrightarrow \exists\, y.\ y \in B(x)$
shows $\exists\, z[M].\ z \in Pi^M(A,\ B)$
⟨*proof*⟩

end — *M_Pi_assumptions_choice*

context *M_cardinal_AC*
begin

24.1 Strengthened Forms of Existing Theorems on Cardinals

lemma *cardinal_rel_eqpoll_rel*: $M(A) \Longrightarrow |A|^M \approx^M A$
⟨*proof*⟩

lemmas *cardinal_rel_idem* = *cardinal_rel_eqpoll_rel* [*THEN cardinal_rel_cong*, *simp*]

lemma *cardinal_rel_eqE*: $|X|^M = |Y|^M \Longrightarrow M(X) \Longrightarrow M(Y) \Longrightarrow X \approx^M Y$
⟨*proof*⟩

lemma *cardinal_rel_eqpoll_rel_iff*: $M(X) \Longrightarrow M(Y) \Longrightarrow |X|^M = |Y|^M \longleftrightarrow X \approx^M Y$
⟨*proof*⟩

lemma *cardinal_rel_disjoint_Un*:
$[\!| |A|^M=|B|^M;\ |C|^M=|D|^M;\ A \cap C = 0;\ B \cap D = 0;\ M(A);\ M(B);\ M(C);\ M(D) |\!]$
$\Longrightarrow |A \cup C|^M = |B \cup D|^M$
⟨*proof*⟩

lemma *lepoll_rel_imp_cardinal_rel_le*: $A \lesssim^M B \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow |A|^M \le |B|^M$
⟨*proof*⟩

lemma *cadd_rel_assoc*: $[\![M(i);\ M(j);\ M(k)]\!] \Longrightarrow (i \oplus^M j) \oplus^M k = i \oplus^M (j \oplus^M k)$
⟨*proof*⟩

lemma *cmult_rel_assoc*: $[\![M(i);\ M(j);\ M(k)]\!] \Longrightarrow (i \otimes^M j) \otimes^M k = i \otimes^M (j \otimes^M k)$
⟨*proof*⟩

lemma *cadd_cmult_distrib*: $[\![M(i);\ M(j);\ M(k)]\!] \Longrightarrow (i \oplus^M j) \otimes^M k = (i \otimes^M k) \oplus^M (j \otimes^M k)$
⟨*proof*⟩

lemma *InfCard_rel_square_eq*: $InfCard^M(|A|^M) \Longrightarrow M(A) \Longrightarrow A \times A \approx^M A$
⟨*proof*⟩

24.2 The relationship between cardinality and le-pollence

lemma *Card_rel_le_imp_lepoll_rel*:
assumes $|A|^M \le |B|^M$
and *types*: $M(A)$ $M(B)$
shows $A \lesssim^M B$

⟨*proof*⟩

lemma *le_Card_rel_iff*: $Card^M(K) ==> M(K) \Longrightarrow M(A) \Longrightarrow |A|^M \leq K \longleftrightarrow A \lesssim^M K$
⟨*proof*⟩

lemma *cardinal_rel_0_iff_0* [*simp*]: $M(A) \Longrightarrow |A|^M = 0 \longleftrightarrow A = 0$
⟨*proof*⟩

lemma *cardinal_rel_lt_iff_lesspoll_rel*:
assumes *i*: $Ord(i)$ **and**
types: $M(i)$ $M(A)$
shows $i < |A|^M \longleftrightarrow i \prec^M A$
⟨*proof*⟩

lemma *cardinal_rel_le_imp_lepoll_rel*: $i \leq |A|^M ==> M(i) \Longrightarrow M(A) \Longrightarrow i \lesssim^M A$
⟨*proof*⟩

24.3 Other Applications of AC

We have an example of instantiating a locale involving higher order variables inside a proof, by using the assumptions of the first order, active locale.

lemma *surj_rel_implies_inj_rel*:
assumes *f*: $f \in surj^M(X, Y)$ **and**
types: $M(f)$ $M(X)$ $M(Y)$
shows $\exists\, g[M].\ g \in inj^M(Y,X)$
⟨*proof*⟩

Kunen's Lemma 10.20

lemma *surj_rel_implies_cardinal_rel_le*:
assumes *f*: $f \in surj^M(X, Y)$ **and**
types:$M(f)$ $M(X)$ $M(Y)$
shows $|Y|^M \leq |X|^M$
⟨*proof*⟩

end — *M_cardinal_AC*

The set-theoretic universe.

abbreviation
Universe :: $i \Rightarrow o$ ($\langle \mathcal{V} \rangle$) **where**
$\mathcal{V}(x) \equiv True$

lemma *separation_absolute*: *separation*($\mathcal{V}$, P)
⟨*proof*⟩

lemma *univalent_absolute*:
assumes *univalent*($\mathcal{V}$, A, P) $P(x, b)$ $x \in A$

shows $P(x,\ y) \Longrightarrow y = b$
⟨*proof*⟩

lemma *replacement_absolute*: *strong_replacement*($\mathcal{V}$, P)
⟨*proof*⟩

lemma *Union_ax_absolute*: *Union_ax*($\mathcal{V}$)
⟨*proof*⟩

lemma *upair_ax_absolute*: *upair_ax*($\mathcal{V}$)
⟨*proof*⟩

lemma *power_ax_absolute*:*power_ax*($\mathcal{V}$)
⟨*proof*⟩

locale *M_cardinal_UN* = *M_Pi_assumptions_choice* _ *K X* **for** *K X* +
assumes
— The next assumption is required by *Least_closed*
X_witness_in_M: $w \in X(x) \Longrightarrow M(x)$
and
lam_m_replacement:$M(f) \Longrightarrow$ *strong_replacement*(M,
$\lambda x\ y.\ y = \langle x,\ \mu\ i.\ x \in X(i),\ f\ `\ (\mu\ i.\ x \in X(i))\ `\ x\rangle$)
and
inj_replacement:
$M(x) \Longrightarrow$ *strong_replacement*(M, $\lambda y\ z.\ y \in inj^M(X(x),\ K) \wedge z = \{\langle x,\ y\rangle\}$)
strong_replacement(M, $\lambda x\ y.\ y = inj^M(X(x),\ K)$)
strong_replacement(M,
$\lambda x\ z.\ z = Sigfun(x,\ \lambda i.\ inj^M(X(i),\ K))$)
$M(r) \Longrightarrow$ *strong_replacement*(M,
$\lambda x\ y.\ y = \langle x,\ minimum(r,\ inj^M(X(x),\ K))\rangle$)

begin

lemma *UN_closed*: $M(\bigcup i{\in}K.\ X(i))$
⟨*proof*⟩

Kunen's Lemma 10.21

lemma *cardinal_rel_UN_le*:
assumes K: $InfCard^M(K)$
shows $(\bigwedge i.\ i{\in}K \Longrightarrow |X(i)|^M \le K) \Longrightarrow |\bigcup i{\in}K.\ X(i)|^M \le K$
⟨*proof*⟩

end — *M_cardinal_UN*

end

25 Relativization of Finite Functions

theory *FiniteFun_Relative*

imports
Lambda_Replacement
begin

lemma *FiniteFunI* :
assumes $f{\in}Fin(A{\times}B)$ $function(f)$
shows $f \in A$ -||> B
⟨*proof*⟩

25.1 The set of finite binary sequences

We implement the poset for adding one Cohen real, the set $2^{<\omega}$ of finite binary sequences.

definition
$seqspace :: [i,i] \Rightarrow i$ ($‹\_^{<}\_›$ $[100,1]100$) **where**
$B^{<\alpha} \equiv \bigcup n{\in}\alpha.\ (n{\to}B)$

lemma *seqspaceI*[*intro*]: $n{\in}\alpha \Longrightarrow f{:}n{\to}B \Longrightarrow f{\in}B^{<\alpha}$
⟨*proof*⟩

lemma *seqspaceD*[*dest*]: $f{\in}B^{<\alpha} \Longrightarrow \exists\, n{\in}\alpha.\ f{:}n{\to}B$
⟨*proof*⟩

locale *M_pre_seqspace* = *M_trancl* + *M_replacement* + *M_Pi*
begin

lemma *function_space_subset_Pow_rel*:
assumes $n{\in}\omega$ $M(B)$
shows $n{\to}B \subseteq Pow^M(\bigcup(\omega{\to}^M B))$
⟨*proof*⟩

lemma *seqspace_subset_Pow_rel*:
assumes $M(B)$
shows $B^{<\omega} \subseteq Pow^M(\bigcup(\omega{\to}^M B))$
⟨*proof*⟩

lemma *seqspace_imp_M*:
assumes $x \in B^{<\omega}$ $M(B)$
shows $M(x)$
⟨*proof*⟩

lemma *seqspace_eq_Collect*:
assumes $M(B)$
shows $B^{<\omega} = \{z \in Pow^M(\bigcup(\omega{\to}^M B)).\ \exists\, x[M].\ \exists\, n[M].\ n \in \omega \wedge z \in x \wedge x = n \to^M B\}$
⟨*proof*⟩

end — *M_pre_seqspace*

locale *M_seqspace* = *M_pre_seqspace* +
assumes
seqspace_separation: $M(B) \Longrightarrow$ *separation*$(M,\lambda z.\ \exists\, x[M].\ \exists\, n[M].\ n \in \omega \wedge z \in x \wedge x = n \rightarrow^{M} B)$
begin

lemma *seqspace_closed*:
$M(B) \Longrightarrow M(B^{<\omega})$
⟨*proof*⟩

end — *M_seqspace*

25.2 Representation of finite functions

A function $f \in A \rightarrow_{fin} B$ can be represented by a function $g \in |f| \rightarrow A \times B$. It is clear that f can be represented by any $g' = g{\cdot}\pi$, where π is a permutation $\pi \in dom(g) \rightarrow dom(g)$. We use this representation of $A \rightarrow_{fin} B$ to prove that our model is closed under $\_ \rightarrow_{fin} \_$.

A function $g \in n \rightarrow A \times B$ that is functional in the first components.

definition *cons_like* :: $i \Rightarrow o$ **where**
$cons\_like(f) \equiv \forall\ i{\in}domain(f)\ .\ \forall\, j{\in}i\ .\ fst(f`i) \neq fst(f`j)$

definition *FiniteFun_iso* :: $[i,i,i,i] \Rightarrow o$ **where**
$FiniteFun\_iso(A,B,n,g,f) \equiv\ (\forall\ i{\in}n\ .\ g`i \in f) \wedge (\forall\ ab{\in}f.\ (\exists\ i{\in}n.\ g`i{=}ab))$

From a function $g \in n \rightarrow A \times B$ we obtain a finite function in A -||> B.

definition *to_FiniteFun* :: $i \Rightarrow i$ **where**
$to\_FiniteFun(f) \equiv \{f`i.\ i{\in}domain(f)\}$

definition *FiniteFun_Repr* :: $[i,i] \Rightarrow i$ **where**
$FiniteFun\_Repr(A,B) \equiv \{f \in (A{\times}B)^{<\omega}\ .\ cons\_like(f)\ \}$

locale *M_FiniteFun* = *M_seqspace* +
assumes
separation_is_function : *separation*$(M,\ is\_function(M))$
begin

lemma *cons_like_separation* : *separation*$(M,\lambda f.\ cons\_like(f))$
⟨*proof*⟩

lemma *supset_separation*: *separation*$(M,\ \lambda\ x.\ \exists\, a.\ \exists\, b.\ x = \langle a,b\rangle \wedge b \subseteq a)$
⟨*proof*⟩

lemma *to_finiteFun_replacement*: *strong_replacement*$(M,\ \lambda x\ y.\ y = range(x))$
⟨*proof*⟩

lemma *fun_range_eq*: $f{\in}A{\rightarrow}B \Longrightarrow \{f`i\ .\ i{\in}domain(f)\ \} = range(f)$

⟨*proof*⟩

lemma *FiniteFun_fst_type*:
assumes $h \in A \text{-}||> B$ $p \in h$
shows $fst(p) \in domain(h)$
⟨*proof*⟩

lemma *FinFun_closed*:
$M(A) \Longrightarrow M(B) \Longrightarrow M(\bigcup \{n \to A \times B \;.\; n \in \omega\})$
⟨*proof*⟩

lemma *cons_like_lt* :
assumes $n \in \omega$ $f \in succ(n) \to A \times B$ *cons_like*(f)
shows *restrict*$(f,n) \in n \to A \times B$ *cons_like*(*restrict*(f,n))
⟨*proof*⟩

A finite function $f \in A \text{-}||> B$ can be represented by a function $g \in n \to A \times B$, with $n = |f|$.

lemma *FiniteFun_iso_intro1*:
assumes $f \in (A \text{-}||> B)$
shows $\exists\, n \in \omega \;.\; \exists\, g \in n \to A \times B.$ *FiniteFun_iso*$(A,B,n,g,f) \wedge$ *cons_like*(g)
⟨*proof*⟩

All the representations of $f \in A \text{-}||> B$ are equal.

lemma *FiniteFun_isoD* :
assumes $n \in \omega$ $g \in n \to A \times B$ $f \in A \text{-}||> B$ *FiniteFun_iso*(A,B,n,g,f)
shows *to_FiniteFun*$(g) = f$
⟨*proof*⟩

lemma *to_FiniteFun_succ_eq* :
assumes $n \in \omega$ $f \in succ(n) \to A$
shows *to_FiniteFun*$(f) =$ *cons*$(f`n$,*to_FiniteFun*(*restrict*$(f,n)))$
⟨*proof*⟩

If $g \in n \to A \times B$ is *cons_like*, then it is a representation of *to_FiniteFun*(g).

lemma *FiniteFun_iso_intro_to*:
assumes $n \in \omega$ $g \in n \to A \times B$ *cons_like*(g)
shows *to_FiniteFun*$(g) \in (A \text{-}||> B) \wedge$ *FiniteFun_iso*$(A,B,n,g,$*to_FiniteFun*$(g))$
⟨*proof*⟩

lemma *FiniteFun_iso_intro2*:
assumes $n \in \omega$ $f \in n \to A \times B$ *cons_like*(f)
shows $\exists\;\; g \in (A \text{-}||> B)$. *FiniteFun_iso*(A,B,n,f,g)
⟨*proof*⟩

lemma *FiniteFun_eq_range_Repr* :
shows $\{$*range*(h) . $h \in$ *FiniteFun_Repr*(A,B) $\} = \{$*to_FiniteFun*(h) . $h \in$ *FiniteFun_Repr*(A,B) $\}$
⟨*proof*⟩

lemma *FiniteFun_eq_to_FiniteFun_Repr* :
  shows *A*-||>*B* = {*to_FiniteFun*(*h*) . *h* ∈ *FiniteFun_Repr*(*A*,*B*) }
    (**is** *?Y*=*?X*)
⟨*proof*⟩

lemma *FiniteFun_Repr_closed* :
  assumes *M*(*A*) *M*(*B*)
  shows *M*(*FiniteFun_Repr*(*A*,*B*))
  ⟨*proof*⟩

lemma *to_FiniteFun_closed*:
  assumes *M*(*A*) *f*∈*A*
  shows *M*(*range*(*f*))
  ⟨*proof*⟩

lemma *To_FiniteFun_Repr_closed* :
  assumes *M*(*A*) *M*(*B*)
  shows *M*({*range*(*h*) . *h* ∈ *FiniteFun_Repr*(*A*,*B*) })
  ⟨*proof*⟩

lemma *FiniteFun_closed*[*intro*,*simp*] :
  assumes *M*(*A*) *M*(*B*)
  shows *M*(*A* -||> *B*)
  ⟨*proof*⟩

end — *M_FiniteFun*

end

26 Library of basic *ZF* results

theory *ZF_Library_Relative*
  imports
    Aleph_Relative — must be before *Transitive_Models.Cardinal_AC_Relative*!
    Cardinal_AC_Relative
    FiniteFun_Relative
begin

locale *M_Pi_assumption_repl* = *M_Pi_replacement* +
  fixes *A f*
  assumes *A_in_M*: *M*(*A*) **and**
  f_repl : *lam_replacement*(*M*,*f*) **and**
  f_closed : ∀ *x*[*M*]. *M*(*f*(*x*))
begin

sublocale *M_Pi_assumptions M A f*
    ⟨*proof*⟩

end — *M_Pi_assumption_repl*

no_notation *sum* (**infixr** ‹+› *65*)
notation *oadd* (**infixl** ‹+› *65*)

lemma (**in** *M_cardinal_arith_jump*) *csucc_rel_cardinal_rel*:
  assumes *Ord*(κ) *M*(κ)
  shows $(|\kappa|^{M+})^M = (\kappa^+)^M$
⟨*proof*⟩

lemma (**in** *M_cardinal_arith_jump*) *csucc_rel_le_mono*:
  assumes $\kappa \leq \nu$ *M*(κ) *M*(ν)
  shows $(\kappa^+)^M \leq (\nu^+)^M$
⟨*proof*⟩

lemma (**in** *M_cardinal_AC*) *cardinal_rel_succ_not_0*: $|A|^M = succ(n) \Longrightarrow M(A) \Longrightarrow M(n) \Longrightarrow A \neq 0$
  ⟨*proof*⟩

⟨*ML*⟩

notation *Finite_to_one_rel* (‹*Finite'_to'_one–'*(_,_*')*›)

abbreviation
  Finite_to_one_r_set :: [*i*,*i*,*i*] ⇒ *i* (‹*Finite'_to'_one–'*(_,_*')*›) **where**
  $Finite\_to\_one^M(X,Y) \equiv Finite\_to\_one\_rel(\#\#M,X,Y)$

locale *M_ZF_library* = *M_aleph* + *M_FiniteFun*
begin

lemma *Finite_Collect_imp*: *Finite*({x∈X . *Q*(x)}) ⟹ *Finite*({x∈X . *M*(x) ∧ *Q*(x)})
  (**is** *Finite*(*?A*) ⟹ *Finite*(*?B*))
  ⟨*proof*⟩

lemma *Finite_to_one_relI*[*intro*]:
  assumes $f{:}X \to^M Y$ ⋀y. y∈Y ⟹ *Finite*({x∈X . f‘x = y})
    and *types*:*M*(*f*) *M*(*X*) *M*(*Y*)
  shows $f \in Finite\_to\_one^M(X,Y)$
  ⟨*proof*⟩

lemma *Finite_to_one_relI'*[*intro*]:
  assumes $f{:}X \to^M Y$ ⋀y. y∈Y ⟹ *Finite*({x∈X . *M*(x) ∧ f‘x = y})
    and *types*:*M*(*f*) *M*(*X*) *M*(*Y*)
  shows $f \in Finite\_to\_one^M(X,Y)$
  ⟨*proof*⟩

lemma *Finite_to_one_relD*[*dest*]:
  $f \in Finite\_to\_one^M(X,Y) \Longrightarrow f{:}X \to^M Y$
  $f \in Finite\_to\_one^M(X,Y) \Longrightarrow y{\in}Y \Longrightarrow M(Y) \Longrightarrow Finite(\{x{\in}X\ .\ M(x) \wedge f`x = y\})$
  ⟨*proof*⟩

lemma *Diff_bij_rel*:
  assumes $\forall A{\in}F.\ X \subseteq A$
    and *types*: $M(F)\ M(X)$ **shows** $(\lambda A{\in}F.\ A\text{-}X) \in bij^M(F, \{A\text{-}X.\ A{\in}F\})$
⟨*proof*⟩

lemma *function_space_rel_nonempty*:
  assumes $b{\in}B$ **and** *types*: $M(B)\ M(A)$
  shows $(\lambda x{\in}A.\ b) : A \to^M B$
⟨*proof*⟩

lemma *mem_function_space_rel*:
  assumes $f \in A \to^M y\ M(A)\ M(y)$
  shows $f \in A \to y$
  ⟨*proof*⟩

lemmas *range_fun_rel_subset_codomain* = *range_fun_subset_codomain*[*OF mem_function_space_rel*]

end — *M_ZF_library*

context *M_Pi_assumptions*
begin

lemma *mem_Pi_rel*: $f \in Pi^M(A,B) \Longrightarrow f \in Pi(A,\ B)$
  ⟨*proof*⟩

lemmas *Pi_rel_rangeD* = *Pi_rangeD*[*OF mem_Pi_rel*]

lemmas *rel_apply_Pair* = *apply_Pair*[*OF mem_Pi_rel*]

lemmas *rel_apply_rangeI* = *apply_rangeI*[*OF mem_Pi_rel*]

lemmas *Pi_rel_range_eq* = *Pi_range_eq*[*OF mem_Pi_rel*]

lemmas *Pi_rel_vimage_subset* = *Pi_vimage_subset*[*OF mem_Pi_rel*]

end — *M_Pi_assumptions*

context *M_ZF_library*
begin

lemmas *rel_apply_in_range* = *apply_in_codomain_Ord*[*OF _ _ mem_function_space_rel*]

lemmas *rel_range_eq_image* = *ZF_Library.range_eq_image*[*OF mem_function_space_rel*]

lemmas *rel_Image_sub_codomain* = *Image_sub_codomain*[*OF mem_function_space_rel*]

lemma *rel_inj_to_Image*: ⟦*f*:$A \to^M B$; $f \in inj^M(A,B)$; $M(A)$; $M(B)$⟧ ⟹ $f \in inj^M(A, f``A)$
⟨*proof*⟩

lemma *inj_rel_imp_surj_rel*:
fixes *f b*
defines [*simp*]: *if*$x(x) \equiv$ *if* $x \in range(f)$ *then* $converse(f)`x$ *else* b
assumes $f \in inj^M(B,A)$ $b \in B$ **and** *types*: $M(f)$ $M(B)$ $M(A)$
shows $(\lambda x \in A.$ *if*$x(x)) \in surj^M(A,B)$
⟨*proof*⟩

lemma *function_space_rel_disjoint_Un*:
assumes $f \in A \to^M B$ $g \in C \to^M D$ $A \cap C = 0$
and *types*:$M(A)$ $M(B)$ $M(C)$ $M(D)$
shows $f \cup g \in (A \cup C) \to^M (B \cup D)$
⟨*proof*⟩

lemma *restrict_eq_imp_Un_into_function_space_rel*:
assumes $f \in A \to^M B$ $g \in C \to^M D$ $restrict(f, A \cap C) = restrict(g, A \cap C)$
and *types*:$M(A)$ $M(B)$ $M(C)$ $M(D)$
shows $f \cup g \in (A \cup C) \to^M (B \cup D)$
⟨*proof*⟩

lemma *lepoll_relD*[*dest*]: $A \lesssim^M B \Longrightarrow \exists f[M].\ f \in inj^M(A, B)$
⟨*proof*⟩
lemma *lepoll_relI*[*intro*]: $f \in inj^M(A, B) \Longrightarrow M(f) \Longrightarrow A \lesssim^M B$
⟨*proof*⟩

lemma *eqpollD*[*dest*]: $A \approx^M B \Longrightarrow \exists f[M].\ f \in bij^M(A, B)$
⟨*proof*⟩
lemma *bij_rel_imp_eqpoll_rel*[*intro*]: $f \in bij^M(A,B) \Longrightarrow M(f) \Longrightarrow A \approx^M B$
⟨*proof*⟩

lemma *restrict_bij_rel*:— Unused
assumes $f \in inj^M(A,B)$ $C \subseteq A$
and *types*:$M(A)$ $M(B)$ $M(C)$
shows $restrict(f,C) \in bij^M(C, f``C)$
⟨*proof*⟩

lemma *range_of_subset_eqpoll_rel*:
assumes $f \in inj^M(X,Y)$ $S \subseteq X$
and *types*:$M(X)$ $M(Y)$ $M(S)$
shows $S \approx^M f `` S$
⟨*proof*⟩

lemmas *inj_rel_is_fun* = *inj_is_fun*[*OF mem_inj_rel*]

lemma *inj_rel_bij_rel_range*: $f \in inj^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow f \in bij^M(A,range(f))$
⟨*proof*⟩

lemma *bij_rel_is_inj_rel*: $f \in bij^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow f \in inj^M(A,B)$
⟨*proof*⟩

lemma *inj_rel_weaken_type*: $[|\ f \in inj^M(A,B);\ \ B \subseteq D;\ M(A);\ M(B);\ M(D)\ |] ==> f \in inj^M(A,D)$
⟨*proof*⟩

lemma *bij_rel_converse_bij_rel* [*TC*]: $f \in bij^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) ==> converse(f): bij^M(B,A)$
⟨*proof*⟩

lemma *bij_rel_is_fun_rel*: $f \in bij^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow f \in A \to^M B$
⟨*proof*⟩

lemmas *bij_rel_is_fun* = *bij_rel_is_fun_rel*[*THEN mem_function_space_rel*]

lemma *comp_bij_rel*:
$g \in bij^M(A,B) \Longrightarrow f \in bij^M(B,C) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow M(C) \Longrightarrow (f\ O\ g) \in bij^M(A,C)$
⟨*proof*⟩

lemma *inj_rel_converse_fun*: $f \in inj^M(A,B) \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow converse(f) \in range(f) \to^M A$
⟨*proof*⟩

lemma *fg_imp_bijective_rel*:
assumes $f \in A \to^M B\ \ g \in B \to^M A\ \ f\ O\ g = id(B)\ g\ O\ f = id(A)\ M(A)\ M(B)$
shows $f \in bij^M(A,B)$
⟨*proof*⟩

end — *M_ZF_library*

26.1 Discipline for *cexp*

⟨*ML*⟩

context *M_ZF_library*
begin

⟨*ML*⟩
⟨*proof*⟩

⟨*ML*⟩ ⟨*proof*⟩

end — *M_ZF_library*

⟨*ML*⟩
notation *is_cexp_fm* (‹·_ ^↑– *is* _·›)
⟨*ML*⟩

abbreviation
  cexp_r :: [*i*,*i*,*i*⇒*o*] ⇒ *i* (‹_ ^↑–,–›) **where**
  cexp_r(*x*,*y*,*M*) ≡ *cexp_rel*(*M*,*x*,*y*)

abbreviation
  cexp_r_set :: [*i*,*i*,*i*] ⇒ *i* (‹_ ^↑–,–›) **where**
  cexp_r_set(*x*,*y*,*M*) ≡ *cexp_rel*(##*M*,*x*,*y*)

context *M_ZF_library*
begin

lemma *Card_rel_cexp_rel*: *M*(κ) ⟹ *M*(ν) ⟹ *Card*^{M}(κ$^{↑ν,M}$)
  ⟨*proof*⟩
declare *conj_cong*[*cong*]

lemma *eq_csucc_rel_ord*:
  Ord(*i*) ⟹ *M*(*i*) ⟹ (*i*$^+$)^M = (|*i*|$^{M+}$)^M
  ⟨*proof*⟩

lemma *lesspoll_succ_rel*:
  assumes *Ord*(κ) *M*(κ)
  shows κ ≲^{M} (κ$^+$)^M
  ⟨*proof*⟩

lemma *lesspoll_rel_csucc_rel*:
  assumes *Ord*(κ)
    and *types*:*M*(κ) *M*(*d*)
  shows *d* ≺^{M} (κ$^+$)^M ⟷ *d* ≲^{M} κ
⟨*proof*⟩

lemma *Infinite_imp_nats_lepoll*:
  assumes *Infinite*(*X*) *n* ∈ ω
  shows *n* ≲ *X*
  ⟨*proof*⟩

lemma *nepoll_imp_nepoll_rel* :
  assumes ¬ *x* ≈ *X* *M*(*x*) *M*(*X*)
  shows ¬ (*x* ≈^{M} *X*)
  ⟨*proof*⟩

lemma *Infinite_imp_nats_lepoll_rel*:
  assumes *Infinite*(*X*) *n* ∈ ω

and *types*: $M(X)$
shows $n \lesssim^M X$
⟨*proof*⟩

lemma *lepoll_rel_imp_lepoll*: $A \lesssim^M B \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow A \lesssim B$
⟨*proof*⟩

lemma *zero_lesspoll_rel*: **assumes** $0<\kappa$ $M(\kappa)$ **shows** $0 \prec^M \kappa$
⟨*proof*⟩

lemma *lepoll_rel_nat_imp_Infinite*: $\omega \lesssim^M X \Longrightarrow M(X) \Longrightarrow Infinite(X)$
⟨*proof*⟩

lemma *InfCard_rel_imp_Infinite*: $InfCard^M(\kappa) \Longrightarrow M(\kappa) \Longrightarrow Infinite(\kappa)$
⟨*proof*⟩

lemma *lt_surj_rel_empty_imp_Card_rel*:
assumes $Ord(\kappa)$ $\bigwedge\alpha.\ \alpha < \kappa \Longrightarrow surj^M(\alpha,\kappa) = 0$
and *types*:$M(\kappa)$
shows $Card^M(\kappa)$
⟨*proof*⟩

end — *M_ZF_library*

⟨*ML*⟩

notation *mono_map_rel* (‹*mono′_map–′*(_,_,_,_′)›)

abbreviation
mono_map_r_set :: [*i*,*i*,*i*,*i*,*i*]⇒*i* (‹*mono′_map–′*(_,_,_,_′)›) **where**
$mono\_map^M(a,r,b,s) \equiv mono\_map\_rel(\#\#M,a,r,b,s)$

context *M_ZF_library*
begin

lemma *mono_map_rel_char*:
assumes $M(a)$ $M(b)$
shows $mono\_map^M(a,r,b,s) = \{f \in mono\_map(a,r,b,s)\ .\ M(f)\}$
⟨*proof*⟩

Just a sample of porting results on *mono_map*

lemma *mono_map_rel_mono*:
assumes
$f \in mono\_map^M(A,r,B,s)$ $B \subseteq C$
and *types*:$M(A)$ $M(B)$ $M(C)$
shows
$f \in mono\_map^M(A,r,C,s)$
⟨*proof*⟩

lemma *nats_le_InfCard_rel*:
  assumes $n \in \omega$ $InfCard^M(\kappa)$
  shows $n \leq \kappa$
  ⟨*proof*⟩

lemma *nat_into_InfCard_rel*:
  assumes $n \in \omega$ $InfCard^M(\kappa)$
  shows $n \in \kappa$
  ⟨*proof*⟩

lemma *Finite_lesspoll_rel_nat*:
  assumes $Finite(x)$ $M(x)$
  shows $x \prec^M nat$
⟨*proof*⟩

lemma *Finite_cardinal_rel_in_nat* [*simp*]:
  assumes $Finite(A)$ $M(A)$ **shows** $|A|^M \in \omega$
⟨*proof*⟩

lemma *Finite_cardinal_rel_eq_cardinal*:
  assumes $Finite(A)$ $M(A)$ **shows** $|A|^M = |A|$
⟨*proof*⟩

lemma *Finite_imp_cardinal_rel_cons*:
  assumes *FA*: $Finite(A)$ **and** *a*: $a \notin A$ **and** *types*:$M(A)$ $M(a)$
  shows $|cons(a,A)|^M = succ(|A|^M)$
  ⟨*proof*⟩

lemma *Finite_imp_succ_cardinal_rel_Diff*:
  assumes $Finite(A)$ $a \in A$ $M(A)$
  shows $succ(|A\text{-}\{a\}|^M) = |A|^M$
⟨*proof*⟩

lemma *InfCard_rel_Aleph_rel*:
  notes *Aleph_rel_zero*[*simp*]
  assumes $Ord(\alpha)$
    and *types*: $M(\alpha)$
  shows $InfCard^M(\aleph_\alpha{}^M)$
⟨*proof*⟩

lemmas *Limit_Aleph_rel* = *InfCard_rel_Aleph_rel*[*THEN InfCard_rel_is_Limit*]

bundle *Ord_dests* = *Limit_is_Ord*[*dest*] *Card_rel_is_Ord*[*dest*]
bundle *Aleph_rel_dests* = *Aleph_rel_cont*[*dest*]
bundle *Aleph_rel_intros* = *Aleph_rel_increasing*[*intro*!]
bundle *Aleph_rel_mem_dests* = *Aleph_rel_increasing*[*OF ltI*, *THEN ltD*, *dest*]

lemma *f_imp_injective_rel*:
  assumes $f \in A \to^M B$ $\forall x{\in}A.\ d(f\ `\ x) = x$ $M(A)$ $M(B)$

shows $f \in inj^M(A,\ B)$
⟨*proof*⟩

lemma *lam_injective_rel*:
assumes $\bigwedge x.\ x \in A \Longrightarrow c(x) \in B$
$\bigwedge x.\ x \in A \Longrightarrow d(c(x)) = x$
$\forall\, x[M].\ M(c(x))$ *lam_replacement*(M,c)
$M(A)\ M(B)$
shows $(\lambda x{\in}A.\ c(x)) \in inj^M(A,\ B)$
⟨*proof*⟩

lemma *f_imp_surjective_rel*:
assumes $f \in A \to^M B$ $\bigwedge y.\ y \in B \Longrightarrow d(y) \in A$ $\bigwedge y.\ y \in B \Longrightarrow f\ `\ d(y) = y$
$M(A)\ M(B)$
shows $f \in surj^M(A,\ B)$
⟨*proof*⟩

lemma *lam_surjective_rel*:
assumes $\bigwedge x.\ x \in A \Longrightarrow c(x) \in B$
$\bigwedge y.\ y \in B \Longrightarrow d(y) \in A$
$\bigwedge y.\ y \in B \Longrightarrow c(d(y)) = y$
$\forall\, x[M].\ M(c(x))$ *lam_replacement*(M,c)
$M(A)\ M(B)$
shows $(\lambda x{\in}A.\ c(x)) \in surj^M(A,\ B)$
⟨*proof*⟩

lemma *lam_bijective_rel*:
assumes $\bigwedge x.\ x \in A \Longrightarrow c(x) \in B$
$\bigwedge y.\ y \in B \Longrightarrow d(y) \in A$
$\bigwedge x.\ x \in A \Longrightarrow d(c(x)) = x$
$\bigwedge y.\ y \in B \Longrightarrow c(d(y)) = y$
$\forall\, x[M].\ M(c(x))$ *lam_replacement*(M,c)
$M(A)\ M(B)$
shows $(\lambda x{\in}A.\ c(x)) \in bij^M(A,\ B)$
⟨*proof*⟩

lemma *function_space_rel_eqpoll_rel_cong*:
assumes
$A \approx^M A'$ $B \approx^M B'$ $M(A)\ M(A')\ M(B)\ M(B')$
shows
$A \to^M B \approx^M A' \to^M B'$
⟨*proof*⟩

lemma *curry_eqpoll_rel*:
fixes $\nu 1\ \nu 2\ \kappa$
assumes $M(\nu 1)\ M(\nu 2)\ M(\kappa)$
shows $\nu 1 \to^M (\nu 2 \to^M \kappa) \approx^M \nu 1 \times \nu 2 \to^M \kappa$
⟨*proof*⟩

lemma *Pow_rel_eqpoll_rel_function_space_rel*:
fixes *d X*
notes *bool_of_o_def* [*simp*]
defines [*simp*]:$d(A) \equiv (\lambda x{\in}X.\ bool\_of\_o(x{\in}A))$
— the witnessing map for the thesis:
assumes $M(X)$
shows $Pow^M(X) \approx^M X \rightarrow^M 2$
⟨*proof*⟩

lemma *Pow_rel_bottom*: $M(B) \Longrightarrow 0 \in Pow^M(B)$
⟨*proof*⟩

lemma *cantor_surj_rel*:
assumes $M(f)$ $M(A)$
shows $f \notin surj^M(A,Pow^M(A))$
⟨*proof*⟩

lemma *cantor_inj_rel*: $M(f) \Longrightarrow M(A) \Longrightarrow f \notin inj^M(Pow^M(A),A)$
⟨*proof*⟩

end — *M_ZF_library*

end

27 Lambda-replacements required for cardinal inequalities

theory *Replacement_Lepoll*
imports
ZF_Library_Relative
begin

definition
lepoll_assumptions1 :: [*i*⇒*o*,*i*,[*i*,*i*]⇒*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
lepoll_assumptions1(*M*,*A*,*F*,*S*,*fa*,*K*,*x*,*f*,*r*) ≡ ∀ *x*∈*S*. *strong_replacement*(*M*, λ*y z*. *y* ∈ *F*(*A*, *x*) ∧ *z* = {⟨*x*, *y*⟩})

definition
lepoll_assumptions2 :: [*i*⇒*o*,*i*,[*i*,*i*]⇒*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
lepoll_assumptions2(*M*,*A*,*F*,*S*,*fa*,*K*,*x*,*f*,*r*) ≡ *strong_replacement*(*M*, λ*x z*. *z* = *Sigfun*(*x*, *F*(*A*)))

definition
lepoll_assumptions3 :: [*i*⇒*o*,*i*,[*i*,*i*]⇒*i*,*i*,*i*,*i*,*i*,*i*] ⇒ *o* **where**
lepoll_assumptions3(*M*,*A*,*F*,*S*,*fa*,*K*,*x*,*f*,*r*) ≡ *strong_replacement*(*M*, λ*x y*. *y* = *F*(*A*, *x*))

definition

lepoll_assumptions4 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions4$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M,\ \lambda x\ y.\ y = \langle x,\ minimum(r,\ F(A,\ x))\rangle)$

definition
lepoll_assumptions5 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions5$(M,A,F,S,fa,K,x,f,r) \equiv$
strong_replacement$(M,\ \lambda x\ y.\ y = \langle x,\ \mu\ i.\ x \in F(A,\ i),\ f\ `\ (\mu\ i.\ x \in F(A,\ i))\ `\ x\rangle)$

definition
lepoll_assumptions6 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions6$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M,\ \lambda y\ z.\ y \in inj^M(F(A,\ x),S) \wedge z = \{\langle x,\ y\rangle\})$

definition
lepoll_assumptions7 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions7$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M,\ \lambda x\ y.\ y = inj^M(F(A,\ x),S))$

definition
lepoll_assumptions8 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions8$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M,\ \lambda x\ z.\ z = Sigfun(x,\ \lambda i.\ inj^M(F(A,\ i),S)))$

definition
lepoll_assumptions9 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions9$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M,\ \lambda x\ y.\ y = \langle x,\ minimum(r,\ inj^M(F(A,\ x),S))\rangle)$

definition
lepoll_assumptions10 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions10$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*
$(M,\ \lambda x\ z.\ z = Sigfun(x,\ \lambda k.\ if\ k \in range(f)\ then\ F(A,\ converse(f)\ `\ k)\ else\ 0))$

definition
lepoll_assumptions11 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions11$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M,\ \lambda x\ y.\ y = (if\ x \in range(f)\ then\ F(A,\ converse(f)\ `\ x)\ else\ 0))$

definition
lepoll_assumptions12 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions12$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*$(M,\ \lambda y\ z.\ y \in F(A,\ converse(f)\ `\ x) \wedge z = \{\langle x,\ y\rangle\})$

definition
lepoll_assumptions13 :: $[i \Rightarrow o,i,[i,i] \Rightarrow i,i,i,i,i,i,i,i] \Rightarrow o$ **where**
lepoll_assumptions13$(M,A,F,S,fa,K,x,f,r) \equiv$ *strong_replacement*
$(M,\ \lambda x\ y.\ y = \langle x,\ minimum(r,\ if\ x \in range(f)\ then\ F(A,converse(f)\ `\ x)$

else 0)⟩)

definition
lepoll_assumptions14 :: [*i⇒o,i,*[*i,i*]*⇒i,i,i,i,i,i,i*] ⇒ *o* **where**
lepoll_assumptions14(*M,A,F,S,fa,K,x,f,r*) ≡ *strong_replacement*
(*M*, λ*x y*. *y* = ⟨*x*, μ *i*. *x* ∈ (*if i* ∈ *range*(*f*) *then F*(*A*, *converse*(*f*) ‘ *i*) *else*
0),
fa ‘ (μ *i*. *x* ∈ (*if i* ∈ *range*(*f*) *then F*(*A*, *converse*(*f*) ‘ *i*) *else*
0)) ‘ *x*⟩)

definition
lepoll_assumptions15 :: [*i⇒o,i,*[*i,i*]*⇒i,i,i,i,i,i,i*] ⇒ *o* **where**
lepoll_assumptions15(*M,A,F,S,fa,K,x,f,r*) ≡ *strong_replacement*
(*M*, λ*y z*. *y* ∈ *inj*^{M}(*if x* ∈ *range*(*f*) *then F*(*A*, *converse*(*f*) ‘ *x*) *else 0,K*) ∧
z = {⟨*x*, *y*⟩})

definition
lepoll_assumptions16 :: [*i⇒o,i,*[*i,i*]*⇒i,i,i,i,i,i,i*] ⇒ *o* **where**
lepoll_assumptions16(*M,A,F,S,fa,K,x,f,r*) ≡ *strong_replacement*(*M*, λ*x y*. *y* =
inj^{M}(*if x* ∈ *range*(*f*) *then F*(*A*, *converse*(*f*) ‘ *x*) *else 0,K*))

definition
lepoll_assumptions17 :: [*i⇒o,i,*[*i,i*]*⇒i,i,i,i,i,i,i*] ⇒ *o* **where**
lepoll_assumptions17(*M,A,F,S,fa,K,x,f,r*) ≡ *strong_replacement*
(*M*, λ*x z*. *z* = *Sigfun*(*x*, λ*i*. *inj*^{M}(*if i* ∈ *range*(*f*) *then F*(*A*, *converse*(*f*)
‘ *i*) *else 0,K*)))

definition
lepoll_assumptions18 :: [*i⇒o,i,*[*i,i*]*⇒i,i,i,i,i,i,i*] ⇒ *o* **where**
lepoll_assumptions18(*M,A,F,S,fa,K,x,f,r*) ≡ *strong_replacement*
(*M*, λ*x y*. *y* = ⟨*x*, *minimum*(*r*, *inj*^{M}(*if x* ∈ *range*(*f*) *then F*(*A*, *converse*(*f*)
‘ *x*) *else 0,K*))⟩)

lemmas *lepoll_assumptions_defs*[*simp*] = *lepoll_assumptions1_def*
lepoll_assumptions2_def lepoll_assumptions3_def lepoll_assumptions4_def
lepoll_assumptions5_def lepoll_assumptions6_def lepoll_assumptions7_def
lepoll_assumptions8_def lepoll_assumptions9_def lepoll_assumptions10_def
lepoll_assumptions11_def lepoll_assumptions12_def lepoll_assumptions13_def
lepoll_assumptions14_def lepoll_assumptions15_def lepoll_assumptions16_def
lepoll_assumptions17_def lepoll_assumptions18_def

definition *if_range_F* **where**
[*simp*]: *if_range_F*(*H,f,i*) ≡ *if i* ∈ *range*(*f*) *then H*(*converse*(*f*) ‘ *i*) *else 0*

definition *if_range_F_else_F* **where**
if_range_F_else_F(*H,b,f,i*) ≡ *if b=0 then if_range_F*(*H,f,i*) *else H*(*i*)

lemma (**in** *M_basic*) *lam_Least_assumption_general*:
assumes

separations:
$\forall A'[M].\ separation(M, \lambda y.\ \exists x{\in}A'.\ y = \langle x, \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i)\rangle)$
and
mem_F_bound:$\bigwedge x\ c.\ x{\in}F(A,c) \Longrightarrow c \in range(f) \cup U(A)$
and
types:$M(A)\ M(b)\ M(f)\ M(U(A))$
shows *lam_replacement*$(M,\lambda x\ .\ \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i))$
⟨*proof*⟩

lemma (**in** *M_basic*) *lam_Least_assumption_ifM_b0*:
fixes F
defines $F \equiv \lambda\_\ x.\ if\ M(x)\ then\ x\ else\ 0$
assumes
separations:
$\forall A'[M].\ separation(M, \lambda y.\ \exists x{\in}A'.\ y = \langle x, \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),0,f,i)\rangle)$
and
types:$M(A)\ M(f)$
shows *lam_replacement*$(M,\lambda x\ .\ \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),0,f,i))$
(**is** *lam_replacement*$(M,\lambda x\ .\ Least(?P(x)))$)
⟨*proof*⟩

lemma (**in** *M_replacement*) *lam_Least_assumption_ifM_bnot0*:
fixes F
defines $F \equiv \lambda\_\ x.\ if\ M(x)\ then\ x\ else\ 0$
assumes
lam_replacement_minimum:*lam_replacement*$(M, \lambda p.\ minimum(fst(p),snd(p)))$
and
separations:
$\forall A'[M].\ separation(M, \lambda y.\ \exists x{\in}A'.\ y = \langle x, \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i)\rangle)$
separation(M,Ord)
and
types:$M(A)\ M(f)$
and
$b{\neq}0$
shows *lam_replacement*$(M,\lambda x\ .\ \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i))$
(**is** *lam_replacement*$(M,\lambda x\ .\ Least(?P(x)))$)
⟨*proof*⟩

lemma (**in** *M_replacement*) *lam_Least_assumption_drSR_Y*:
fixes $F\ r'\ D$
defines $F \equiv drSR\_Y(r',D)$
assumes $\forall A'[M].\ separation(M, \lambda y.\ \exists x{\in}A'.\ y = \langle x, \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i)\rangle)$
$M(A)\ M(b)\ M(f)\ M(r')$
and
lam_replacement_minimum:*lam_replacement*$(M, \lambda p.\ minimum(fst(p),snd(p)))$
shows *lam_replacement*$(M,\lambda x\ .\ \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i))$
⟨*proof*⟩

locale *M_replacement_lepoll* = *M_replacement* + *M_inj* +

fixes F
assumes
$F\_type[simp]$: $M(A) \Longrightarrow \forall\, x[M].\ M(F(A,x))$
and
$lam\_lepoll\_assumption\_F$:$M(A) \Longrightarrow lam\_replacement(M,F(A))$
and
— Here b is a Boolean.
$lam\_Least\_assumption$:$M(A) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow$
$lam\_replacement(M,\lambda x\ .\ \mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i))$
and
F_args_closed: $M(A) \Longrightarrow M(x) \Longrightarrow x \in F(A,i) \Longrightarrow M(i)$
and
$lam\_replacement\_inj\_rel$:$lam\_replacement(M,\ \lambda p.\ inj^M(fst(p),snd(p)))$
and
$lam\_replacement\_minimum$:$lam\_replacement(M,\ \lambda p.\ minimum(fst(p),snd(p)))$
begin

declare $if\_range\_F\_else\_F\_def[simp]$

lemma $lepoll\_assumptions1$:
assumes $types[simp]$:$M(A)\ M(S)$
shows $lepoll\_assumptions1(M,A,F,S,fa,K,x,f,r)$
⟨*proof*⟩

lemma $lepoll\_assumptions2$:
assumes $types[simp]$:$M(A)\ M(S)$
shows $lepoll\_assumptions2(M,A,F,S,fa,K,x,f,r)$
⟨*proof*⟩

lemma $lepoll\_assumptions3$:
assumes $types[simp]$:$M(A)$
shows $lepoll\_assumptions3(M,A,F,S,fa,K,x,f,r)$
⟨*proof*⟩

lemma $lepoll\_assumptions4$:
assumes $types[simp]$:$M(A)\ M(r)$
shows $lepoll\_assumptions4(M,A,F,S,fa,K,x,f,r)$
⟨*proof*⟩

lemma lam_Least_closed :
assumes $M(A)\ M(b)\ M(f)$
shows $\forall\, x[M].\ M(\mu\ i.\ x \in if\_range\_F\_else\_F(F(A),b,f,i))$
⟨*proof*⟩

lemma $lepoll\_assumptions5$:
assumes
$types[simp]$:$M(A)\ M(f)$
shows $lepoll\_assumptions5(M,A,F,S,fa,K,x,f,r)$
⟨*proof*⟩

lemma *lepoll_assumptions6*:
assumes *types*[*simp*]:$M(A)$ $M(S)$ $M(x)$
shows *lepoll_assumptions6*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemma *lepoll_assumptions7*:
assumes *types*[*simp*]:$M(A)$ $M(S)$ $M(x)$
shows *lepoll_assumptions7*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemma *lepoll_assumptions8*:
assumes *types*[*simp*]:$M(A)$ $M(S)$
shows *lepoll_assumptions8*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemma *lepoll_assumptions9*:
assumes *types*[*simp*]:$M(A)$ $M(S)$ $M(r)$
shows *lepoll_assumptions9*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemma *lepoll_assumptions10*:
assumes *types*[*simp*]:$M(A)$ $M(f)$
shows *lepoll_assumptions10*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemma *lepoll_assumptions11*:
assumes *types*[*simp*]:$M(A)$ $M(f)$
shows *lepoll_assumptions11*(M, A, F, S, fa, K, x, f, r)
⟨*proof*⟩

lemma *lepoll_assumptions12*:
assumes *types*[*simp*]:$M(A)$ $M(x)$ $M(f)$
shows *lepoll_assumptions12*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemma *lepoll_assumptions13*:
assumes *types*[*simp*]:$M(A)$ $M(r)$ $M(f)$
shows *lepoll_assumptions13*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemma *lepoll_assumptions14*:
assumes *types*[*simp*]:$M(A)$ $M(f)$ $M(fa)$
shows *lepoll_assumptions14*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemma *lepoll_assumptions15*:
assumes *types*[*simp*]:$M(A)$ $M(x)$ $M(f)$ $M(K)$
shows *lepoll_assumptions15*(M,A,F,S,fa,K,x,f,r)

⟨*proof*⟩

lemma *lepoll_assumptions16*:
assumes *types*[*simp*]:$M(A)$ $M(f)$ $M(K)$
shows *lepoll_assumptions16*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemma *lepoll_assumptions17*:
assumes *types*[*simp*]:$M(A)$ $M(f)$ $M(K)$
shows *lepoll_assumptions17*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemma *lepoll_assumptions18*:
assumes *types*[*simp*]:$M(A)$ $M(K)$ $M(f)$ $M(r)$
shows *lepoll_assumptions18*(M,A,F,S,fa,K,x,f,r)
⟨*proof*⟩

lemmas *lepoll_assumptions* = *lepoll_assumptions1 lepoll_assumptions2*
lepoll_assumptions3 lepoll_assumptions4 lepoll_assumptions5
lepoll_assumptions6 lepoll_assumptions7 lepoll_assumptions8
lepoll_assumptions9 lepoll_assumptions10 lepoll_assumptions11
lepoll_assumptions12 lepoll_assumptions13 lepoll_assumptions14
lepoll_assumptions15 lepoll_assumptions16
lepoll_assumptions17 lepoll_assumptions18

end — *M_replacement_lepoll*

end

28 Cardinal Arithmetic under Choice

theory *Cardinal_Library_Relative*
imports
Replacement_Lepoll
begin

locale *M_library* = *M_ZF_library* + *M_cardinal_AC* +
assumes
separation_cardinal_rel_lesspoll_rel: $M(\kappa) \Longrightarrow$ *separation*$(M,\ \lambda x\ .\ x \prec^M \kappa)$
begin

declare *eqpoll_rel_refl* [*simp*]

28.1 Miscellaneous

lemma *cardinal_rel_RepFun_apply_le*:
assumes $S \in A \to B$ $M(S)$ $M(A)$ $M(B)$
shows $|\{S`a\ .\ a \in A\}|^M \le |A|^M$
⟨*proof*⟩

lemma *cardinal_rel_RepFun_le*:
  assumes *lrf*:*lam_replacement*(*M*,*f*) **and** *f_closed*:$\forall\, x[M].\ M(f(x))$ **and** $M(X)$
  shows $|\{f(x)\ .\ x \in X\}|^M \leq |X|^M$
  ⟨*proof*⟩

lemma *subset_imp_le_cardinal_rel*: $A \subseteq B \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow |A|^M \leq |B|^M$
  ⟨*proof*⟩

lemma *lt_cardinal_rel_imp_not_subset*: $|A|^M < |B|^M \Longrightarrow M(A) \Longrightarrow M(B) \Longrightarrow \neg\ B \subseteq A$
  ⟨*proof*⟩

lemma *cardinal_rel_lt_csucc_rel_iff*:
  Card_rel(*M*,*K*) $\Longrightarrow$ $M(K) \Longrightarrow M(K') \Longrightarrow |K'|^M < (K^+)^M \longleftrightarrow |K'|^M \leq K$
  ⟨*proof*⟩

end — *M_library*

locale *M_cardinal_UN_nat* = *M_cardinal_UN* _ ω *X* **for** *X*
begin
lemma *cardinal_rel_UN_le_nat*:
  assumes $\bigwedge i.\ i{\in}\omega \Longrightarrow |X(i)|^M \leq \omega$
  shows $|\bigcup i{\in}\omega.\ X(i)|^M \leq \omega$
⟨*proof*⟩

end — *M_cardinal_UN_nat*

locale *M_cardinal_UN_inj* = *M_library* +
  j:*M_cardinal_UN* _ *J* +
  y:*M_cardinal_UN* _ *K* $\lambda k.$ *if* $k{\in}$*range*(*f*) *then* *X*(*converse*(*f*)‘*k*) *else* *0* **for** *J K* *f* +
assumes
  f_inj: $f \in$ *inj_rel*(*M*,*J*,*K*)
begin

lemma *inj_rel_imp_cardinal_rel_UN_le*:
  notes [*dest*] = *InfCard_is_Card Card_is_Ord*
  fixes *Y*
  defines $Y(k) \equiv$ *if* $k{\in}$*range*(*f*) *then* *X*(*converse*(*f*)‘*k*) *else* *0*
  assumes *InfCard*^{M}(*K*) $\bigwedge i.\ i{\in}J \Longrightarrow |X(i)|^M \leq K$
  shows $|\bigcup i{\in}J.\ X(i)|^M \leq K$
⟨*proof*⟩

end — *M_cardinal_UN_inj*

locale *M_cardinal_UN_lepoll* = *M_library* + *M_replacement_lepoll* _ λ_. *X* +

j:*M_cardinal_UN _ J* **for** *J*
begin

lemma *lepoll_rel_imp_cardinal_rel_UN_le*:
notes [*dest*] = *InfCard_is_Card Card_is_Ord*
assumes $InfCard^M(K)$ $J \lesssim^M K$ $\bigwedge i.\ i{\in}J \Longrightarrow |X(i)|^M \leq K$
$M(K)$
shows $|\bigcup i{\in}J.\ X(i)|^M \leq K$
⟨*proof*⟩

end — *M_cardinal_UN_lepoll*

context *M_library*
begin

lemma *cardinal_rel_lt_csucc_rel_iff'*:
includes *Ord_dests*
assumes *Card_rel*(M,κ)
and *types*:$M(\kappa)$ $M(X)$
shows $\kappa < |X|^M \longleftrightarrow (\kappa^+)^M \leq |X|^M$
⟨*proof*⟩

lemma *lepoll_rel_imp_subset_bij_rel*:
assumes $M(X)$ $M(Y)$
shows $X \lesssim^M Y \longleftrightarrow (\exists\, Z[M].\ Z \subseteq Y \wedge Z \approx^M X)$
⟨*proof*⟩

The following result proves to be very useful when combining *cardinal_rel* and *eqpoll_rel* in a calculation.

lemma *cardinal_rel_Card_rel_eqpoll_rel_iff*:
Card_rel(M,κ) $\Longrightarrow$ $M(\kappa)$ $\Longrightarrow$ $M(X)$ $\Longrightarrow$ $|X|^M = \kappa \longleftrightarrow X \approx^M \kappa$
⟨*proof*⟩

lemma *lepoll_rel_imp_lepoll_rel_cardinal_rel*:
assumes$X \lesssim^M Y$ $M(X)$ $M(Y)$
shows $X \lesssim^M |Y|^M$
⟨*proof*⟩

lemma *lepoll_rel_Un*:
assumes *InfCard_rel*(M,κ) $A \lesssim^M \kappa$ $B \lesssim^M \kappa$ $M(A)$ $M(B)$ $M(\kappa)$
shows $A \cup B \lesssim^M \kappa$
⟨*proof*⟩

lemma *cardinal_rel_Un_le*:
assumes *InfCard_rel*(M,κ) $|A|^M \leq \kappa$ $|B|^M \leq \kappa$ $M(\kappa)$ $M(A)$ $M(B)$
shows $|A \cup B|^M \leq \kappa$
⟨*proof*⟩

lemma *Finite_cardinal_rel_iff'*: $M(i) \Longrightarrow Finite(|i|^M) \longleftrightarrow Finite(i)$

⟨*proof*⟩

lemma *cardinal_rel_subset_of_Card_rel*:
assumes *Card_rel*(*M*,γ) $a \subseteq \gamma$ *M*(a) *M*(γ)
shows $|a|^M < \gamma \vee |a|^M = \gamma$
⟨*proof*⟩

lemma *cardinal_rel_cases*:
includes *Ord_dests*
assumes *M*(γ) *M*(*X*)
shows *Card_rel*(*M*,γ) $\Longrightarrow |X|^M < \gamma \longleftrightarrow \neg\ |X|^M \geq \gamma$
⟨*proof*⟩

end — *M_library*

28.2 Countable and uncountable sets

⟨*ML*⟩

notation *countable_rel* (‹*countable*–'(_')›)

abbreviation
countable_r_set :: [*i*,*i*]⇒*o* (‹*countable*–'(_')›) **where**
$countable^M(i) \equiv$ *countable_rel*(##*M*,*i*)

context *M_library*
begin

lemma *countableI*[*intro*]: $X \lesssim^M \omega \Longrightarrow$ *countable_rel*(*M*,*X*)
⟨*proof*⟩

lemma *countableD*[*dest*]: *countable_rel*(*M*,*X*) $\Longrightarrow X \lesssim^M \omega$
⟨*proof*⟩

lemma *countable_rel_iff_cardinal_rel_le_nat*: *M*(*X*) $\Longrightarrow$ *countable_rel*(*M*,*X*)
$\longleftrightarrow |X|^M \leq \omega$
⟨*proof*⟩

lemma *lepoll_rel_countable_rel*: $X \lesssim^M Y \Longrightarrow$ *countable_rel*(*M*,*Y*) $\Longrightarrow$ *M*(*X*)
$\Longrightarrow$ *M*(*Y*) $\Longrightarrow$ *countable_rel*(*M*,*X*)
⟨*proof*⟩
lemma *surj_rel_countable_rel*:
countable_rel(*M*,*X*) $\Longrightarrow f \in$ *surj_rel*(*M*,*X*,*Y*) $\Longrightarrow$ *M*(*X*) $\Longrightarrow$ *M*(*Y*) $\Longrightarrow$ *M*(*f*)
$\Longrightarrow$ *countable_rel*(*M*,*Y*)
⟨*proof*⟩

lemma *Finite_imp_countable_rel*: *Finite_rel*(*M*,*X*) $\Longrightarrow$ *M*(*X*) $\Longrightarrow$ *countable_rel*(*M*,*X*)
⟨*proof*⟩

end — *M_library*

lemma (**in** *M_cardinal_UN_lepoll*) *countable_rel_imp_countable_rel_UN*:
assumes *countable_rel*(M,J) $\bigwedge i.\ i\in J \Longrightarrow$ *countable_rel*(M,$X(i)$)
shows *countable_rel*(M,$\bigcup i\in J.\ X(i)$)
⟨*proof*⟩

locale *M_cardinal_library* = *M_library* + *M_replacement* +
assumes
lam_replacement_inj_rel:*lam_replacement*(M, $\lambda x.\ inj^M(fst(x),snd(x))$)
and
cardinal_lib_assms1:
$M(A) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow$
separation(M, $\lambda y.\ \exists x\in A.\ y = \langle x,\ \mu\ i.\ x \in$ *if_range_F_else_F*($\lambda x.$ *if* $M(x)$ *then* x *else* 0,b,f,i)⟩)
and
cardinal_lib_assms2:
$M(A') \Longrightarrow M(G) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow$
separation(M, $\lambda y.\ \exists x\in A'.\ y = \langle x,\ \mu\ i.\ x \in$ *if_range_F_else_F*($\lambda a.$ *if* $M(a)$ *then* $G`a$ *else* 0,b,f,i)⟩)
and
cardinal_lib_assms3:
$M(A') \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow M(F) \Longrightarrow$
separation(M, $\lambda y.\ \exists x\in A'.\ y = \langle x,\ \mu\ i.\ x \in$ *if_range_F_else_F*($\lambda a.$ *if* $M(a)$ *then* $F\text{-}``\{a\}$ *else* 0,b,f,i)⟩)
and
cardinal_rel_separation :
separation(M, $\lambda\langle x,y\rangle.$ *cardinal_rel*(M,x) = y)
and
separation_cardinal_rel_lt :
$M(\gamma) \Longrightarrow$ *separation*(M, $\lambda Z\ .$ *cardinal_rel*(M,Z) $< \gamma$)

begin

lemma *cdlt_assms*: $M(x) \Longrightarrow M(Q) \Longrightarrow$ *separation*(M, $\lambda a\ .\ \forall s\in x.\ \langle s,\ a\rangle \in Q$)
⟨*proof*⟩

lemma *countable_rel_union_countable_rel*:
assumes $\bigwedge x.\ x \in C \Longrightarrow$ *countable_rel*(M,x) *countable_rel*(M,C) $M(C)$
shows *countable_rel*(M,$\bigcup C$)
⟨*proof*⟩

end — *M_cardinal_library*

abbreviation
uncountable_rel :: $[i\Rightarrow o,i]\Rightarrow o$ **where**
uncountable_rel(M,X) $\equiv \neg$ *countable_rel*(M,X)

context *M_library*

begin

lemma *uncountable_rel_iff_nat_lt_cardinal_rel*:
$M(X) \Longrightarrow$ *uncountable_rel*$(M,X) \longleftrightarrow \omega < |X|^M$
⟨*proof*⟩

lemma *uncountable_rel_not_empty*: *uncountable_rel*$(M,X) \Longrightarrow X \neq 0$
⟨*proof*⟩

lemma *uncountable_rel_imp_Infinite*: *uncountable_rel*$(M,X) \Longrightarrow M(X) \Longrightarrow$ *Infinite*(X)
⟨*proof*⟩

lemma *uncountable_rel_not_subset_countable_rel*:
assumes *countable_rel*(M,X) *uncountable_rel*(M,Y) $M(X)$ $M(Y)$
shows $\neg\ (Y \subseteq X)$
⟨*proof*⟩

28.3 Results on Aleph_rels

lemma *nat_lt_Aleph_rel1*: $\omega < \aleph_1{}^M$
⟨*proof*⟩

lemma *zero_lt_Aleph_rel1*: $0 < \aleph_1{}^M$
⟨*proof*⟩

lemma *le_Aleph_rel1_nat*: $M(k) \Longrightarrow$ *Card_rel*$(M,k) \Longrightarrow k<\aleph_1{}^M \Longrightarrow k \leq \omega$
⟨*proof*⟩

lemma *lesspoll_rel_Aleph_rel_succ*:
assumes *Ord*(α)
and *types*:$M(\alpha)$ $M(d)$
shows $d \prec^M \aleph_{succ(\alpha)}{}^M \longleftrightarrow d \lesssim^M \aleph_\alpha{}^M$
⟨*proof*⟩

lemma *cardinal_rel_Aleph_rel* [*simp*]: *Ord*$(\alpha) \Longrightarrow M(\alpha) \Longrightarrow |\aleph_\alpha{}^M|^M = \aleph_\alpha{}^M$
⟨*proof*⟩
lemma *Aleph_rel_lesspoll_rel_increasing*:
includes *Aleph_rel_intros*
assumes $M(b)$ $M(a)$
shows $a < b \Longrightarrow \aleph_a{}^M \prec^M \aleph_b{}^M$
⟨*proof*⟩

lemma *uncountable_rel_iff_subset_eqpoll_rel_Aleph_rel1*:
includes *Ord_dests*
assumes $M(X)$
notes *Aleph_rel_zero*[*simp*] *Card_rel_nat*[*simp*] *Aleph_rel_succ*[*simp*]
shows *uncountable_rel*$(M,X) \longleftrightarrow (\exists\, S[M].\ S \subseteq X \wedge S \approx^M \aleph_1{}^M)$
⟨*proof*⟩

lemma *UN_if_zero*: $M(K) \Longrightarrow (\bigcup x{\in}K.\ \mathit{if}\ M(x)\ \mathit{then}\ G\ `\ x\ \mathit{else}\ 0) = (\bigcup x{\in}K.\ G\ `\ x)$
⟨*proof*⟩

lemma *mem_F_bound1*:
fixes $F\ G$
defines $F \equiv \lambda\_\ x.\ \mathit{if}\ M(x)\ \mathit{then}\ G`x\ \mathit{else}\ 0$
shows $x{\in}F(A,c) \Longrightarrow c \in (\mathit{range}(f) \cup \mathit{domain}(G))$
⟨*proof*⟩

end — *M_library*

context *M_cardinal_library*
begin

lemma *lt_Aleph_rel_imp_cardinal_rel_UN_le_nat*: $\mathit{function}(G) \Longrightarrow \mathit{domain}(G) \lesssim^M \omega \Longrightarrow$
$\forall\, n{\in}\mathit{domain}(G).\ |G`n|^M{<}{\aleph_1}^M \Longrightarrow M(G) \Longrightarrow |\bigcup n{\in}\mathit{domain}(G).\ G`n|^M{\le}\omega$
⟨*proof*⟩

lemma *Aleph_rel1_eq_cardinal_rel_vimage*: $f{:}{\aleph_1}^M{\to^M}\omega \Longrightarrow \exists\, n{\in}\omega.\ |f\text{-}``\{n\}|^M = {\aleph_1}^M$
⟨*proof*⟩

lemma *eqpoll_rel_Aleph_rel1_cardinal_rel_vimage*:
assumes $Z \approx^M ({\aleph_1}^M)\ f \in Z \to^M \omega\ M(Z)$
shows $\exists\, n{\in}\omega.\ |f\text{-}``\{n\}|^M = {\aleph_1}^M$
⟨*proof*⟩

end — *M_cardinal_library*

28.4 Applications of transfinite recursive constructions

locale *M_cardinal_library_extra* = *M_cardinal_library* +
assumes
replacement_trans_apply_image:
$M(f) \Longrightarrow M(\beta) \Longrightarrow \mathit{Ord}(\beta) \Longrightarrow$
$\mathit{strong\_replacement}(M,\ \lambda x\ y.\ x{\in}\beta \wedge y = \langle x,\ \mathit{transrec}(x,\ \lambda a\ g.\ f\ `\ (g\ ``\ a))\rangle)$
begin

lemma *rec_constr_type_rel*:
assumes $f{:}\mathit{Pow\_rel}(M,G){\to^M}\ G\ \mathit{Ord}(\alpha)\ M(G)$
shows $M(\alpha) \Longrightarrow \mathit{rec\_constr}(f,\alpha) \in G$
⟨*proof*⟩

lemma *rec_constr_closed* :
assumes $f{:}\mathit{Pow\_rel}(M,G){\to^M}\ G\ \mathit{Ord}(\alpha)\ M(G)\ M(\alpha)$
shows $M(\mathit{rec\_constr}(f,\alpha))$

⟨*proof*⟩

lemma *lambda_rec_constr_closed* :
assumes $Ord(\gamma)$ $M(\gamma)$ $M(f)$ f:$Pow\_rel(M,G)\to^M G$ $M(G)$
shows $M(\lambda\alpha\in\gamma\ .\ rec\_constr(f,\alpha))$
⟨*proof*⟩

The next lemma is an application of recursive constructions. It works under the assumption that whenever the already constructed subsequence is small enough, another element can be added.

lemma *bounded_cardinal_rel_selection*:
includes *Ord_dests*
assumes
$\bigwedge Z.\ |Z|^M < \gamma \Longrightarrow Z \subseteq G \Longrightarrow M(Z) \Longrightarrow \exists\, a\in G.\ \forall\, s\in Z.\ <s,a>\in Q\ \ b\in G$
$Card\_rel(M,\gamma)$
$M(G)$ $M(Q)$ $M(\gamma)$
shows
$\exists\, S[M].\ S : \gamma \to^M G \wedge (\forall\, \alpha \in \gamma.\ \forall\, \beta \in \gamma.\ \ \alpha<\beta \longrightarrow <S`\alpha,S`\beta>\in Q)$
⟨*proof*⟩

The following basic result can, in turn, be proved by a bounded-cardinal_rel selection.

lemma *Infinite_iff_lepoll_rel_nat*: $M(Z) \Longrightarrow Infinite(Z) \longleftrightarrow \omega \lesssim^M Z$
⟨*proof*⟩

lemma *Infinite_InfCard_rel_cardinal_rel*: $Infinite(Z) \Longrightarrow M(Z) \Longrightarrow InfCard\_rel(M,|Z|^M)$
⟨*proof*⟩

lemma (**in** *M_trans*) *mem_F_bound2*:
fixes F A
defines $F \equiv \lambda\_\ x.\ if\ M(x)\ then\ A\text{-``}\{x\}\ else\ 0$
shows $x\in F(A,c) \Longrightarrow c \in (range(f) \cup range(A))$
⟨*proof*⟩

lemma *Finite_to_one_rel_surj_rel_imp_cardinal_rel_eq*:
assumes $F \in Finite\_to\_one\_rel(M,Z,Y) \cap surj\_rel(M,Z,Y)$ $Infinite(Z)$ $M(Z)$
$M(Y)$
shows $|Y|^M = |Z|^M$
⟨*proof*⟩

lemma *cardinal_rel_map_Un*:
assumes $Infinite(X)$ $Finite(b)$ $M(X)$ $M(b)$
shows $|\{a \cup b\ .\ a \in X\}|^M = |X|^M$
⟨*proof*⟩

end — *M_cardinal_library_extra*

context *M_library*
begin

28.5 Results on relative cardinal exponentiation

lemma *cexp_rel_eqpoll_rel_cong*:
assumes
$A \approx^M A'$ $B \approx^M B'$ $M(A)$ $M(A')$ $M(B)$ $M(B')$
shows
$A^{\uparrow B,M} = A'^{\uparrow B',M}$
⟨*proof*⟩

lemma *cexp_rel_cexp_rel_cmult*:
assumes $M(\kappa)$ $M(\nu 1)$ $M(\nu 2)$
shows $(\kappa^{\uparrow \nu 1,M})^{\uparrow \nu 2,M} = \kappa^{\uparrow \nu 2 \otimes^M \nu 1,M}$
⟨*proof*⟩

lemma *cardinal_rel_Pow_rel*: $M(X) \Longrightarrow |Pow\_rel(M,X)|^M = 2^{\uparrow X,M}$ — Perhaps it's better with |X|
⟨*proof*⟩

lemma *cantor_cexp_rel*:
assumes *Card_rel*(M,ν) $M(\nu)$
shows $\nu < 2^{\uparrow \nu,M}$
⟨*proof*⟩

lemma *countable_iff_lesspoll_rel_Aleph_rel_one*:
notes *iff_trans*[*trans*]
assumes $M(C)$
shows *countable*$^M(C) \longleftrightarrow C \prec^M \aleph_1{}^M$
⟨*proof*⟩

lemma *countable_iff_le_rel_Aleph_rel_one*:
notes *iff_trans*[*trans*]
assumes $M(C)$
shows *countable*$^M(C) \longleftrightarrow |C|^M \prec^M \aleph_1{}^M$
⟨*proof*⟩

end — *M_library*

lemma (**in** *M_cardinal_library*) *countable_fun_imp_countable_image*:
assumes $f{:}C \to^M B$ *countable*$^M(C)$ $\bigwedge c.\ c{\in}C \Longrightarrow$ *countable*$^M(f`c)$
$M(C)$ $M(B)$
shows *countable*$^M(\bigcup (f``C))$
⟨*proof*⟩

end

29 The Delta System Lemma, Relativized

theory *Delta_System_Relative*
imports
Cardinal_Library_Relative
begin

⟨*ML*⟩

locale *M_delta* = *M_cardinal_library_extra* +
assumes
countable_lepoll_assms:
$M(G) \Longrightarrow M(A) \Longrightarrow M(b) \Longrightarrow M(f) \Longrightarrow$ *separation*$(M, \lambda y.\ \exists\, x{\in}A.$
$y = \langle x, \mu\ i.\ x \in$ *if_range_F_else_F*$(\lambda x.\ \{xa \in G\ .\ x \in xa\}, b, f, i)\rangle)$
begin

lemma *disjoint_separation*: $M(c) \Longrightarrow$ *separation*$(M, \lambda\ x.\ \exists\, a.\ \exists\, b.\ x{=}\langle a,b\rangle \wedge a \cap b = c)$
⟨*proof*⟩

lemma (**in** *M_trans*) *mem_F_bound6*:
fixes *F G*
defines $F \equiv \lambda\_\ x.$ *Collect*$(G, (\in)(x))$
shows $x{\in}F(G,c) \Longrightarrow c \in (\mathit{range}(f) \cup \bigcup G)$
⟨*proof*⟩

lemma *delta_system_Aleph_rel1*:
assumes $\forall\, A{\in}F.$ *Finite*(A) $F \approx^M \aleph_1{}^M$ $M(F)$
shows $\exists\, D[M].\ D \subseteq F \wedge$ *delta_system*$(D) \wedge D \approx^M \aleph_1{}^M$
⟨*proof*⟩

lemma *delta_system_uncountable_rel*:
assumes $\forall\, A{\in}F.$ *Finite*(A) *uncountable_rel*(M,F) $M(F)$
shows $\exists\, D[M].\ D \subseteq F \wedge$ *delta_system*$(D) \wedge D \approx^M \aleph_1{}^M$
⟨*proof*⟩

end — *M_delta*

end

30 Relative DC

theory *Pointed_DC_Relative*
imports
Cardinal_Library_Relative

begin

consts *dc_witness* :: $i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow i$
primrec
wit0 : $dc\_witness(0,A,a,s,R) = a$
witrec : $dc\_witness(succ(n),A,a,s,R) = s`\{x \in A.\ \langle dc\_witness(n,A,a,s,R),x\rangle \in R\}$

lemmas $dc\_witness\_def = dc\_witness\_nat\_def$

⟨*ML*⟩

schematic_goal *sats_is_dc_witness_fm_auto*:
assumes $na < length(env)$ $e < length(env)$
shows
$na \in \omega \Longrightarrow$
$A \in \omega \Longrightarrow$
$a \in \omega \Longrightarrow$
$s \in \omega \Longrightarrow$
$R \in \omega \Longrightarrow$
$e \in \omega \Longrightarrow$
$env \in list(Aa) \Longrightarrow$
$0 \in Aa \Longrightarrow$
$is\_dc\_witness(\#\#Aa,\ nth(na,\ env),\ nth(A,\ env),\ nth(a,\ env),\ nth(s,\ env),$
$nth(R,\ env),\ nth(e,\ env)) \longleftrightarrow$
$Aa,\ env \models ?fm(nat,\ A,\ a,\ s,\ R,\ e)$
⟨*proof*⟩

⟨*ML*⟩
⟨*proof*⟩

definition *dcwit_body* :: $[i,i,i,i,i] \Rightarrow o$ **where**
$dcwit\_body(A,a,g,R) \equiv \lambda p.\ snd(p) = dc\_witness(fst(p),\ A,\ a,\ g,\ R)$

⟨*ML*⟩

context *M_replacement*
begin

lemma *dc_witness_closed*[*intro*,*simp*]:
assumes $M(n)$ $M(A)$ $M(a)$ $M(s)$ $M(R)$ $n \in nat$
shows $M(dc\_witness(n,A,a,s,R))$
⟨*proof*⟩

lemma *dc_witness_rel_char*:
assumes $M(A)$
shows $dc\_witness\_rel(M,n,A,a,s,R) = dc\_witness(n,A,a,s,R)$
⟨*proof*⟩

lemma (**in** *M_basic*) *first_section_closed*:
assumes
$M(A)$ $M(a)$ $M(R)$

shows $M(\{x \in A \,.\, \langle a,\, x\rangle \in R\})$
⟨*proof*⟩

lemma *witness_into_A* [*TC*]:
assumes $a{\in}A$
$\forall\, X[M].\ X{\neq}0 \wedge X{\subseteq}A \longrightarrow s`X{\in}A$
$\forall\, y{\in}A.\ \{x{\in}A.\ \langle y,x\rangle{\in}R\ \} \neq 0\ n{\in}nat$
$M(A)\ M(a)\ M(s)\ M(R)$
shows $dc\_witness(n,\, A,\, a,\, s,\, R){\in}A$
⟨*proof*⟩

end — *M_replacement*

locale $M\_DC = M\_trancl + M\_replacement + M\_eclose +$
assumes
separation_is_dcwit_body:
$M(A) \Longrightarrow M(a) \Longrightarrow M(g) \Longrightarrow M(R) \Longrightarrow separation(M,is\_dcwit\_body(M,\, A,\, a,\, g,\, R))$
and
dcwit_replacement:$Ord(na) \Longrightarrow$
$M(na) \Longrightarrow$
$M(A) \Longrightarrow$
$M(a) \Longrightarrow$
$M(s) \Longrightarrow$
$M(R) \Longrightarrow$
transrec_replacement
$(M,\, \lambda n\ f\ ntc.$
is_nat_case
$(M,\, a,$
$\lambda m\ bmfm.$
$\exists\, fm[M].\ \exists\, cp[M].$
$is\_apply(M,\, f,\, m,\, fm) \wedge$
$is\_Collect(M,\, A,\, \lambda x.\ \exists\, fmx[M].\ (M(x) \wedge fmx \in R) \wedge pair(M,\, fm,\, x,\, fmx),\, cp) \wedge$
$is\_apply(M,\, s,\, cp,\, bmfm),$
$n,\, ntc),na)$
begin

lemma *is_dc_witness_iff*:
assumes $Ord(na)\ M(na)\ M(A)\ M(a)\ M(s)\ M(R)\ M(res)$
shows $is\_dc\_witness(M,\, na,\, A,\, a,\, s,\, R,\, res) \longleftrightarrow res = dc\_witness\_rel(M,\, na,\, A,\, a,\, s,\, R)$
⟨*proof*⟩

lemma *dcwit_body_abs*:
$fst(x) \in \omega \Longrightarrow M(A) \Longrightarrow M(a) \Longrightarrow M(g) \Longrightarrow M(R) \Longrightarrow M(x) \Longrightarrow$
$is\_dcwit\_body(M,A,a,g,R,x) \longleftrightarrow dcwit\_body(A,a,g,R,x)$
⟨*proof*⟩

lemma *Lambda_dc_witness_closed*:
assumes $g \in Pow^M(A)\text{-}\{0\} \to A$ $a{\in}A$ $\forall\, y{\in}A.\ \{x \in A\ .\ \langle y,\, x\rangle \in R\} \neq 0$
$M(g)$ $M(A)$ $M(a)$ $M(R)$
shows $M(\lambda n{\in}nat.\ dc\_witness(n,A,a,g,R))$
⟨*proof*⟩

lemma *witness_related*:
assumes $a{\in}A$
$\forall\, X[M].\ X{\neq}0 \wedge X{\subseteq}A \longrightarrow s`X{\in}X$
$\forall\, y{\in}A.\ \{x{\in}A.\ \langle y,x\rangle{\in}R\ \} \neq 0$ $n{\in}nat$
$M(a)$ $M(A)$ $M(s)$ $M(R)$ $M(n)$
shows $\langle dc\_witness(n,\ A,\ a,\ s,\ R), dc\_witness(succ(n),\ A,\ a,\ s,\ R)\rangle{\in}R$
⟨*proof*⟩

lemma *witness_funtype*:
assumes $a{\in}A$
$\forall\, X[M].\ X{\neq}0 \wedge X{\subseteq}A \longrightarrow s`X \in A$
$\forall\, y{\in}A.\ \{x{\in}A.\ \langle y,x\rangle{\in}R\} \neq 0$
$M(A)$ $M(a)$ $M(s)$ $M(R)$
shows $(\lambda n{\in}nat.\ dc\_witness(n,\ A,\ a,\ s,\ R)) \in nat \to A$ (**is** *?f* $\in$ _ $\to$ _)
⟨*proof*⟩

lemma *witness_to_fun*:
assumes $a{\in}A$
$\forall\, X[M].\ X{\neq}0 \wedge X{\subseteq}A \longrightarrow s`X{\in}A$
$\forall\, y{\in}A.\ \{x{\in}A.\ \langle y,x\rangle{\in}R\ \} \neq 0$
$M(A)$ $M(a)$ $M(s)$ $M(R)$
shows $\exists f \in nat{\to}A.\ \forall\, n{\in}nat.\ f`n = dc\_witness(n,A,a,s,R)$
⟨*proof*⟩

end — *M_DC*

locale *M_library_DC* = *M_library* + *M_DC*
begin

lemma *AC_M_func*:
assumes $\bigwedge x.\ x \in A \Longrightarrow (\exists\, y.\ y \in x)$ $M(A)$
shows $\exists f \in A \to^M \bigcup(A).\ \forall\, x \in A.\ f`x \in x$
⟨*proof*⟩

lemma *non_empty_family*: $[\!|\ 0 \notin A;\ \ x \in A\ |\!] ==> \exists\, y.\ y \in x$
⟨*proof*⟩

lemma *AC_M_func0*: $0 \notin A \Longrightarrow M(A) \Longrightarrow \exists f \in A \to^M \bigcup(A).\ \forall\, x \in A.\ f`x \in x$
⟨*proof*⟩

lemma *AC_M_func_Pow_rel*:
assumes $M(C)$
shows $\exists f \in (Pow^M(C)\text{-}\{0\}) \to^M C.\ \forall\, x \in Pow^M(C)\text{-}\{0\}.\ f`x \in x$

⟨*proof*⟩

theorem *pointed_DC*:
assumes $\forall x{\in}A.\ \exists y{\in}A.\ \langle x,y\rangle{\in}\ R\ M(A)\ M(R)$
shows $\forall a{\in}A.\ \exists f \in nat{\to}^M A.\ f`0 = a \wedge (\forall n \in nat.\ \langle f`n,f`succ(n)\rangle{\in}R)$
⟨*proof*⟩

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

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

corollary *DC_on_A_x_nat* :
assumes $(\forall x{\in}A{\times}nat.\ \exists y{\in}A.\ \langle x,\langle y,succ(snd(x))\rangle\rangle \in R)\ a{\in}A\ M(A)\ M(R)$
shows $\exists f \in nat{\to}^M A.\ f`0 = a \wedge (\forall n \in nat.\ \langle\langle f`n,n\rangle,\langle f`succ(n),succ(n)\rangle\rangle{\in}R)$ (**is** $\exists x{\in}\_.?P(x)$)
⟨*proof*⟩

lemma *aux_sequence_DC* :
assumes $\forall x{\in}A.\ \forall n{\in}nat.\ \exists y{\in}A.\ \langle x,y\rangle \in S`n$
$R{=}\{\langle\langle x,n\rangle,\langle y,m\rangle\rangle \in (A{\times}nat){\times}(A{\times}nat).\ \langle x,y\rangle{\in}S`m\ \}$
shows $\forall\ x{\in}A{\times}nat\ .\ \exists y{\in}A.\ \langle x,\langle y,succ(snd(x))\rangle\rangle \in R$
⟨*proof*⟩

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

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

end — *M_library_DC*

end

31 Cohen forcing notions

theory *Partial_Functions_Relative*
imports
Cardinal_Library_Relative
begin

In this theory we introduce bounded partial functions and its relative version; for historical reasons the relative version is based on a proper definition

of partial functions.

We note that finite partial functions are easier and are used to prove some lemmas about finite sets in the theory *Transitive_Models.ZF_Library_Relative*.

definition
Fn :: $[i,i,i] \Rightarrow i$ **where**
$Fn(\kappa,I,J) \equiv \bigcup \{y \ . \ d \in Pow(I),\ y{=}(d{\to}J) \wedge d{\prec}\kappa\}$

lemma *domain_function_lepoll* :
assumes *function*(r)
shows *domain*(r) $\lesssim r$
⟨*proof*⟩

lemma *function_lepoll*:
assumes $r{:}d{\to}J$
shows $r \lesssim d$
⟨*proof*⟩

lemma *function_eqpoll* :
assumes $r{:}d{\to}J$
shows $r \approx d$
⟨*proof*⟩

lemma *Fn_char* : $Fn(\kappa,I,J) = \{f \in Pow(I{\times}J) \ . \ function(f) \wedge f \prec \kappa\}$ (**is** *?L=?R*)
⟨*proof*⟩

lemma *zero_in_Fn*:
assumes $0 < \kappa$
shows $0 \in Fn(\kappa,\ I,\ J)$
⟨*proof*⟩

lemma *Fn_nat_eq_FiniteFun*: $Fn(nat,I,J) = I$ -||> J
⟨*proof*⟩

lemma *Fn_nat_subset_Pow*: $Fn(\kappa,I,J) \subseteq Pow(I{\times}J)$
⟨*proof*⟩

lemma *FnI*:
assumes $p : d \to J\ d \subseteq I\ d \prec \kappa$
shows $p \in Fn(\kappa,I,J)$
⟨*proof*⟩

lemma *FnD*[*dest*]:
assumes $p \in Fn(\kappa,I,J)$
shows $\exists\, d.\ p : d \to J \wedge d \subseteq I \wedge d \prec \kappa$
⟨*proof*⟩

lemma *Fn_is_function*: $p \in Fn(\kappa,I,J) \Longrightarrow function(p)$
⟨*proof*⟩

lemma *Fn_csucc*:
assumes $Ord(\kappa)$
shows $Fn(csucc(\kappa),I,J) = \bigcup\{y\ .\ d \in Pow(I),\ y{=}(d{\to}J) \wedge d{\lesssim}\kappa\}$
⟨*proof*⟩

definition
$FnleR :: i \Rightarrow i \Rightarrow o$ (**infixl** ‹⊇› *50*) **where**
$f \supseteq g \equiv g \subseteq f$

lemma *FnleR_iff_subset* [*iff*]: $f \supseteq g \longleftrightarrow g \subseteq f$
⟨*proof*⟩

definition
$Fnlerel :: i \Rightarrow i$ **where**
$Fnlerel(A) \equiv Rrel(\lambda x\ y.\ x \supseteq y, A)$

definition
$Fnle :: [i,i,i] \Rightarrow i$ **where**
$Fnle(\kappa,I,J) \equiv Fnlerel(Fn(\kappa,I,J))$

lemma *FnleI*[*intro*]:
assumes $p \in Fn(\kappa,I,J)$ $q \in Fn(\kappa,I,J)$ $p \supseteq q$
shows $\langle p,q\rangle \in Fnle(\kappa,I,J)$
⟨*proof*⟩

lemma *FnleD*[*dest*]:
assumes $\langle p,q\rangle \in Fnle(\kappa,I,J)$
shows $p \in Fn(\kappa,I,J)$ $q \in Fn(\kappa,I,J)$ $p \supseteq q$
⟨*proof*⟩

definition *PFun_Space_Rel* :: $[i,i{\Rightarrow}o,\ i] \Rightarrow i$ (‹_ –⇀ _›)
where $A \rightharpoonup^M B \equiv \{f \in Pow(A{\times}B)\ .\ M(f) \wedge function(f)\}$

lemma (**in** *M_library*) *PFun_Space_subset_Powrel* :
assumes $M(A)$ $M(B)$
shows $A \rightharpoonup^M B = \{f \in Pow^M(A{\times}B)\ .\ function(f)\}$
⟨*proof*⟩

lemma (**in** *M_library*) *PFun_Space_closed* :
assumes $M(A)$ $M(B)$
shows $M(A \rightharpoonup^M B)$
⟨*proof*⟩

lemma *pfun_is_function* :
$f \in A \rightharpoonup^M B \Longrightarrow function(f)$
⟨*proof*⟩

lemma *pfun_range* :

$f \in A \rightharpoonup^M B \Longrightarrow range(f) \subseteq B$
⟨*proof*⟩

lemma *pfun_domain* :
$f \in A \rightharpoonup^M B \Longrightarrow domain(f) \subseteq A$
⟨*proof*⟩

lemma *Un_filter_fun_space_closed*:
assumes $G \subseteq I \to J \bigwedge f\ g\ .\ f{\in}G \Longrightarrow g{\in}G \Longrightarrow \exists\, d{\in}I{\to}\ J\ .\ d \supseteq f \wedge d \supseteq g$
shows $\bigcup G \in Pow(I{\times}J)\ function(\bigcup G)$
⟨*proof*⟩

lemma *Un_filter_is_fun* :
assumes $G \subseteq I \to J \bigwedge f\ g\ .\ f{\in}G \Longrightarrow g{\in}G \Longrightarrow \exists\, d{\in}I{\to}\ J\ .\ d{\supseteq}f \wedge d{\supseteq}g\ G{\neq}0$
shows $\bigcup G \in I \to J$
⟨*proof*⟩

context *M_Pi*
begin

lemma *mem_function_space_relD*:
assumes $f \in function\_space\_rel(M,A,y)\ M(A)\ M(y)$
shows $f \in A \to y$ **and** $M(f)$
⟨*proof*⟩

lemma *pfunI* :
assumes $C{\subseteq}A\ f \in C \to^M B\ M(C)\ M(B)$
shows $f{\in}\ A \rightharpoonup^M B$
⟨*proof*⟩

lemma *zero_in_PFun_rel*:
assumes $M(I)\ M(J)$
shows $0 \in I \rightharpoonup^M J$
⟨*proof*⟩

lemma *pfun_subsetI* :
assumes $f \in A \rightharpoonup^M B\ g{\subseteq}f\ M(g)$
shows $g{\in}\ A \rightharpoonup^M B$
⟨*proof*⟩

lemma *pfun_Un_filter_closed*:
assumes $G \subseteq I{\rightharpoonup^M}\ J \bigwedge f\ g\ .\ f{\in}G \Longrightarrow g{\in}G \Longrightarrow \exists\, d{\in}I{\rightharpoonup^M}\ J\ .\ d{\supseteq}f \wedge d{\supseteq}g$
shows $\bigcup G \in Pow(I{\times}J)\ function(\bigcup G)$
⟨*proof*⟩

lemma *pfun_Un_filter_closed''*:
assumes $G \subseteq I{\rightharpoonup^M}\ J \bigwedge f\ g\ .\ f{\in}G \Longrightarrow g{\in}G \Longrightarrow \exists\, d{\in}G\ .\ d{\supseteq}f \wedge d{\supseteq}g$
shows $\bigcup G \in Pow(I{\times}J)\ function(\bigcup G)$
⟨*proof*⟩

lemma *pfun_Un_filter_closed'*:
assumes $G \subseteq I \rightharpoonup^M J$ $\bigwedge f\ g\ .\ f{\in}G \Longrightarrow g{\in}G \Longrightarrow \exists\, d{\in}G\ .\ d{\supseteq}f \wedge d{\supseteq}g\ M(G)$
shows $\bigcup G \in I \rightharpoonup^M J$
⟨*proof*⟩

lemma *pfunD* :
assumes $f \in A \rightharpoonup^M B$
shows $\exists\, C[M].\ C{\subseteq}A \wedge f \in C{\rightarrow}B$
⟨*proof*⟩

lemma *pfunD_closed* :
assumes $f \in A \rightharpoonup^M B$
shows $M(f)$
⟨*proof*⟩

lemma *pfun_singletonI* :
assumes $x \in A$ $b \in B$ $M(A)$ $M(B)$
shows $\{\langle x,b\rangle\} \in A \rightharpoonup^M B$
⟨*proof*⟩

lemma *pfun_unionI* :
assumes $f \in A \rightharpoonup^M B$ $g \in A \rightharpoonup^M B$ $domain(f) \cap domain(g) = 0$
shows $f \cup g \in A \rightharpoonup^M B$
⟨*proof*⟩

lemma (**in** *M_library*) *pfun_restrict_eq_imp_compat*:
assumes $f \in I \rightharpoonup^M J$ $g \in I \rightharpoonup^M J$ $M(J)$
$restrict(f,\ domain(f) \cap domain(g)) = restrict(g,\ domain(f) \cap domain(g))$
shows $f \cup g \in I \rightharpoonup^M J$
⟨*proof*⟩

lemma *FiniteFun_pfunI* :
assumes $f \in A \text{-}||{>}\ B$ $M(A)$ $M(B)$
shows $f \in A \rightharpoonup^M B$
⟨*proof*⟩

lemma *PFun_FiniteFunI* :
assumes $f \in A \rightharpoonup^M B$ $Finite(f)$
shows $f \in A \text{-}||{>}\ B$
⟨*proof*⟩

end — *M_Pi*

definition
Fn_rel :: $[i{\Rightarrow}o,i,i,i] \Rightarrow i$ (‹*Fn*–'(_,_,_')›) **where**
$Fn\_rel(M,\kappa,I,J) \equiv \{f \in I \rightharpoonup^M J\ .\ f \prec^M \kappa\}$
— Eventually we can define *Fn_rel* as the relativization of *Fn*

context *M_library*
begin

lemma *Fn_rel_subset_PFun_rel* : $Fn^M(\kappa,I,J) \subseteq I -\!\!\rightarrow^M J$
⟨*proof*⟩

lemma *Fn_relI*[*intro*]:
assumes $f : d \to J$ $d \subseteq I$ $f \prec^M \kappa$ $M(d)$ $M(J)$ $M(f)$
shows $f \in Fn\_rel(M,\kappa,I,J)$
⟨*proof*⟩

lemma *Fn_relD*[*dest*]:
assumes $p \in Fn\_rel(M,\kappa,I,J)$
shows $\exists\, C[M].\ C \subseteq I \wedge p : C \to J \wedge p \prec^M \kappa$
⟨*proof*⟩

lemma *Fn_rel_is_function*:
assumes $p \in Fn\_rel(M,\kappa,I,J)$
shows *function*(p) $M(p)$ $p \prec^M \kappa$ $p \in I -\!\!\rightarrow^M J$
⟨*proof*⟩

lemma *Fn_rel_mono*:
assumes $p \in Fn\_rel(M,\kappa,I,J)$ $\kappa \prec^M \kappa'$ $M(\kappa)$ $M(\kappa')$
shows $p \in Fn\_rel(M,\kappa',I,J)$
⟨*proof*⟩

lemma *Fn_rel_mono'*:
assumes $p \in Fn\_rel(M,\kappa,I,J)$ $\kappa \lesssim^M \kappa'$ $M(\kappa)$ $M(\kappa')$
shows $p \in Fn\_rel(M,\kappa',I,J)$
⟨*proof*⟩

lemma *Fn_csucc*:
assumes *Ord*(κ) $M(\kappa)$
shows $Fn\_rel(M,(\kappa^+)^M,I,J) = \{p \in I -\!\!\rightarrow^M J\ .\ p \lesssim^M \kappa\}$ (**is** *?L=?R*)
⟨*proof*⟩

lemma *Finite_imp_lesspoll_nat*:
assumes *Finite*(A)
shows $A \prec nat$
⟨*proof*⟩

lemma *FinD_Finite* :
assumes $a \in Fin(A)$
shows *Finite*(a)
⟨*proof*⟩

lemma *Fn_rel_nat_eq_FiniteFun*:
assumes $M(I)$ $M(J)$
shows I -||> $J = Fn\_rel(M,\omega,I,J)$

⟨*proof*⟩

lemma *Fn_nat_abs*:
assumes $M(I)$ $M(J)$
shows $Fn(nat,I,J) = Fn\_rel(M,\omega,I,J)$
⟨*proof*⟩

lemma *Fn_rel_singletonI*:
assumes $x \in I$ $j \in J$ $1 \prec^M \kappa$ $M(\kappa)$ $M(I)$ $M(J)$
shows $\{\langle x,j\rangle\} \in Fn^M(\kappa,I,J)$
⟨*proof*⟩

end — *M_library*

definition
$Fnle\_rel :: [i \Rightarrow o,i,i,i] \Rightarrow i$ (‹*Fnle–'*(_,_,_')›) **where**
$Fnle\_rel(M,\kappa,I,J) \equiv Fnlerel(Fn^M(\kappa,I,J))$

abbreviation
$Fn\_r\_set :: [i,i,i,i] \Rightarrow i$ (‹*Fn–'*(_,_,_')›) **where**
$Fn\_r\_set(M) \equiv Fn\_rel(\#\#M)$

abbreviation
$Fnle\_r\_set :: [i,i,i,i] \Rightarrow i$ (‹*Fnle–'*(_,_,_')›) **where**
$Fnle\_r\_set(M) \equiv Fnle\_rel(\#\#M)$

context *M_library*
begin

lemma *Fnle_relI*[*intro*]:
assumes $p \in Fn\_rel(M,\kappa,I,J)$ $q \in Fn\_rel(M,\kappa,I,J)$ $p \supseteq q$
shows $\langle p,\ q\rangle \in Fnle\_rel(M,\kappa,I,J)$
⟨*proof*⟩

lemma *Fnle_relD*[*dest*]:
assumes $\langle p,\ q\rangle \in Fnle\_rel(M,\kappa,I,J)$
shows $p \in Fn\_rel(M,\kappa,I,J)$ $q \in Fn\_rel(M,\kappa,I,J)$ $p \supseteq q$
⟨*proof*⟩

lemma *Fn_rel_closed*[*intro*,*simp*]:
assumes $M(\kappa)$ $M(I)$ $M(J)$
shows $M(Fn^M(\kappa,I,J))$
⟨*proof*⟩

lemma *Fn_rel_subset_Pow*:
assumes $M(\kappa)$ $M(I)$ $M(J)$
shows $Fn^M(\kappa,I,J) \subseteq Pow(I \times J)$
⟨*proof*⟩

lemma *Fnle_rel_closed*[*intro*,*simp*]:
  assumes $M(\kappa)$ $M(I)$ $M(J)$
  shows $M(Fnle^M(\kappa,I,J))$
  ⟨*proof*⟩

lemma *zero_in_Fn_rel*:
  assumes $0<\kappa$ $M(\kappa)$ $M(I)$ $M(J)$
  shows $0 \in Fn^M(\kappa,\ I,\ J)$
  ⟨*proof*⟩

lemma *zero_top_Fn_rel*:
  assumes $p \in Fn^M(\kappa,\ I,\ J)$ $0<\kappa$ $M(\kappa)$ $M(I)$ $M(J)$
  shows $\langle p,\ 0\rangle \in Fnle^M(\kappa,\ I,\ J)$
  ⟨*proof*⟩

lemma *preorder_on_Fnle_rel*:
  assumes $M(\kappa)$ $M(I)$ $M(J)$
  shows *preorder_on*$(Fn^M(\kappa,\ I,\ J),\ Fnle^M(\kappa,\ I,\ J))$
  ⟨*proof*⟩

end — *M_library*

context *M_cardinal_library*
begin

lemma *lesspoll_nat_imp_lesspoll_rel*:
  assumes $A \prec \omega$ $M(A)$
  shows $A \prec^M \omega$
⟨*proof*⟩

lemma *Finite_imp_lesspoll_rel_nat*:
  assumes *Finite*(A) $M(A)$
  shows $A \prec^M \omega$
  ⟨*proof*⟩

end — *M_cardinal_library*

context *M_cardinal_library_extra*
begin

lemma *InfCard_rel_lesspoll_rel_Un*:
  includes *Ord_dests*
  assumes *InfCard_rel*(M,κ) $A \prec^M \kappa$ $B \prec^M \kappa$
    and *types*: $M(\kappa)$ $M(A)$ $M(B)$
  shows $A \cup B \prec^M \kappa$
⟨*proof*⟩

lemma *Fn_rel_unionI*:

assumes $p \in Fn^M(\kappa,I,J)$ $q \in Fn^M(\kappa,I,J)$ $InfCard^M(\kappa)$
$M(\kappa)$ $M(I)$ $M(J)$ $domain(p) \cap domain(q) = 0$
shows $p \cup q \in Fn^M(\kappa,I,J)$
⟨*proof*⟩

lemma *restrict_eq_imp_compat_rel*:
assumes $p \in Fn^M(\kappa,\ I,\ J)$ $q \in Fn^M(\kappa,\ I,\ J)$ $InfCard^M(\kappa)$ $M(J)$ $M(\kappa)$
$restrict(p,\ domain(p) \cap domain(q)) = restrict(q,\ domain(p) \cap domain(q))$
shows $p \cup q \in Fn^M(\kappa,\ I,\ J)$
⟨*proof*⟩

lemma *InfCard_rel_imp_n_lesspoll_rel* :
assumes $InfCard^M(\kappa)$ $M(\kappa)$ $n \in \omega$
shows $n \prec^M \kappa$
⟨*proof*⟩

lemma *cons_in_Fn_rel*:
assumes $x \notin domain(p)$ $p \in Fn^M(\kappa,I,J)$ $x \in I$ $j \in J$ $InfCard^M(\kappa)$
$M(\kappa)$ $M(I)$ $M(J)$
shows $cons(\langle x,j\rangle,\ p) \in Fn^M(\kappa,I,J)$
⟨*proof*⟩

lemma *dense_dom_dense*:
assumes $x \in I$ $InfCard^M(\kappa)$ $M(I)$ $M(J)$ $z \in J$ $M(\kappa)$ $p \in Fn^M(\kappa,\ I,\ J)$
shows $\exists\, d \in \{\ p \in Fn^M(\kappa,\ I,\ J)\ .\ x \in domain(p)\ \}.\ \langle d,p\rangle \in Fnle^M(\kappa,\ I,\ J)$
⟨*proof*⟩

lemma (**in** *M_cardinal_library*) *domain_lepoll_rel* :
assumes $function(r)$ $M(r)$
shows $domain(r) \lesssim^M r$
⟨*proof*⟩

lemma *dense_surj_dense*:
assumes $x \in J$ $InfCard^M(\kappa)$ $M(I)$ $M(J)$ $M(\kappa)$ $p \in Fn^M(\kappa,\ I,\ J)$ $\kappa \lesssim^M I$
shows $\exists\, d \in \{\ p \in Fn^M(\kappa,\ I,\ J)\ .\ x \in range(p)\ \}.\ \langle d,p\rangle \in Fnle^M(\kappa,\ I,\ J)$
⟨*proof*⟩

end — *M_cardinal_library_extra*

end

References

[1] E. Gunther, M. Pagano, P. Sánchez Terraf, Formalization of Forcing in Isabelle/ZF, arXiv e-prints, in: N. Peltier, V. Sofronie-Stokkermans (Eds.), Automated Reasoning. 10th International Joint Conference, IJCAR 2020, Paris, France, July 1–4, 2020, Proceedings, Part II, Lecture Notes in Artificial Intelligence **12167**, Springer International Publishing: 221–235 (2020).

[2] Y.N. Moschovakis, "Notes on Set Theory", Springer Texts in Electrical Engineering, Springer-Verlag (1994).

[3] L.C. Paulson, The relative consistency of the axiom of choice mechanized using Isabelle/ZF, *LMS J. Comput. Math.* **6**: 198–248 (2003). Appendix A available electronically at http://www.lms.ac.uk/jcm/6/lms2003-001/appendix-a/.