Axiomatic theory of hereditarily finite sets and its fragments.

Štěpán Holub Zuzana Haniková

July 14, 2026

Funded by the Czech Science Foundation grant GAČR 25-16489S.

Abstract

Axiomatization of the theory of hereditarily finite sets in classical first-order logic is systematically explored and formalized. Our list of axioms includes the usual ones of the Zermelo-Fraenkel set theory, as well as some of their less usual variants and alternatives. The strongest theory considered is ZFfin, obtained from ZF by replacing the axiom of infinity with its negation and adding the axiom of transitive closure. Of particular interest are the axioms used in the set fragment of the Alternative Set Theory by Vopěnka [10, 7, 8, 9], which provide a different approach to axiomatizing ZFfin. The hierarchy of fragments of ZFfin is represented as a hierarchy of locales.

Set-theoretic membership, the only basic predicate in the language of the theory, is rendered as a polymorphic binary predicate. Accordingly, first-order formulas of this language are understood as predicates over that type. ZFfin is not finitely axiomatizable, so it is necessary to use axiom schemata. To that end, an inductive definition of set-theoretically definable predicates is introduced and their basic properties are elaborated.

A major aim of the formalization is to study several axioms of finiteness and to formalize the equivalence of the fragments in which these axioms are used. In particular, we formalize the equivalence of ZFfin and the set fragment of Vopěnka's Alternative Set Theory, where the latter uses the schema of induction for sets as a finiteness principle. Next to that, Tarski finiteness and Dedekind finiteness are used. The formalization also focuses on the axioms of transitive closure, regularity, and the axiom schema of epsilon induction and their interplay in the context of various fragments.

Hereditarily finite sets, previously formalized by Paulson, satisfy the theory ZFfin. To demonstrate independence of some statements, several non-standard models of specific fragments are constructed. We formalize the Bernays-Rieger method of permutation models, and using the construction of a model dating back to [6] we formalize the proof that neither the scheme of regularity nor the existence of a transitive closure follow from regularity for finite sets.

This formalization takes inspiration from various sources but does not follow closely any of them. In addition to already cited literature, it covers some results from [1], [2], [4] and [3].

Contents

1 Fragments of Zermelo-Fraenkel set theory without infinity **2**
1.1 Preliminaries . . . 2
1.2 Signature . . . 2
1.2.1 Membership and basic set-theoretic notions . . . 2
1.2.2 Set theoretical notions . . . 3
1.2.3 Set relations and properties . . . 7
1.3 Axiomatizations of hereditarily finite sets . . . 12
1.3.1 Finiteness axioms . . . 12
1.3.2 Extensionality . . . 13
1.3.3 Empty set . . . 14
1.3.4 Set pair . . . 16
1.3.5 Set successor . . . 17
1.3.6 Regularity . . . 22
1.3.7 Separation . . . 24
1.3.8 Transitive superset . . . 34
1.3.9 Replacement . . . 35
1.3.10 Powerset . . . 41
1.3.11 Union . . . 43
1.3.12 Successor induction . . . 59
1.3.13 Negation of inf . . . 66
1.3.14 Dedekind finite . . . 68
1.3.15 Tarski finite . . . 69
1.3.16 Set induction and regularity schema . . . 70
1.3.17 Summary of dependencies . . . 73

2 Models and counter-models **80**
2.1 Hereditarily finite sets . . . 80
2.2 Permutation models . . . 81
2.3 Regularity . . . 84
2.4 Independence of transitive superset . . . 90

References **100**

Chapter 1

Fragments of Zermelo-Fraenkel set theory without infinity

theory *ZFfin*
imports *Main*
begin

We explore axiomatizations of the theory of hereditarily finite sets, their fragments and relationships between them.

1.1 Preliminaries

lemma *ex-or-iff-or* [*simp*]: $(\exists\, w.\ (w = x \lor w = y) \land P\ u\ w) \longleftrightarrow (P\ u\ x \lor P\ u\ y)$
by *blast*

— A tautology recording the behaviour of bounded quantifiers. The logical equivalence between the axiom schemata *regsch* and *epsind* below is a particular instance.
lemma *abstract-foundation-iff*: $((\exists\, x.\ P\ x) \longrightarrow (\exists\, x.\ B\ x \land P\ x)) \longleftrightarrow (\forall\ x.\ B\ x \longrightarrow \neg\ P\ x) \longrightarrow (\forall\ x.\ \neg\ P\ x)$
by *auto*

1.2 Signature

1.2.1 Membership and basic set-theoretic notions

locale *set-signature* =
fixes *membership* :: $'a \Rightarrow\ 'a \Rightarrow bool$ (**infix** ε *50*)

begin

General relation between regularity schema and epsilon-induction scheme. Applying the above remark about B-induction and B-foundation

thm *abstract-foundation-iff*[*of* λ *x*. $\neg$ *P x* λ *x*. ($\forall$ *y*. *y* ε *x* $\longrightarrow$ (*P y*)), *unfolded not-not*, *symmetric*]
thm *abstract-foundation-iff*[*of* λ *y*. *P y* λ *x*. ($\forall$ *y*. *y* ε *x* $\longrightarrow$ $\neg$ (*P y*)), *unfolded not-not*]

1.2.2 Set theoretical notions

We use "M" (as in "Menge") for our defined versions of set-theoretical concepts, built upon membership.

definition *subsetM* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (- $\subseteq_M$ - [*51*,*51*] *53*) **where**
subsetM x y $\equiv$ ($\forall$ *z*. *z* ε *x* $\longrightarrow$ *z* ε *y*)

lemma *subsetM-refl*[*simp*]: $x \subseteq_M x$
unfolding *subsetM-def* **by** *blast*

lemma *subsetM-trans*[*simp*]: **assumes** $x \subseteq_M y$ $y \subseteq_M z$ **shows** $x \subseteq_M z$
using *assms subsetM-def* **by** *simp*

definition *proper-subsetM* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (- $\subset_M$ - [*50*,*50*] *50*) **where**
proper-subsetM x y $\equiv$ ($\forall$ *z*::$'a$. *z* ε *x* $\longrightarrow$ *z* ε *y*) $\wedge$ $x \neq y$

named-theorems *set-defs*
lemmas[*set-defs*] = *proper-subsetM-def subsetM-def*

lemma *proper-subset-def'*: $x \subset_M y \longleftrightarrow x \subseteq_M y \wedge x \neq y$
unfolding *set-defs* **by** *blast*

lemma *proper-subsetI*: $x \subseteq_M y \Longrightarrow x \neq y \Longrightarrow x \subset_M y$
unfolding *set-defs* **by** *blast*

lemma *proper-subsetM-irrefl*[*simp*]: $\neg\, x \subset_M x$
unfolding *set-defs* **by** *simp*

definition *transM*:: $'a \Rightarrow$ *bool* **where**
transM x $\equiv$ $\forall$ *y*. *y* ε *x* $\longrightarrow$ $y \subseteq_M x$

Hilbert's description operator *The* is used to define sets that are determined by their elements. Resulting set-theoretic operations can and will be used in appropriate contexts only that guarantee the existence of corresponding sets, and their unicity (provable with extensionality).

definition *collectM* :: ($'a \Rightarrow$ *bool*) $\Rightarrow$ $'a$ **where**
collectM Q $\equiv$ *THE z* . ($\forall$ *u* . (*u* ε *z* $\longleftrightarrow$ *Q u*))

definition *emptysetM* ($\emptyset$) **where**
emptysetM $\equiv$ *collectM* (λ *x*. $x \neq x$)

definition *separationM* :: $'a \Rightarrow$ ($'a \Rightarrow$ *bool*) $\Rightarrow$ $'a$ **where**
separationM y Q $\equiv$ *collectM* (λ *u*. *u* ε *y* $\wedge$ *Q u*)

definition *powersetM* :: $'a \Rightarrow 'a$ ($\mathfrak{P}$) **where**
powersetM $x \equiv$ *collectM* $(\lambda\ u.\ u \subseteq_M x)$

definition *binunionM* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** $\cup_M$ *55*) **where**
binunionM $x\ y \equiv$ *collectM* $(\lambda\ u.\ u\ \varepsilon\ x \vee u\ \varepsilon\ y)$

definition *intersectionM* :: $'a \Rightarrow 'a \Rightarrow 'a$ (- $\cap_M$ - [*55*,*54*] *60*) **where**
intersectionM $x\ y \equiv$ *collectM* $(\lambda\ u.\ u\ \varepsilon\ x \wedge u\ \varepsilon\ y)$

lemma *intersection-symm*: $x \cap_M y = y \cap_M x$
unfolding *intersectionM-def* **by** *metis*

definition *unionM* :: $'a \Rightarrow 'a$ ($\bigcup_M$) **where**
unionM $x \equiv$ *collectM* $(\lambda\ u.\ (\exists\ v.\ v\ \varepsilon\ x \wedge u\ \varepsilon\ v))$

definition *differenceM* :: $'a \Rightarrow 'a \Rightarrow 'a$ (- $\backslash_M$ -) **where**
differenceM $x\ y \equiv$ *collectM* $(\lambda\ u.\ u\ \varepsilon\ x \wedge \neg\ u\ \varepsilon\ y\ )$

definition *singletonM* :: $'a \Rightarrow 'a$ ($\{$-$\}_M$ *70*) **where**
singletonM $x \equiv$ *collectM* $(\lambda\ u.\ u = x)$

definition *pairM* :: $'a \Rightarrow 'a \Rightarrow 'a$ ($\{$-,-$\}_M$ [*51*,*51*] *60*) **where**
pairM $x\ y \equiv$ *collectM* $(\lambda\ u.\ u = x \vee u = y)$

— Set successor. Also known as adjunction.
definition *setsucM* :: $'a \Rightarrow 'a \Rightarrow 'a$ ($\mathfrak{suc}$) **where**
setsucM $x\ y \equiv$ *collectM* $(\lambda\ u.\ u\ \varepsilon\ x \vee u = y)$

definition *replaceM* :: $('a \Rightarrow 'a \Rightarrow$ *bool*$) \Rightarrow 'a \Rightarrow 'a$ **where**
replaceM $P\ x \equiv$ *collectM* $(\lambda\ u.\ \exists\ v.\ v\ \varepsilon\ x \wedge P\ v\ u)$

definition *leastM* :: $('a \Rightarrow$ *bool*$) \Rightarrow 'a$ ($\bigcap_M$) **where**
leastM $Q \equiv$ *collectM* $(\lambda\ u.\ \forall\ y.\ Q\ y \longrightarrow u\ \varepsilon\ y)$

Least transitive superset

definition *least-tsM* :: $'a \Rightarrow 'a$ **where**
least-tsM $x = \bigcap_M (\lambda\ y.$ *transM* $y \wedge x \subseteq_M y)$

definition *ordered-pairM* :: $'a \Rightarrow 'a \Rightarrow 'a$ ($\langle$-,-$\rangle$ [*51*,*51*] *60*) **where**
ordered-pairM $a\ b \equiv \{\{a\}_M,\{a,b\}_M\}_M$

definition *relationM* :: $'a \Rightarrow$ *bool* **where**
relationM $x \equiv \forall\ v.\ v\ \varepsilon\ x \longrightarrow (\exists\ a\ b.\ v = \langle a,b\rangle)$

definition *rel-inverseM* :: $'a \Rightarrow 'a$ **where**
rel-inverseM $r \equiv$ *collectM* $(\lambda\ u.\ \exists\ a\ b.\ \langle a,b\rangle\ \varepsilon\ r \wedge u = \langle b,a\rangle)$

definition *functionM* :: $'a \Rightarrow$ *bool* **where**

functionM x ≡ *relationM x* ∧ (∀ *a b c*. ⟨*a*,*b*⟩ ε *x* ∧ ⟨*a*,*c*⟩ ε *x* ⟶ *b* = *c*)

lemma *funD*: *functionM f* ⟹ ⟨*a*,*b*⟩ ε *f* ⟹ ⟨*a*,*c*⟩ ε *f* ⟹ *b* = *c*
unfolding *functionM-def* **by** *blast*

definition *domM* :: *'a* ⇒ *'a* **where**
domM r ≡ *collectM* (λ *u*. ∃ *v*. ⟨*u*,*v*⟩ ε *r*)

definition *rngM* :: *'a* ⇒ *'a* **where**
rngM r ≡ *collectM* (λ *u*. ∃ *v*. ⟨*v*,*u*⟩ ε *r*)

definition *one-oneM* :: *'a* ⇒ *bool* **where**
one-oneM f ≡ *functionM f* ∧ (∀ *a b c*. ⟨*b*,*a*⟩ ε *f* ∧ ⟨*c*,*a*⟩ ε *f* ⟶ *b* = *c*)

lemma *one-one-inj*: *one-oneM f* ⟹ ⟨*b*,*a*⟩ ε *f* ⟹ ⟨*c*,*a*⟩ ε *f* ⟹ *b* = *c*
unfolding *one-oneM-def* **by** *blast*

lemma *one-oneI*: **assumes** ∀ *v*. *v* ε *f* ⟶ (∃ *a b*. *v* = ⟨*a*,*b*⟩) (∀ *a b c*. ⟨*b*,*a*⟩ ε *f* ∧ ⟨*c*,*a*⟩ ε *f* ⟶ *b* = *c*) (∀ *a b c*. ⟨*a*,*b*⟩ ε *f* ∧ ⟨*a*,*c*⟩ ε *f* ⟶ *b* = *c*)
shows *one-oneM f*
using *assms* **unfolding** *one-oneM-def functionM-def relationM-def* **by** *blast+*

lemma *one-oneD1*: *one-oneM f* ⟹ ∀ *v*. *v* ε *f* ⟶ (∃ *a b*. *v* = ⟨*a*,*b*⟩)
unfolding *one-oneM-def functionM-def relationM-def* **by** *blast*

lemma *one-oneD2*: *one-oneM f* ⟹ (∀ *a b c*. ⟨*b*,*a*⟩ ε *f* ∧ ⟨*c*,*a*⟩ ε *f* ⟶ *b* = *c*)
unfolding *one-oneM-def functionM-def relationM-def* **by** *blast*

lemma *one-oneD3*: *one-oneM f* ⟹ (∀ *a b c*. ⟨*a*,*b*⟩ ε *f* ∧ ⟨*a*,*c*⟩ ε *f* ⟶ *b* = *c*)
unfolding *one-oneM-def functionM-def relationM-def* **by** *blast*

definition *set-equivalent* :: *'a* ⇒ *'a* ⇒ *bool* (- $\approx_M$ - [*51*,*51*] *52*) **where**
set-equivalent x y ≡ (∃ *f*. *one-oneM f* ∧ *x* = *domM f* ∧ *y* = *rngM f*)

definition *cartesian-productM* :: *'a* ⇒ *'a* ⇒ *'a* (- $\times_M$ - [*51*,*51*] *60*) **where**
cartesian-productM x y ≡ *collectM* (λ *u*. ∃ v_1 v_2. v_1 ε *x* ∧ v_2 ε *y* ∧ *u* = ⟨v_1,v_2⟩)

definition *composeM* :: *'a* ⇒ *'a* ⇒ *'a* (- $\circ_M$ - [*51*,*51*] *60*) **where**
f $\circ_M$ *g* = *collectM* (λ *u*. ∃ *a b c*. ⟨*a*,*b*⟩ ε *g* ∧ ⟨*b*,*c*⟩ ε *f* ∧ *u* = ⟨*a*,*c*⟩)

definition *tarski-fin* :: *'a* ⇒ *bool* **where**
tarski-fin x ≡ ∀ *y*. (∀ *z*. *z* ε *y* ⟶ *z* $\subseteq_M$ *x*) ⟶ (∃ *z*. *z* ε *y*) ⟶ (∃ *z*. *z* ε *y* ∧ ¬(∃ *w*. *w* ε *y* ∧ *z* $\subset_M$ *w*))

lemma *tarski-subset-tarski*: **assumes** *tarski-fin x y* $\subseteq_M$ *x* **shows** *tarski-fin y*
using *assms* **unfolding** *tarski-fin-def subsetM-def* **by** *presburger*

definition *dedekind-fin* :: *'a* ⇒ *bool* **where**
dedekind-fin x ≡ ∀ *y*. (*y* $\subset_M$ *x* ⟶ ¬ *x* $\approx_M$ *y*)

— x is regular if it contains no infinite ε-decreasing chain as a subset
definition *regular* :: $'a \Rightarrow bool$ **where**
$regular\ x \equiv \forall\ y.\ y \subseteq_M x \longrightarrow (\exists\ z.\ z\ \varepsilon\ y) \longrightarrow (\exists\ z.\ z\ \varepsilon\ y \wedge (\forall\ v.\ \neg\ (v\ \varepsilon\ z \wedge v\ \varepsilon\ y)))$

— Ordinals and natural numbers

definition *epschain* :: $'a \Rightarrow bool$ **where**
$epschain\ x \equiv transM\ x \wedge (\forall\ y\ z.\ y\ \varepsilon\ x \wedge z\ \varepsilon\ x \longrightarrow y\ \varepsilon\ z \vee y = z \vee z\ \varepsilon\ y)$

— In theories that do not rely on the regularity axiom, ordinals are explicitly assumed not to contain infinite ε-decreasing chains
definition *ordM* :: $'a \Rightarrow bool$ **where**
$ordM\ x \equiv epschain\ x \wedge regular\ x$

lemma *ordM-I*: $regular\ x \Longrightarrow epschain\ x \Longrightarrow ordM\ x$
using *ordM-def* **by** *blast*

lemma *ordM-regular*[*simp*]: $ordM\ x \Longrightarrow regular\ x$
unfolding *ordM-def* **by** *blast*

lemma *ordM-D1*: $ordM\ x \Longrightarrow y\ \varepsilon\ x \Longrightarrow y \subseteq_M x$
using *ordM-def* **unfolding** *transM-def epschain-def* **by** *blast*

lemma *ordM-trans*: **assumes** $ordM\ v\ \ w\ \varepsilon\ u\ u\ \varepsilon\ v$ **shows** $w\ \varepsilon\ v$
using *assms ordM-D1 subsetM-def* **by** *blast*

lemma *ordM-D2*: $ordM\ x \Longrightarrow y\ \varepsilon\ x \Longrightarrow z\ \varepsilon\ x \Longrightarrow y\ \varepsilon\ z \vee y = z \vee z\ \varepsilon\ y$
using *ordM-def epschain-def* **by** *blast*

definition *is-sucM* :: $'a \Rightarrow bool$ **where**
$is\text{-}sucM\ x \equiv \exists\ y.\ (\forall\ z.\ z\ \varepsilon\ x \longleftrightarrow z\ \varepsilon\ y \vee z = y)$

— A natural number is an ordinal x such that: x and all its elements are either empty or a successor
definition *natM* :: $'a \Rightarrow bool$ **where**
$natM\ x \equiv ordM\ x \wedge (\forall\ v.\ (v\ \varepsilon\ x \vee v = x) \longrightarrow (\exists\ u.\ u\ \varepsilon\ v) \longrightarrow is\text{-}sucM\ v)$

lemma *natM-I*: $ordM\ x \Longrightarrow is\text{-}sucM\ x \Longrightarrow (\bigwedge v.\ \exists\ u.\ u\ \varepsilon\ v \Longrightarrow v\ \varepsilon\ x \Longrightarrow is\text{-}sucM\ v) \Longrightarrow natM\ x$
unfolding *natM-def* **by** *blast*

lemma *nat-is-suc*: $natM\ x \Longrightarrow \exists\ u.\ u\ \varepsilon\ x \Longrightarrow is\text{-}sucM\ x$
unfolding *natM-def* **by** *blast*

lemma *nat-mem-is-suc*: $natM\ x \Longrightarrow v\ \varepsilon\ x \Longrightarrow \exists\ u.\ u\ \varepsilon\ v \Longrightarrow is\text{-}sucM\ v$
unfolding *natM-def* **by** *blast*

lemma *natM-D*: *natM x* $\Longrightarrow$ *ordM x*
using *natM-def* **by** *blast*

definition *cardinality* :: $'a \Rightarrow 'a \Rightarrow bool$ **where**
$cardinality\ x\ n \equiv natM\ n \wedge x \approx_M n$

lemmas[*set-defs*] = *transM-def epschain-def ordM-def tarski-fin-def is-sucM-def natM-def functionM-def relationM-def one-oneM-def rngM-def domM-def regular-def cardinality-def set-equivalent-def*

1.2.3 Set relations and properties

We represent a set formula as the predicate it defines. A predicate assigns truth values to assignments. We consider countably many variables x_i, represented by natural numbers. An assignment maps variables to elements of the domain, i.e., it maps *nat* to $'a$. Predicates defined by set formulas are defined inductively.

inductive *SetFormulaPredicate* :: $((nat \Rightarrow 'a) \Rightarrow bool) \Rightarrow bool$
where
SFP-mem: $\bigwedge m\ n.\ SetFormulaPredicate\ (\lambda\ \Xi.\ (\Xi\ m)\ \varepsilon\ (\Xi\ n))$ — formula $x_m\ \varepsilon\ x_n$
| *SFP-eq*: $\bigwedge m\ n.\ SetFormulaPredicate\ (\lambda\ \Xi.\ (\Xi\ m) = (\Xi\ n))$ — formula $x_m = x_n$
| *SFP-neg*: $SetFormulaPredicate\ P \Longrightarrow SetFormulaPredicate\ (\lambda\ \Xi.\ \neg\ P\ \Xi)$ — formula $\neg\ \varphi$
| *SFP-disj*: $SetFormulaPredicate\ P \Longrightarrow SetFormulaPredicate\ Q$
$\Longrightarrow SetFormulaPredicate\ (\lambda\ \Xi.\ P\ \Xi \vee Q\ \Xi)$ — formula $\varphi \vee \psi$
| *SFP-all*: $\bigwedge n.\ SetFormulaPredicate\ P \Longrightarrow SetFormulaPredicate\ (\lambda\ \Xi.\ \forall\ a.\ P\ (\Xi(n:=a)))$
— formula $\forall\ x_n.\ \varphi$

named-theorems *SFP-rules*
lemmas[*SFP-rules*] = *SFP-mem SFP-eq SFP-neg SFP-disj SFP-all*

Every set-theoretically definable predicate depends on finitely many values. Such values correspond to free variables.

lemma *bounded-free*: **assumes** *SetFormulaPredicate P*
shows $\exists\ m.\ \forall\ \Xi\ \Xi'.\ (\forall\ i < m.\ \Xi\ i = \Xi'\ i) \longrightarrow P(\Xi) = P\ (\Xi')$
proof (*rule SetFormulaPredicate.induct*[*OF assms*, *of* $\lambda\ Q.\ (\exists\ m.\ \forall\ \Xi\ \Xi'.\ (\forall\ i < m.\ \Xi\ i = \Xi'\ i) \longrightarrow Q(\Xi) = Q\ (\Xi'))$])
let $?Q = \lambda\ x.\ True$
show $\exists max.\ \forall \Xi\ \Xi'.\ (\forall i< max.\ \Xi\ i = \Xi'\ i) \longrightarrow (\Xi\ m\ \varepsilon\ \Xi\ n) = (\Xi'\ m\ \varepsilon\ \Xi'\ n)$ **for** *m n* :: *nat*
by (*rule exI*[*of - Suc*(*max m n*)]) *force*
show $\exists max.\ \forall \Xi\ \Xi'.\ (\forall i< max.\ \Xi\ i = \Xi'\ i) \longrightarrow (\Xi\ m = \Xi\ n) = (\Xi'\ m = \Xi'\ n)$ **for** *m n* :: *nat*
by (*rule exI*[*of - Suc*(*max m n*)]) *force*
next
fix $Q :: (nat \Rightarrow 'a) \Rightarrow bool$

assume $\exists\, m.\ \forall \Xi\ \Xi'.\ (\forall\, i{<}m.\ \Xi\ i = \Xi'\ i) \longrightarrow Q\ \Xi = Q\ \Xi'$
then obtain *m* **where** $\forall \Xi\ \Xi'.\ (\forall\, i{<}m.\ \Xi\ i = \Xi'\ i) \longrightarrow Q\ \Xi = Q\ \Xi'$
by *blast*
then have $\forall \Xi\ \Xi'.\ (\forall\, i{<}m.\ \Xi\ i = \Xi'\ i) \longrightarrow (\neg\ Q\ \Xi) = (\neg\ Q\ \Xi')$
by *simp*
thus $\exists\, m.\ \forall \Xi\ \Xi'.\ (\forall\, i{<}m.\ \Xi\ i = \Xi'\ i) \longrightarrow (\neg\ Q\ \Xi) = (\neg\ Q\ \Xi')$
by *blast*
next
fix $Q\ R :: (nat \Rightarrow 'a) \Rightarrow bool$
assume $\exists\, m.\ \forall \Xi\ \Xi'.\ (\forall\, i{<}m.\ \Xi\ i = \Xi'\ i) \longrightarrow Q\ \Xi = Q\ \Xi'$ $\exists\, m.\ \forall \Xi\ \Xi'.\ (\forall\, i{<}m.\ \Xi\ i = \Xi'\ i) \longrightarrow R\ \Xi = R\ \Xi'$
then obtain *max-Q max-R* **where** $\forall \Xi\ \Xi'.\ (\forall\, i{<}max\text{-}Q.\ \Xi\ i = \Xi'\ i) \longrightarrow Q\ \Xi = Q\ \Xi'$ $\forall \Xi\ \Xi'.\ (\forall\, i{<}max\text{-}R.\ \Xi\ i = \Xi'\ i) \longrightarrow R\ \Xi = R\ \Xi'$
by *blast*
hence $\forall \Xi\ \Xi'.\ (\forall\, i{<}(max\ max\text{-}Q\ max\text{-}R).\ \Xi\ i = \Xi'\ i) \longrightarrow (Q\ \Xi \vee R\ \Xi) = (Q\ \Xi' \vee R\ \Xi')$
unfolding *less-max-iff-disj* **by** *metis*
thus $\exists\, m.\ \forall \Xi\ \Xi'.\ (\forall\, i{<}m.\ \Xi\ i = \Xi'\ i) \longrightarrow (Q\ \Xi \vee R\ \Xi) = (Q\ \Xi' \vee R\ \Xi')$
by *blast*
next
fix $Q :: (nat \Rightarrow 'a) \Rightarrow bool$ **and** $n :: nat$
assume $\exists\, m.\ \forall \Xi\ \Xi'.\ (\forall\, i{<}m.\ \Xi\ i = \Xi'\ i) \longrightarrow Q\ \Xi = Q\ \Xi'$
then obtain *max* **where** *max*: $(\forall\, i{<}max.\ \Xi\ i = \Xi'\ i) \longrightarrow Q\ \Xi = Q\ \Xi'$ **for** $\Xi\ \Xi'$
by *blast*
have $(\forall\, i{<}max.\ \Xi\ i = \Xi'\ i) \longrightarrow (\forall\, a.\ Q\ (\Xi(n := a))) = (\forall\, a.\ Q\ (\Xi'(n := a)))$ **for** $\Xi\ \Xi'$
using $max[of\ \Xi(n := \text{-})\ \Xi'(n := \text{-})]$ **by** *force*
thus $\exists\, m.\ \forall \Xi\ \Xi'.\ (\forall\, i{<}m.\ \Xi\ i = \Xi'\ i) \longrightarrow (\forall\, a.\ Q\ (\Xi(n := a))) = (\forall\, a.\ Q\ (\Xi'(n := a)))$
by *blast*
qed

— Renaming variables in any way (not necessarily permuting them) preserves the property of being a set-theoretically definable predicate.
lemma *transform-variables*: **assumes** *SetFormulaPredicate P*
shows *SetFormulaPredicate* $(\lambda\ \Xi.\ P\ (\lambda\ n.\ \Xi(f\ n)))$
using *assms*
proof(*induction arbitrary*: *f rule*: *SetFormulaPredicate.inducts*)
case (*SFP-mem m n*)
then show *?case*
using *set-signature.SFP-mem* **by** *blast*
next
case (*SFP-eq m n*)
then show *?case*
using *set-signature.SFP-eq* **by** *blast*
next
case (*SFP-neg P*)
then show *?case*
using *SetFormulaPredicate.SFP-neg* **by** *simp*

next
  case (*SFP-disj P Q*)
  then show *?case* **using** *SetFormulaPredicate.SFP-disj* **by** *simp*
next
  case (*SFP-all P n*)
  then show *?case*
  proof−
    obtain k **where** k: $(\forall i < k.\ \Xi\ i = \Xi'\ i) \Longrightarrow P\ \Xi = P\ \Xi'$ **for** $\Xi\ \Xi' :: nat \Rightarrow 'a$
      using *bounded-free*[*OF* ‹*SetFormulaPredicate P*›] **by** *blast*
    obtain M **where** *M-bound*: $(\forall\ b < k.\ f\ b < M)$
    using *finite-imageI*[*OF finite-Collect-less-nat*, *of f k*, *unfolded finite-nat-set-iff-bounded*]
      by *blast*
    let $?M = Suc\ (M + n)$
    let $?Q = \lambda a\ \Xi.\ P\ ((\lambda b.\ \Xi\ (f\ b))(n{:=}a))$
    have M: $(\forall i < ?M.\ \Xi\ i = \Xi'\ i) \Longrightarrow ?Q\ a\ \Xi = ?Q\ a\ \Xi'$ **for** $\Xi\ \Xi' :: nat \Rightarrow 'a$
and a
      by (*rule k*[*of* $(\lambda b.\ \Xi\ (f\ b))(n := a)$ $(\lambda b.\ \Xi'\ (f\ b))(n := a)$]) (*use M-bound* **in**
auto)
    have *fsimp*: $(\lambda b.\ (\Xi(p := a))\ ((f(n := p))\ b)) = (\lambda b.\ (\Xi(p := a))\ (f\ b))(n{:=}a)$
for $\Xi :: nat \Rightarrow 'a$ **and** $a\ p$
      by *auto*
    have *M-irrelevant*: $P\ ((\lambda b.\ (\Xi(?M := a))\ (f\ b))(n := a)) = P\ ((\lambda b.\ \Xi\ (f\ b))(n := a))$ **for** $\Xi\ a$
      by (*rule M*) *simp*
    show *?thesis*
      using *set-signature.SFP-all*[*OF SFP-all*(*2*)[*of f*(*n*:=*?M*)], *of ?M*]
      unfolding *fsimp M-irrelevant*.
  qed
qed

lemma *update-variable*[*intro*]: **assumes** *SetFormulaPredicate P*
  shows *SetFormulaPredicate* $(\lambda\ \Xi.\ P(\Xi(n{:=}\ \Xi\ m)))$
proof−
  have *aux*: $(\lambda\ a.\ \Xi\ ((id(n{:=}m))\ a)) = (\Xi\ (n{:=}\Xi\ m))$ **for** $\Xi :: nat \Rightarrow 'a$
    by *force*
  show *?thesis*
    by (*rule transform-variables*[*OF assms*, *of id* (*n*:=*m*), *unfolded aux*])
qed

lemma *swap-variables*: **assumes** *SetFormulaPredicate P*
  shows *SetFormulaPredicate* $(\lambda\ \Xi.\ P\ (\Xi(k := \Xi\ l,\ l := \Xi\ k)))$
proof−
  have $(\lambda n.\ \Xi\ ((id(k := l,\ l := k))\ n)) = \Xi(k := \Xi\ l,\ l := \Xi\ k)$ **for** $\Xi :: nat \Rightarrow 'a$
    by *fastforce*
  from *transform-variables*[*OF assms*, *of id*(*k*:=*l*,*l*:=*k*), *unfolded this*]
  show *?thesis*.
qed

A rule allowing to get rid of quantifiers when proving that a predicate is a

SetFormulaPredicate

lemma *sfp-all-rule4* [*SFP-rules*]: **assumes** $\bigwedge$ *m*. *SetFormulaPredicate* (λ Ξ. *Q* (Ξ *m*) (Ξ *k1*) (Ξ *k2*) (Ξ *k3*) (Ξ *k4*))
shows *SetFormulaPredicate* (λ Ξ. $\forall$ *y*. *Q y* (Ξ *k1*) (Ξ *k2*) (Ξ *k3*) (Ξ *k4*))
using *SFP-all*[*OF assms*, *of k1+k2+k3+k4+1 k1+k2+k3+k4+1*] **by** *simp*

named-theorems *logsimps*
lemmas[*logsimps*] = *not-not*
lemma *reduce-ex* [*logsimps*]: ($\lambda\Xi$. ($\exists$ *z*. *Q z* Ξ)) = ($\lambda\Xi$. $\neg$ ($\forall$ *z*. $\neg$ (*Q z* Ξ)))
by *blast*
lemma *reduce-and* [*logsimps*]: (λ Ξ. (*P* Ξ) $\wedge$ (*Q* Ξ)) = (λ Ξ. ($\neg$ (($\neg$ (*P* Ξ)) $\vee$ ($\neg$ (*Q* Ξ)))))
by *blast*
lemma *reduce-imp*[*logsimps*]: (λ Ξ. (*P* Ξ) $\longrightarrow$ (*Q* Ξ)) = (λ Ξ. ($\neg$ (*P* Ξ) $\vee$ (*Q* Ξ)))
by *blast*
lemma *reduce-iff*[*logsimps*]: (λ Ξ. (*P* Ξ) $\longleftrightarrow$ (*Q* Ξ)) = ($\lambda\Xi$. $\neg$ ($\neg$ ($\neg$ *P* Ξ $\vee$ *Q* Ξ) $\vee$ $\neg$ ($\neg$ *Q* Ξ $\vee$ *P* Ξ)))
unfolding *iff-conv-conj-imp* **unfolding** *logsimps* **by** *blast*

lemma *SFP-const*[*intro*,*simp*]: *SetFormulaPredicate* ($\lambda\Xi$. *b*)
by (*induct b*) (*use SFP-eq*[*of 0 0*] *SFP-neg*[*OF SFP-eq*[*of 0 0*]] **in** *force*)+

lemma *SFP-ex*: **assumes** *SetFormulaPredicate P* **shows** *SetFormulaPredicate* (λ Ξ. ($\exists$ *z*. *P* (Ξ (*m* := *z*))))
unfolding *logsimps* **by** (*rule SFP-neg*, *rule SFP-all*, *rule SFP-neg*, *fact*)

lemma *SFP-replace*: **assumes** *SetFormulaPredicate P* **and** *fresh-m*: $\forall$ Ξ Ξ'. ($\forall$ *i*<*m*. Ξ *i* = Ξ' *i*) $\longrightarrow$ *P* Ξ = *P* Ξ'
shows *SetFormulaPredicate*($\lambda\Xi$. $\exists$ *z*. $\forall$ *v*. (*v* ε *z*) = ($\exists$ *u*. *u* ε Ξ *m* $\wedge$ *P* (Ξ (*0*:=*u*,*1*:=*v*))))
proof–
let *?f* = *id*(*0*:= *m*+*1*, *1*:= *m*+*2*)
let *?P* = λ Ξ. *P* (Ξ(*0* := Ξ(*m*+*1*),*1*:=Ξ(*m*+*2*)))
have *idsimp*: (λ*n*. Ξ (*?f n*)) = Ξ (*0*:= Ξ(*m*+*1*),*1* :=Ξ(*m*+*2*)) **for** Ξ :: *nat* $\Rightarrow$ *'a* **by** *fastforce*
have *Psimp*: *P* (Ξ(*m* + *3* := *z*, *m* + *2* := *v*, *m* + *1* := *u*, *0* := (Ξ(*m* + *3* := *z*, *m* + *2* := *v*, *m* + *1* := *u*)) (*m* + *1*),*1* := (Ξ(*m* + *3* := *z*, *m* + *2* := *v*, *m* + *1* := *u*)) (*m* + *2*)))
= *P* (Ξ (*0*:=*u*,*1*:=*v*)) **for** Ξ *v u z*
by (*rule fresh-m*[*rule-format*]) *force*
have *sfp*: *SetFormulaPredicate ?P*
using *transform-variables*[*OF assms*(*1*), *of ?f*] **unfolding** *idsimp*.
have *SetFormulaPredicate* ($\lambda\Xi$. $\neg$ Ξ (*m*+*1*) ε Ξ *m* $\vee$ $\neg$ *?P* Ξ)
by (*rule SFP-rules*)+ *fact*
from *SFP-all*[*OF this*, *of m*+*1*, *simplified fun-upd-same fun-upd-other*]
have *SetFormulaPredicate* ($\lambda\Xi$. $\forall$ *z*. $\neg$ *z* ε Ξ *m* $\vee$ $\neg$ *?P* (Ξ(*m* + *1* := *z*)))
unfolding *logsimps* **by** *simp*
have *SetFormulaPredicate* (λ Ξ. (Ξ (*m*+*2*) ε Ξ (*m*+*3*)) = ($\exists$ *u*. *u* ε Ξ *m* $\wedge$ *?P* (Ξ(*m*+*1* := *u*))))

unfolding *logsimps* **by** (*rule SFP-rules* | *fact*)+
from *SFP-all*[*OF this*, *of m+2*, *simplified fun-upd-same fun-upd-other*]
have *SetFormulaPredicate* ($\lambda\Xi.\ \forall\, a.\ (a\ \varepsilon\ \Xi\ (m + 3)) = (\exists\, u.\ u\ \varepsilon\ \Xi\ m \wedge\ ?P\ (\Xi(m + 2 := a,\ m + 1 := u))))$ **by** *simp*
note *aux* = *SFP-all*[*OF SFP-neg*[*OF this*], *of m+3*, *simplified fun-upd-same fun-upd-other*]
have *SetFormulaPredicate*
($\lambda\Xi.\ \exists\ z.\ (\forall\, v.\ (v\ \varepsilon\ z) = (\exists\, u.\ u\ \varepsilon\ \Xi\ m \wedge\ ?P\ (\Xi(m + 3 := z,\ m + 2 := v,\ m + 1 := u)))))$
unfolding *logsimps* **using** *SFP-neg*[*OF aux*[*unfolded logsimps*]] **by** *simp*
then show *SetFormulaPredicate* ($\lambda\Xi.\ \exists\, z.\ \forall\, v.\ (v\ \varepsilon\ z) = (\exists\, u.\ u\ \varepsilon\ \Xi\ m \wedge P\ (\Xi(0 := u,\ 1 := v))))$
unfolding *Psimp*.
qed

A (binary) relation is set-theoretically definable if the predicate it defines by assigning values to variables x_0 and x_1 is set-theoretically definable. The value of the relation therefore cannot depend on other variables than x_0 and x_1. In other words, if a formula φ defining a set-theoretically definable relation contains a free variable x_k, $1 < k$, then it is equivalent to $\forall\ x_k.\ \varphi$

definition *SetRelation* :: ($'a \Rightarrow\ 'a \Rightarrow bool) \Rightarrow bool$
where
SetRelation P $\equiv$ *SetFormulaPredicate* ($\lambda\ \Xi.\ P\ (\Xi\ 0)\ (\Xi\ 1)$)

A set-theoretically definable property is defined similarly.

definition *SetProperty* :: ($'a \Rightarrow\ bool) \Rightarrow bool$
where
SetProperty P $\equiv$ *SetFormulaPredicate* ($\lambda\ \Xi.\ P\ (\Xi\ 0)$)

Auxiliary proofs for specific useful set-relations and set-properties

lemma *SR-neg*[*simp*, *intro*]: *SetRelation P* $\Longrightarrow$ *SetRelation* ($\lambda\ x\ y.\ \neg\ P\ x\ y$)
unfolding *SetRelation-def* **by** (*rule SFP-rules*)

lemma *SR-sym*[*simp*, *intro*]: **assumes** *SetRelation P* **shows** *SetRelation* ($\lambda\ x\ y.\ P\ y\ x$)
using *swap-variables*[*OF assms*[*unfolded SetRelation-def*], *of 0 1*, *simplified fun-upd-same fun-upd-other*]
unfolding *SetRelation-def*.

lemma *SR-neq*[*simp*]: *SetRelation* ($\neq$)
unfolding *SetRelation-def* **using** *SFP-eq SFP-neg* **by** *blast*

lemma *SP-const*[*simp*]: *SetProperty* ($\lambda\ x.\ b$)
unfolding *SetProperty-def* **using** *SFP-const*.

lemma *SP-neg*[*simp*, *intro*]: *SetProperty P* $\Longrightarrow$ *SetProperty* ($\lambda\ x.\ \neg\ P\ x$)
unfolding *SetProperty-def* **using** *SFP-neg*.

lemma *SP-tarski*[*simp*]: *SetProperty tarski-fin*
  unfolding *SetProperty-def logsimps set-defs* **by** (*rule SFP-rules*)+

lemma *SP-setsuc*[*simp*]: *SetProperty is-sucM*
  unfolding *SetProperty-def logsimps set-defs* **by** (*rule SFP-rules*)+

lemma *SP-nat*[*simp*]: *SetProperty natM*
  unfolding *SetProperty-def logsimps set-defs* **by** (*rule SFP-rules*)+

lemma *empty-mem-ord'*: **assumes** *ordM x* ∃ *z. z ε x*
  shows ∃ *emp.* (∀ *u.* ¬ *u ε emp*) ∧ *emp ε x*
proof–
  obtain *u* **where** *u ε x* ∀ *y. y ε u* ⟶ ¬ *y ε x*
    using ‹*ordM x*› ‹∃ *z. z ε x*› *ordM-def regular-def subsetM-refl* **by** *metis*
  have *y ε u* ⟶ *y ε x* **for** *y*
    using ‹*ordM x*› ‹*u ε x*› *ordM-D1 subsetM-def* **by** *blast*
  hence ¬ *y ε u* **for** *y*
    using ‹∀ *y. y ε u* ⟶ ¬ *y ε x*› **by** *simp*
  then show *?thesis*
    using ‹*u ε x*› **by** *blast*
qed

end

1.3 Axiomatizations of hereditarily finite sets

1.3.1 Finiteness axioms

The aim of each of the axioms is to guarantee that each element of the domain will be finite.

Usual axiom: there is no inductive set.

locale *L-fin* = *set-signature* +
  assumes *fin*: ¬ (∃ *x.* ∅ *ε x* ∧ (∀ *y. y ε x* ⟶ *setsucM y y ε x*))

Induction schema for set formulas, a.k.a. set induction schema or just set induction. Not to be confused with epsilon induction.

locale *L-setind* = *set-signature* +
  assumes *setind*: ⋀ *P. SetFormulaPredicate P* ⟹
    P(Ξ(*0*:= ∅)) ⟶ (∀ *x y. P*(Ξ(*0*:= *x*)) ⟶ *P* (Ξ(*0*:= *setsucM x y*))) ⟶ (∀ *x.* *P*(Ξ(*0*:= *x*)))
begin

lemma *setind-SP*: **assumes** *SetProperty P* **and** *P* ∅ **and** *step*: ∀ *x y. P x* ⟶ *P* (*setsucM x y*)
  shows *P x*
  using *setind*[*of* λ Ξ. *P* (Ξ *0*), *folded SetProperty-def*, *OF assms*(*1*), *simplified*] ‹*P* ∅› *step* **by** *blast*

lemma *setind-var*: **assumes** *SetFormulaPredicate P* $P(\Xi(n{:=}\ \emptyset))$ **and** *step*: $\forall\ x\ y.\ P(\Xi(n{:=}\ x)) \longrightarrow P\ (\Xi(n{:=}\ setsucM\ x\ y))$
shows $P(\Xi(n{:=}\ x))$
proof–
from *bounded-free*[*OF* ‹*SetFormulaPredicate P*›]
obtain *m* **where** *m-def*: $P\ \Xi = P\ \Xi'$ **if** $\forall i{<}m.\ \Xi\ i = \Xi'\ i$ **for** $\Xi\ \Xi'$
by *blast*
let $?f = id(0 := Suc\ (n + m),\ n{:=}\ 0)$
let $?Q = \lambda\ \Xi.\ (P\ (\lambda\ b.\ \Xi\ (?f\ \ b)))$
let $?X = \Xi(Suc\ (n + m){:=}\ \Xi\ 0)$
have *sfpq*: *SetFormulaPredicate ?Q*
using *transform-variables*[*OF* ‹*SetFormulaPredicate P*›] **by** *simp*
have *small*: $\forall\ \ i < m.\ (?X\ (0 := u))\ (?f\ i) = (\Xi(n := u))\ i$ **for** *u*
by *auto*
have *equiv*: $(?Q\ (?X\ (0 := u))) \longleftrightarrow (P\ (\Xi(n := u)))$ **for** *u*
by (*rule m-def*[*of* $\lambda\ b.\ (?X\ (0 := u))(?f\ b)\ \Xi(n := u)$]) (*rule small*)
show $P(\Xi(n{:=}\ x))$
unfolding *equiv*[*symmetric*]
by (*rule setind*[*rule-format*, *OF sfpq*, *of ?X*], *unfold equiv*)
(*use assms* **in** *blast*)+
qed

end

locale *L-dedekind* = *set-signature* +
assumes *dedekind*: $\forall\ \ x.$ *dedekind-fin x*

locale *L-tarski* = *set-signature* +
assumes *tarski*: $\forall\ \ y.$ *tarski-fin y*

1.3.2 Extensionality

locale *L-setext* = *set-signature* +
assumes *setext*: $x = y \longleftrightarrow (\forall\ \ z.\ z\ \varepsilon\ x \longleftrightarrow z\ \varepsilon\ y)$

begin

set in HOL is defined as a class. Therefore, the following comprehension is an axiom.

lemma $x \in \{u.\ P\ u\} \longleftrightarrow P\ x$
using *mem-Collect-eq*.

In case of (our) M-sets, only uniqueness, not existence is guaranteed by extensionality.

lemma *collect-def'*:
assumes $\forall\ \ u.\ (u\ \varepsilon\ w \longleftrightarrow Q\ u)$
shows *collectM* $Q = w$
proof–

let *?P* = λ *z*. ($\forall$ *u*. *u* ε *z* $\longleftrightarrow$ *Q u*)
have *unique*: $z' = w$ **if** *?P* z' **for** z'
unfolding *setext*[*rule-format*, *of* z' *w*]
using *that assms* **by** *blast*
then show *?thesis*
using *theI*[*of ?P w*, *OF assms unique*] **unfolding** *collectM-def* **by** *blast*
qed

lemma *collect-def-ex*: — A formulation useful in proofs
assumes $\exists$ *w*. $\forall$ *u* . (*u* ε *w* $\longleftrightarrow$ *Q u*)
shows *u* ε *collectM Q* $\longleftrightarrow$ *Q u*
using *assms collect-def′ assms* **by** *auto*

lemma *proper-subset-diff*[*simp*] : $y \subset_M x \Longrightarrow \exists$ *z*. *z* ε *x* $\wedge$ $\neg$ *z* ε *y*
unfolding *proper-subsetM-def setext* **by** *blast*

lemma *subsetM-antisym*: $y \subseteq_M x \Longrightarrow x \subseteq_M y \Longrightarrow x = y$
unfolding *subsetM-def setext* **by** *blast*

lemma *proper-subsetM-trans*[*simp*]: **assumes** $x \subset_M y$ $y \subset_M z$ **shows** $x \subset_M z$
proof (*unfold proper-subsetM-def*, *rule conjI*)
show ($\forall$ *u*. *u* ε *x* $\longrightarrow$ *u* ε *z*)
using *assms proper-subsetM-def* **by** *simp*
show $x \neq z$
by (*rule notI*) (*use assms proper-subsetM-def setext*[*of z y*] **in** *auto*)
qed

lemma *emptyset-mem-ord*: **assumes** *ordM x* $\exists$ *z*. *z* ε *x* **shows** $\emptyset$ ε *x*
using *empty-mem-ord′*[*OF assms*] *collect-def′*[*of - λ x. x $\neq$ x*] *emptysetM-def* **by** *force*

end

1.3.3 Empty set

locale *L-empty* = *set-signature* +
assumes *empty*: $\exists$ *x*. $\forall$ *y*. ($\neg$ *y* ε *x*)

Since the empty set is defined*collectM*, and hence Hilbert's *The*, nothing useful can be said about $\emptyset$ without extensionality.

locale *L-setext-empty* = *L-setext* + *L-empty*

begin

— Here we can already prove that $\emptyset$, has the intended meaning.
lemma *empty-is-empty*[*simp*]: $\forall$ *y*. ($\neg$ *y* ε $\emptyset$)
proof–
have *ex*: $\exists$ *x*. $\forall$ *z*. *z* ε *x* $\longleftrightarrow$ $z \neq z$
using *empty* **by** *auto*

from *collect-def-ex*[*OF this*, *folded emptysetM-def*]
show *?thesis*
unfolding *emptysetM-def collectM-def* **by** *auto*
qed

lemma *emptyset-def′*[*set-defs*]: $x = \emptyset \longleftrightarrow (\forall\ u.\ \neg\ u\ \varepsilon\ x)$
unfolding *setext* **by** *simp*

lemma *empty-mem-false* [*set-defs*]: $u\ \varepsilon\ \emptyset \longleftrightarrow$ *False*
using *emptyset-def′* **by** *simp*

lemma *setsuc-empty-sing*: *setsucM* $\emptyset\ x = \{x\}_M$
unfolding *singletonM-def setsucM-def empty-mem-false* **by** *simp*

lemma *SFP-empty-n* [*simp*]: *SetFormulaPredicate* $(\lambda\Xi.\ \Xi\ n = \emptyset)$
unfolding *SetProperty-def logsimps set-defs* **by** (*rule SFP-rules*)+

lemma *SP-empty*[*simp*]: *SetProperty* $(\lambda x.\ x = \emptyset)$
unfolding *SetProperty-def* **by** *simp*

lemma *SFP-empty-n′* [*simp*]: *SetFormulaPredicate* $(\lambda\Xi.\ \forall\, u.\ \neg\ u\ \varepsilon\ \Xi\ n)$
using *SFP-empty-n emptyset-def′* **by** *simp*

lemma *empty-dedekind*[*simp*]: **shows** *dedekind-fin* $\emptyset$
unfolding *dedekind-fin-def set-defs* **by** *auto*

lemma *subset-of-empty*[*simp*]: $u \subseteq_M \emptyset \longleftrightarrow u = \emptyset$
unfolding *subsetM-def emptyset-def′* **by** *simp*

lemma *empty-tarski*[*simp*]: **shows** *tarski-fin* $\emptyset$
unfolding *tarski-fin-def proper-subsetM-def* **by** *auto*

lemma *nemp-setI*[*intro*]: $x\ \varepsilon\ y \Longrightarrow y \neq \emptyset$
by *force*

lemma *nemp-setI-ex*: $\exists\ x.\ x\ \varepsilon\ y \Longrightarrow y \neq \emptyset$
by *force*

lemma *empty-is-subset*: $\emptyset \subseteq_M A$
unfolding *subsetM-def* **by** *force*

lemma *empty-regular*[*simp*]: *regular* $\emptyset$
unfolding *regular-def* **by** *simp*

lemma *empty-trans*[*simp*]: *transM* $\emptyset$
unfolding *transM-def* **by** *simp*

lemma *emp-natM*[*simp*]: *natM* $\emptyset$
by (*simp add*: *natM-def ordM-def epschain-def*)

lemma *binunion-emp*[*simp*]: $\emptyset \cup_M\ t = t\ t \cup_M \emptyset = t$
  unfolding *setext*[*of - t*] *binunionM-def* **using** *empty-is-empty*
    collect-def'[*of t* λ *u*. *u* ε *t*] **by** *simp-all*

end

1.3.4 Set pair

Pairing is a simple set construction which can be easily obtained by set successor and empty set. It is interesting on its own, in particular in a context without the empty set axiom.

locale *L-pair* = *set-signature* +
  assumes *pair*: $\forall\ x\ y.\ \exists\ z.\ \forall\ v.\ v\ \varepsilon\ z \longleftrightarrow v = x \lor v = y$

locale *L-setext-pair* = *L-setext* + *L-pair*

begin

lemma *singleton-def'*[*set-defs*]: $u\ \varepsilon\ \{y\}_M \longleftrightarrow\ u = y$
proof–
  have $\exists\ x.\ \forall\ u.\ u\ \varepsilon\ x \longleftrightarrow\ u = y$
    using *pair* **by** *meson*
  from *collect-def-ex*[*OF this*[*rule-format*], *folded singletonM-def*]
  show *?thesis*.
qed

lemma *singleton-eq*[*set-defs*]: $u = \{y\}_M \longleftrightarrow\ (\forall\ v.\ v\ \varepsilon\ u \longleftrightarrow v = y)$
  unfolding *setext set-defs* **by** *blast*

lemma *pair-def'*[*set-defs*]: $u\ \varepsilon\ \{x,y\}_M \longleftrightarrow u = x \lor u = y$
  using *collect-def-ex*[*OF pair*[*rule-format*], *folded pairM-def*].

lemma *pair-eq* [*set-defs*]: $w = \{x,y\}_M \longleftrightarrow (\forall\ u.\ (u\ \varepsilon\ w) \longleftrightarrow\ u = x \lor u = y)$
  unfolding *setext*[*of w*] *set-defs* **by** *simp*

lemma *regular-not-self-mem*: **assumes** *regular x* **shows** $\neg\ x\ \varepsilon\ x$
proof
  assume $x\ \varepsilon\ x$
  with *assms*[*unfolded regular-def*, *rule-format*, *of* $\{x\}_M$]
  show *False*
    unfolding *subsetM-def singleton-def'* **by** *blast*
qed

lemma *ordered-pair-unique*[*simp*]: $\langle u,v\rangle = \langle u',v'\rangle \longleftrightarrow u = u' \land v = v'$
  unfolding *setext*[*of* $\{$-,-$\}_M$] *ordered-pairM-def* **using** *pair-def' singleton-def'* **by** *metis*

lemma *sing-eq-iff*: $\{a\}_M = \{b\}_M \longleftrightarrow a = b$

unfolding *setext*[*of* {-}$_M$] *singleton-def′* **by** *metis*

lemma *sing-eq-pair-iff*: $\{a\}_M = \{b,c\}_M \longleftrightarrow a = b \wedge b = c$
unfolding *setext*[*of* {-}$_M$] *pair-def′ singleton-def′* **by** *metis*

lemma *sing-eq-pair-iff′*: $\{b,c\}_M = \{a\}_M \longleftrightarrow a = b \wedge b = c$
using *sing-eq-pair-iff* **by** *metis*

lemma *SFP-mem-ordered-pair*[*SFP-rules*]: *SetFormulaPredicate* (λ Ξ. Ξ k ε $\langle\Xi$ l, Ξ $m\rangle$)
unfolding *ordered-pairM-def set-defs logsimps* **by** (*rule SFP-rules*)+

lemma *SFP-eq-ordered-pair* [*SFP-rules*]: *SetFormulaPredicate* (λ Ξ. Ξ k = $\langle\Xi$ l,Ξ $m\rangle$)
unfolding *setext logsimps* **by** (*rule SFP-rules*)+

lemma *SFP-ordered-pair-mem*[*SFP-rules*]: *SetFormulaPredicate* ($\lambda\Xi$. $\langle\Xi$ k,Ξ $l\rangle$ ε Ξ m)
proof–
have *SetFormulaPredicate* ($\lambda\Xi$. $\exists$ u. $u = \langle\Xi\ k,\Xi\ l\rangle \wedge u\ \varepsilon\ \Xi\ m$)
unfolding *setext logsimps* **by** (*rule SFP-rules*)+
thus *?thesis*
by *simp*
qed

end

1.3.5 Set successor

Also known as adjunction. Enables constructing sets in steps. This axiom plays an important role in the fragment of set theory axiomatized by extensionality, empty set and set successor (see *L-setext-empty-setsuc* below), mutually interpretable with Robinson's arithmetic Q.

locale *L-setsuc* = *set-signature* +
assumes *setsuc*: $\forall$ x y. $\exists$ z. $\forall$ u. $u\ \varepsilon\ z \longleftrightarrow u\ \varepsilon\ x \vee u = y$

locale *L-setext-setsuc* = *L-setext* + *L-setsuc*
— some statements do not need the empty set

begin

lemma *setsuc-def′*[*set-defs*, *simp*]: $u\ \varepsilon$ *setsucM* x $y \longleftrightarrow (u\ \varepsilon\ x \vee u = y)$
using *collect-def-ex*[*OF setsuc*[*rule-format*]] **unfolding** *setsucM-def*.

lemma *setsuc-triv*[*simp*]: $y\ \varepsilon\ x \Longrightarrow$ *setsucM* x $y = x$
unfolding *setext*[*of* - x] *setsuc-def′* **by** *blast*

lemma *is-suc-def'*: *is-sucM x* $\longleftrightarrow$ ($\exists$ *y*. *x* = *setsucM y y*)
 unfolding *is-sucM-def setsuc-def' setext*..

lemma *trans-suc-trans*: *transM x* $\Longrightarrow$ *transM* (*setsucM x x*)
 unfolding *transM-def set-defs* **by** *blast*

lemma *epschain-suc-epschain*: *epschain x* $\Longrightarrow$ *epschain* (*setsucM x x*)
 unfolding *epschain-def* **using** *trans-suc-trans*
 using *setsuc-def'* **by** *auto*

lemma *ord-pred-fun*: **assumes** *ordM x ordM y setsucM x x* = *setsucM y y*
 shows *x* = *y*
 using *assms* **unfolding** *setext*[*of x*] *setext*[*of setsucM x x*] *set-defs* **by** *metis*

lemma *SFP-setsuc-mem*[*SFP-rules*]: *SetFormulaPredicate* ($\lambda\Xi$. Ξ *k* ε *setsucM* (Ξ *l*) (Ξ *m*))
 unfolding *setsuc-def'* **by** (*rule SFP-rules*)+

lemma *SFP-setsuc-eq*[*SFP-rules*]: *SetFormulaPredicate* ($\lambda\Xi$. Ξ *k* = *setsucM* (Ξ *l*) (Ξ *m*))
 unfolding *setsuc-def' setext logsimps* **by** (*rule SFP-rules*)+

end

locale *L-empty-setsuc* = *L-empty* + *L-setsuc*
— some statements do not need extensionality

begin

sublocale *L-pair*
proof (*unfold-locales*, *rule allI*, *rule allI*)
 fix *x y*
 obtain *empty* **where** *emp*: $\forall$ *u*. $\neg$ *u* ε *empty*
  using *empty* **by** *blast*
 obtain *sing* **where** $\forall$ *u*. *u* ε *sing* $\longleftrightarrow$ *u* = *x*
  using *setsuc*[*rule-format*, *of empty x*] *emp* **by** *blast*
 then obtain *pair* **where** $\forall$ *v*. (*v* ε *pair*) = (*v* = *x* $\vee$ *v* = *y*)
  using *setsuc*[*rule-format*, *of sing y*] **by** *blast*
 then show $\exists$ *z*. $\forall$ *v*. (*v* ε *z*) = (*v* = *x* $\vee$ *v* = *y*)
  by *blast*
qed

lemma *singleton-setsuc*: **shows** $\exists$ *y*. $\forall$ *u*. *u* ε *y* $\longleftrightarrow$ *u* = *z*
proof–
 obtain *emp* **where** $\forall$ *u*. $\neg$ *u* ε *emp*
  using *empty* **by** *blast*
 then show *?thesis*
  using *setsuc*[*rule-format*, *of emp z*] **by** *blast*
qed

lemma *pair-setsuc*: **shows** $\exists\ y.\ \forall\ u.\ u\ \varepsilon\ y \longleftrightarrow u = z_1 \lor u = z_2$
proof–
  obtain *z1* **where** $\forall\ u.\ u\ \varepsilon\ z1 \longleftrightarrow u = z_1$
    using *singleton-setsuc* **by** *blast*
  then show *?thesis*
    using *setsuc*[*rule-format*, *of z1* z_2] **by** *simp*
qed

lemma *triple-setsuc*: **shows** $\exists\ y.\ \forall\ u.\ u\ \varepsilon\ y \longleftrightarrow u = z_1 \lor u = z_2 \lor u = z_3$
proof–
  obtain *z2* **where** $\forall\ u.\ u\ \varepsilon\ z2 \longleftrightarrow u = z_1 \lor u = z_2$
    using *pair-setsuc* **by** *blast*
  then show *?thesis*
    using *setsuc*[*rule-format*, *of z2* z_3] **by** *simp*
qed

lemma *regular-antisym-mem*: **assumes** *regular x* $z_1\ \varepsilon\ x\ z_2\ \varepsilon\ x$ **shows** $\neg\ (z_1\ \varepsilon\ z_2 \land z_2\ \varepsilon\ z_1)$
proof
  obtain *pair* **where** *pair*: $\forall\ u.\ u\ \varepsilon\ pair \longleftrightarrow u = z_1 \lor u = z_2$
    using *pair-setsuc* **by** *blast*
  assume $z_1\ \varepsilon\ z_2 \land z_2\ \varepsilon\ z_1$
  with ‹*regular x*›[*unfolded regular-def*, *rule-format*, *of pair*]
  show *False*
    unfolding *subsetM-def* **using** ‹$z_1\ \varepsilon\ x$› ‹$z_2\ \varepsilon\ x$› *pair* **by** *auto*
qed

lemma *regular-antisym2-mem*: **assumes** *regular x* $z_1\ \varepsilon\ x\ z_2\ \varepsilon\ x\ z_3\ \varepsilon\ x$ **shows** $\neg\ (z_1\ \varepsilon\ z_2 \land z_2\ \varepsilon\ z_3 \land z_3\ \varepsilon\ z_1)$
proof
  obtain *triple* **where** *triple*: $\forall\ u.\ u\ \varepsilon\ triple \longleftrightarrow u = z_1 \lor u = z_2 \lor u = z_3$
    using *triple-setsuc* **by** *blast*
  assume $z_1\ \varepsilon\ z_2 \land z_2\ \varepsilon\ z_3 \land z_3\ \varepsilon\ z_1$
  with ‹*regular x*›[*unfolded regular-def*, *rule-format*, *of triple*]
  show *False*
    unfolding *subsetM-def* **using** ‹$z_1\ \varepsilon\ x$› ‹$z_2\ \varepsilon\ x$› ‹$z_3\ \varepsilon\ x$› *triple* **by** *auto*
qed

lemma *ord-mem-ord*: **assumes** *ordM v u* ε *v* **shows** *ordM u*
  unfolding *ordM-def epschain-def conj-assoc*[*symmetric*]
proof (*rule conjI*, *rule conjI*)
  have *regular v u* $\subseteq_M$ *v*
    using ‹*ordM v*› *ordM-D1* ‹$u\ \varepsilon\ v$› *ordM-def* **by** *blast+*
  then show *regular u*
    unfolding *regular-def* **using** *subsetM-trans*[*OF* - ‹$u \subseteq_M v$›] **by** *blast*
  have $u \subseteq_M v$ *epschain v regular v transM v*
    using *assms* **unfolding** *natM-def ordM-def transM-def epschain-def* **by** *blast+*
  thus $\forall y\ z.\ y\ \varepsilon\ u \land z\ \varepsilon\ u \longrightarrow y\ \varepsilon\ z \lor y = z \lor z\ \varepsilon\ y$

```
    using ‹ordM v› unfolding ordM-def subsetM-def epschain-def by blast
  show transM u
    unfolding transM-def set-defs
  proof (rule allI, rule impI, rule allI, rule impI)
    fix z y
    assume y ε u z ε y
    hence z ε v y ε v
      using ‹u ε v› ‹transM v› unfolding transM-def set-defs by blast+
    have u ≠ z
     using regular-antisym-mem[OF ‹regular v› ‹y ε v› ‹z ε v›] ‹y ε u› ‹z ε y› by
blast
    have ¬ u ε z
      using regular-antisym2-mem[OF ‹regular v› ‹u ε v› ‹z ε v› ‹y ε v›] ‹y ε u›
‹z ε y› by blast
    show z ε u
      using ordM-D2[OF ‹ordM v› ‹u ε v› ‹z ε v›] ‹u ≠ z› ‹¬ u ε z› by blast
  qed
qed

end

locale L-setext-empty-setsuc = L-setext + L-empty + L-setsuc

begin

sublocale L-setext-empty
 by unfold-locales

sublocale L-setext-setsuc
  by unfold-locales

sublocale L-empty-setsuc
  by unfold-locales

sublocale L-setext-pair
  by unfold-locales

lemma empty-mem-ord: assumes ordM x x ≠ ∅ shows ∅ ε x
proof–
  have ∃ z. z ε x
    using ‹x ≠ ∅›  by (simp add: setext)
  then obtain u where u ε x ∀ y. y ε u ⟶ ¬ y ε x
    using ‹ordM x›  ordM-def regular-def subsetM-refl by metis
  have y ε u ⟶ y ε x for y
    using ‹ordM x› ‹u ε x› ordM-D1 subsetM-def by blast
  hence u = ∅
    using ‹∀ y. y ε u ⟶ ¬ y ε x›
    by (simp add: emptyset-def′)
  then show ∅ ε x
```

using ‹$u \varepsilon x$› **by** *blast*
qed

lemma *SP-injective* [*simp*]: *SetProperty* ($\lambda x.\ \forall\, a\ b\ c.\ \langle b,a\rangle\ \varepsilon\ x \wedge \langle c,a\rangle\ \varepsilon\ x \longrightarrow b = c$)
unfolding *set-defs logsimps SetProperty-def* **by** (*rule SFP-rules*)+

lemma *SP-unique-image* [*simp*]: *SetProperty* ($\lambda x.\ \forall\, a\ b\ c.\ \langle a,b\rangle\ \varepsilon\ x \wedge \langle a,c\rangle\ \varepsilon\ x \longrightarrow b = c$)
unfolding *set-defs logsimps SetProperty-def* **by** (*rule SFP-rules*)+

lemma *SFP-compose* [*simp,intro*]: *SetFormulaPredicate* ($\lambda\ \Xi.\ \exists\, a\ b\ c.\ \langle a,b\rangle\ \varepsilon\ \Xi\ k \wedge \langle b,c\rangle\ \varepsilon\ \Xi\ l \wedge \Xi\ m = \langle a,c\rangle$)
proof−
have *sfp*: *SetFormulaPredicate* ($\lambda\ \Xi.\ \langle \Xi\ (k+l+m+3),\Xi\ (k+l+m+4)\rangle\ \varepsilon\ \Xi\ k \wedge \langle \Xi\ (k+l+m+4),\Xi\ (k+l+m+5)\rangle\ \varepsilon\ \Xi\ l \wedge \Xi\ m = \langle \Xi\ (k+l+m+3),\ \Xi\ (k+l+m+5)\rangle$)
unfolding *logsimps* **by** (*rule SFP-rules*)+
show *?thesis*
using *SFP-ex*[*OF SFP-ex*[*OF SFP-ex*], *of - k+l+m+3 k+l+m+4 k+l+m+5*, *OF sfp*] **by** *simp*
qed

lemma *SP-function*[*simp*]: *SetProperty* (λ *x. functionM x*)
unfolding *SetProperty-def set-defs logsimps* **by** (*rule SFP-rules*)+

lemma *SP-one-one*[*simp*]: *SetProperty* (λ *x. one-oneM x*)
unfolding *SetProperty-def set-defs logsimps* **by** (*rule SFP-rules*)+

lemma *dom-SR* [*simp*]: *SetRelation* ($\lambda x\ u.\ \exists\, w.\ x = \langle u,w\rangle$)
unfolding *SetRelation-def logsimps* **by** (*rule SFP-rules*)+

lemma *rng-SR* [*simp*]: *SetRelation* ($\lambda x\ u.\ \exists\, w.\ x = \langle w,u\rangle$)
unfolding *SetRelation-def logsimps* **by** (*rule SFP-rules*)+

end

locale *L-binunion* = *set-signature* +
assumes *binunion*: $\forall\ x\ y.\ \exists\ z.\ \forall\ v.\ v\ \varepsilon\ z \longleftrightarrow (v\ \varepsilon\ x \vee v\ \varepsilon\ y)$

locale *L-setext-binunion* = *L-setext* + *L-binunion*

begin

lemma *binunion-def'*[*set-defs*]: $u\ \varepsilon\ x \cup_M y \longleftrightarrow u\ \varepsilon\ x \vee u\ \varepsilon\ y$
unfolding *binunionM-def*
using *collect-def-ex*[*OF binunion*[*rule-format*], *of u*]
by *force*

end

locale *L-setext-pair-binunion* = *L-setext* + *L-pair* + *L-binunion*
— A specific minimalist combination of axioms that guarantees the existence of both ordered pair and set successor

begin

sublocale *L-setext-binunion*
  by *unfold-locales*

sublocale *L-setext-pair*
  by *unfold-locales*

sublocale *L-setsuc*
proof (*unfold-locales*, (*rule allI*)+, *rule exI*, *rule allI*)
  show $u\ \varepsilon\ x \cup_M \{y\}_M \longleftrightarrow (u\ \varepsilon\ x \vee u = y)$ **for** $x\ y\ u$
    unfolding *binunion-def′ singleton-def′* **by** *blast*
qed

sublocale *L-setext-setsuc*
  by *unfold-locales*

lemma *SFP-ordered-pair-setsuc-eq*[*SFP-rules*]: *SetFormulaPredicate* $(\lambda\Xi.\ \Xi\ k = \langle \Xi\ l,\ setsucM\ (\Xi\ m)\ (\Xi\ n)\rangle)$
proof−
— An explicit proof avoiding too many quantifiers, beyond reach of $(\bigwedge m.\ SetFormulaPredicate\ (\lambda\Xi.\ ?Q\ (\Xi\ m)\ (\Xi\ ?k1.0)\ (\Xi\ ?k2.0)\ (\Xi\ ?k3.0)\ (\Xi\ ?k4.0))) \Longrightarrow SetFormulaPredicate\ (\lambda\Xi.\ \forall y.\ ?Q\ y\ (\Xi\ ?k1.0)\ (\Xi\ ?k2.0)\ (\Xi\ ?k3.0)\ (\Xi\ ?k4.0))$
   let $?Q = \lambda\ \Xi.\ \Xi\ (m+n+l+k+1) = setsucM\ (\Xi\ m)\ (\Xi\ n) \wedge \Xi\ k = \langle \Xi\ l, \Xi\ (m+n+l+k+1)\rangle$
  thm *SFP-ex*[*of ?Q m+n+l+k+1*, *simplified*]
  show *?thesis*
   by (*rule SFP-ex*[*of ?Q m+n+l+k+1*, *simplified*], *unfold logsimps*) (*rule SFP-rules*)+
qed

end

1.3.6 Regularity

locale *L-reg* = *set-signature* +
  assumes *reg*: $\forall\ x.\ (\exists\ y.\ y\ \varepsilon\ x) \longrightarrow (\exists\ y.\ y\ \varepsilon\ x \wedge (\forall\ v.\ \neg\ (v\ \varepsilon\ y \wedge v\ \varepsilon\ x)))$

begin

lemma *any-reg*[*simp*]: *regular x*
  using *reg* **unfolding** *regular-def* **by** *blast*

Note that $\neg\ x\ \varepsilon\ x$ cannot be proved without existence of a singleton.

end

lemma (**in** *set-signature*) *reg-all-regular-iff*:
  L-reg (ε) $\longleftrightarrow$ ($\forall$ *x*. *regular x*)
proof
  assume *L-reg* (ε)
  then interpret *L-reg* (ε)**.**
  show $\forall$ *x*. *regular x*
    using *reg* **unfolding** *regular-def* **by** *blast*
next
  assume *all-reg*: $\forall$ *x*. *regular x*
  show *L-reg* (ε)
    by *unfold-locales* (*meson all-reg regular-def set-signature.subsetM-refl*)
qed

Schema of regularity

locale *L-regsch* = *set-signature* +
  assumes *regsch*: *SetFormulaPredicate P* $\Longrightarrow$ ($\exists$ *x*. *P* (Ξ (*0*:=*x*))) $\longrightarrow$ ($\exists$ *x*. *P* (Ξ (*0*:=*x*)) $\wedge$ ($\forall$ *y*. *y* ε *x* $\longrightarrow$ $\neg$ *P* (Ξ (*0*:=*y*))))

begin

regsch is stronger than normal regularity in the absence of infinity axiom, since one cannot prove the existence of transitive supersets/closures. See *not-reg-setind-implies-regsch* in*ZFfin-regsch-model.thy*

sublocale *L-reg*
proof
  have *SFP*: *SetFormulaPredicate* ($\lambda\Xi$. Ξ *0* ε Ξ *1*)
    by (*rule SFP-rules*)
  show $\forall$ *x*. ($\exists$ *y*. *y* ε *x*) $\longrightarrow$ ($\exists$ *y*. *y* ε *x* $\wedge$ ($\forall$ *v*. $\neg$ (*v* ε *y* $\wedge$ *v* ε *x*)))
    using *regsch*[*OF SFP*, *unfolded fun-upd-apply*] **by** *force*
qed

lemma *regsch-epsind*:
  SetFormulaPredicate Q $\Longrightarrow$ ($\forall$ *x*. ($\forall$ *y*. (*y* ε *x* $\longrightarrow$ *Q* (Ξ (*0*:=*y*)))) $\longrightarrow$ *Q* (Ξ (*0*:=*x*))) $\longrightarrow$ ($\forall$ *x*. *Q* (Ξ (*0*:=*x*)))
  using *regsch*[*of* λ *x*. $\neg$ *Q x* Ξ] **using** *SFP-neg* **by** *blast*

lemma *regsch-mem-not-refl*: $\neg$ *u* ε *u*
proof
  assume *u* ε *u*
  hence *ex*: $\exists$ *x*. (*undefined*(*1* := *u*, *0* := *x*)) *0* = (*undefined*(*1* := *u*, *0* := *x*)) *1*
    by *force*
  have *sfp*: *SetFormulaPredicate* ($\lambda\Xi$. (Ξ *0*) = (Ξ *1*))
    by (*rule SFP-rules*)
  from *regsch*[*of* λ Ξ. (Ξ *0*) = (Ξ *1*), *rule-format*, *OF sfp ex*]
  show *False*
    using ‹*u* ε *u*› **by** *auto*
qed

end

Epsilon induction schema is logically equivalent to the regularity schema (that is, regardless of what ε means), see $((\exists x.\ ?P\ x) \longrightarrow (\exists x.\ ?B\ x \wedge ?P\ x)) = ((\forall x.\ ?B\ x \longrightarrow \neg\ ?P\ x) \longrightarrow (\forall x.\ \neg\ ?P\ x))$ above.

locale *L-epsind* = *set-signature* +
assumes *epsind*: $SetFormulaPredicate\ Q \Longrightarrow (\forall\ x.\ (\forall y.\ (y\ \varepsilon\ x \longrightarrow Q\ (\Xi\ (0{:=}y)))) \longrightarrow\ Q\ (\Xi\ (0{:=}x))) \longrightarrow\ (\forall\ x.\ Q\ (\Xi\ (0{:=}x)))$

begin

lemma *epsind-regsch*: **assumes** $SetFormulaPredicate\ P\ \exists\ x.\ P\ (\Xi\ (0{:=}x))$
shows $\exists\ x.\ P\ (\Xi\ (0{:=}x)) \wedge (\forall\ y.\ y\ \varepsilon\ x \longrightarrow \neg\ P\ (\Xi\ (0{:=}y)))$
proof–
have *SP*: $SetFormulaPredicate\ (\lambda\ x.\ \neg\ P\ x)$
using ‹*SetFormulaPredicate P*› **by** (*simp add*: *SFP-rules*)
show *?thesis*
unfolding *abstract-foundation-iff*[*of* $\lambda\ x.\ \forall\ y.\ y\ \varepsilon\ x \longrightarrow \neg\ P\ (\Xi(0{:=}y))\ \lambda\ x.\ \neg\ P\ (\Xi(0{:=}x))$, *unfolded not-not*]
using *epsind*[*OF SP*, *rule-format*, *of* Ξ] ‹$\exists\ x.\ P\ (\Xi\ (0{:=}x))$›
by *blast+*
qed

sublocale *L-regsch*
using *epsind-regsch* **by** *unfold-locales blast*

end

1.3.7 Separation

Separation is often an alternative to set successor. It allows one to construct "from above" many sets that set successor builds "from below".

locale *L-sep* = *set-signature* +
assumes *sep*: $SetFormulaPredicate\ P \Longrightarrow \forall x.\ \exists\ z.\ \forall\ u.\ (u\ \varepsilon\ z \longleftrightarrow\ u\ \varepsilon\ x \wedge P\ (\Xi(0{:=}\ u)))$

begin

sublocale *L-empty*
by *unfold-locales* (*use sep*[*OF SFP-const*[*of False*]] **in** *force*)

lemma *sep-var*: **assumes** *SetFormulaPredicate P* **shows** $\forall x.\ \exists\ z.\ \forall\ u.\ (u\ \varepsilon\ z \longleftrightarrow u\ \varepsilon\ x \wedge P\ (\Xi(n{:=}\ u)))$
using *sep*[*OF swap-variables*, *OF assms*, *of* $\Xi(n{:=}\ \Xi\ 0)\ n\ 0$]
by (*cases* $n = 0$, *simp*) (*simp del*: *neq0-conv add*: *fun-upd-twist*[*of n 0*])

lemma *sep-SP*: **assumes** *SetProperty P* **shows** $\forall x.\ \exists\ z.\ \forall\ u.\ (u\ \varepsilon\ z \longleftrightarrow\ u\ \varepsilon\ x \wedge P\ u)$
using *sep*[*OF assms*[*unfolded SetProperty-def*]] **by** *simp*

lemma *sep-SR*: **assumes** *SetRelation P* **shows** $\forall x.\ \exists\ z.\ \forall\ u.\ (u\ \varepsilon\ z \longleftrightarrow\ u\ \varepsilon\ x \wedge P\ u\ r)$
using *sep*[*OF assms*[*unfolded SetRelation-def*]] **by** *simp*

lemma *singleton-sep*: **assumes** $z\ \varepsilon\ x$
shows $\exists\ y.\ \forall\ u.\ u\ \varepsilon\ y \longleftrightarrow u = z$
using *sep*[*rule-format*, *of* $\lambda\Xi.\ \Xi\ 0 = \Xi\ 1\ x$ *undefined*(*1*:=*z*), *OF SFP-eq*, *unfolded fun-upd-apply*] *assms* **by** *auto*

lemma *pair-sep*: **assumes** $z_1\ \varepsilon\ x\ z_2\ \varepsilon\ x$
shows $\exists\ y.\ \forall\ u.\ u\ \varepsilon\ y \longleftrightarrow u = z_1 \vee u = z_2$
proof–
have *SetFormulaPredicate* $(\lambda\Xi.\ \Xi\ 0 = \Xi\ 1 \vee \Xi\ 0 = \Xi\ 2)$
by (*rule SFP-rules*)+
from *sep*[*rule-format*, *OF this*, *of x undefined*($1:=z_1,2:=z_2$), *simplified*]
show $\exists y.\ \forall u.\ (u\ \varepsilon\ y) = (u = z_1 \vee u = z_2)$
using *assms* **by** *blast*
qed

lemma *triple-sep*: **assumes** $z_1\ \varepsilon\ x\ z_2\ \varepsilon\ x\ z_3\ \varepsilon\ x$
shows $\exists\ y.\ \forall\ u.\ u\ \varepsilon\ y \longleftrightarrow u = z_1 \vee u = z_2 \vee u = z_3$
proof–
have *SetFormulaPredicate* $(\lambda\Xi.\ \Xi\ 0 = \Xi\ (Suc\ 0) \vee \Xi\ 0 = \Xi\ 2 \vee \Xi\ 0 = \Xi\ 3)$
by (*rule SFP-rules*)+
from *sep*[*rule-format*, *OF this*, *of x undefined*($1:=z_1,2:=z_2,3:=z_3$), *simplified*]
show $\exists\ y.\ \forall\ u.\ u\ \varepsilon\ y \longleftrightarrow u = z_1 \vee u = z_2 \vee u = z_3$
using *assms* **by** *blast*
qed

lemma *regular-not-self-mem-sep*: **assumes** *regular x* **shows** $\neg\ x\ \varepsilon\ x$
proof
assume $x\ \varepsilon\ x$
then obtain *t* **where** *t*: $\forall\ u.\ u\ \varepsilon\ t \longleftrightarrow u = x$
using *singleton-sep* **by** *blast*
from *assms*[*unfolded regular-def*, *rule-format*, *of t*]
show *False*
unfolding *subsetM-def t*[*rule-format*] **using** ‹$x\ \varepsilon\ x$› **by** *blast*
qed

lemma *regular-antisym-mem-sep*: **assumes** *regular* $x\ y_1\ \varepsilon\ x\ y_2\ \varepsilon\ x$ **shows** $\neg\ (y_1\ \varepsilon\ y_2 \wedge y_2\ \varepsilon\ y_1)$
proof
obtain *t* **where** *t*: $\forall\ u.\ u\ \varepsilon\ t \longleftrightarrow u = y_1 \vee u = y_2$
using *pair-sep*[*OF assms*(*2*–)] **by** *blast*
assume $y_1\ \varepsilon\ y_2 \wedge y_2\ \varepsilon\ y_1$

with ‹*regular x*›[*unfolded regular-def*, *rule-format*, *of t*]
show *False*
unfolding *subsetM-def t*[*rule-format*] **using** *assms regular-def* **by** *blast*
qed

lemma *regular-antisym2-mem-sep*: **assumes** *regular x* y_1 ε *x* y_2 ε *x* y_3 ε *x* **shows** $\neg$ (y_1 ε y_2 $\wedge$ y_2 ε y_3 $\wedge$ y_3 ε y_1)
proof
obtain *t* **where** *t*: $\forall$ *u*. *u* ε *t* $\longleftrightarrow$ *u* = y_1 $\vee$ *u* = y_2 $\vee$ *u* = y_3
using *triple-sep*[*OF assms*(*2*−)] **by** *blast*
assume y_1 ε y_2 $\wedge$ y_2 ε y_3 $\wedge$ y_3 ε y_1
with ‹*regular x*›[*unfolded regular-def*, *rule-format*, *of t*]
show *False*
unfolding *set-defs t*[*rule-format*] **using** *assms regular-def* **by** *blast*
qed

lemma *ord-mem-ord-sep*: **assumes** *ordM v u* ε *v* **shows** *ordM u*
unfolding *ordM-def epschain-def conj-assoc*[*symmetric*]
proof (*rule conjI*)+
have *regular v u* $\subseteq_M$ *v*
using ‹*ordM v*› *ordM-D1* ‹*u* ε *v*› *ordM-def* **by** *blast*+
then show *regular u*
unfolding *regular-def* **using** *subsetM-trans*[*OF* - ‹*u* $\subseteq_M$ *v*›] **by** *blast*
have *u* $\subseteq_M$ *v epschain v regular v transM v*
using *assms* **unfolding** *natM-def ordM-def transM-def epschain-def* **by** *blast*+
thus $\forall$ *y z*. *y* ε *u* $\wedge$ *z* ε *u* $\longrightarrow$ *y* ε *z* $\vee$ *y* = *z* $\vee$ *z* ε *y*
using ‹*ordM v*› **unfolding** *ordM-def subsetM-def epschain-def* **by** *blast*
show *transM u*
unfolding *transM-def set-defs*
proof (*rule allI*, *rule impI*)+
fix *z y*
assume *y* ε *u z* ε *y*
hence *z* ε *v y* ε *v*
using ‹*u* ε *v*› ‹*transM v*› **unfolding** *transM-def set-defs* **by** *blast*+
have *u* $\neq$ *z*
using *regular-antisym-mem-sep*[*OF* ‹*regular v*› ‹*y* ε *v*› ‹*z* ε *v*›] ‹*y* ε *u*› ‹*z* ε *y*› **by** *blast*
have $\neg$ *u* ε *z*
using *regular-antisym2-mem-sep*[*OF* ‹*regular v*› ‹*u* ε *v*› ‹*z* ε *v*› ‹*y* ε *v*›] ‹*y* ε *u*› ‹*z* ε *y*› **by** *blast*
show *z* ε *u*
using *ordM-D2*[*OF* ‹*ordM v*› ‹*u* ε *v*› ‹*z* ε *v*›] ‹*u* $\neq$ *z*› ‹$\neg$ *u* ε *z*› **by** *blast*
qed
qed

end

The interest of this locale is to investigate intersections of various kinds.

locale *L-setext-sep* = *L-setext* + *L-sep*

begin

sublocale *L-setext-empty*
by *unfold-locales*

lemma *separ-def-SFP*: **assumes** *SetFormulaPredicate P*
shows $u\ \varepsilon\ separationM\ y\ (\lambda\ x.\ P(\Xi(n{:=}x))) \longleftrightarrow (u\ \varepsilon\ y \wedge (P\ (\Xi(n{:=}\ u))))$
using *collect-def-ex*[*OF sep-var*[*rule-format*], *OF assms*] *separationM-def* **by** *simp*

lemma *separ-def-SP*: **assumes** *SetProperty P*
shows $u\ \varepsilon\ separationM\ y\ (\lambda\ x.\ P\ x) \longleftrightarrow (u\ \varepsilon\ y \wedge P\ u)$
using *separ-def-SFP*[*OF assms*[*unfolded SetProperty-def*], *of - - - 0*] **by** *simp*

lemma *separ-def-SR*: **assumes** *SetRelation P*
shows $u\ \varepsilon\ separationM\ y\ (\lambda\ x.\ P\ x\ r) \longleftrightarrow (u\ \varepsilon\ y \wedge P\ u\ r)$
using *separ-def-SFP*[*OF assms*[*unfolded SetRelation-def*], *of - - - 0*] **by** *simp*

lemma *least-def'*:
assumes $Q\ (\Xi(n{:=}w))$ **and** *SetFormulaPredicate Q*
shows $u\ \varepsilon \bigcap_M\ (\lambda\ x.\ Q\ (\Xi(n{:=}x))) \longleftrightarrow (\forall\ \ y.\ Q\ (\Xi(n{:=}y)) \longrightarrow u\ \varepsilon\ y)$
proof−
from *bounded-free*[*OF assms*(*2*)]
obtain m **where** m: $Q\ \Xi = Q\ \Xi'$ **if** $\forall i.\ i < m \longrightarrow \Xi\ i = \Xi'\ i$ **for** $\Xi\ \Xi'$
by *blast*
have k: $Q\ (\Xi(m{+}n{+}1 := x,\ n := y)) \longleftrightarrow Q\ (\Xi(n := y))$ **for** $x\ y$
by (*rule m*) *force+*
have *aux*: $(\Xi(n := a))\ n = a$ **for** Ξ **and** $a{::}'a$
by *simp*
have *sfp*: $SetFormulaPredicate\ (\lambda\ \Xi.\ (\forall\ \ y.\ Q\ (\Xi(n{:=}y)) \longrightarrow ((\Xi(n{:=}y))\ (m{+}n{+}1))\ \varepsilon\ y))$
by (*rule SFP-all*[*of* $\lambda\ \Xi.\ \ Q\ (\Xi) \longrightarrow (\Xi)\ (m{+}n{+}1)\ \varepsilon\ (\Xi\ n)$, *of n*, *unfolded aux*], *unfold logsimps*) (*rule* | *fact*)+
have *iff*: $(v\ \varepsilon\ w \wedge (\forall y.\ Q\ (\Xi(n := y)) \longrightarrow v\ \varepsilon\ y)) \longleftrightarrow (\forall y.\ Q\ (\Xi(n := y)) \longrightarrow v\ \varepsilon\ y)$ **for** v
using *assms*(*1*) **by** *blast*
have *sep*: $(v\ \varepsilon\ separationM\ w\ (\lambda x.\ \forall y.\ Q\ (\Xi(n := y)) \longrightarrow x\ \varepsilon\ y)) = (\forall y.\ Q\ (\Xi(n := y)) \longrightarrow v\ \varepsilon\ y)$ **for** v
using *separ-def-SFP*[*of - v*, *OF sfp*, *of w* Ξ $m{+}n{+}1$] *iff*[*of v*]
unfolding k **by** *force*
show *?thesis*
unfolding *leastM-def* **using** *collect-def'*
using *collect-def'*[*of separationM w* $(\lambda x.\ \forall y.\ Q\ (\Xi(0 := y)) \longrightarrow u\ \varepsilon\ y)$] *sep* **by** *force*
qed

lemma *least-def-ex*:
$\exists\ \ w.\ Q\ (\Xi(n := w)) \Longrightarrow SetFormulaPredicate\ Q \Longrightarrow$

$u\ \varepsilon \bigcap_M (\lambda\ x.\ Q\ (\Xi(n{:=}x))) \longleftrightarrow (\forall\ y.\ Q\ (\Xi(n{:=}y)) \longrightarrow u\ \varepsilon\ y)$
using *least-def'* **by** *force*

lemma *intersection-def'*[*set-defs*]: $z\ \varepsilon\ x \cap_M y \longleftrightarrow z\ \varepsilon\ x \wedge z\ \varepsilon\ y$
proof–
have *SetFormulaPredicate* $(\lambda\Xi.\ \Xi\ 0\ \varepsilon\ \Xi\ (Suc\ 0))$
by (*rule SFP-rules*)
from *separ-def-SFP*[*OF this*, *of z x*, *rule-format*, *of undefined*(*1*:=*y*) *0*, *simplified*]

show $(z\ \varepsilon\ x \cap_M y) = (z\ \varepsilon\ x \wedge z\ \varepsilon\ y)$
unfolding *intersectionM-def separationM-def*.
qed

lemma *intersect-subset*: $x \cap_M y \subseteq_M x\ x \cap_M y \subseteq_M y$
unfolding *set-defs* **by** *simp-all*

lemma *intersect-regular*: **assumes** *regular x regular y*
shows *regular* $(x \cap_M y)$
using *assms* **unfolding** *regular-def intersection-def' subsetM-def* **by** *blast*

lemma *intersect-transM*: **assumes** *transM x transM y*
shows *transM* $(x \cap_M y)$
using *assms* **unfolding** *intersection-def' transM-def subsetM-def* **by** *blast*

lemma *intersect-epschain*: **assumes** *epschain x epschain y*
shows *epschain* $(x \cap_M y)$
using *assms intersect-transM* **unfolding** *intersection-def' epschain-def subsetM-def* **by** *blast*

lemma *intersect-ordM*: **assumes** *ordM x ordM y*
shows *ordM* $(x \cap_M y)$
using *assms intersect-transM intersect-regular* **unfolding** *ordM-def*
unfolding *intersection-def' epschain-def* **by** *simp*

lemma *ordM-subset-mem*: **assumes** *ordM x ordM y* $y \subset_M x$
shows $y\ \varepsilon\ x$
proof–
have *SR*: *SetRelation* $(\lambda\ v\ y.\ \neg\ v\ \varepsilon\ y)$
unfolding *SetRelation-def* **by** (*rule SFP-rules*)+
have *regular x*
using ‹*ordM x*› **by** *simp*
have *dif*: $u\ \varepsilon$ *separationM x* $(\lambda\ v.\ \neg\ v\ \varepsilon\ y) \longleftrightarrow u\ \varepsilon\ x \wedge \neg\ u\ \varepsilon\ y$ **for** *u*
using *separ-def-SR*[*OF SR*].
hence *separationM* $x\ (\lambda\ v.\ \neg\ v\ \varepsilon\ y) \subseteq_M x$
unfolding *subsetM-def* **by** *blast*
from ‹*regular x*›[*unfolded regular-def*, *rule-format*, *OF this*, *unfolded dif*]
obtain *c* **where** $c\ \varepsilon\ x\ \neg\ c\ \varepsilon\ y$ **and** *c*: $\forall\, v.\ v\ \varepsilon\ c \longrightarrow (v\ \varepsilon\ y \vee \neg\ v\ \varepsilon\ x)$
using *proper-subset-diff*[*OF* ‹$y \subset_M x$›] **by** *blast*
have $c \subseteq_M x$

using ‹$c \varepsilon x$› *assms*(*1*) *natM-def ordM-D1* **by** *blast*
hence $c \subseteq_M y$
using ‹$y \subset_M x$› c **unfolding** *subsetM-def proper-subsetM-def* **by** *blast*
have $y = c$
proof (*rule subsetM-antisym*[*OF* ‹$c \subseteq_M y$›], *unfold subsetM-def*, *rule allI*, *rule impI*)
fix z **assume** $z \varepsilon y$
hence $z \varepsilon x$
using ‹$y \subset_M x$› **unfolding** *proper-subsetM-def* **by** *blast*
from *ordM-D2*[*OF* ‹*ordM* x› *this* ‹$c \varepsilon x$›]
show $z \varepsilon c$
using ‹$\neg c \varepsilon y$› ‹$z \varepsilon y$› *ordM-D1*[*OF* ‹*ordM* y› ‹$z \varepsilon y$›, *unfolded subsetM-def*] **by** *blast*
qed
thus *?thesis*
using ‹$c \varepsilon x$› **by** *blast*
qed

lemma *ordM-total*: **assumes** *ordM* x *ordM* y
shows $x \varepsilon y \vee y \varepsilon x \vee x = y$
using *ordM-def*[*unfolded epschain-def*]
proof–
have *regular* ($x \cap_M y$)
using *intersect-ordM*[*OF* ‹*ordM* x› ‹*ordM* y›] **by** *simp*
from *regular-not-self-mem-sep*[*OF this*]
have $\neg (x \cap_M y \subset_M x \wedge x \cap_M y \subset_M y)$
using *ordM-subset-mem*[*OF assms*(*1*) *intersect-ordM*, *OF assms*]
intersection-def′ assms(*1*,*2*) *ord-mem-ord-sep ordM-subset-mem* **by** *blast*
with *sep-SR*[*rule-format*, *of* λ *v y*. $v \varepsilon y$]
consider $x \cap_M y = x \mid x \cap_M y = y$
unfolding *proper-subsetM-def* **using** *intersection-def′* **by** *blast*
then show *?thesis*
proof (*cases*)
assume $x \cap_M y = x$
hence $x = y \vee x \subset_M y$
unfolding *proper-subsetM-def setext* **using** *intersection-def′* **by** *blast*
with *ordM-subset-mem*[*OF assms*(*2*,*1*)]
show *?thesis*
by *blast*
next
assume $x \cap_M y = y$
hence $x = y \vee y \subset_M x$
unfolding *proper-subsetM-def setext* **using** *intersection-def′* **by** *blast*
with *ordM-subset-mem*[*OF assms*]
show *?thesis*
by *blast*
qed
qed

lemma *nat-mem-nat*: **assumes** $natM\ v\ u \ \varepsilon\ v$ **shows** $natM\ u$
  unfolding *natM-def*
proof
  show $ordM\ u$
    using *assms ord-mem-ord-sep* **unfolding** *natM-def* **by** *blast*
  show $\forall v.\ v\ \varepsilon\ u \lor v = u \longrightarrow (\exists u.\ u\ \varepsilon\ v) \longrightarrow is\text{-}sucM\ v$
    using *assms natM-D nat-mem-is-suc ordM-D1 subsetM-def* **by** *blast*
qed

lemma *set-of-ords-regular*: **assumes** $\forall\ y.\ y\ \varepsilon\ s \longrightarrow ordM\ y$
  shows *regular s*
  unfolding *regular-def*
proof (*rule allI*, *rule impI*, *rule impI*)
  fix y **assume** $y \subseteq_M s\ \exists z.\ z\ \varepsilon\ y$
  then obtain z **where** $z\ \varepsilon\ y$
    by *blast*
  have $ordM\ z$
    using ‹$y \subseteq_M s$› ‹$z\ \varepsilon\ y$› *assms subsetM-def* **by** *blast*
  hence *regular z*
    by *simp*
  show $\exists u.\ u\ \varepsilon\ y \land (\forall v.\ \neg\ (v\ \varepsilon\ u \land v\ \varepsilon\ y))$
  proof (*cases* $\forall v.\ \neg\ (v\ \varepsilon\ z \land v\ \varepsilon\ y)$, *use* ‹$z\ \varepsilon\ y$› **in** *blast*)
    assume a: $\neg\ (\forall v.\ \neg\ (v\ \varepsilon\ z \land v\ \varepsilon\ y))$
    from *sep-SR*[*of* $\lambda\ x\ y.\ x\ \varepsilon\ y$, *rule-format*, *of z*, *unfolded SetRelation-def*, *OF SFP-mem*]
    obtain u **where** u: $\forall\ v.\ v\ \varepsilon\ u \longleftrightarrow v\ \varepsilon\ z \land v\ \varepsilon\ y$
      by *auto*
    hence $u \subseteq_M z\ \exists\ v.\ v\ \varepsilon\ u$
      using a **unfolding** *u*[*rule-format*] *set-defs* **by** *blast+*
    from ‹*regular z*›[*unfolded regular-def*, *rule-format*, *OF this*]
    obtain m **where** $m\ \varepsilon\ u\ \forall v.\ \neg\ (v\ \varepsilon\ m \land v\ \varepsilon\ u)$
      by *blast*
    hence $m\ \varepsilon\ y \land (\forall v.\ \neg\ (v\ \varepsilon\ m \land v\ \varepsilon\ y))$
      unfolding *u*[*rule-format*] **using** ‹$ordM\ z$› *ordM-trans* **by** *presburger*
    thus *?thesis*
      by *blast*
  qed
qed

lemma *intersect-natM*: **assumes** $natM\ x\ natM\ y$
  shows $natM\ (x \cap_M y)$
proof (*cases* $x = x \cap_M y$, *use assms* **in** *argo*)
  assume $x \neq x \cap_M y$
  show $natM\ (x \cap_M y)$
  proof (*cases* $x \cap_M y = \emptyset$)
    assume $x \cap_M y \neq \emptyset$
    show $natM\ (x \cap_M y)$
    proof (*rule natM-I*)
      show $\bigwedge v.\ \exists u.\ u\ \varepsilon\ v \Longrightarrow v\ \varepsilon\ x \cap_M y \Longrightarrow is\text{-}sucM\ v$

using ‹*natM x*›[*unfolded natM-def*] **unfolding** *intersection-def'* **by** *blast*
show *is-sucM* $(x \cap_M y)$
proof (*rule nat-mem-is-suc*)
have $\neg\ x\ \varepsilon\ x \cap_M y$
using *assms*(*1*) *natM-def ordM-def regular-not-self-mem-sep* **unfolding** *intersection-def'* **by** *fast*
thus $(x \cap_M y)\ \varepsilon\ x$
using *ordM-total*[*OF - intersect-ordM*, *of x x y*] ‹$\neg\ x = x \cap_M y$› *natM-D assms* **by** *blast*
show $\exists\, u.\ u\ \varepsilon\ x \cap_M y$
using ‹$x \cap_M y \neq \emptyset$› *emptyset-def'* **by** *auto*
qed *fact*
qed (*simp add*: *assms intersect-ordM natM-D*)
qed *simp*
qed

end

locale *L-setext-setsuc-sep* = *L-setext* + *L-setsuc* + *L-sep*

begin

We want to prove that ordinals are closed under the successor operation. However, regularity is a problem. It is not necessarily preserved by taking successors.

Separation is needed to avoid the following pathological situation: consider an epschain $C = \{c_z \mid z \in \mathbb{Z}\}$ of type $\mathbb{Z}$ where moreover $\emptyset \in c_z$ for each z. That is, $u\ \varepsilon\ c_m$ iff $u = c_k$ with $k < m$, or $u = \emptyset$. We postulate that an infinite subclass of C is a set iff it contains c_0. Now, c_z are ordinals for all $z \leq 0$. But c_1 is not regular, since it contains an infinite descending chain $\{c_k \mid k \leq 0\}$ as a subset.

sublocale *L-setext-empty-setsuc*
by *unfold-locales*

sublocale *L-setext-sep*
by *unfold-locales*

lemma *regular-setsuc*: **assumes** *transM x regular x regular y*
shows *regular* (*setsucM x y*)
proof (*cases* $y\ \varepsilon\ x$)
assume $y\ \varepsilon\ x$
hence *setsucM x y* = *x*
unfolding *setext*[*of - x*] *set-defs* **by** *blast*
thus *regular* (*setsucM x y*)
using ‹*regular x*› **by** *simp*
next
assume $\neg\ y\ \varepsilon\ x$
show *regular* (*setsucM x y*)

unfolding *regular-def*
proof (*rule allI*, *rule impI*, *rule impI*)
fix w
assume $w \subseteq_M$ *setsucM* $x\ y$ $\exists\ z.\ z\ \varepsilon\ w$
have *SetFormulaPredicate* ($\lambda\Xi.\ \Xi\ 0 \neq \Xi\ 1$)
by (*rule SFP-rules*)+
from *sep-SR*[*of* $\lambda\ x\ y.\ x \neq y$, *unfolded SetRelation-def*, *OF this*]
obtain w' **where** w': $\bigwedge\ z.\ z\ \varepsilon\ w' \longleftrightarrow z\ \varepsilon\ w \wedge z \neq y$
by *presburger*
have $w' \subseteq_M x$
using ‹$w \subseteq_M$ *setsucM* $x\ y$› w' *subsetM-def setsuc-def'* **by** *auto*
show $\exists z.\ z\ \varepsilon\ w \wedge (\forall v.\ \neg\ (v\ \varepsilon\ z \wedge v\ \varepsilon\ w))$
proof (*cases* $y\ \varepsilon\ w$)
assume $y\ \varepsilon\ w$
show *?thesis*
proof (*cases* $w' = \emptyset$)
assume $w' = \emptyset$
have $w = \{y\}_M$
using w' *empty-is-empty* ‹$y\ \varepsilon\ w$› **unfolding** *setext*[*of* w] *set-defs* ‹$w' = \emptyset$› *singleton-def'* **by** *metis*
hence $y\ \varepsilon\ w \wedge (\forall v.\ \neg\ (v\ \varepsilon\ y \wedge v\ \varepsilon\ w))$
unfolding ‹$w = \{y\}_M$› *set-defs* **using** *regular-not-self-mem*[*OF* ‹*regular* y›] *singleton-def'*
by *auto*
then show *?thesis*
by *blast*
next
assume $\neg\ w' = \emptyset$
hence $\exists\ z.\ z\ \varepsilon\ w'$
by (*simp add*: *emptyset-def'*)
from ‹*regular* x›[*unfolded regular-def*, *rule-format*, *OF* ‹$w' \subseteq_M x$› *this*]
obtain m **where** $m\ \varepsilon\ w'$ **and** m: $\forall v.\ \neg\ (v\ \varepsilon\ m \wedge v\ \varepsilon\ w')$
by *blast*
hence $m\ \varepsilon\ w$
using w' **by** *blast*
have $\neg\ y\ \varepsilon\ m$
using ‹*transM* x›[*unfolded transM-def*] ‹$m\ \varepsilon\ w'$› ‹$w' \subseteq_M x$› ‹$\neg\ y\ \varepsilon\ x$› *subsetM-def* **by** *blast*
have $\forall v.\ \neg\ (v\ \varepsilon\ m \wedge v\ \varepsilon\ w)$
using m **unfolding** w' **using** ‹$\neg\ y\ \varepsilon\ m$› **by** *blast*
then show *?thesis*
using ‹$m\ \varepsilon\ w$› **by** *blast*
qed
next
assume $\neg\ y\ \varepsilon\ w$
hence $w \subseteq_M x$
using ‹$w \subseteq_M$ *setsucM* $x\ y$› w' *subsetM-def*
by (*metis* ‹$w' \subseteq_M x$›)
from ‹*regular* x›[*unfolded regular-def*, *rule-format*, *OF this* ‹$\exists\ z.\ z\ \varepsilon\ w$›]

show *?thesis*.
qed
qed
qed

lemma *ord-suc-ord*: **assumes** *ordM x* **shows** *ordM* (*setsucM x x*)
proof(*rule ordM-I*)
show *epschain*: *epschain* (*setsucM x x*)
using *assms*
using *ordM-def epschain-suc-epschain* **by** *blast*
show *regular* (*setsucM x x*)
unfolding *regular-def* **using** *assms epschain-def ordM-def regular-def regular-setsuc* **by** *blast*
qed

lemma *nat-suc-nat*: **assumes** *natM x* **shows** *natM* (*setsucM x x*)
proof–
have $(\forall v.\ v\ \varepsilon\ x \lor v = x \longrightarrow (\exists u.\ u\ \varepsilon\ v) \longrightarrow (\exists y.\ \forall z.\ (z\ \varepsilon\ v) = (z\ \varepsilon\ y \lor z = y))) \Longrightarrow$
$(\forall v.\ v\ \varepsilon\ setsucM\ x\ x \lor v = setsucM\ x\ x \longrightarrow (\exists u.\ u\ \varepsilon\ v) \longrightarrow (\exists y.\ \forall z.\ (z\ \varepsilon\ v) = (z\ \varepsilon\ y \lor z = y)))$
using *setsuc-def'* **by** *auto*
thus *natM* (*setsucM x x*)
using *assms ord-suc-ord* **unfolding** *natM-def is-sucM-def* **by** *fast*
qed

lemma *one-natM*[*simp*]: *natM* $(\{\emptyset\}_M)$
using *nat-suc-nat*[*OF emp-natM*] **unfolding** *setsuc-empty-sing*.

lemma *ord-mem-suc*: **assumes** *ordM x* **and** $y\ \varepsilon\ x$
shows $setsucM\ y\ y\ \varepsilon\ x \lor setsucM\ y\ y = x$
proof–
have *ordM y*
using *ord-mem-ord*[*OF* ‹*ordM x*› ‹$y\ \varepsilon\ x$›].
have *ordM*(*setsucM y y*)
by (*simp add*: ‹*ordM y*› *ord-suc-ord*)
have $\neg\ x\ \varepsilon\ setsucM\ y\ y$
proof
assume $x\ \varepsilon\ setsucM\ y\ y$
then consider $x\ \varepsilon\ y \mid x = y$
unfolding *setsuc-def'* **by** *blast*
then show *False*
by (*cases*) (*use* ‹$y\ \varepsilon\ x$› ‹*ordM x*› *ordM-def ordM-trans regular-not-self-mem* **in** *blast*)+
qed
with *ordM-total*[*OF* ‹*ordM x*› ‹*ordM*(*setsucM y y*)›]
show *?thesis*
by *blast*
qed

lemma *ord-limit-or-suc*: **assumes** *ordM x*
  shows *is-sucM x* $\vee$ ($\forall$ *u*. *u* ε *x* $\longrightarrow$ *setsucM u u* ε *x*)
  using *assms is-suc-def′ ord-mem-suc* **by** *auto*

lemma **assumes** *regular x regular y*
  shows *regular* (*setsucM x y*)
  unfolding *regular-def setsuc-def′*
  oops
— not true, consider $\{x\}$ ε x. Then $x \cup \{x\}$ contains a subset $\{x,\{x\}\}$ which is a cycle

end

1.3.8 Transitive superset

The existence of transitive supersets needs to be assumed explicitly if the axiom of infinity is absent (or negated). It is related to regularity, since it enables obtaining an infinite set from infinite descending epsilon chains.

locale *L-ts* = *set-signature* +
  assumes *ts*: $\forall$ *x*. $\exists$ *z*. (*transM z* $\wedge$ ($x \subseteq_M z$))

locale *L-setext-sep-ts* = *L-setext* + *L-sep* + *L-ts*

begin

sublocale *L-setext-sep*
  by *unfold-locales*

lemma *least-ts-def′* : *u* ε *least-tsM x* $\longleftrightarrow$ ($\forall$ *v*. *transM v* $\wedge$ $x \subseteq_M v$ $\longrightarrow$ *u* ε *v*)
   (**is** *u* ε *least-tsM x* $\longleftrightarrow$ ($\forall$ *v*. *?Q v* $\longrightarrow$ *u* ε *v*))
  by (*rule least-def-ex*[*of* λ Ξ. *transM* (Ξ *0*) $\wedge$ Ξ *1* $\subseteq_M$ Ξ *0 undefined*(*1*:=*x*) *0*, *simplified*, *OF ts*[*rule-format*], *folded least-tsM-def*], *unfold logsimps set-defs*) (*rule SFP-rules*)+

lemma *least-ts-is-transitive*: *transM* (*least-tsM x*)
  using *least-ts-def′* **unfolding** *transM-def subsetM-def* **by** *blast*

end

locale *L-setext-sep-reg* = *L-setext* + *L-sep* + *L-reg*

locale *L-setext-sep-reg-ts* = *L-setext* + *L-sep* + *L-reg* + *L-ts*

begin

sublocale *L-setext-sep-ts*
  by *unfold-locales*

The schema of regularity follows from regularity and transitive superset.

```
sublocale L-regsch
proof (unfold-locales, rule impI)
  fix P Ξ
  assume SetFormulaPredicate P
  assume ∃ x. P (Ξ(0:=x))
  then obtain x where P (Ξ(0:=x))
    by blast
  have x ⊆M (least-tsM x)
    using least-ts-def′ subsetM-def by blast
  define w where w = separationM (least-tsM x) (λ x. P (Ξ(0:=x)))
  then have w: u ε w ⟷ u ε least-tsM x ∧ P (Ξ(0:=u)) for u
    using separ-def-SFP[OF ‹SetFormulaPredicate P›] by blast
  show ∃x. P (Ξ(0 := x)) ∧ (∀y. y ε x ⟶ ¬ P (Ξ(0 := y)))
  proof(cases)
    assume ∃ v. v ε w
    have (∃ v. v ε w)⟶(∃ v. (v ε w ∧ ( ∀ t. (t ε v  ⟶ ¬ t ε w ) ) ) )
      using reg by blast
    then have (∃ v. (v ε w ∧ ( ∀ t. (t ε v  ⟶ ¬ t ε w ) ) ) ) using ‹∃ v. v ε w›
by blast
    then obtain v where v ε w and v: t ε v  ⟶ ¬ t ε w for t by blast
    have P (Ξ(0 := v))
      using w ‹v ε w› by blast
    have v ε least-tsM x
      using w ‹v ε w› by blast
    have v ⊆M least-tsM x
      using least-ts-is-transitive ‹v ε least-tsM x› transM-def by blast
    have t ε v ⟶ ¬ P (Ξ(0 := t)) for t
      using ‹v ⊆M least-tsM x› w v unfolding subsetM-def by blast
    then have ∀ t. (t ε v ⟶ ¬ P (Ξ(0 := t))) by blast
    then have ∃x. P (Ξ(0 := x))∧ (∀y. y ε x ⟶ ¬ P (Ξ(0 := y)))
      using ‹P (Ξ(0 := v))› by blast
    then show ?thesis by blast
  next
    assume  ¬ (∃ v. v ε w)
    then have ∀ v. (v ε least-tsM x ⟶ ¬ P (Ξ(0 := v)) )
      using w  by blast
    then have  ∀ v. (v ε x ⟶ ¬ P (Ξ(0 := v)) )
       using subsetM-def ‹x ⊆M least-tsM x› by blast
     then show ?thesis
       using ‹P (Ξ(0 := x))› by blast
  qed
qed

end
```

1.3.9 Replacement

locale *L-repl* = *set-signature* +

assumes *replf*: *SetFormulaPredicate* $P \Longrightarrow (\forall\ u.\ \exists!\ v.\ P\ (\Xi(\mathit{0}{:=}u,\ \mathit{1}{:=}\ v))) \longrightarrow (\forall\ x.\ \exists\ z.\ \forall\ v.\ v\ \varepsilon\ z \longleftrightarrow (\exists\ u.\ u\ \varepsilon\ x \wedge P\ (\Xi(\mathit{0}{:=}u,\ \mathit{1}{:=}\ v))))$
— We can replace x_0 by the unique x_1 satisfying $P(x_0,x_1)$. Note the presence of parameters x_i, $i \geq 2$

locale *L-setext-empty-repl* = *L-setext* + *L-empty* + *L-repl*

begin

Replacing using total functions is equivalent to replacing using partial functions.

lemma *replp*: **assumes** *SetFormulaPredicate P* **and** *func*: $\forall\ u\ v\ w.\ P\ (\Xi(\mathit{0}{:=}u,\ \mathit{1}{:=}\ v)) \wedge P\ (\Xi(\mathit{0}{:=}u,\ \mathit{1}{:=}\ w)) \longrightarrow v = w$
shows $\forall\ x.\ \exists\ z.\ (\forall\ v.\ v\ \varepsilon\ z \longleftrightarrow (\exists\ u.\ u\ \varepsilon\ x \wedge P\ (\Xi(\mathit{0}{:=}u,\ \mathit{1}{:=}\ v))))$
proof
fix *x*
show $\exists z.\ \forall v.\ (v\ \varepsilon\ z) = (\exists u.\ u\ \varepsilon\ x \wedge P\ (\Xi(\mathit{0}{:=}u,\ \mathit{1}{:=}\ v)))$
proof (*cases* $\exists\ u\ v.\ u\ \varepsilon\ x \wedge P\ (\Xi(\mathit{0}{:=}u,\ \mathit{1}{:=}\ v))$)
assume $\nexists u\ v.\ u\ \varepsilon\ x \wedge P\ (\Xi(\mathit{0}{:=}u,\ \mathit{1}{:=}\ v))$
then show $\exists z.\ \forall v.\ (v\ \varepsilon\ z) = (\exists u.\ u\ \varepsilon\ x \wedge P\ (\Xi(\mathit{0}{:=}u,\ \mathit{1}{:=}\ v)))$
using *exI*[*of* $\lambda\ z.\ \forall v.\ (v\ \varepsilon\ z) = (\exists u.\ u\ \varepsilon\ x \wedge P\ (\Xi(\mathit{0}{:=}u,\ \mathit{1}{:=}\ v)))\ \emptyset$] *empty* **by** *blast*
next
assume *ex*: $\exists\ u\ v.\ u\ \varepsilon\ x \wedge P\ (\Xi(\mathit{0}{:=}u,\ \mathit{1}{:=}\ v))$
then obtain *v* **where** *ex*: $\exists u.\ u\ \varepsilon\ x \wedge P\ (\Xi(\mathit{0}{:=}u,\ \mathit{1}{:=}\ v))$
by *blast*
from *bounded-free*[*OF assms*(*1*)]
obtain *k* **where** *k0*: $\forall \Xi\ \Xi'.\ (\forall i{<}k.\ \Xi\ i = \Xi'\ i) \longrightarrow P\ \Xi = P\ \Xi'$
by *blast*
let *?k* = *Suc* (*max k 1*)
have *k*: $P(\Xi) \longleftrightarrow P(\Xi(?k{:=}\ a))$ $\mathit{1} <$ *?k* **for** Ξ *a*
by (*rule k0*[*rule-format, of* $\Xi\ \Xi(?k{:=}\ a)$], *force*) *simp*
have *twist*: $\Xi(\mathit{Suc}\ (\mathit{max}\ k\ \mathit{1}) := a,\ \mathit{0} := b,\ \mathit{1} := c) = \Xi(\mathit{0} := b,\ \mathit{1} := c,\ \mathit{Suc}\ (\mathit{max}\ k\ \mathit{1}) := a)$ **for** *a b c*
by *auto*
have *canc*: $P(\Xi(?k{:=}a,\mathit{0}{:=}b,\mathit{1}{:=}c)) = P(\Xi(\mathit{0}{:=}b,\mathit{1}{:=}c))$ $\Xi(?k{:=}a,\mathit{0}{:=}b,\mathit{1}{:=}c,\mathit{1}{:=}d) = \Xi(?k{:=}a,\mathit{0}{:=}b,\mathit{1}{:=}d)$
$(\Xi(?k := a,\ \mathit{0} := b,\ \mathit{1} := c))\ \mathit{1} = c$ $(\Xi(?k := a,\ \mathit{0} := b,\ \mathit{1} := c))\ ?k = a$ **for** *a b c d*
unfolding *twist* **using** *k*(*1*) **by** *auto*
define *Q* :: (*nat* ⇒ *'a*) ⇒ *bool* **where**
$Q = (\lambda\ \Xi.\ (\mathit{if}\ \exists\ c.\ P\ (\Xi(\mathit{1}{:=}\ c))\ \mathit{then}\ P\ \Xi\ \ \mathit{else}\ \Xi\ \mathit{1} = \Xi\ ?k))$
have $\exists!\ v'.\ Q\ (\Xi(?k{:=}v,\mathit{0}{:=}u,\mathit{1}{:=}v'))$ **for** *u*
unfolding *Q-def canc* **using** *ex func* **by** *metis*
hence *ex1*: $\forall\ u.\ \exists!\ w.\ Q\ (\Xi(?k{:=}v,\mathit{0}{:=}u,\mathit{1}{:=}w))$
by *simp*
have *SetFormulaPredicate Q*
unfolding *Q-def* **by** (*simp*, *unfold logsimps*) (*rule SFP-rules* | *fact*)+
from *replf*[*rule-format*, *OF this ex1*[*rule-format*], *of x*]

have *Qset*: $\exists z.\ \forall w.\ (w\ \varepsilon\ z) = (\exists u.\ u\ \varepsilon\ x \wedge Q(\Xi(?k{:=}v,0{:=}u,1{:=}w)))$.
have *P-iff-Q*: $(\exists u.\ u\ \varepsilon\ x \wedge P\ (\Xi(0 := u,\ 1 := w))) \longleftrightarrow (\exists u.\ u\ \varepsilon\ x \wedge Q\ (\Xi(?k{:=}v,\ 0 := u,\ 1 := w)))$ **for** w
unfolding *Q-def canc* **using** *ex* **by** *auto*
show *?thesis*
using *Qset* **unfolding** *P-iff-Q*.
qed
qed

lemma *replp-vars*: **assumes** *SetFormulaPredicate* P **and** *func*: $\forall\ u\ v\ w.\ P\ (\Xi(k{:=}u,\ l{:=}\ v)) \wedge P\ (\Xi(k{:=}u,\ l{:=}\ w)) \longrightarrow v = w$
shows $\forall\ x.\ \exists\ z.\ (\forall\ v.\ v\ \varepsilon\ z \longleftrightarrow (\exists\ u.\ u\ \varepsilon\ x \wedge P\ (\Xi(k{:=}u,\ l{:=}\ v))))$
proof–
from *bounded-free*[*OF* ‹*SetFormulaPredicate P*›]
obtain m **where** m: $P\ \Xi = P\ \Xi'$ **if** $\forall i{<}m.\ \Xi\ i = \Xi'\ i$ **for** $\Xi\ \Xi'$
by *blast*
let $?m = Suc\ (k + l + m)$
let $?f = id(0 := ?m,\ 1 := Suc\ ?m,\ k{:=}\ 0, l := 1)$
let $?Q = \lambda\ X.\ (P\ (\lambda\ b.\ X\ (?f\ \ b)))$
let $?X = \Xi(?m{:=}\ \Xi\ 0, (Suc\ ?m){:=}\ \Xi\ 1)$
have *sfpq*: *SetFormulaPredicate ?Q*
using *transform-variables*[*OF* ‹*SetFormulaPredicate P*›] **by** *simp*
have *small*: $\forall\ i < m.\ (\Xi(k := u,\ l := v))\ i = (?X\ (0 := u,\ 1 := v))\ (?f\ i)$ **for** $u\ v$
by *auto*
have *equiv*: $(P\ (\Xi(k := u,\ l := v))) \longleftrightarrow (?Q\ (?X\ (0 := u,\ 1 := v)))$ **for** $u\ v$
by (*rule* m[*of* $\Xi(k := u,\ l := v)\ \lambda\ b.\ (?X\ (0 := u,\ 1 := v))(?f\ b)$]) *fact*
then have *funcq*: $(\forall u\ v\ w.\ ?Q\ (?X(0 := u,\ 1 := v)) \wedge ?Q\ (?X(0 := u,\ 1 := w)) \longrightarrow v = w)$
using *func* **by** *blast*
show *?thesis*
using *replp*[*OF sfpq funcq*] **unfolding** *equiv*.
qed

lemma *repl-SR*: **assumes** *SetRelation* $P\ \forall\ u\ v\ w.\ P\ u\ v \wedge P\ u\ w \longrightarrow v = w$
shows $\forall\ x.\ \exists\ z.\ (\forall\ v.\ v\ \varepsilon\ z \longleftrightarrow (\exists\ u.\ u\ \varepsilon\ x \wedge P\ u\ v))$
using *assms* **unfolding** *SetRelation-def* **using** *replp*[*of* $\lambda\Xi.\ P\ (\Xi\ 0)\ (\Xi\ 1)$] **by** *fastforce*

lemma *replace-def'*:
assumes *SetFormulaPredicate* $P\ \forall\ u\ v\ w.\ P(\Xi(k{:=}u,\ l{:=}\ v)) \wedge P(\Xi(k{:=}u,\ l{:=}\ w)) \longrightarrow v = w$
shows $\forall\ v.\ v\ \varepsilon\ replaceM\ (\lambda\ u\ v.\ P(\Xi(k{:=}u,\ l{:=}\ v)))\ x \longleftrightarrow (\exists\ u.\ u\ \varepsilon\ x \wedge (P(\Xi(k{:=}u,\ l{:=}\ v))))$
using *collect-def-ex*[*of* $\lambda\ v.\ (\exists u.\ u\ \varepsilon\ x \wedge P(\Xi(k{:=}u,\ l{:=}\ v)))$, *OF replp-vars*[*OF assms*, *rule-format*], *folded replaceM-def*] **by** *blast*

Separation can be obtained from replacement.

sublocale *L-setext-sep*

proof (*unfold-locales*, *rule allI*)
  fix P Ξ x
  assume *sfp*: *SetFormulaPredicate* P
  — The idea is clear: we replace x_0 with itself, that is, with $x_1 = x_0$. A technical complication is that we have to rename parameters x_i which for separation include x_1 while for replacement they have to start with x_2
  have *t*: $(\lambda n.\ \mathit{if}\ n = 0\ \mathit{then}\ v\ \mathit{else}\ ((\lambda i.\ \Xi\ (i - 1))(0 := v))\ (n{+}1)) = \Xi(0 := v)$ **for** v **and** $\Xi :: \mathit{nat} \Rightarrow {'a}$
    by *auto*
  have *SetFormulaPredicate* $(\lambda\ \Xi.\ (\Xi\ 0) = (\Xi\ 1) \wedge P\ (\lambda\ n.\ \Xi\ (n{+}1)))$
    unfolding *logsimps* **by** (*rule SFP-rules*)+ (*rule transform-variables*[*OF sfp*])
  from *replp*[*rule-format*, *OF this*, *of* $\lambda\ i.\ \Xi\ (i - 1)\ x$]
  show $\exists z.\ \forall u.\ (u\ \varepsilon\ z) = (u\ \varepsilon\ x \wedge P\ (\Xi(0 := u)))$
    by (*simp del*: *One-nat-def add*: *t*)
qed

end

Separation does not follow from replacement without the empty set axiom. We show it by constructing a counter-model.

theorem *not-repl-ext-implies-separ*: $\neg$ ($\forall$ *mem*. *L-repl mem* $\wedge$ *L-setext mem* $\longrightarrow$ *L-sep mem*)
proof–
  define *loop* :: $'a \Rightarrow {'a} \Rightarrow \mathit{bool}$ **where** *loop* $\equiv \lambda\ x\ y.\ x = y$
  interpret *eq-mem* : *set-signature loop*.
  have *loop-ext*: *L-setext loop*
    by (*unfold-locales*, *unfold loop-def*) *force*
  have *loop-repl*: *L-repl loop*
  proof (*unfold-locales*, *rule impI*)
    fix $P :: (\mathit{nat} \Rightarrow {'a}) \Rightarrow \mathit{bool}$ **and** Ξ
    assume $\forall u.\ \exists!v.\ P\ (\Xi(0 := u,\ 1 := v))$
    then show $\forall x.\ \exists z.\ \forall v.\ \mathit{loop}\ v\ z = (\exists u.\ \mathit{loop}\ u\ x \wedge P\ (\Xi(0 := u,\ 1 := v)))$
      unfolding *loop-def* **by** *metis*
  qed
  have *SP*: *eq-mem.SetFormulaPredicate* ($\lambda\ \Xi.$ *False*)
    by *simp*
  have $\neg$ *L-sep loop*
    unfolding *L-sep-def* **using** *loop-def SP* **by** *blast*
  show *?thesis*
    using ‹$\neg$ *L-sep loop*› *loop-ext loop-repl* **by** *auto*
qed

locale *L-setext-empty-setsuc-repl* = *L-setext* + *L-empty* + *L-setsuc* + *L-repl*

begin

sublocale *L-setext-empty-setsuc*
  by *unfold-locales*

sublocale *L-setext-empty-repl*
  by *unfold-locales*

lemma *tarski-setsuc-tarski*: **assumes** *tarski-fin x* **shows** *tarski-fin* (*setsucM x y*)
  unfolding *tarski-fin-def*
proof (*rule*, *rule*, *rule*)
  — fix a system w for which we want to find the maximum
  fix w **assume** w: $\forall z.\ z\ \varepsilon\ w \longrightarrow z \subseteq_M$ *setsucM x y* **and** $\exists\ z.\ z\ \varepsilon\ w$
    — remove y from each element of w, call the result w'
  let *?P* $= \lambda\ \Xi.\ \forall\ a.\ a\ \varepsilon\ \Xi\ 1 \longleftrightarrow a\ \varepsilon\ \Xi\ 0 \wedge a \neq \Xi\ 2$
  have *sfp*: *SetFormulaPredicate* ($\lambda\Xi.\ \forall v.\ (v\ \varepsilon\ \Xi\ 1) = (v\ \varepsilon\ \Xi\ 0 \wedge v \neq \Xi\ 2)$)
    unfolding *logsimps* **by** (*rule SFP-rules*)+
  have *sfp'*: *SetFormulaPredicate* ($\lambda\Xi.\ \Xi\ 0 \neq \Xi\ 1$)
      by (*rule SFP-rules*)+
  have *aux*: $(\Xi(2{::}nat := y,\ 0 := u,\ 1 := v))\ 0 = u$
      $(\Xi(2 := y,\ 0 := u,\ 1 := v))\ 1 = v$
      $(\Xi(2 := y,\ 0 := u,\ 1 := v))\ 2 = y$ **for** $u\ v\ \Xi$
    by *simp-all*
  have $\exists\ w'.\ \forall\ z.\ z\ \varepsilon\ w' \longleftrightarrow (\exists\ u.\ u\ \varepsilon\ w \wedge (\forall\ v.\ v\ \varepsilon\ z \longleftrightarrow v\ \varepsilon\ u \wedge v \neq y))$

   by (*rule replp*[*rule-format*, *of ?P undefined*(*2*:= *y*) *w*, *unfolded aux*], *fact*)
   (*unfold setext*, *simp*)
  then obtain w' **where** w': $\forall\ z.\ z\ \varepsilon\ w' \longleftrightarrow (\exists\ u.\ u\ \varepsilon\ w \wedge (\forall\ v.\ v\ \varepsilon\ z \longleftrightarrow v\ \varepsilon$
$u \wedge v \neq y))$
    by *blast*
  — get the maximum of w'
  obtain *wmax'* **where** *wmax'*: *wmax'* $\varepsilon\ w' \wedge (\nexists w.\ w\ \varepsilon\ w' \wedge$ *wmax'* $\subset_M w)$
  proof (*rule exE*[*OF assms*[*unfolded tarski-fin-def*, *rule-format*, *of w'*], *of thesis*])

    show $z \subseteq_M x$ **if** $z\ \varepsilon\ w'$ **for** z
      using ‹$z\ \varepsilon\ w'$›[*unfolded w'*[*rule-format*, *of z*]] w **unfolding** *subsetM-def*
setsuc-def' **by** *auto*
    show $\exists x.\ x\ \varepsilon\ w'$
    proof–
      obtain u **where** $u\ \varepsilon\ w$
        using ‹$\exists\ z.\ z\ \varepsilon\ w$› **by** *blast*
      from *sep*[*rule-format*, *OF sfp'*, *of u undefined*(*1*:=*y*)]
      obtain u' **where** u': $\forall\ v.\ v\ \varepsilon\ u' \longleftrightarrow v\ \varepsilon\ u \wedge v \neq y$
        by *auto*
      show $\exists x.\ x\ \varepsilon\ w'$
        unfolding *w'*[*rule-format*] **using** ‹$u\ \varepsilon\ w$› u' **by** *blast*
    qed
  qed
  have $\neg\ y\ \varepsilon$ *wmax'* *wmax'* $\varepsilon\ w'\ \nexists w.\ w\ \varepsilon\ w' \wedge$ *wmax'* $\subset_M w$
    using w' *wmax'* **by** *blast*+
  — Either *wmax'* or *setsucM wmax' y* is the desired maximum of w
  show $\exists z.\ z\ \varepsilon\ w \wedge (\nexists v.\ v\ \varepsilon\ w \wedge z \subset_M v)$
  proof(*cases*)

assume *setsucM wmax′ y ε w*
have ($\nexists$ *v. v ε w* ∧ *setsucM wmax′ y* $\subset_M$ *v*)
proof
assume ∃ *v. v ε w* ∧ *setsucM wmax′ y* $\subset_M$ *v*
then obtain *v* **where** *v ε w* **and** *setsucM wmax′ y* $\subset_M$ *v*
by *blast*
from *sep*[*rule-format*, *OF sfp′*, *of v undefined*(*1*:=*y*)]
obtain *v′* **where** *v′*: ⋀ *t. t ε v′* ⟷ *t ε v* ∧ *t* ≠ *y*
by *auto*
have *v′ ε w′*
using ‹*v ε w*› *v′ w′* **by** *auto*
have *wmax′* $\subset_M$ *v′*
using ‹*setsucM wmax′ y* $\subset_M$ *v*› ‹¬ *y ε wmax′*› **unfolding** *set-defs setext*[*of wmax′*] *setext*[*of - v*] *v′* **by** *blast*
show *False*
using *wmax′* ‹*v′ ε w′*› ‹*wmax′* $\subset_M$ *v′*› **by** *blast*
qed
thus *?thesis*
using ‹*setsucM wmax′ y ε w*› **by** *blast*
next
assume ¬ *setsucM wmax′ y ε w*
hence *wmax′ ε w*
proof–
obtain *wmax* **where** *wmax ε w* **and** *wmax*: ∀ *v*. (*v ε wmax′*) = (*v ε wmax* ∧ *v* ≠ *y*)
using ‹*wmax′ ε w′*›[*unfolded w′*[*rule-format*, *of wmax′*]] ‹¬ *y ε wmax′*› **by** *blast*
have ¬ *y ε wmax*
proof
assume *y ε wmax*
hence *wmax* = *setsucM wmax′ y*
unfolding *setext*[*of wmax*] *set-defs wmax*[*rule-format*] **by** *blast*
then show *False*
using ‹¬ *setsucM wmax′ y ε w*› ‹*wmax ε w*› **by** *blast*
qed
hence *wmax* = *wmax′*
using *setext wmax* **by** *blast*
then show *wmax′ ε w*
using ‹*wmax ε w*› **by** *blast*
qed
have ($\nexists$ *v. v ε w* ∧ *wmax′* $\subset_M$ *v*)
proof
assume ∃ *v. v ε w* ∧ *wmax′* $\subset_M$ *v*
then obtain *v* **where** *v ε w* **and** *wmax′* $\subseteq_M$ *v* **and** *wmax′* ≠ *v*
unfolding *proper-subset-def′* **by** *blast*
obtain *v′* **where** *v′*: ⋀ *t. t ε v′* ⟷ *t ε v* ∧ *t* ≠ *y*
using *sep*[*rule-format*, *OF sfp′*, *of v undefined*(*1*:=*y*)] **by** *auto*
have *v′ ε w′*
using ‹*v ε w*› *v′ w′* **by** *auto*

have $wmax' \subseteq_M v'$
using ‹$wmax' \subseteq_M v$› ‹$\neg\ y\ \varepsilon\ wmax'$› **unfolding** *setext*[*of - v*] v' *set-defs* **by** *blast*
hence $wmax' = v'$
using *wmax'* ‹$v'\ \varepsilon\ w'$› **unfolding** *proper-subset-def'* **by** *blast*
have $v = setsucM\ wmax'\ y$
using ‹$wmax' \neq v$› **unfolding** ‹$wmax' = v'$› *setext*[*of v*] *setext*[*of - v*] *set-defs* v' **by** *blast*
then show *False*
using ‹$\neg\ setsucM\ wmax'\ y\ \varepsilon\ w$› ‹$v\ \varepsilon\ w$› **by** *blast*
qed
thus *?thesis*
using ‹$wmax'\ \varepsilon\ w$› **by** *blast*
qed
qed

end

1.3.10 Powerset

locale *L-power* = *set-signature* +
assumes
power: $\forall\ x.\ \exists\ y.\ (\forall\ u.\ u\ \varepsilon\ y \longleftrightarrow u \subseteq_M x)$

locale *L-setext-empty-power* = *L-setext* + *L-empty* + *L-power*

begin

sublocale *L-setext-empty*
by *unfold-locales*

lemma *powerset-def'*[*set-defs*]: $u\ \varepsilon\ (\mathfrak{P}\ x) \longleftrightarrow u \subseteq_M x$
using *collect-def-ex*[*OF power*[*rule-format*], *folded powersetM-def*].

lemma $\emptyset\ \varepsilon\ \mathfrak{P}\ \emptyset$
using *empty-is-subset powerset-def'* **by** *blast*

lemma *set-one-mem* [*simp*]: $u\ \varepsilon\ \mathfrak{P}\ \emptyset \longleftrightarrow u = \emptyset$
unfolding *powerset-def' subsetM-def emptyset-def'* **by** *force*

lemma *set-one-def*: $\mathfrak{P}\ \emptyset = \{\emptyset\}_M$
unfolding *singletonM-def* **using** *collect-def'*[*of* $\mathfrak{P}\ \emptyset$] **unfolding** *set-one-mem* **by** *force*

lemma *set-two-mem*: $u\ \varepsilon\ \mathfrak{P}\ (\mathfrak{P}\ \emptyset) \longleftrightarrow u = \emptyset \lor u = \{\emptyset\}_M$
unfolding *powerset-def' set-one-def*
proof
show $u = \emptyset \lor u = \{\emptyset\}_M$ **if** $u \subseteq_M \{\emptyset\}_M$
using *that*[*unfolded subsetM-def*] *emptyset-def'*

unfolding *setext*[*of* - $\{\emptyset\}_M$] *set-one-mem*[*unfolded set-one-def*] **by** *blast*
show $u = \emptyset \vee u = \{\emptyset\}_M \Longrightarrow u \subseteq_M \{\emptyset\}_M$
using *empty-is-subset subsetM-refl* **by** *blast*
qed

end

locale *L-setext-empty-power-repl* = *L-setext* + *L-empty* + *L-power* + *L-repl*

— Subsumes many already existing locales. In particular, we obtain set successor from powerset and replacement

begin

sublocale *L-setext-empty-repl*
by *unfold-locales*

sublocale *L-setext-empty-power*
by *unfold-locales*

sublocale *L-setext-empty-setsuc*
proof (*unfold-locales*, *rule allI*, *rule allI*)
fix *x y*
let $?P = (\lambda\, \Xi.\ ((\exists\ q.\ q\ \varepsilon\ (\Xi\ (0::nat))) \wedge (\forall\ q.\ (q\ \varepsilon\ (\Xi\ 0) \longleftrightarrow q = (\Xi\ 1))) \vee (\Xi\ 1 = \Xi\ 2 \wedge \neg(\exists\ z.\ \forall\ q.\ (q\ \varepsilon\ (\Xi\ 0) \longleftrightarrow q = z)))))$
have *sfp*: *SetFormulaPredicate ?P*
unfolding *logsimps set-defs* **by** (*rule SFP-rules*)+
have *func*: $\forall\, u\ v\ w.$
$((\exists\, q.\ q\ \varepsilon\ u) \wedge (\forall\, q.\ (q\ \varepsilon\ u) \longleftrightarrow (q = v)) \vee v = y \wedge (\nexists\, z.\ \forall\, q.\ (q\ \varepsilon\ u) \longleftrightarrow (q = z))) \wedge$
$((\exists\, q.\ q\ \varepsilon\ u) \wedge (\forall\, q.\ (q\ \varepsilon\ u) \longleftrightarrow (q = w)) \vee w = y \wedge (\nexists\, z.\ \forall\, q.\ (q\ \varepsilon\ u) \longleftrightarrow (q = z))) \longrightarrow$
$v = w$
by *blast*
have *aux*: $(undefined(2 :: nat := y,\ 0 := u,\ 1 := v))\ 0 = u$ $(undefined(2 :: nat := y,\ 0 := u,\ 1 := v))\ 1 = v$ $(undefined(2 :: nat := y,\ 0 := u,\ 1 := v))\ 2 = y$ **for** *u v*
by *simp-all*
have $\exists\, z.\ \forall\, v.\ (v\ \varepsilon\ z) =$
$(\exists\, u.\ u\ \varepsilon\ \mathfrak{P}\ x \wedge ((\exists\, q.\ q\ \varepsilon\ u) \wedge (\forall\, q.\ (q\ \varepsilon\ u) = (q = v)) \vee v = y \wedge (\forall\, z.\ \exists\, q.\ (q\ \varepsilon\ u) = (q \neq z))))$
using *replp*[*OF sfp*, *simplified*, *of undefined*(*2*:=*y*), *unfolded fun-upd-same*, *OF func*[*simplified*]] **by** *simp*
— singletons from $\mathfrak{P}\ x$ are replaced by their elements, other sets by y
then obtain *s* **where** *s*: $\forall\, v.\ (v\ \varepsilon\ s) =$
$(\exists\, u.\ u\ \varepsilon\ \mathfrak{P}\ x \wedge ((\exists\, q.\ q\ \varepsilon\ u) \wedge (\forall\, q.\ (q\ \varepsilon\ u) = (q = v)) \vee v = y \wedge (\forall\, z.\ \exists\, q.\ (q\ \varepsilon\ u) = (q \neq z))))$
by *auto*
have $v\ \varepsilon\ s \longleftrightarrow (v\ \varepsilon\ x \vee v = y)$ **for** *v*

proof
show $v\ \varepsilon\ s \Longrightarrow v\ \varepsilon\ x \lor v = y$
unfolding *s*[*rule-format*] *powerset-def'*[*unfolded subsetM-def*] **by** *blast*
show $v\ \varepsilon\ x \lor v = y \Longrightarrow v\ \varepsilon\ s$
proof (*erule disjE*)
assume $v\ \varepsilon\ x$
obtain *singv* **where** *singv*: $singv\ \varepsilon\ \mathfrak{P}\ x\ \forall a.\ (a\ \varepsilon\ singv) \longleftrightarrow a = v$
proof–
have *sfp*: *SetFormulaPredicate* $(\lambda\ \Xi.\ \Xi\ 0 = \emptyset \land \Xi\ 1 = \Xi\ 2)$
unfolding *logsimps set-defs* **by** (*rule SFP-rules*)+
then obtain *singv* **where** $\forall a.\ (a\ \varepsilon\ singv) \longleftrightarrow a = v$
using *replp*[*OF sfp*, *rule-format*, *of undefined*($2 := v$) $\mathfrak{P}\ \emptyset$, *simplified*] **by** *blast*
then have $singv\ \varepsilon\ \mathfrak{P}\ x$
using ‹$v\ \varepsilon\ x$› **by** (*simp add*: *powerset-def' subsetM-def*)
from *that*[*OF this* ‹$\forall a.\ (a\ \varepsilon\ singv) \longleftrightarrow a = v$›]
show *thesis*.
qed
then show $v\ \varepsilon\ s$
unfolding *s*[*rule-format*] **by** *blast*
next
assume $v = y$
hence $\emptyset\ \varepsilon\ \mathfrak{P}\ x \land \emptyset \subseteq_M x \land (v = y \land (\nexists z.\ \forall q.\ (q\ \varepsilon\ \emptyset) = (q = z)))$
unfolding *set-defs* **by** *force*
then show $v\ \varepsilon\ s$
unfolding *s*[*rule-format*] **by** *blast*
qed
qed
then show $\exists z.\ \forall u.\ (u\ \varepsilon\ z) = (u\ \varepsilon\ x \lor u = y)$
by *blast*
qed

sublocale *L-setext-empty-setsuc-repl*
by *unfold-locales*

end

1.3.11 Union

locale *L-union* = *set-signature* +
assumes *union*: $\forall\ x.\ \exists\ y. \forall\ v.\ v\ \varepsilon\ y \longleftrightarrow (\exists\ u.\ u\ \varepsilon\ x \land v\ \varepsilon\ u)$

locale *L-setext-union* = *L-setext* + *L-union*

begin

lemma *union-def'*[*set-defs*]: $u\ \varepsilon\ \bigcup_M x \longleftrightarrow (\exists\ v.\ v\ \varepsilon\ x \land u\ \varepsilon\ v)$
using *collect-def-ex*[*OF union*[*rule-format*, *of x*], *folded unionM-def*[*of x*]].

lemma *union-trans-trans*: **assumes** $\forall\ v.\ v\ \varepsilon\ x \longrightarrow transM\ v$
  shows $transM\ (\bigcup_M\ x)$
  using *assms* **unfolding** *transM-def union-def′ subsetM-def* **by** *blast*

end

locale *L-setext-sep-binunion-ts* = *L-setext* + *L-sep* + *L-binunion* + *L-ts*

begin

sublocale *L-setext-binunion*
  by *unfold-locales*

sublocale *L-setext-sep-ts*
  by *unfold-locales*

lemma *trans-union*: $transM\ u \Longrightarrow transM\ v \Longrightarrow transM\ (u \cup_M\ v)$
  by (*simp add*: *binunion-def′ subsetM-def transM-def*)

lemma *leastTS-union*: $least\text{-}tsM\ (u \cup_M\ v) = least\text{-}tsM\ u \cup_M\ least\text{-}tsM\ v$
  unfolding *setext least-ts-def′ binunion-def′*
proof (*rule allI*, *rule iffI*)
  fix z
  assume *trans*: $\forall x.\ transM\ x \wedge u \cup_M\ v \subseteq_M\ x \longrightarrow z\ \varepsilon\ x$
  show $(\forall x.\ transM\ x \wedge u \subseteq_M\ x \longrightarrow z\ \varepsilon\ x) \vee (\forall x.\ transM\ x \wedge v \subseteq_M\ x \longrightarrow z\ \varepsilon\ x)$
  proof (*rule disjCI*)
    assume $\neg\ (\forall x.\ transM\ x \wedge v \subseteq_M\ x \longrightarrow z\ \varepsilon\ x)$
    then obtain x **where** $transM\ x\ v \subseteq_M\ x\ \neg\ z\ \varepsilon\ x$
      by *blast*
    show $\forall x.\ transM\ x \wedge u \subseteq_M\ x \longrightarrow z\ \varepsilon\ x$
    proof (*rule allI*, *rule impI*)
      fix y
      assume $transM\ y \wedge u \subseteq_M\ y$
      with *trans-union*[*OF* ‹*transM x*›, *of y*]
      have $transM\ (x \cup_M\ y) \wedge u \cup_M\ v \subseteq_M\ x \cup_M\ y$
        using ‹$v \subseteq_M\ x$› *binunion-def′ subsetM-def* **by** *auto*
      from *trans*[*rule-format*, *OF this*]
      show $z\ \varepsilon\ y$
        using ‹$\neg\ z\ \varepsilon\ x$› **unfolding** *binunion-def′* **by** *blast*
    qed
  qed
next
  fix z
  assume *or*: $(\forall x.\ transM\ x \wedge u \subseteq_M\ x \longrightarrow z\ \varepsilon\ x) \vee (\forall x.\ transM\ x \wedge v \subseteq_M\ x \longrightarrow z\ \varepsilon\ x)$
  show $\forall y.\ transM\ y \wedge u \cup_M\ v \subseteq_M\ y \longrightarrow z\ \varepsilon\ y$
  proof (*rule allI*, *rule impI*)
    fix y
    assume $transM\ y \wedge u \cup_M\ v \subseteq_M\ y$

hence *ins*: $transM\ y \wedge u \subseteq_M\ y\ transM\ y \wedge v \subseteq_M\ y$
unfolding *subsetM-def binunion-def′* **by** *auto*
show $z\ \varepsilon\ y$
by (*rule disjE*[*OF or*, *rule-format*]) (*use ins* **in** *blast*)+
qed
qed

end

The following locale is called Elementary Set Theory (EST) in Baratella and Ferro, "A theory of Sets with the Negation of the Axiom of Infinity", 1993.

locale *L-setext-empty-union-repl-pair* = *L-setext* + *L-empty* + *L-union* + *L-repl* + *L-pair*

begin

— binunion is the union of a pair
sublocale *L-setext-pair-binunion*
proof (*unfold-locales*, *rule allI*, *rule allI*)
fix $x\ y$
from *pair*[*rule-format*, *of x y*]
obtain *pair* **where** *pair*: $\forall v.\ (v\ \varepsilon\ pair) = (v = x \vee v = y)$
by *blast*
show $\exists z.\ \forall v.\ (v\ \varepsilon\ z) = (v\ \varepsilon\ x \vee v\ \varepsilon\ y)$
using *union*[*rule-format*, *of pair*] **unfolding** *pair*[*rule-format*] **by** *simp*
qed

end

The locale *L-setext-empty-union-repl-pair* (i.e., the fragment EST also used by Baratella and Ferro [1]) is the right context in which to show that the schema of regularity yields existence of transitive closures (least transitive supersets)

locale *L-setext-empty-union-repl-pair-regsch* = *L-setext* + *L-empty* + *L-union* + *L-repl* + *L-pair* + *L-regsch*

begin

sublocale *L-setext-empty-union-repl-pair*
by *unfold-locales*

sublocale *L-setext-empty-repl*
by *unfold-locales*

sublocale *L-setext-union*
by *unfold-locales*

lemma *transitive-closure-ex*:

shows $\forall\ x.\ \exists\ z.\ x \subseteq_M z \wedge$ *transM* $z \wedge (\forall\ u.\ u\ \varepsilon\ z \longleftrightarrow (\forall\ t.\ x \subseteq_M t \wedge$ *transM* $t \longrightarrow u\ \varepsilon\ t))$
(**is** $\forall\ x.\ \exists\ z.$ *?TC x z* **is** $\forall\ x.$ *?hasTC x*)
proof
fix x' :: *'a*
have *ind-rule*: *SetFormulaPredicate* ($\lambda\Xi$. *?hasTC*(Ξ *0*)) $\Longrightarrow$ ($\bigwedge x.$ ($\bigwedge y.$ $y\ \varepsilon\ x \Longrightarrow$ *?hasTC y*) $\Longrightarrow$ *?hasTC x*) $\Longrightarrow$ *?hasTC* x'
using *regsch-epsind*[*rule-format*, *of* $\lambda\ \Xi$. *?hasTC*(Ξ *0*) *undefined* x', *unfolded fun-upd-same*] **by** *argo*
show $\exists z.\ x' \subseteq_M z \wedge$ *transM* $z \wedge (\forall u.\ (u\ \varepsilon\ z) = (\forall t.\ x' \subseteq_M t \wedge$ *transM* $t \longrightarrow u\ \varepsilon\ t))$
proof (*rule ind-rule*)
fix *x*
define *TC* **where** *TC = ?TC — TC x z*
have *sfp-tc*: *SetFormulaPredicate* ($\lambda\Xi$. *TC* (Ξ *0*) (Ξ *1*))
unfolding *TC-def set-defs logsimps* **by** (*rule SFP-rules*)+
have *tc-fun*: $\forall$ *u v w. TC u v* $\wedge$ *TC u w* $\longrightarrow$ *v = w*
unfolding *TC-def setext* **by** *blast*
show *SetFormulaPredicate* ($\lambda\Xi$. *?hasTC*(Ξ *0*))
unfolding *TC-def set-defs logsimps* **by** (*rule SFP-rules*)+
assume *IP*: *?hasTC y* **if** $y\ \varepsilon\ x$ **for** *y*
define *tcs* **where** *tcs* = $\bigcup_M$ (*replaceM TC x*)
have *repl-x*: $\forall v.$ ($v\ \varepsilon$ *replaceM TC x*) $\longleftrightarrow$ ($\exists u.\ u\ \varepsilon\ x \wedge$ *TC u v*)
by (*rule replace-def'*[*OF sfp-tc*, *of - 0 1 x*, *simplified fun-upd-same fun-upd-other*]) *fact*
hence *transM tcs*
unfolding *TC-def tcs-def union-def' transM-def subsetM-def* **by** *blast*
hence *tcs-mem*: $w\ \varepsilon$ *tcs* $\longleftrightarrow$ ($\exists\ y.\ y\ \varepsilon\ x \wedge (\exists\ v.$ *TC y v* $\wedge\ w\ \varepsilon\ v$)) **for** *w*
unfolding *tcs-def* **using** *union-def' repl-x* **by** *auto*
show *?hasTC x*
proof (*rule exI*[*of - tcs* $\cup_M$ *x*], *fold conj-assoc*, *rule conjI*, *rule conjI*)
show $x \subseteq_M$ *tcs* $\cup_M$ *x*
unfolding *tcs-def set-defs* **by** *blast*
show *trans*: *transM* (*tcs* $\cup_M$ *x*)
unfolding *transM-def binunion-def'*
proof (*rule allI*, *rule impI*, *erule disjE*)
show $y \subseteq_M$ *tcs* $\cup_M$ *x* **if** $y\ \varepsilon$ *tcs* **for** *y*
using ‹*transM tcs*› *that* **unfolding** *transM-def binunion-def' subsetM-def* **by** *blast*
show $y \subseteq_M$ *tcs* $\cup_M$ *x* **if** $y\ \varepsilon\ x$ **for** *y*
unfolding *subsetM-def tcs-def binunion-def' union-def' repl-x*[*rule-format*]

proof (*rule allI*, *rule impI*, *rule disjI1*)
fix *z*
assume $z\ \varepsilon\ y$
obtain *w* **where** *TC y w*
unfolding *TC-def* **using** *IP*[*OF* ‹$y\ \varepsilon\ x$›] **by** *blast*
hence $z\ \varepsilon\ w$
using ‹$z\ \varepsilon\ y$› **unfolding** *TC-def subsetM-def* **by** *blast*

hence $(y \varepsilon x \wedge TC\ y\ w) \wedge z \varepsilon w$
using ‹$y \varepsilon x$› ‹$TC\ y\ w$› **by** *blast*
then show $\exists v. (\exists u. u \varepsilon x \wedge TC\ u\ v) \wedge z \varepsilon v$
by *blast*
qed
qed
show $\forall u. (u \varepsilon tcs \cup_M x) \longleftrightarrow (\forall t. x \subseteq_M t \wedge transM\ t \longrightarrow u \varepsilon t)$
proof (*rule*, *rule*, *rule*, *rule*)
— *tcs* $\cup_M$ *x* is the least transitive superset
fix *u t*
assume $u \varepsilon tcs \cup_M x\ x \subseteq_M t \wedge transM\ t$
consider $\exists y. y \varepsilon x \wedge (\exists w. TC\ y\ w \wedge u \varepsilon w) \mid u \varepsilon x$
using ‹$u \varepsilon tcs \cup_M x$› **unfolding** *set-defs tcs-def repl-x*[*rule-format*] **by** *blast*
then show $u \varepsilon t$
proof (*cases*)
assume $\exists y. y \varepsilon x \wedge (\exists w. TC\ y\ w \wedge u \varepsilon w)$
then obtain *y w* **where** *y-w*: $y \varepsilon x\ TC\ y\ w\ u \varepsilon w$
by *blast*
have $y \subseteq_M t$
using ‹$y \varepsilon x$› ‹$x \subseteq_M t \wedge transM\ t$› **unfolding** *transM-def subsetM-def* **by** *blast*
then show $u \varepsilon t$
using ‹$x \subseteq_M t \wedge transM\ t$› *y-w* **unfolding** *TC-def* **by** *blast*
qed (*use* ‹$x \subseteq_M t \wedge transM\ t$› *subsetM-def* **in** *blast*) — the trivial case $u \varepsilon x$
qed (*use* ‹$x \subseteq_M tcs \cup_M x$› *local.trans* **in** *blast*) — the trivial implication: *tcs* $\cup_M$ *x* is a transitive superset
qed
qed
qed

sublocale *L-setext-sep-reg-ts*
using *transitive-closure-ex* **by** *unfold-locales blast*

end

locale *L-setext-empty-power-union-repl* = *L-setext* + *L-empty* + *L-power* + *L-union* + *L-repl*

— ZF without infinity and regularity. Axioms enabling reasoning about functions, cardinality, and therefore about Dedekind finiteness.

begin

sublocale *L-setext-empty-power-repl*
by *unfold-locales*

sublocale *L-setext-setsuc-sep*
by *unfold-locales*

sublocale *L-setext-union*
by *unfold-locales*

sublocale *L-setext-empty-union-repl-pair*
by *unfold-locales*

lemma *ordered-pair-mem*: **assumes** $v\ \varepsilon\ x\ v'\ \varepsilon\ y$ **shows** $\langle v,v'\rangle\ \varepsilon\ \mathfrak{P}\ (\mathfrak{P}\ (x \cup_M\ y))$
proof–
have $\{v\}_M\ \varepsilon\ \mathfrak{P}\ (x \cup_M\ y)\ \{v,v'\}_M\ \varepsilon\ \mathfrak{P}\ (x \cup_M\ y)$
using ‹$v\ \varepsilon\ x$› ‹$v'\ \varepsilon\ y$› **unfolding** *set-defs* **by** *blast+*
thus *?thesis*
unfolding *setext*[*of* - $\mathfrak{P}$ -] *powerset-def'*[*of* - $\mathfrak{P}$ -]
pair-def' subsetM-def ordered-pairM-def **by** *blast*
qed

lemma *ordered-pair-mem-union*: **assumes** $\langle u,v\rangle\ \varepsilon\ r$ **shows** $u\ \varepsilon\ \bigcup_M\ (\bigcup_M\ r)\ v\ \varepsilon\ \bigcup_M\ (\bigcup_M\ r)$
proof–
have $(\langle u,v\rangle\ \varepsilon\ r \wedge \{u,v\}_M\ \varepsilon\ \langle u,v\rangle) \wedge u\ \varepsilon\ \{u,v\}_M\ (\langle u,v\rangle\ \varepsilon\ r \wedge \{u,v\}_M\ \varepsilon\ \langle u,v\rangle) \wedge v\ \varepsilon\ \{u,v\}_M$
using *assms* **unfolding** *pair-def' ordered-pairM-def* **by** *blast+*
then show $u\ \varepsilon\ \bigcup_M\ (\bigcup_M\ r)\ v\ \varepsilon\ \bigcup_M\ (\bigcup_M\ r)$
unfolding *union-def'* **by** *blast+*
qed

lemma *car-prod-ex*: $\exists\ z.\ \forall\ u.\ u\ \varepsilon\ z \longleftrightarrow (\exists\ v\ v'.\ v\ \varepsilon\ x \wedge v'\ \varepsilon\ y \wedge u = \langle v,v'\rangle)$
proof–
have *SetFormulaPredicate* $(\lambda\Xi.\ \exists\, v\ v'.\ v\ \varepsilon\ \Xi\ \mathit{1} \wedge v'\ \varepsilon\ \Xi\ \mathit{2} \wedge \Xi\ \mathit{0} = \langle v,v'\rangle)$
unfolding *set-defs logsimps* **by** (*rule SFP-rules*)+
from *sep*[*rule-format*, *of* - $\mathfrak{P}\ (\mathfrak{P}\ (x \cup_M\ y))$ *undefined*(*1*:=*x*,*2*:=*y*), *OF this*, *simplified*]
show *?thesis*
using *ordered-pair-mem* **by** *metis*
qed

lemma *car-prod-def'*: $u\ \varepsilon\ x \times_M\ y \longleftrightarrow (\exists\ v\ v'.\ v\ \varepsilon\ x \wedge v'\ \varepsilon\ y \wedge u = \langle v,v'\rangle)$
unfolding *cartesian-productM-def*[*rule-format*]
using *collect-def-ex*[*OF car-prod-ex*, *of* - *x y*] *ordered-pair-mem* **by** *blast*

lemma *rel-inv-def'*: $u\ \varepsilon$ *rel-inverseM* $r \longleftrightarrow (\exists\ a\ b.\ \langle a,b\rangle\ \varepsilon\ r \wedge u = \langle b,a\rangle)$
proof–
have *SR*: *SetRelation* $(\lambda u\ r.\ \exists\, a\ b.\ \langle a,b\rangle\ \varepsilon\ r \wedge u = \langle b,a\rangle)$
unfolding *SetRelation-def logsimps* **by** (*rule SFP-rules*)+
have *eq*: $(u\ \varepsilon\ \bigcup_M\ (\bigcup_M\ r) \times_M \bigcup_M\ (\bigcup_M\ r) \wedge (\exists\, a\ b.\ \langle a,b\rangle\ \varepsilon\ r \wedge u = \langle b,a\rangle))$
$\longleftrightarrow (\exists\, a\ b.\ \langle a,b\rangle\ \varepsilon\ r \wedge u = \langle b,a\rangle)$ **for** u
unfolding *car-prod-def'*[*rule-format*] **using** *ordered-pair-mem-union* **by** *blast*
show *?thesis*
unfolding *rel-inverseM-def*

using *collect-def-ex*[*of* λ *v*. ($\exists$ *a b*. $\langle a,b\rangle$ ε *r* $\wedge$ *v* = $\langle b,a\rangle$)]
sep-SR[*rule-format*,*OF SR*, *of* $\bigcup_M$ ($\bigcup_M$ *r*) $\times_M$ $\bigcup_M$ ($\bigcup_M$ *r*) *r*, *unfolded eq*]
by *simp*
qed

lemma *dom-def'*: *u* ε *domM r* $\longleftrightarrow$ ($\exists$ *v*. $\langle u,v\rangle$ ε *r*)
unfolding *domM-def*
proof (*rule collect-def-ex*)
have *aux*: ($\exists$ *v*. $\langle u,v\rangle$ ε *r*) $\longleftrightarrow$ ($\exists$ *x*. *x* ε *r* $\wedge$ ($\exists$ *w*. *x* = $\langle u,w\rangle$)) **for** *u*
by *blast*
show $\exists$ *w*. $\forall$ *u*. (*u* ε *w*) = ($\exists$ *v*. $\langle u,v\rangle$ ε *r*)
unfolding *aux*
by (*rule repl-SR*[*rule-format*, *of* λ *x u*. $\exists$ *w*. *x* = $\langle u,w\rangle$ *r*]) *force*+
qed

lemma *rng-def'*: *u* ε *rngM r* $\longleftrightarrow$ ($\exists$ *v*. $\langle v,u\rangle$ ε *r*)
unfolding *rngM-def*
proof (*rule collect-def-ex*)
have *aux*: ($\exists$ *v*. $\langle v,u\rangle$ ε *r*) $\longleftrightarrow$ ($\exists$ *x*. *x* ε *r* $\wedge$ ($\exists$ *w*. *x* = $\langle w,u\rangle$)) **for** *u*
by *blast*
show $\exists$ *w*. $\forall$ *u*. (*u* ε *w*) = ($\exists$ *v*. $\langle v,u\rangle$ ε *r*)
unfolding *aux*
by (*rule repl-SR*[*rule-format*, *of* λ *x u*. $\exists$ *w*. *x* = $\langle w,u\rangle$ *r*]) *force*+
qed

lemma *SFP-dom*[*simp*, *intro*]: *SetFormulaPredicate* (λ Ξ. Ξ *k* = *domM* (Ξ *l*))
unfolding *setext dom-def' logsimps* **by** (*rule SFP-rules*)+

lemma *SFP-rng*[*simp*, *intro*]: *SetFormulaPredicate* (λ Ξ. Ξ *k* = *rngM* (Ξ *l*))
unfolding *setext rng-def' logsimps* **by** (*rule SFP-rules*)+

lemmas[*SFP-rules*] = *SFP-rng*[*unfolded set-defs logsimps*] *SFP-dom*[*unfolded set-defs logsimps*]

lemma *SetRelation* (λ*x y*. $\exists$*f*. *one-oneM f* $\wedge$ *x* = *domM f* $\wedge$ *y* = *rngM f*)
(**is** *SetRelation* (λ *x y*. ($\exists$ *f*. *?P x y f*)))
unfolding *set-defs logsimps SetRelation-def* **by** (*rule SFP-rules*)+

lemma *SR-equiv*[*simp*]: *SetRelation* (λ *x y*. *x* $\approx_M$ *y*)
unfolding *SetRelation-def set-defs logsimps* **by** (*rule SFP-rules*)+

lemma *SP-dedekind*[*simp*]: *SetProperty dedekind-fin*
unfolding *SetProperty-def dedekind-fin-def one-oneM-def set-equivalent-def set-defs logsimps* **by** (*rule SFP-rules*)+

lemma *SR-card*: *SetRelation cardinality*
unfolding *set-defs logsimps SetRelation-def* **by** (*rule SFP-rules*)+

lemma *rel-inv-dom-rng*: *domM* (*rel-inverseM r*) = *rngM r rngM* (*rel-inverseM r*)

$=$ *domM r*
unfolding *setext*[*of - rngM -*] *setext*[*of - domM -*] *dom-def'* *rng-def'* *rel-inv-def'* *ordered-pair-unique* **by** *blast+*

lemma *rel-inv-rel*: *relationM f* $\Longrightarrow$ *relationM* (*rel-inverseM f*)
unfolding *relationM-def* *rel-inv-def'* **by** *blast*

lemma *one-one-inv*: *one-oneM f* $\Longrightarrow$ *one-oneM* (*rel-inverseM f*)
unfolding *one-oneM-def* *rel-inv-def'* *functionM-def* *ordered-pair-unique* **using** *rel-inv-rel* **by** *auto*

lemma *dedekind-setsuc-dedekind*: **assumes** *dedekind-fin x* **shows** *dedekind-fin* (*setsucM x t*)
proof (*cases* $t\ \varepsilon\ x$)
assume $\neg\ t\ \varepsilon\ x$
show *?thesis*
unfolding *dedekind-fin-def*
proof (*rule allI*, *rule impI*)
have *IH*: $u \subset_M x \Longrightarrow x \approx_M u \Longrightarrow$ *False* **for** u
using *assms* **unfolding** *dedekind-fin-def* **by** *blast*
show $\neg$ *setsucM x t* $\approx_M y$ **if** $y \subset_M$ *setsucM x t* **for** y
proof
assume *setsucM x t* $\approx_M y$
then obtain f **where** *f-bij*: *one-oneM f* **and** *f-dom*: *domM f* $=$ *setsucM x t* **and** *f-rng*: *rngM f* $= y$
unfolding *set-equivalent-def* **by** *force*
obtain *ft* **where** $\langle t, ft\rangle\ \varepsilon\ f$
using *f-dom*[*unfolded setext dom-def'*, *rule-format*, *of t*] **unfolding** *setsuc-def'* **by** *force*
— From this we construct a bijection g between strict subset y' of x and x There are two different constructions. Depending on whether removing the pair *fx*,x from f yields directly a bijection of x on its subset, or whether a modification is needed.
have $ft\ \varepsilon\ y$
using ‹*rngM f* $= y$› ‹$\langle t,ft\rangle\ \varepsilon\ f$› **unfolding** *setext rng-def'* **by** *blast*
consider $t\ \varepsilon\ y \wedge ft \neq t \mid y \subseteq_M x \vee ft = t$
using ‹$y \subset_M$ *setsucM x t*› **unfolding** *subsetM-def* *proper-subsetM-def* *setsuc-def'* **by** *blast*
then show *False*
proof (*cases*)
assume $y \subseteq_M x \vee ft = t$
— First construction just removes $\langle t,ft\rangle$
from *sep-SR*[*rule-format*, *OF SR-neq*]
obtain y' **where** y': $\forall\ u.\ u\ \varepsilon\ y' \longleftrightarrow u\ \varepsilon\ y \wedge u \neq ft$
by *presburger*
from *sep-SR*[*rule-format*, *OF SR-neq*]
obtain g **where** g: $\forall\ u.\ u\ \varepsilon\ g \longleftrightarrow u\ \varepsilon\ f \wedge u \neq \langle t,ft\rangle$
by *presburger*
show *False*

proof (*rule IH*)
show $y' \subset_M x$
using y' ‹$y \subseteq_M x \vee ft = t$›[*unfolded subsetM-def*] ‹$ft\ \varepsilon\ y$› *proper-subsetM-def* ‹$y \subset_M$ *setsucM* $x\ t$›[*unfolded proper-subsetM-def setsuc-def′ setext*[*of y*]] **by** *metis*
show $x \approx_M y'$
proof–
have *one-oneM g*
using ‹*one-oneM f*› *g* **unfolding** *one-oneM-def functionM-def relationM-def* **by** *blast*
have *domM* $g = x$
using *funD*[*OF conjunct1*[*OF f-bij*[*unfolded one-oneM-def*]] ‹$\langle t,ft\rangle\ \varepsilon\ f$›] *f-dom* ‹$\neg\ t\ \varepsilon\ x$›
unfolding *setext*[*of domM -*] *dom-def′* y'[*rule-format*] *g*[*rule-format*] *one-oneM-def functionM-def setsuc-def′ ordered-pair-unique*
by *auto*
have *rngM* $g = y'$
using *one-one-inj*[*OF f-bij* ‹$\langle t,ft\rangle\ \varepsilon\ f$›] *f-rng*
unfolding *setext*[*of rngM -*] *rng-def′* y'[*rule-format*] *g*[*rule-format*] *ordered-pair-unique*
by *blast*
show $x \approx_M y'$
unfolding *set-equivalent-def* **using** ‹*rngM* $g = y'$› ‹*domM* $g = x$› ‹*one-oneM g*› **by** *blast*
qed
qed
next
assume $t\ \varepsilon\ y \wedge ft \neq t$
then obtain *pt* **where** $\langle pt,t\rangle\ \varepsilon\ f$
using *f-rng*[*unfolded setext rng-def′*, *rule-format*, *of t*] **by** *blast*
hence $pt\ \varepsilon\ x$
using *f-dom*[*unfolded setext*[*of domM -*] *dom-def′ setsuc-def′*]
one-oneD3[*OF f-bij*, *rule-format*, *of t ft t*] ‹$\langle pt,t\rangle\ \varepsilon\ f$› ‹$\langle t,ft\rangle\ \varepsilon\ f$› ‹$t\ \varepsilon\ y \wedge ft \neq t$› **by** *blast*
then show *False*
proof–
— the second construction requires to modify the mapping *f* by adding $\langle fx,px\rangle$, not just to restrict *f*
from *sep-SR*[*rule-format*, *OF SR-neq*]
obtain y' **where** y': $\forall\ u.\ u\ \varepsilon\ y' \longleftrightarrow u\ \varepsilon\ y \wedge u \neq t$
by *force*
have *sfp*: *SetFormulaPredicate* ($\lambda\ \Xi.\ \Xi\ 0 = \langle \Xi\ 1,\Xi\ 2\rangle \vee (\Xi\ 0\ \varepsilon\ \Xi\ 4 \wedge \Xi\ 0 \neq \langle \Xi\ 3,\Xi\ 2\rangle \wedge \Xi\ 0 \neq \langle \Xi\ 1,\Xi\ 3\rangle)$)
unfolding *logsimps* **by** (*rule SFP-rules*)+
obtain *g* **where** *g*: $\forall\ u.\ u\ \varepsilon\ g \longleftrightarrow (u = \langle pt,ft\rangle \vee (u\ \varepsilon\ f \wedge u \neq \langle t,ft\rangle \wedge u \neq \langle pt,t\rangle))$
using *sep*[*rule-format*, *OF sfp*, *of setsucM f* ($\langle pt,ft\rangle$) *undefined*(*1*:=*pt*, *2*:=*ft*, *3*:= *t* ,*4*:=*f*), *simplified*] **by** *auto*
show *False*
proof (*rule IH*)

show $y' \subset_M x$
using ‹$y \subset_M$ *setsucM x t*› ‹$t \varepsilon y \wedge ft \neq t$› **unfolding** *proper-subsetM-def* y'[*rule-format*] *setsuc-def'* *setext*[*of y*] *setext*[*of y'*] **by** *blast*
show $x \approx_M y'$
proof−
have *one-oneM g*
by (*rule one-oneI*, *unfold g*[*rule-format*] *ordered-pair-unique*)
(*use one-oneD1*[*OF f-bij*] **in** *blast*,
use one-oneD2[*OF f-bij*] ‹$\langle t,ft\rangle \varepsilon f$› **in** *blast*,
use one-oneD3[*OF f-bij*] ‹$\langle pt,t\rangle \varepsilon f$› **in** *blast*)
have *domM g* = *x*
unfolding *setext*[*of domM* -] *dom-def'*
proof (*rule allI*, *rule iffI*)
fix *z* **assume** $\exists v.\ \langle z,v\rangle \varepsilon g$
show $z \varepsilon x$
using ‹$\exists v.\ \langle z,v\rangle \varepsilon g$› ‹$pt \varepsilon x$› *one-oneD3*[*OF f-bij*, *rule-format*, *of t ft* -] ‹$\langle t,ft\rangle \varepsilon f$› *f-dom*[*unfolded setext*[*of domM* -] *setsuc-def' dom-def'*] **unfolding** *ordered-pair-unique*
g[*rule-format*] **by** *blast*
next
fix *z* **assume** $z \varepsilon x$
then show $\exists v.\ \langle z,v\rangle \varepsilon g$
unfolding *g*[*rule-format*] **using** *f-dom*[*unfolded setext*[*of domM* -] *setsuc-def' dom-def'*] **by** (*cases z* = *pt*) (*use* ‹$\neg\ t \varepsilon x$› **in** *auto*)
qed
have *rngM g* = y'
unfolding *setext*[*of rngM* -] *rng-def'*
proof (*rule allI*, *rule iffI*)
fix *z* **assume** $\exists v.\ \langle v,z\rangle \varepsilon g$
show $z \varepsilon y'$
using
one-oneD2[*OF f-bij*]
‹$\exists v.\ \langle v,z\rangle \varepsilon g$› ‹$ft \varepsilon y$› ‹$t \varepsilon y \wedge ft \neq t$› ‹$\langle pt,t\rangle \varepsilon f$› ‹$\langle t,ft\rangle \varepsilon f$›
unfolding *f-rng*[*unfolded setext*[*of rngM* -] *rng-def'*, *rule-format*, *symmetric*, *of z*] y'[*rule-format*] *ordered-pair-unique g*[*rule-format*] **by** *blast*
next
fix *z* **assume** $z \varepsilon y'$
show $\exists v.\ \langle v,z\rangle \varepsilon g$
using ‹$z \varepsilon y'$›[*unfolded* y'[*rule-format*]]
unfolding *f-rng*[*unfolded setext*[*of rngM* -] *rng-def'*, *rule-format*, *symmetric*, *of z*] *g*[*rule-format*] *ordered-pair-unique* **by** *blast*
qed
show $x \approx_M y'$
unfolding *set-equivalent-def* **using** ‹*rngM g* = y'› ‹*domM g* = *x*› ‹*one-oneM g*› **by** *blast*
qed
qed
qed
qed

qed
qed
qed (*simp add*: *assms*)

lemma *set-equivalent-sym*: $x \approx_M y \longleftrightarrow y \approx_M x$
proof–
have $x \approx_M y \Longrightarrow y \approx_M x$ **for** x y
proof–
assume $x \approx_M y$
then obtain f **where** f : *one-oneM f* x = *domM f* y = *rngM f*
unfolding *set-equivalent-def* **by** *blast*
show $y \approx_M x$
unfolding *set-equivalent-def*
by (*rule exI*[*of - rel-inverseM f*]) (*simp add*: *one-one-inv*[*OF* ‹*one-oneM f*›] *f*(*2*,*3*) *rel-inv-dom-rng*)
qed
thus *?thesis*
by *blast*
qed

lemma *compose-def'*: $u\ \varepsilon\ f \circ_M g \longleftrightarrow (\exists\, a\ b\ c.\ \langle a,b\rangle\ \varepsilon\ g \wedge \langle b,c\rangle\ \varepsilon\ f \wedge u = \langle a,c\rangle)$
proof–
have *ex*: $\exists\, w.\ \forall\, u.\ (u\ \varepsilon\ w) = (u\ \varepsilon$ *domM g* $\times_M$ *rngM f* $\wedge (\exists\, a\ b\ c.\ \langle a,b\rangle\ \varepsilon\ g \wedge \langle b,c\rangle\ \varepsilon\ f \wedge u = \langle a,c\rangle))$
using *sep*[*rule-format*, *OF SFP-compose*[*of 1 2 0*], *of domM g* $\times_M$ *rngM f undefined*(*1*:=*g*, *2*:= *f*)]
by *simp*
have *aux*: $u\ \varepsilon$ *domM g* $\times_M$ *rngM f* $\wedge (\exists\, a\ b\ c.\ \langle a,b\rangle\ \varepsilon\ g \wedge \langle b,c\rangle\ \varepsilon\ f \wedge u = \langle a,c\rangle) \longleftrightarrow (\exists\, a\ b\ c.\ \langle a,b\rangle\ \varepsilon\ g \wedge \langle b,c\rangle\ \varepsilon\ f \wedge u = \langle a,c\rangle)$ **for** u
using *dom-def' rng-def' car-prod-def'* **by** *auto*
have $u\ \varepsilon\ f \circ_M g \longleftrightarrow (\exists\, a\ b\ c.\ \langle a,b\rangle\ \varepsilon\ g \wedge \langle b,c\rangle\ \varepsilon\ f \wedge u = \langle a,c\rangle)$ (**is** $u\ \varepsilon\ f \circ_M g \longleftrightarrow$ *?Q u*)
unfolding *composeM-def*
using *collect-def-ex*[*of* λ *u*. *?Q u*, *OF ex*[*unfolded aux*]]**.**
then show *?thesis*
unfolding *car-prod-def' dom-def'*[*of - g*] *rng-def'*[*of - f*] **by** *blast*
qed

lemma *rel-comp*: **assumes** *relationM f relationM g* **shows** *relationM* $(f \circ_M g)$
using *assms* **unfolding** *relationM-def compose-def'* **by** *blast*

lemma *fun-comp*: **assumes** *functionM f functionM g* **shows** *functionM* $(f \circ_M g)$
using *assms rel-comp* **unfolding** *functionM-def compose-def' ordered-pair-unique* **by** *blast*

lemma *one-one-comp*: **assumes** *one-oneM f one-oneM g* **shows** *one-oneM* $(f \circ_M g)$
using *assms fun-comp* **unfolding** *one-oneM-def compose-def' ordered-pair-unique* **by** *blast*

lemma *set-equivalent-trans*: **assumes** $x \approx_M y$ $y \approx_M z$ **shows** $x \approx_M z$
proof–
  obtain *f1* **where** *f1*: *one-oneM f1* **and** $x = domM\ f1$ $y = rngM\ f1$
    using ‹$x \approx_M y$› **unfolding** *set-equivalent-def* **by** *blast*
  obtain *f2* **where** *f2*: *one-oneM f2* **and** $y = domM\ f2$ $z = rngM\ f2$
    using ‹$y \approx_M z$› **unfolding** *set-equivalent-def* **by** *blast*
  have *one-oneM* $(f2 \circ_M f1)$
    using *f1 f2 one-one-comp* **by** *blast*
  have $x = domM\ (f2 \circ_M f1)$
    unfolding *dom-def′ setext* **unfolding** *compose-def′ ordered-pair-unique*
   proof (*rule allI*, *rule iffI*)
    fix *u* **assume** $\exists v\ a\ b\ c.\ \langle a,b\rangle\ \varepsilon\ f1 \wedge \langle b,c\rangle\ \varepsilon\ f2 \wedge u = a \wedge v = c$
    then show $u\ \varepsilon\ x$
      unfolding ‹$x = domM\ f1$›[*unfolded setext dom-def′*, *rule-format*] **by** *blast*
   next
    fix *u* **assume** $u\ \varepsilon\ x$
    then obtain *b* **where** ‹$\langle u,b\rangle\ \varepsilon\ f1$› $b\ \varepsilon\ y$
      unfolding ‹$x = domM\ f1$›[*unfolded setext dom-def′*, *rule-format*]
              ‹$y = rngM\ f1$›[*unfolded setext rng-def′*, *rule-format*] **by** *blast*
    then obtain *c* **where** ‹$\langle b,c\rangle\ \varepsilon\ f2$› $c\ \varepsilon\ z$
      unfolding ‹$y = domM\ f2$›[*unfolded setext dom-def′*, *rule-format*]
              ‹$z = rngM\ f2$›[*unfolded setext rng-def′*, *rule-format*] **by** *blast*
    show $\exists v\ a\ b\ c.\ \langle a,b\rangle\ \varepsilon\ f1 \wedge \langle b,c\rangle\ \varepsilon\ f2 \wedge u = a \wedge v = c$
      using ‹$\langle u,b\rangle\ \varepsilon\ f1$› ‹$\langle b,c\rangle\ \varepsilon\ f2$› **by** *auto*
   qed
  have $z = rngM\ (f2 \circ_M f1)$
    unfolding *rng-def′ setext* **unfolding** *compose-def′ ordered-pair-unique*
   proof (*rule allI*, *rule iffI*)
    fix *u* **assume** $\exists v\ a\ b\ c.\ \langle a,b\rangle\ \varepsilon\ f1 \wedge \langle b,c\rangle\ \varepsilon\ f2 \wedge v = a \wedge u = c$
    then show $u\ \varepsilon\ z$
      unfolding ‹$z = rngM\ f2$›[*unfolded setext rng-def′*, *rule-format*] **by** *blast*
   next
    fix *u* **assume** $u\ \varepsilon\ z$
    then obtain *b* **where** ‹$\langle b,u\rangle\ \varepsilon\ f2$› $b\ \varepsilon\ y$
      unfolding ‹$y = domM\ f2$›[*unfolded setext dom-def′*, *rule-format*]
              ‹$z = rngM\ f2$›[*unfolded setext rng-def′*, *rule-format*] **by** *blast*
    then obtain *c* **where** ‹$\langle c,b\rangle\ \varepsilon\ f1$› $c\ \varepsilon\ x$
      unfolding ‹$x = domM\ f1$›[*unfolded setext dom-def′*, *rule-format*]
              ‹$y = rngM\ f1$›[*unfolded setext rng-def′*, *rule-format*] **by** *blast*
    show $\exists v\ a\ b\ c.\ \langle a,b\rangle\ \varepsilon\ f1 \wedge \langle b,c\rangle\ \varepsilon\ f2 \wedge v = a \wedge u = c$
      using ‹$\langle b,u\rangle\ \varepsilon\ f2$› ‹$\langle c,b\rangle\ \varepsilon\ f1$› **by** *auto*
   qed
  show *?thesis*
     unfolding *set-equivalent-def*
    using ‹*one-oneM* $(f2 \circ_M f1)$› ‹$x = domM\ (f2 \circ_M f1)$› ‹$z = rngM\ (f2 \circ_M f1)$›
by *blast*
qed

lemma *union-of-ords-regular*: **assumes** $\forall\ y.\ y\ \varepsilon\ s \longrightarrow$ *ordM y*
shows *regular* ($\bigcup_M\ s$)
using *assms ord-mem-ord set-of-ords-regular union-def'* **by** *blast*

lemma *union-of-ords-ord*: **assumes** $\forall\ y.\ y\ \varepsilon\ s \longrightarrow$ *ordM y*
shows *ordM* ($\bigcup_M\ s$)
proof–
have *t*: *transM* ($\bigcup_M\ s$)
using *assms epschain-def ordM-def union-trans-trans* **by** *blast*
have *r*: *regular* ($\bigcup_M\ s$)
by (*simp add*: *assms union-of-ords-regular*)
have *o*: *ordM v* **if** $v\ \varepsilon$ ($\bigcup_M\ s$) **for** *v*
using *assms*(*1*) *ord-mem-ord that union-def'* **by** *blast*
show *ordM* ($\bigcup_M\ s$)
unfolding *ordM-def epschain-def* **using** *t r o ordM-total* **by** *blast*
qed

lemma *union-ord*: **assumes** *ordM x* **shows** *ordM* ($\bigcup_M\ x$)
using *assms ord-mem-ord union-of-ords-ord* **by** *blast*

lemma *non-limit-union-ord-mem*: **assumes** $\forall\ u.\ u\ \varepsilon\ s \longrightarrow$ *ordM u is-sucM* ($\bigcup_M$ *s*)
shows $\bigcup_M\ s\ \varepsilon\ s$
proof–
obtain *m* **where** *m*: $\forall\ z.\ z\ \varepsilon \bigcup_M\ s \longleftrightarrow z\ \varepsilon\ m \lor z = m$
using ‹*is-sucM* ($\bigcup_M\ s$)› *is-sucM-def* **by** *blast*
then obtain *ymax* **where** *ymax* ε *s m* ε *ymax*
unfolding *union-def'* **by** *blast*
from *assms*(*1*)[*rule-format*, *OF* ‹*ymax* ε *s*›]
have $\bigcup_M\ s \subseteq_M$ *ymax*
unfolding *subsetM-def m*[*rule-format*] **using** ‹*m* ε *ymax*› **using** *ordM-trans*[*of ymax - m*] **by** *blast*
then have *ymax* = $\bigcup_M\ s$
using ‹*ymax* ε *s*› **unfolding** *setext subsetM-def subsetM-def union-def'* **by** *blast*
then show $\bigcup_M\ s\ \varepsilon\ s$
using ‹*ymax* ε *s*› **by** *blast*
qed

lemma *limit-ord*: **assumes** $\forall\ u.\ u\ \varepsilon\ s \longrightarrow$ *ordM u* $\land$ *setsucM u u* ε *s*
shows $\neg$ *is-sucM* ($\bigcup_M\ s$)
by (*metis assms non-limit-union-ord-mem ordM-regular regular-not-self-mem-sep setsuc-def'*
union-def')

lemma *not-limit-ord*: **assumes** *ordM x is-sucM x*
shows *x* = *setsucM* ($\bigcup_M\ x$) ($\bigcup_M\ x$)
proof–
obtain *m* **where** *m*: *x* = *setsucM m m*
using ‹*is-sucM x*› **unfolding** *setext is-sucM-def setsuc-def'* **by** *blast*

hence *ordM m*
using ‹*ordM x*› *ord-mem-ord setsuc-def'* **by** *presburger*
have $m = \bigcup_M x$
unfolding *m setext*[*of m*] *union-def' setsuc-def'*
using *ordM-trans*[*OF* ‹*ordM m*›] **by** *blast*
then show *?thesis*
using *m* **by** *force*
qed

lemma *natM-fin*: **assumes** $s \subseteq_M x$ $s \neq \emptyset$ $\forall u.\ u\ \varepsilon\ s \longrightarrow$ *setsucM u u* ε *s*
shows $\neg$ *natM x*
proof
have $\exists u.\ u\ \varepsilon \bigcup_M s$
using *assms union-def'*[*of - s*] *setsuc-def' emptyset-def'* **by** *metis*
assume *natM x*
from *natM-D*[*OF this*]
have $\bigcup_M s \subseteq_M x$
unfolding *subsetM-def union-def'* **using** *ordM-trans* ‹$s \subseteq_M x$›[*unfolded subsetM-def*] **by** *blast*
have *ordM* $(\bigcup_M s)$
using ‹*ordM x*› *union-of-ords-ord ord-mem-ord* ‹$s \subseteq_M x$›[*unfolded subsetM-def*] **by** *blast*
have *natM* $(\bigcup_M s)$
using *ordM-subset-mem*[*OF* ‹*ordM x*› ‹*ordM* $(\bigcup_M s)$›, *unfolded proper-subset-def'*] ‹*natM x*›
nat-mem-nat ‹$\bigcup_M s \subseteq_M x$› **by** *blast*
from *nat-is-suc*[*OF this* ‹$\exists u.\ u\ \varepsilon \bigcup_M s$›]
show *False*
using *assms limit-ord natM-D*[*OF nat-mem-nat*[*OF* ‹*natM x*›]] **unfolding** *subsetM-def* **by** *force*
qed

lemma *nat-induction-sfp*: **assumes** *SetFormulaPredicate P* **and** $P\ (\Xi(0{:=}\emptyset))$ **and**
step: $\bigwedge x.$ *natM x* $\Longrightarrow P\ (\Xi(0{:=}x)) \Longrightarrow P\ (\Xi(0{:=}$*setsucM x x*$))$ **and** *natM x*
shows $P\ (\Xi(0{:=}x))$
proof (*rule ccontr*)
assume $\neg P\ (\Xi(0{:=}x))$
define *v* **where** $v =$ *separationM x* $(\lambda x.\ P(\Xi(0{:=}x)))$
from *separ-def-SFP*[*OF* ‹*SetFormulaPredicate P*›]
have *v*: $u\ \varepsilon\ v \longleftrightarrow u\ \varepsilon\ x \wedge P(\Xi(0{:=}u))$ **for** *u*
unfolding *v-def* **by** *simp*
hence $\emptyset\ \varepsilon\ v$
using ‹$P\ (\Xi(0{:=}\emptyset))$› ‹$\neg P\ (\Xi(0{:=}x))$› ‹*natM x*› *emp-natM empty-is-empty ordM-total* **unfolding** *natM-def* **by** *fast*
have (*setsucM y y*) ε *v* **if** *y* ε *v* **for** *y*
unfolding *v*[*of setsucM y y*] **using** ‹$\neg P\ (\Xi(0{:=}x))$› ‹*natM x*› *that*[*unfolded v*[*of y*]] *step*[*of y*] *ord-mem-suc*[*of x y*] *nat-mem-nat*[*of x y*] *natM-D*[*OF* ‹*natM x*›]
by *fast*

thus *False*
using ‹$\emptyset\ \varepsilon\ v$› *notE*[*OF natM-fin*[*of v x*], *OF* - - - ‹*natM x*›] *v*
by (*metis emptyset-def′ subsetM-def*)
qed

— Schema of induction for natural numbers
lemma *nat-induction-sp*: **assumes** *SetProperty P* **and** $P\ \emptyset$ **and** *natM x* **and**
step: $\bigwedge x.\ natM\ x \Longrightarrow P\ x \Longrightarrow P\ (setsucM\ x\ x)$
shows *P x*
using *assms* **unfolding** *SetProperty-def* **using** *nat-induction-sfp* **by** *force*

theorem *nat-tarski-fin*: **assumes** *natM x* **shows** *tarski-fin x*
using *nat-induction-sp*[*OF SP-tarski empty-tarski* ‹*natM x*›] *tarski-setsuc-tarski*
by *blast*

lemma *nat-dedekind-finite*: **assumes** *natM x* **shows** *dedekind-fin x*
using *nat-induction-sp*[*OF SP-dedekind empty-dedekind* ‹*natM x*›] *dedekind-setsuc-dedekind*
by *blast*

lemma *card-emp*: *cardinality* $\emptyset\ \emptyset$
proof–
have *natM* $\emptyset$
by *simp*
moreover have *one-oneM* $\emptyset$
unfolding *one-oneM-def functionM-def relationM-def* **by** *simp*
moreover have $\emptyset = domM\ \emptyset\ \emptyset = rngM\ \emptyset$
unfolding *setext dom-def′ rng-def′* **by** *simp-all*
ultimately show *?thesis*
unfolding *cardinality-def set-equivalent-def*
using *exI*[*of* $\lambda\ f.\ \ one\text{-}oneM\ f \wedge \emptyset = domM\ f \wedge \emptyset = rngM\ f\ \emptyset$] **by** *blast*
qed

lemma *nat-equiv-unique*: **assumes** *natM x natM y* $x \approx_M y$
shows $x = y$
proof (*rule ccontr*)
assume $x \neq y$
have $\neg\ y\ \varepsilon\ x$
using ‹$x \neq y$› *assms nat-dedekind-finite*[*unfolded dedekind-fin-def*, *rule-format*, *OF* ‹*natM x*›] *ordM-D1*[*OF natM-D*[*OF* ‹*natM x*›]] *subsetM-def proper-subset-def′*
by *blast*
have $\neg\ x\ \varepsilon\ y$
using ‹$x \neq y$› *assms nat-dedekind-finite*[*unfolded dedekind-fin-def*, *rule-format*, *OF* ‹*natM y*›] *ordM-D1*[*OF natM-D*[*OF* ‹*natM y*›]]
subsetM-def proper-subset-def′ set-equivalent-sym[*of x y*] **by** *blast*
show *False*
using ‹$\neg\ x\ \varepsilon\ y$› ‹$\neg\ y\ \varepsilon\ x$› ‹$x \neq y$› *ordM-total*[*OF natM-D natM-D*, *OF assms*(*1*,*2*)] **by** *blast*
qed

lemma *card-fun*: **assumes** *cardinality x n cardinality x m* **shows** $n = m$
using *assms* **unfolding** *cardinality-def* **using** *nat-equiv-unique set-equivalent-sym set-equivalent-trans* **by** *blast*

end

locale *L-setext-empty-power-union-repl-reg* = *L-setext* + *L-empty* + *L-power* + *L-union* + *L-repl*
+ *L-reg*

begin

Some consequences of the axiom of regularity

sublocale *L-setext-empty-power-union-repl*
by *unfold-locales*

lemma *mem-not-refl*: $\neg\ x\ \varepsilon\ x$
by (*simp add*: *regular-not-self-mem*)

lemma *mem-antisym*: $\neg\ (u\ \varepsilon\ v \wedge v\ \varepsilon\ u)$
by (*meson any-reg regular-antisym-mem setsuc*)

lemma *suc-unique*: **assumes** *setsucM c c* = *setsucM d d*
shows $c = d$
proof (*rule ccontr*)
assume $c \neq d$
from *assms*[*unfolded setext setsuc-def'*[*of* - $\cup_M$ -], *unfolded setsuc-def'*]
have $c\ \varepsilon\ d\ d\ \varepsilon\ c$
using ‹$c \neq d$› **by** *blast+*
thus *False*
using *mem-antisym* **by** *blast*
qed

lemma *mem-neq-singleton*: $x \neq \{x\}_M$
by (*metis reg singleton-def'*)

lemma *suc-subset*[*simp*]: $z \subset_M$ *setsucM z z*
unfolding *proper-subsetM-def setext setsuc-def'*
singleton-def' **using** *mem-not-refl*[*of z*] **by** *blast*

lemma *card-setsuc*: **assumes** $\neg\ y\ \varepsilon\ x$ *cardinality x m* **shows** *cardinality* (*setsucM x y*) (*setsucM m m*)
proof−
obtain *f* **where** *one-oneM f x* = *domM f m* = *rngM f natM m*
using ‹*cardinality x m*› **unfolding** *cardinality-def set-equivalent-def* **by** *blast*
let *?f* = *setsucM f* (⟨*y*,*m*⟩)
have *natM* (*setsucM m m*)
by (*simp add*: ‹*natM m*› *nat-suc-nat*)
have *setsucM x y* = *domM ?f*

using ‹x = *domM f*› **unfolding** *setext dom-def′* **by** *auto*
have *setsucM m m* = *rngM ?f*
using ‹m = *rngM f*› **unfolding** *setext rng-def′* **by** *auto*
have *relationM ?f*
unfolding *relationM-def*
using ‹*one-oneM f*› *functionM-def one-oneM-def relationM-def* **by** *auto*
then have *functionM ?f*
unfolding *functionM-def*
using ‹*one-oneM f*› ‹x = *domM f*› ‹¬ y ε x› *dom-def′ funD one-oneD3* **by** *auto*
then have *one-oneM ?f*
unfolding *one-oneM-def*
using ‹¬ y ε x› *one-one-inj*[*OF* ‹*one-oneM f*›] ‹m = *rngM f*› *mem-not-refl*[*of* m]
rng-def′ **by** *auto*
show *?thesis*
unfolding *cardinality-def set-equivalent-def*
using ‹*setsucM m m* = *rngM ?f*› ‹*natM* (*setsucM m m*)› ‹*one-oneM ?f*› ‹*setsucM x y* = *domM ?f*› **by** *blast*
qed

end

1.3.12 Successor induction

An important fragment of the theory of hereditarily finite sets. The finiteness principle used is the induction schema for set formulas. This route to axiomatizating ZFfin is developed in Vopěnka: Mathematics in the alternative set theory [10]. Notice that no regularity principles are used in this fragment.

locale *L-setext-empty-setsuc-setind* = *L-setext* + *L-empty* + *L-setsuc* + *L-setind*

begin

sublocale *L-setext-empty-setsuc*
by *unfold-locales*

lemma *binunion-ex*:
shows ∃ z. (∀ u. u ε z ⟷ u ε x ∨ u ε x') (**is** *?P x x′*)
proof−
have *SR*: *SetRelation ?P*
unfolding *SetRelation-def logsimps* **by** (*rule SFP-rules*)+
show *?thesis*
proof (*rule setind*[*rule-format*, *OF SR*[*unfolded SetRelation-def*], *of undefined*(*0*:=x,*1*:=x'), *simplified*], *force*)
fix x y :: ′*a*
assume *ex*: ∃ z. ∀ u. (u ε z) ⟷ (u ε x ∨ u ε x')
then show ∃ z. ∀ u. (u ε z) ⟷ (u ε x ∨ u = y ∨ u ε x')

unfolding *setsuc-def′* **using** *setsuc*[*rule-format*, *of - y*] **by** *metis*
qed
qed

sublocale *L-union*
proof (*unfold-locales*, *rule allI*, *rule setind-SP*[*rule-format*])
show $\exists z. \forall u. (u \varepsilon z) \longleftrightarrow (\exists v. v \varepsilon \emptyset \wedge u \varepsilon v)$
by (*meson empty-is-empty*)
next
fix *x y*
have *aux*: $(\forall u. (u \varepsilon z) \longleftrightarrow (\exists v. (v \varepsilon x \vee v = y) \wedge u \varepsilon v)) \longleftrightarrow (\forall u. (u \varepsilon z) \longleftrightarrow (\exists v. v \varepsilon x \wedge u \varepsilon v) \vee u \varepsilon y)$ **for** *z*
by *blast*
let $?Q = \lambda z. \forall u. (u \varepsilon z) = (\exists v. (v \varepsilon x \vee v = y) \wedge u \varepsilon v)$
assume $\exists z. \forall u. (u \varepsilon z) \longleftrightarrow (\exists v. v \varepsilon x \wedge u \varepsilon v)$
thus $\exists z. \forall u. (u \varepsilon z) \longleftrightarrow (\exists v. v \varepsilon (setsucM\ x\ y) \wedge u \varepsilon v)$
unfolding *setsuc-def′ aux* **using** *binunion-ex*[*rule-format*, *of - y*] **by** *metis*
next
show *SetProperty* $(\lambda a. \exists z. \forall u. (u \varepsilon z) = (\exists v. v \varepsilon a \wedge u \varepsilon v))$
unfolding *SetProperty-def logsimps* **by** (*rule SFP-rules*)+
qed

sublocale *L-repl*
proof (*unfold-locales*, *rule impI*, *rule allI*)
fix *P x* Ξ
assume *SetFormulaPredicate P* **and** *func*: $\forall u. \exists! v. P (\Xi(0 := u, 1 := v))$
from *bounded-free*[*OF* ‹*SetFormulaPredicate P*›]
obtain *m* **where** *m*: $\forall \Xi\ \Xi'. (\forall i<m. \Xi\ i = \Xi'\ i) \longrightarrow P\ \Xi = P\ \Xi'$
by *blast*
hence *small*: $P (\Xi(m := x, 0 := u, 1 := v)) = P (\Xi(0 := u, 1 := v))$ **for** *u v x*
by *simp*
let $?P = \lambda\Xi. \exists z. \forall v. (v \varepsilon z) = (\exists u. u \varepsilon \Xi\ m \wedge P (\Xi\ (0:=u,1:=v)))$
have *sfp*: *SetFormulaPredicate ?P*
by (*rule SFP-replace*) *fact*+
note *aux-rule* = *setind-var*[*rule-format*, *of ?P* Ξ *m x*, *OF sfp*, *unfolded small*, *simplified*, *unfolded One-nat-def*[*symmetric*]]
show $\exists z. \forall v. (v \varepsilon z) = (\exists u. u \varepsilon x \wedge P (\Xi(0 := u, 1 := v)))$
proof (*rule aux-rule*)
fix *x y*
assume $\exists z. \forall v. (v \varepsilon z) = (\exists u. u \varepsilon x \wedge P (\Xi(0 := u, 1 := v)))$
then obtain *z* **where** *z*: $\forall v. (v \varepsilon z) = (\exists u. u \varepsilon x \wedge P (\Xi(0 := u, 1 := v)))$
by *blast*
obtain y' **where** y': $P (\Xi(0 := y, 1 := y'))$
using *func* **by** *blast*
show $\exists z. \forall v. (v \varepsilon z) = (\exists u. (u \varepsilon x \vee u = y) \wedge P (\Xi(0 := u, 1 := v)))$
by (*rule exI*[*of* $\lambda z. \forall v. (v \varepsilon z) = (\exists u. (u \varepsilon x \vee u = y) \wedge P (\Xi(0 := u, 1 := v)))$ *setsucM z y′*])
(*unfold setsuc-def′*, *use y′ z func* **in** *blast*)
qed (*simp only*: *empty*)

qed

sublocale *L-setext-empty-repl*
  by *unfold-locales*

lemma *setsuc-separ*: **assumes** $y\ \varepsilon\ u$
  shows $u = setsucM\ (separationM\ u\ (\lambda\ x.\ x \neq y))\ y$
proof–
  have *sfp*: $SetFormulaPredicate\ (\lambda\Xi.\ \Xi\ 0 \neq \Xi\ 1)$
    by (*rule SFP-rules*)+
  show *?thesis*
    unfolding *setext*[*of u*] *setsuc-def′ separ-def-SR*[*of* (≠) - *u y*, *unfolded SetRelation-def*, *OF sfp*] **using** ‹$y\ \varepsilon\ u$› **by** *blast*
qed

lemma *setsuc-subset-setind*: $u \subseteq_M setsucM\ x\ y \longleftrightarrow u \subseteq_M x \vee (\exists\ u'.\ u' \subseteq_M x \wedge u = setsucM\ u'\ y)$
proof
  assume $u \subseteq_M setsucM\ x\ y$
  show $u \subseteq_M x \vee (\exists u'.\ u' \subseteq_M x \wedge u = setsucM\ u'\ y)$
  proof (*unfold disj-imp*, *rule impI*)
    assume $\neg\ u \subseteq_M x$
    hence $y\ \varepsilon\ u$
      using ‹$u \subseteq_M setsucM\ x\ y$› *subsetM-def setsuc-def′* **by** *auto*
    have $separationM\ u\ (\lambda\ x.\ x \neq y) \subseteq_M x$
      using ‹$u \subseteq_M setsucM\ x\ y$› **unfolding** *subsetM-def*
        separ-def-SR[*of* (≠), *unfolded SetRelation-def*, *OF SFP-neg*[*OF SFP-eq*]] *setsuc-def′* **by** *blast*
    with *setsuc-separ*[*OF* ‹$y\ \varepsilon\ u$›]
    show $\exists u'.\ u' \subseteq_M x \wedge u = setsucM\ u'\ y$
      by *blast*
  qed
next
  assume $u \subseteq_M x \vee (\exists u'.\ u' \subseteq_M x \wedge u = setsucM\ u'\ y)$
  then show $u \subseteq_M setsucM\ x\ y$
  proof
    show $u \subseteq_M x \Longrightarrow u \subseteq_M setsucM\ x\ y$
      unfolding *subsetM-def setsuc-def′* **by** *blast*
    assume $\exists u'.\ u' \subseteq_M x \wedge u = setsucM\ u'\ y$
    then show $u \subseteq_M setsucM\ x\ y$
      unfolding *subsetM-def setext*[*of u*] *setsuc-def′* **by** *blast*
  qed
qed

sublocale *L-power*
proof (*unfold-locales*, *rule allI*)
  fix x
  show $\exists y.\ \forall u.\ (u\ \varepsilon\ y) = u \subseteq_M x$
proof (*rule setind-SP*[*rule-format*])

show $SetProperty\ (\lambda a.\ \exists\, y.\ \forall\, u.\ (u\ \varepsilon\ y) = u \subseteq_M\ a)$
unfolding *SetProperty-def logsimps set-defs* **by** (*rule SFP-rules*)+
show $\exists\, y.\ \forall\, u.\ (u\ \varepsilon\ y) = u \subseteq_M\ \emptyset$
using *singleton-def′*[*of* - $\emptyset$] **by** *auto*
fix $x\ y$
let $?P = \lambda\ \Xi.\ \Xi\ 1 = setsucM\ (\Xi\ 0)\ (\Xi\ 2)$
have *sfp*: *SetFormulaPredicate ?P*
unfolding *setext logsimps set-defs* **by** (*rule SFP-rules*)+
have *ex1*: $\exists!\ v.\ v = u \cup_M \{y\}_M$ **for** u
by *blast*
have *binunion-def′*: $u\ \varepsilon\ x \cup_M\ y \longleftrightarrow u\ \varepsilon\ x \lor u\ \varepsilon\ y$ **for** $u\ x\ y$
using *collect-def-ex*[*OF binunion-ex*[*rule-format*, *of x y*], *folded binunionM-def*]**.**
assume $\exists\, z.\ \forall\, u.\ (u\ \varepsilon\ z) = u \subseteq_M\ x$
then obtain z **where** z: $\forall\, u.\ (u\ \varepsilon\ z) = u \subseteq_M\ x$
by *blast*
obtain z' **where** z': $\forall\ v.\ v\ \varepsilon\ z' \longleftrightarrow (\exists\ u.\ u\ \varepsilon\ z \land v = setsucM\ u\ y)$
using *replf*[*OF sfp*, *of undefined*(*2*:=y), *simplified*] **by** *blast*
show $\exists\, z.\ \forall\, u.\ (u\ \varepsilon\ z) = u \subseteq_M\ setsucM\ x\ y$
by (*rule exI*[*of* $\lambda\ z.(\forall\, u.\ (u\ \varepsilon\ z) = u \subseteq_M\ setsucM\ x\ y)\ z \cup_M\ z'$])
(*unfold binunion-def′ setsuc-subset-setind*, *use z z′* **in** *simp*)
qed
qed

— Axioms of ZF without infinity and regularity are theorems

sublocale *L-setext-empty-power-union-repl*
by *unfold-locales*

lemma *min-subset-ex*: **assumes** $u \neq \emptyset$ **shows** $\exists\ z.\ z\ \varepsilon\ u \land (\nexists\ w.\ w\ \varepsilon\ u \land w \subset_M z)$
proof (*rule mp*[*OF - assms*], *rule setind-SP*[*rule-format*])
show $SetProperty\ (\lambda a.\ a \neq \emptyset \longrightarrow (\exists\, z.\ z\ \varepsilon\ a \land (\nexists\, w.\ w\ \varepsilon\ a \land w \subset_M\ z)))$
unfolding *SetProperty-def set-defs logsimps* **by** (*rule SFP-rules*)+
next
fix $x\ y$
assume *IH*: $x \neq \emptyset \longrightarrow (\exists\, z.\ z\ \varepsilon\ x \land (\nexists\, w.\ w\ \varepsilon\ x \land w \subset_M\ z))$
show $setsucM\ x\ y \neq \emptyset \longrightarrow (\exists\, z.\ z\ \varepsilon\ setsucM\ x\ y \land (\nexists\, w.\ w\ \varepsilon\ setsucM\ x\ y \land w \subset_M\ z))$
proof (*rule impI*, *cases* $x = \emptyset$)
assume $x = \emptyset$
then show $\exists\, z.\ z\ \varepsilon\ setsucM\ x\ y \land (\nexists\, w.\ w\ \varepsilon\ setsucM\ x\ y \land w \subset_M\ z)$
using *setsuc-def′* **by** *force*
next
assume $x \neq \emptyset$
from *IH*[*rule-format*, *OF this*]
obtain u **where** $u\ \varepsilon\ x$ **and** *nex*: $\nexists\, w.\ w\ \varepsilon\ x \land w \subset_M\ u$
by *blast*
show $\exists\, z.\ z\ \varepsilon\ setsucM\ x\ y \land (\nexists\, w.\ w\ \varepsilon\ setsucM\ x\ y \land w \subset_M\ z)$

proof (*cases* $y \subset_M u$)
assume $y \subset_M u$
have $y\ \varepsilon$ *setsucM* $x\ y$
using *setsuc-def′* **by** *auto*
show *?thesis*
proof (*rule exI*[*of* - *y*], *rule conjI*[*OF* ‹$y\ \varepsilon$ *setsucM* $x\ y$›], *rule notI*)
assume $\exists w.\ w\ \varepsilon$ *setsucM* $x\ y \wedge w \subset_M y$
then show *False*
using ‹$y \subset_M u$› ‹$y\ \varepsilon$ *setsucM* $x\ y$› *nex proper-subsetM-trans*[*OF* - ‹$y \subset_M u$›]
by (*metis proper-subsetM-def setsuc-def′*)
qed
next
assume $\neg\ y \subset_M u$
show *?thesis*
by (*rule exI*[*of* - *u*]) (*use setsuc-def′* ‹$u\ \varepsilon\ x$› *nex* ‹$\neg\ y \subset_M u$› **in** *metis*)
qed
qed
qed *simp*

A general variant of Tarski finiteness axiom (also proved in Vopěnka's book). Nonempty sets have maximal (and minimal) elements under inclusion.

lemma *max-subset-ex*: **assumes** $u \neq \emptyset$ **shows** $\exists\ z.\ z\ \varepsilon\ u \wedge (\nexists\ w.\ w\ \varepsilon\ u \wedge z \subset_M w)$
proof (*rule mp*[*OF* - *assms*], *rule setind-SP*[*rule-format*])
show *SetProperty* $(\lambda a.\ a \neq \emptyset \longrightarrow (\exists z.\ z\ \varepsilon\ a \wedge (\nexists w.\ w\ \varepsilon\ a \wedge z \subset_M w)))$
unfolding *SetProperty-def set-defs logsimps* **by** (*rule SFP-rules*)+
next
fix $x\ y$
assume *IH*: $x \neq \emptyset \longrightarrow (\exists z.\ z\ \varepsilon\ x \wedge (\nexists w.\ w\ \varepsilon\ x \wedge z \subset_M w))$
show *setsucM* $x\ y \neq \emptyset \longrightarrow (\exists z.\ z\ \varepsilon$ *setsucM* $x\ y \wedge (\nexists w.\ w\ \varepsilon$ *setsucM* $x\ y \wedge z \subset_M w))$
proof (*rule impI*, *cases* $x = \emptyset$)
assume $x = \emptyset$
then show $\exists z.\ z\ \varepsilon$ *setsucM* $x\ y \wedge (\nexists w.\ w\ \varepsilon$ *setsucM* $x\ y \wedge z \subset_M w)$
using *setsuc-def′* **by** *force*
next
assume $x \neq \emptyset$
from *IH*[*rule-format*, *OF this*]
obtain u **where** $u\ \varepsilon\ x$ **and** *nex*: $\nexists w.\ w\ \varepsilon\ x \wedge u \subset_M w$
by *blast*
show $\exists z.\ z\ \varepsilon$ *setsucM* $x\ y \wedge (\nexists w.\ w\ \varepsilon$ *setsucM* $x\ y \wedge z \subset_M w)$
proof (*cases* $u \subset_M y$)
assume $u \subset_M y$
have $y\ \varepsilon$ *setsucM* $x\ y$
using *setsuc-def′* **by** *auto*
show *?thesis*
proof (*rule exI*[*of* - *y*], *rule conjI*[*OF* ‹$y\ \varepsilon$ *setsucM* $x\ y$›], *rule notI*)
assume $\exists w.\ w\ \varepsilon$ *setsucM* $x\ y \wedge y \subset_M w$

then show *False*
using ‹$u \subset_M y$› ‹$y \varepsilon$ *setsucM* $x\ y$› *nex proper-subsetM-trans*[*OF* - ‹$u \subset_M$ y›]
by (*metis proper-subsetM-def setsuc-def'*)
qed
next
assume $\neg\ u \subset_M y$
show *?thesis*
by (*rule exI*[*of* - *u*]) (*use setsuc-def'* ‹$u \varepsilon x$› *nex* ‹$\neg\ u \subset_M y$› **in** *metis*)
qed
qed
qed *simp*

lemma *max-mem*: **assumes** $P\ n\ n \varepsilon x$ *SetProperty P*
shows $\exists\ m.\ m \varepsilon x \wedge P\ m \wedge (\forall\ k.\ k \varepsilon x \wedge P\ k \longrightarrow \neg\ m \subset_M k)$
proof (*rule ccontr*)
assume *contr*: $\nexists m.\ m \varepsilon x \wedge P\ m \wedge (\forall k.\ k \varepsilon x \wedge P\ k \longrightarrow \neg\ m \subset_M k)$
hence *chain*: $\exists\ k.\ k \varepsilon x \wedge P\ k \wedge\ m \subset_M k$ **if** *a*: $P\ m\ m \varepsilon x$ **for** *m*
using *that* **by** *blast*
define *y* **where** $y = \mathit{separationM}\ x\ P$
from *separ-def-SP*[*OF* ‹*SetProperty P*›, *of* - *x*, *folded y-def*]
have *y-def*: $(u \varepsilon y) \longleftrightarrow (u \varepsilon x \wedge P\ u)$ **for** *u*
by *simp*
have $y \neq \emptyset$
using *assms y-def* **by** *force*
show *False*
using *max-subset-ex*[*OF* ‹$y \neq \emptyset$›] *chain* **unfolding** *y-def* **by** *blast*
qed

sublocale *L-tarski*
using *max-subset-ex*
by *unfold-locales* (*simp add*: *tarski-fin-def*, *metis empty*)

sublocale *L-dedekind*
by *unfold-locales*
(*use setind-SP*[*rule-format*, *OF SP-dedekind empty-dedekind*] *dedekind-setsuc-dedekind* **in** *blast*)

lemma *fun-images*: **assumes** *SetFormulaPredicate P* **and** $\forall\ u.\ (\exists!\ v.\ P(\Xi(0{:=}u,1{:=}v)))$
shows $\exists\ z.\ (\forall\ v.\ v \varepsilon z \longleftrightarrow (\exists\ u.\ u \varepsilon x \wedge P(\Xi(0{:=}u,1{:=}v))))$
proof–
from *bounded-free*[*OF* ‹*SetFormulaPredicate P*›]
obtain *m* **where** $\forall \Xi\ \Xi'.\ (\forall i{<}m.\ \Xi\ i = \Xi'\ i) \longrightarrow P\ \Xi = P\ \Xi'$
by *blast*
hence *m*: $\forall \Xi\ \Xi'.\ (\forall i{<}m{+}2.\ \Xi\ i = \Xi'\ i) \longrightarrow P\ \Xi = P\ \Xi'$
by *simp*
have *small*: $P\ (\Xi(\mathit{Suc}\ (\mathit{Suc}\ m) := x,\ 0 := u,\ \mathit{Suc}\ 0 := v)) = P\ (\Xi(0 := u,\ \mathit{Suc}$ $0 := v))$ **for** *u v x*
by (*rule m*[*rule-format*]) *simp*

let $?P = \lambda\ X.\ \exists\, z.\ \forall\, v.\ (v\ \varepsilon\ z) = (\exists\, u.\ u\ \varepsilon\ (X\ (m+2)) \land P\ (X(0 := u,\ 1 := v)))$
have *SetFormulaPredicate ?P*
using *SFP-replace*[*OF* ‹*SetFormulaPredicate P*› *m*]**.**
note *aux-rule* = *setind-var*[*OF this*, *of* $\Xi((m+2):= x)\ m+2$, *simplified*, *unfolded small*, *folded One-nat-def*]
show $\exists\ z.\ (\forall\ v.\ v\ \varepsilon\ z \longleftrightarrow (\exists\ u.\ u\ \varepsilon\ x \land P(\Xi(0{:=}u,1{:=}v))))$
proof (*rule aux-rule*[*rule-format*])
show $\exists\, z.\ \forall\, v.\ \neg\ v\ \varepsilon\ z$
using *empty* **by** *blast*
next
fix *x y*
assume $\exists\, z.\ \forall\, v.\ (v\ \varepsilon\ z) = (\exists\, u.\ u\ \varepsilon\ x \land P\ (\Xi(0 := u,\ 1 := v)))$
then obtain *z* **where** *z-def*: $\forall\, v.\ (v\ \varepsilon\ z) = (\exists\, u.\ u\ \varepsilon\ x \land P\ (\Xi(0 := u,\ 1 := v)))$
by *blast*
obtain y' **where** $P\ (\Xi(0 := y,\ 1 := y'))$
using *assms*(2) **by** *blast*
have *witness*: $\forall\ v.\ v\ \varepsilon\ setsucM\ z\ y' \longleftrightarrow (\exists\ u.\ u\ \varepsilon\ setsucM\ x\ y \land P\ (\Xi(0 := u,\ 1 := v)))$
unfolding *setsuc-def*$'$ **using** ‹$P\ (\Xi(0 := y,\ 1 := y'))$› *assms*(2) *z-def* **by** *auto*
show $\exists\, z.\ \forall\, v.\ (v\ \varepsilon\ z) = (\exists\, u.\ (u\ \varepsilon\ x \lor u = y) \land P\ (\Xi(0 := u,\ 1 := v)))$
by (*rule exI*[*of - setsucM z y*$'$]) (*use witness* **in** *force*)
qed
qed

sublocale *L-fin*
proof (*unfold-locales*, *rule notI*)
assume $\exists\, x.\ \emptyset\ \varepsilon\ x \land (\forall\, y.\ y\ \varepsilon\ x \longrightarrow setsucM\ y\ y\ \varepsilon\ x)$
then obtain *x* **where** $\emptyset\ \varepsilon\ x$ **and** *suc*: $\forall\, y.\ y\ \varepsilon\ x \longrightarrow setsucM\ y\ y\ \varepsilon\ x$
by *blast*
have *SP*: *SetProperty* ($\lambda\ x.\ regular\ x \land transM\ x$)
unfolding *SetProperty-def* **unfolding** *set-defs logsimps* **by** (*rule SFP-rules*)+
from *sep-SP*[*OF this*, *rule-format*, *of x*]
obtain x' **where** x': $\forall\ u.\ u\ \varepsilon\ x' \longleftrightarrow u\ \varepsilon\ x \land regular\ u \land transM\ u$
by *blast*
have $x' \neq \emptyset\ \emptyset\ \varepsilon\ x'$
using x' ‹$\emptyset\ \varepsilon\ x$› **by** *force*+
have *suc*$'$: $y\ \varepsilon\ x' \longrightarrow setsucM\ y\ y\ \varepsilon\ x'$ **for** *y*
unfolding x'[*rule-format*] **using** *suc* **by** (*simp add*: *regular-setsuc trans-suc-trans*)

from *max-subset-ex*[*OF* ‹$x' \neq \emptyset$›]
obtain *xmax* **where** $xmax\ \varepsilon\ x'\ \nexists\ w.\ w\ \varepsilon\ x' \land xmax \subset_M w$
by *blast*
with *suc*$'$[*rule-format*, *OF* ‹$xmax\ \varepsilon\ x'$›]
have $\neg\ xmax \subset_M setsucM\ xmax\ xmax$
by *blast*
then have $xmax\ \varepsilon\ xmax$
unfolding *set-defs setext*[*of xmax*] **by** *blast*
then show *False*

using ‹*xmax* ε *x'*› *regular-not-self-mem x'* **by** *blast*
qed

theorem *ord-is-suc*: **assumes** *ordM x* **shows** $x = \emptyset \vee$ *is-sucM x*
using *assms empty-mem-ord fin ord-limit-or-suc* **by** *blast*

theorem *ord-iff-nat*: *ordM x* $\longleftrightarrow$ *natM x*
proof (*rule iffI*)
assume *ordM x*
show *natM x*
proof (*cases* $x = \emptyset$)
assume $x \neq \emptyset$
show *natM x*
proof (*rule natM-I*, *fact*)
fix *v* **assume** $\exists$ *u*. *u* ε *v v* ε *x*
then show *is-sucM v*
using *ord-mem-ord*[*OF* ‹*ordM x*› ‹*v* ε *x*›] *ord-is-suc*[*of v*] **by** *fastforce*
qed (*use ord-is-suc*[*OF* ‹*ordM x*›] ‹$x \neq \emptyset$› **in** *auto*)
qed *simp*
qed (*use natM-def* **in** *blast*)

lemma *ord-iff-empty-or-suc*: *ordM x* $\longleftrightarrow$ $x = \emptyset \vee (\exists$ *y*. *ordM y* $\wedge$ *x* = *setsucM y y*)
by (*metis emp-natM not-limit-ord ord-is-suc ord-iff-nat ord-suc-ord union-ord*)

lemma *union-of-ords-mem*: **assumes** $\forall$ *y*. *y* ε *s* $\longrightarrow$ *ordM y* $s \neq \emptyset$
shows $\bigcup_M$ *s* ε *s*
using ‹$s \neq \emptyset$› *emptyset-def'* *non-limit-union-ord-mem*[*OF assms*(*1*)] *ord-is-suc*[*OF union-of-ords-ord*[*OF assms*(*1*)]]
union-def' **by** *metis*

end

1.3.13 Negation of inf

ZF with inf replaced by fin

locale *L-setext-empty-power-union-repl-reg-fin* = *L-setext* + *L-empty* + *L-power* + *L-union* + *L-repl* + *L-reg* + *L-fin*

begin

sublocale *L-setext-empty-power-union-repl-reg*
by *unfold-locales*

sublocale *L-setind*
proof (*unfold-locales*, (*rule impI*)+, *rule allI*, *rule ccontr*)
fix *P* Ξ *x*
assume *SetFormulaPredicate P P* (Ξ (*0*:= $\emptyset$)) **and** *step*: $\forall$ *x y*. *P* (Ξ (*0*:= *x*)) $\longrightarrow$ *P* ((Ξ (*0*:=*setsucM x y*))) **and** $\neg$ (*P* (Ξ (*0*:=*x*)))

```
  have sfp: SetFormulaPredicate (λ Ξ. cardinality (Ξ 0) (Ξ 1))
    unfolding set-defs logsimps by (rule SFP-rules)+
  have cinj: cardinality u v ∧ cardinality u w ⟹ v = w for u v w
    using card-fun by auto
  from sep[OF ‹SetFormulaPredicate P›, rule-format, of𝔓 x Ξ]
  obtain s where s: ∀ u. (u ε s) = (u ε 𝔓 x ∧ P (Ξ(0 := u)))
    by blast
  from replp-vars[rule-format, OF sfp, of Ξ 0 1 s, simplified]
  obtain m where m: ∀ v. (v ε m) = (∃ a. a ε s ∧ cardinality a v)
    using cinj by blast
  have ∅ ε m
    using card-emp exI[of λ x. x ⊆M x ∧ P (Ξ (0:=x, 1:=∅)) ∧ cardinality x ∅ ∅]
‹P (Ξ (0:=∅))›
    unfolding m[rule-format] powerset-def' subsetM-def s[rule-format]
    using emptyset-def'  by auto
  have setsucM n n ε m if n ε m for n
  proof–
    obtain y where y ε 𝔓 x cardinality y n P (Ξ (0:=y))
      using m  ‹n ε m› s by fast
    obtain y' where y' ε x ¬ y' ε y
    using ‹y ε 𝔓 x› ‹P (Ξ (0:=y))› ‹¬ P (Ξ (0:=x))› subsetM-antisym powerset-def'
subsetM-def by blast
    show setsucM n n ε m
      unfolding m[rule-format]
    proof (rule exI[of - setsucM y y'])
      show setsucM y y' ε s ∧ cardinality (setsucM y y') (setsucM n n)
        using card-setsuc[OF ‹¬ y' ε y› ‹cardinality y n›] ‹y ε 𝔓 x› ‹y' ε x›
         step[rule-format, OF ‹P (Ξ(0:= y))›] unfolding powerset-def' subsetM-def
s[rule-format] by simp
    qed
  qed
  then show False
    using fin ‹∅ ε m› by blast
qed

lemma cardinality-ex: ∃! n. natM n ∧ x ≈M n
proof (rule ex-ex1I)
  show ∃  n. natM n ∧ x ≈M n
  proof (rule setind-SP[rule-format])
    show SetProperty (λa. ∃ n. natM n ∧ a ≈M n)
      unfolding SetProperty-def set-defs logsimps by (rule SFP-rules)+
    show ∃ n. natM n ∧ ∅ ≈M n
      using card-emp cardinality-def by blast
    show ∃ n. natM n ∧ setsucM x y ≈M n if ∃ n. natM n ∧ x ≈M n for x y
      using that card-setsuc setsuc-triv unfolding cardinality-def  by metis
  qed
  show natM n ∧ x ≈M n ⟹ natM y ∧ x ≈M y ⟹ n = y for n x y
    using nat-equiv-unique set-equivalent-sym set-equivalent-trans by blast
qed
```

sublocale *L-setext-empty-setsuc-setind*
  by *unfold-locales*

end

1.3.14 Dedekind finite

locale *L-setext-empty-power-union-repl-dedekind* = *L-setext* + *L-empty* + *L-power* + *L-union* + *L-repl* + *L-dedekind*

begin

sublocale *L-setext-empty-power-union-repl*
  by *unfold-locales*

sublocale *L-fin*
proof (*unfold-locales*, *rule notI*)
  assume $\exists x.\ \emptyset\ \varepsilon\ x \wedge (\forall y.\ y\ \varepsilon\ x \longrightarrow setsucM\ y\ y\ \varepsilon\ x)$
  then obtain x' **where** x': $\emptyset\ \varepsilon\ x'$ $\forall y.\ y\ \varepsilon\ x' \longrightarrow setsucM\ y\ y\ \varepsilon\ x'$
    by *blast*
  define x **where** $x = separationM\ x'\ natM$
  from *separ-def-SP*[*OF SP-nat*]
  have x: $u\ \varepsilon\ x \longleftrightarrow u\ \varepsilon\ x' \wedge natM\ u$ **for** u
    unfolding *x-def*.
  have *x-setsuc*: $u\ \varepsilon\ x \longrightarrow setsucM\ u\ u\ \varepsilon\ x$ **for** u
    using $x'(2)$ *nat-suc-nat*[*of u*] **unfolding** x **by** *blast*
  have *SetFormulaPredicate* $(\lambda\ \Xi.\ \exists\ v.\ \Xi\ 0 = \langle v, setsucM\ v\ v\rangle)$
    unfolding *logsimps set-defs* **by** (*rule SFP-rules*)+
  from *sep*[*OF this*, *rule-format*, *of* $x \times_M x$, *simplified*]
  obtain f **where** $u\ \varepsilon\ f \longleftrightarrow (u\ \varepsilon\ x \times_M x\ \wedge (\exists v.\ u = \langle v, setsucM\ v\ v\rangle))$ **for** u
    by *blast*
  hence f: $u\ \varepsilon\ f \longleftrightarrow (\exists\ v.\ v\ \varepsilon\ x \wedge u = \langle v, setsucM\ v\ v\rangle)$ **for** u
    using *car-prod-def′ x-setsuc* **by** *auto*
  have *one-oneM f*
   by (*rule one-oneI*, *unfold f x*, *blast*, *unfold ordered-pair-unique*)
    (*use ord-pred-fun*[*OF natM-D natM-D*] **in** *blast*)+
  have *domM f* = x
    unfolding *setext*[*of - x*] *f dom-def′* **by** *force*
  have $x \approx_M rngM\ f$
    unfolding *set-equivalent-def* **by** (*rule exI*[*of - f*]) (*use* ‹*one-oneM f*› ‹*domM f* = x› **in** *blast*)
  have *rngM f* $\subset_M x$
  proof (*rule proper-subsetI*)
    show *rngM f* $\subseteq_M x$
      unfolding *subsetM-def rng-def′ f ordered-pair-unique* **using** *x-setsuc* **by** *blast*
    show *rngM f* $\neq x$
      unfolding *setext*[*of - x*] *rng-def′ not-all*
    proof (*rule exI*[*of -* $\emptyset$])

have *t*: $\emptyset\ \varepsilon\ x = True$
by (*simp add*: *x x'*(*1*))
have *f*: ($\exists\ v.\ \langle v,\emptyset\rangle\ \varepsilon\ f) = False$
unfolding *f ordered-pair-unique setext*[*of* $\emptyset$] *binunion-def' singleton-def'* **by** *force*
show ($\exists\,v.\ \langle v,\emptyset\rangle\ \varepsilon\ f) \neq (\emptyset\ \varepsilon\ x)$
unfolding *t f* **by** *blast*
qed
qed
from *dedekind*[*unfolded dedekind-fin-def*, *rule-format*, *OF this*]
show *False*
using ‹$x \approx_M$ *rngM f*› **by** *blast*
qed

end

1.3.15 Tarski finite

locale *L-setext-empty-power-sep-setsuc-tarski* = *L-setext* + *L-empty* + *L-power* + *L-sep* + *L-setsuc* + *L-tarski*

begin

sublocale *L-setext-empty-power*
by *unfold-locales*

sublocale *L-setext-empty-setsuc*
by *unfold-locales*

In a suitable context, the axiom of Tarski finitness yields the set induction schema.

sublocale *L-setind*
proof (*unfold-locales*, *rule impI*, *rule impI*, *rule allI*)
fix $P\ \Xi\ x$
assume *set-p*: *SetFormulaPredicate P* **and**
step: ($\forall\,x\ y.\ P\ (\Xi(0 := x)) \longrightarrow P\ (\Xi(0 := setsucM\ x\ y)))$ **and**
$P\ (\Xi(0 := \emptyset))$
show $P\ (\Xi(0 := x))$
proof (*rule ccontr*)
assume $\neg\ P\ (\Xi(0 := x))$
obtain *z* **where** *z-def*: $\bigwedge u.\ (u\ \varepsilon\ z) \longleftrightarrow (u\ \varepsilon\ (\mathfrak{P}\ x) \wedge P\ (\Xi(0 := u)))$
using *sep*[*OF set-p*, *rule-format*, *of* $\mathfrak{P}\ x$] **by** *blast*
have $z \neq \emptyset$
using *z-def* ‹$P\ (\Xi(0 := \emptyset))$› *empty-is-empty empty-is-subset subsetM-def powerset-def'* **by** *blast*
have $z \subseteq_M \mathfrak{P}\ x$
by (*simp add*: *subsetM-def z-def*)
have *neg*: $\exists\ v.\ v\ \varepsilon\ z \wedge u \subset_M v$ **if** $u\ \varepsilon\ z$ **for** *u*
proof–

obtain y **where** $\neg\ y\ \varepsilon\ u$ **and** $y\ \varepsilon\ x$
using ‹$\neg\ P\ (\Xi(0 := x))$› ‹$u\ \varepsilon\ z$›[*unfolded z-def*] *setext*[*of u x*] **unfolding** *subsetM-def powerset-def'* **by** *blast*
have $(setsucM\ u\ y)\ \varepsilon\ z$
using ‹$y\ \varepsilon\ x$› *step*[*rule-format,of u y*] *powerset-def' subsetM-def setsuc-def' that z-def* **by** *auto*
moreover have $u \subset_M (setsucM\ u\ y)$
unfolding *proper-subsetM-def setext*[*of u*] *setsuc-def'*
using ‹$\neg\ y\ \varepsilon\ u$› **by** *blast*
ultimately show *?thesis*
by *blast*
qed
show *False*
using *tarski*[*rule-format, of x, unfolded tarski-fin-def, rule-format, of z*] ‹$z \subseteq_M \mathfrak{P}\ x$› *neg* ‹$z \neq \emptyset$›
unfolding *subsetM-def powerset-def' z-def*
using ‹$P\ (\Xi(0 := \emptyset))$› *empty-is-empty* **by** *blast*
qed
qed

sublocale *L-setext-empty-setsuc-setind*
by *unfold-locales*

It follows in particular that union and replacement are provable in this context.

sublocale *L-repl*
by *unfold-locales*

sublocale *L-union*
by *unfold-locales*

end

1.3.16 Set induction and regularity schema

An apparently minor variation of the set induction schema, which nevertheless yields also the schema of regularity (i.e., epsilon induction). It is used in Pudlák and Sochor [5].

locale *L-setindregsch* = *set-signature* +
assumes *setindregsch*: *SetFormulaPredicate P* $\Longrightarrow$
$P(\Xi(0{:=}\ \emptyset)) \longrightarrow (\forall\ x\ y.\ P(\Xi(0{:=}\ x)) \wedge P(\Xi(0{:=}\ y)) \longrightarrow P(\Xi(0{:=}\ setsucM\ x\ y))) \longrightarrow (\forall\ x.\ P(\Xi(0{:=}\ x)))$

begin

lemma *setindregsch-sp*: **assumes** *SetProperty P* $P\ \emptyset$ $\forall\ x\ y.\ P\ x \wedge P\ y \longrightarrow P\ (setsucM\ x\ y)$
shows $\forall\ x.\ P\ x$

using *setindregsch*[*OF* ‹*SetProperty P*›[*unfolded SetProperty-def*]] *assms* **by** *force*

lemma *setindregsch-var*: **assumes** *SetFormulaPredicate P P*(Ξ(*n*:= ∅)) ∀ *x y*. *P*(Ξ(*n*:= *x*)) ∧ *P*(Ξ(*n*:= *y*))
⟶ *P*(Ξ(*n*:= *setsucM x y*))
shows ∀ *x*. *P*(Ξ(*n*:= *x*))
proof
fix *x*
from *bounded-free*[*OF* ‹*SetFormulaPredicate P*›]
obtain *m* **where** *m*: *P* Ξ = *P* Ξ′ **if** ∀ *i*<*m*. Ξ *i* = Ξ′ *i* **for** Ξ Ξ′
by *blast*
let *?m* = *Suc* (*n* + *m*)
let *?f* = *id*(*0* := *?m*, *n*:= *0*)
let *?Q* = λ *X*. (*P* (λ *b*. *X* (*?f b*)))
let *?X* = Ξ(*?m*:= Ξ *0*)
have *sfpq*: *SetFormulaPredicate ?Q*
using *transform-variables*[*OF* ‹*SetFormulaPredicate P*›] **by** *simp*
have *small*: ∀ *i* < *m*. (Ξ(*n* := *u*)) *i* = (*?X* (*0* := *u*)) (*?f i*) **for** *u*
by *auto*
have *equiv*: (*P* (Ξ(*n* := *u*))) ⟷ (*?Q* (*?X* (*0* := *u*))) **for** *u*
by (*rule m*[*of* Ξ(*n* := *u*) λ *b*. (*?X* (*0* := *u*))(*?f b*)]) *fact*
show *P* (Ξ(*n* := *x*))
unfolding *equiv*
by (*rule setindregsch*[*rule-format*, *OF sfpq*, *of* Ξ(*Suc* (*n* + *m*) := Ξ *0*)], *unfold equiv*[*symmetric*])
(*use assms* **in** *blast*)+
qed

— Induction schema for set formulas is a theorem.

sublocale *L-setind*
by *unfold-locales* (*use setindregsch* **in** *blast*)

end

locale *L-setext-empty-setsuc-setindregsch* = *L-setext* + *L-empty* + *L-setsuc* + *L-setindregsch*

begin

sublocale *L-setext-empty-setsuc-setind*
by *unfold-locales*

lemma *epsind-from-setindregsch-sp*: **assumes** *spp*: *SetProperty P* **and** *indp*: (∀ *x*. (∀ *y*. (*y* ε *x* ⟶ *P y*)) ⟶ *P x*)
shows ∀ *x*. *P x*
proof–
let *?Q* = λ *x*. ∀ *u*. (*u* ε *x* ⟶ *P u*)
have *SetFormulaPredicate* (λΞ. *P* (Ξ *m*)) **for** *m*

using *spp*[*unfolded SetProperty-def*] **by** (*rule transform-variables*)
have *SetProperty ?Q*
unfolding *SetProperty-def set-defs logsimps* **by** (*rule SFP-rules*)+ *fact*
have $\forall$ *v w*. *?Q v* $\wedge$ *?Q w* $\longrightarrow$ *?Q* (*setsucM v w*) — use *?Q* w and *indp* to show P w
using *indp* **unfolding** *setsuc-def'* **by** *presburger*
from *setindregsch-sp*[*OF* ‹*SetProperty ?Q*› - *this*]
have $\forall$ *x*. *?Q x*
by *force*
then show $\forall$ *x*. *P x*
using *indp* **by** *blast*
qed

— Epsilon induction (and hence, regularity schema) is a theorem.

lemma *epsind-from-setindregsch*: **assumes** *sfp*: *SetFormulaPredicate P* **and** *indp*: ($\forall$ *x*. ($\forall$ *y*. (*y* ε *x* $\longrightarrow$ *P*(Ξ(*0*:=*y*)))) $\longrightarrow$ *P*(Ξ(*0*:=*x*)))
shows $\forall$ *x*. *P*(Ξ(*0*:=*x*))
proof
fix *x*
from *bounded-free*[*OF* ‹*SetFormulaPredicate P*›]
obtain *m* **where** $\forall \Xi\ \Xi'.\ (\forall i<m.\ \Xi\ i = \Xi'\ i) \longrightarrow P\ \Xi = P\ \Xi'$
by *blast*
then have *small*: *P*(Ξ(*Suc m*:=*a*, *0*:= *u*)) = *P*(Ξ(*0*:= *u*)) **for** Ξ *a u*
by *simp*
let *?Q* = λ Ξ. $\forall$ *u*. (*u* ε (Ξ (*Suc m*)) $\longrightarrow$ *P*(Ξ(*0*:=*u*)))
have *sfpQ*: *SetFormulaPredicate ?Q*
by (*rule SFP-all*[*of* λ Ξ. (Ξ *0*) ε (Ξ (*m*+*1*)) $\longrightarrow$ *P* Ξ *0*, *simplified*])
(*unfold logsimps set-defs*, (*rule* | *fact*)+)
thm *setindregsch-var*[*OF* ‹*SetFormulaPredicate ?Q*›, *of* - *Suc m*, *unfolded small*, *simplified*]
have *?Q* (Ξ (*0*:=*v*)) $\wedge$ *?Q* (Ξ (*0*:=*w*)) $\longrightarrow$ *?Q* (Ξ (*0*:=($v \cup_M \{w\}_M$))) **for** *v w* Ξ
— use ?Q w and indp to show P w
unfolding *setsuc-def'* **using** *indp*[*rule-format*] **by** *simp*
from *setindregsch-var*[*OF* ‹*SetFormulaPredicate ?Q*›, *of* - *Suc m*, *unfolded small*, *simplified*]
have *?Q* (Ξ(*Suc m*:= *x*))
unfolding *small fun-upd-same* **using** *indp* **by** *metis*
then show *P* (Ξ(*0* := *x*))
using *indp* **unfolding** *small fun-upd-same* **by** *blast*
qed

sublocale *L-epsind*
by (*unfold-locales*) (*use epsind-from-setindregsch* **in** *blast*)

sublocale *L-setext-empty-union-repl-pair-regsch*
by *unfold-locales*

end

1.3.17 Summary of dependencies

theorem *epsind-regsch-iff*:
L-epsind mem $\longleftrightarrow$ *L-regsch mem*
proof
assume *L-epsind mem*
from *L-epsind.epsind-regsch*[*OF this*]
show *L-regsch mem*
by *unfold-locales blast*
next
assume *L-regsch mem*
from *L-regsch.regsch-epsind*[*OF this*]
show *L-epsind mem*
by *unfold-locales*
qed

We give additional names to some important collections of axioms. Here is a "canonical" axiomatization of the theory of hereditarily finite sets.

locale *ZFfin* = *L-setext* + *L-empty* + *L-power* + *L-union* + *L-repl* + *L-fin* + *L-epsind*

begin
sublocale *L-setext-empty-power-union-repl-reg-fin*
by *unfold-locales*
sublocale *L-regsch*
by *unfold-locales*
sublocale *L-reg*
by *unfold-locales*
sublocale *L-setsuc*
by *unfold-locales*
sublocale *L-sep*
by *unfold-locales*
sublocale *L-setind*
by *unfold-locales*
sublocale *L-dedekind*
by *unfold-locales*
sublocale *L-tarski*
by *unfold-locales*
end

This is the list of axioms for sets from Vopěnka's book [10].

locale *ASTset* = *L-setext* + *L-empty* + *L-setsuc* + *L-setind* + *L-regsch*

begin

Vopěnka [10] shows that all axioms of *ZFfin* are provable in *ASTset*

sublocale *ZFfin*
proof–
interpret *L-setext-empty-setsuc-setind*

by *unfold-locales*
interpret *L-epsind*
by (*simp add*: *L-regsch-axioms epsind-regsch-iff*)
show *ZFfin* (ε)
by *unfold-locales*
qed

The variation of set induction which combines it with regularity schema is provable from set induction schema and epsilon induction (i.e., regularity schema). Taken for granted in [5].

sublocale *L-setindregsch*
proof (*unfold-locales*, *rule impI*, *rule impI*, *rule allI*)
fix $P :: (nat \Rightarrow 'a) \Rightarrow bool$ **and** Ξ **and** x
let $?P = \lambda\ y.\ P\ (\Xi(0 := y))$
assume *sfp*: *SetFormulaPredicate P*
show $?P\ x$ **if** $?P\ \emptyset$ **and** *step*: $(\forall\, x\ y.\ ?P\ x \wedge\ ?P\ y \longrightarrow\ ?P\ (setsucM\ x\ y))$
proof–
— In order to complete the proof by ε-induction, it is enough to show that all sets inherit the property $?P$. Call this inheritance property Q
let $?Q = \lambda\ w.\ (\forall\ \ u.\ u\ \varepsilon\ w \longrightarrow\ ?P\ u) \longrightarrow\ ?P\ w$
— We show that all sets satisfy Q by set-induction.

— But first some work has to be done. Note that $?Q$ depends on free variables present in P. We therefore have to reformulate $?Q$ to reflect this. We formulate Q as a property of a fresh variable $\Xi\ (m+1)$
from *bounded-free*[*OF sfp*]
obtain m **where** m: $\forall \Xi\ \Xi'.\ (\forall\, i{<}m.\ \Xi\ i = \Xi'\ i) \longrightarrow P\ \Xi = P\ \Xi'$
by *blast*
have *fresh*: $P\ (\Xi(m + k := a,\ 0 := b)) = P\ (\Xi(0 := b))$ **for** $a\ b\ k\ \Xi$
— Indeed, P does not depend on any $\Xi\ (m{+}k)$
using m[*rule-format*, *of* $\Xi(m + k := a,\ 0 := b)\ \Xi(0 := b)$] **by** *simp*

let $?Q' = \lambda\ \Xi.\ (\forall\ \ u.\ u\ \varepsilon\ \Xi\ (m{+}1) \longrightarrow P\ (\Xi(0 := u))) \longrightarrow P\ (\Xi(0 := \Xi\ (m{+}1)))$
— $?Q'\ \Xi$ now says: x_{m+1} inherits P from its elements
have *SetFormulaPredicate* $?Q'$
— The technical part: showing that $?Q'$ is set formula predicate
proof–
have *SetFormulaPredicate* $(\lambda\Xi.\ \forall\, u.\ u\ \varepsilon\ \Xi\ (Suc\ m) \longrightarrow P\ (\Xi(0 := u)))$
by (*rule SFP-all*[*of* $\lambda\Xi.\ \Xi\ (m{+}2)\ \varepsilon\ \Xi\ (Suc\ m) \longrightarrow P\ (\Xi(0 := \Xi(m{+}2)))$ $m{+}2$,
simplified fun-upd-same fun-upd-other, *unfolded fresh*])
(*unfold logsimps*, *rule+*, *fact*)
show *SetFormulaPredicate* $?Q'$
unfolding *logsimps* **by** (*rule SFP-rules*)+
(*simp*, *fact*, *simp add*: *sfp update-variable*)
qed
— This yields the desired form of the induction in terms of $?Q$
have *Q-ind-rule*: $?Q\ \emptyset \Longrightarrow (\forall\ \ x\ y.\ ?Q\ x \longrightarrow\ ?Q\ (setsucM\ x\ y)) \Longrightarrow \forall\ \ x.\ ?Q\ x$

using
setind-var[*rule-format*, *OF* ‹*SetFormulaPredicate ?Q′*›, *of* Ξ *m+1*]
unfolding *fresh fun-upd-same* **by** (*rule*, *blast+*)

— Back to the main proof.
have ∀ *w*. *?Q w*
proof (*rule Q-ind-rule*)
show *?Q* ∅
using ‹*?P* ∅› **by** *blast*
— the empty set inherits *P*, since it satisfies *P*
show ∀ *x y*. *?Q x* ⟶ *?Q* (*setsucM x y*)
proof (*rule allI*, *rule allI*, *rule impI*, *rule impI*)
fix *x y*
assume ‹*?Q x*› **and** ∀ *u*. *u* ε *setsucM x y* ⟶ *?P u*
hence *?P x ?P y*
using ‹*?Q x*› **unfolding** *setsuc-def′* **by** *blast+*
thus *?P* (*setsucM x y*)
using *step* **by** *blast*
qed
qed
thus *?P x*
using *epsind*[*OF* ‹*SetFormulaPredicate P*›] **unfolding** *fun-upd-same fresh* **by** *metis*
qed
qed

end

ZFfin and *ASTset* are two different axiomatizations of the same theory.

sublocale *ASTset* ⊆ *ZFfin*
by *unfold-locales*
sublocale *ZFfin* ⊆ *ASTset*
by (*simp add*: *ASTset-def L-empty-axioms L-regsch-axioms L-setext-axioms L-setind-axioms L-setsuc-axioms*)

In the AST, set induction and epsilon induction (i.e., regularity schema) can be replaced by the variant *setindregsch*

theorem *setindregsch-ast*: *L-setext-empty-setsuc-setindregsch mem* ⟷ *ASTset mem*
proof
assume *L-setext-empty-setsuc-setindregsch mem*
then interpret *L-setext-empty-setsuc-setindregsch mem*.
show *ASTset mem*
by *unfold-locales*
next
assume *ASTset mem*
then interpret *ASTset mem*.
show *L-setext-empty-setsuc-setindregsch mem*
by *unfold-locales*

qed

theorem *zffin-ast*: *ZFfin mem* $\longleftrightarrow$ *ASTset mem*
proof
  assume *ZFfin mem*
  then interpret *ZFfin mem*.
  show *ASTset mem*
    by *unfold-locales*
next
  assume *ASTset mem*
  then interpret *ASTset mem*.
  show *ZFfin mem*
    by *unfold-locales*
qed

Ambivalence of the separation schema and the replacement schema.

theorem *repl-implies-sep*:
  shows *L-setext-empty-repl mem* $\Longrightarrow$ *L-sep mem*
proof–
  assume *L-setext-empty-repl mem*
  then interpret *L-setext-empty-repl mem*.
  show *L-sep mem*
    by *unfold-locales*
qed

Under certain finiteness assumptions, separation entails replacement. See [2] for details.

theorem *sep-implies-repl*:
  shows *L-setext-empty-power-sep-setsuc-tarski mem* $\Longrightarrow$ *L-repl mem*
proof–
  assume *L-setext-empty-power-sep-setsuc-tarski mem*
  then interpret *L-setext-empty-power-sep-setsuc-tarski mem*.
  show *L-repl mem*
    by *unfold-locales*
qed

Entailment between different finiteness principles, in their natural contexts.

The following is included in Vopěnka [10] by exploring the consequences of induction for set formulas.

theorem (**in** *L-setext-empty-setsuc*)
  shows *setind-implies-tarski*: *L-setind* (ε) $\Longrightarrow$ *L-tarski* (ε) **and**
      setind-implies-fin-by-setsuc: *L-setind* (ε) $\Longrightarrow$ *L-fin* (ε) **and**
    setind-implies-dedekind-by-setsuc: *L-setind* (ε) $\Longrightarrow$ *L-dedekind* (ε)
proof–
  assume *L-setind* (ε)
  then interpret *L-setind*
    by *blast*
  interpret *L-setext-empty-setsuc-setind*

by *unfold-locales*
show *L-tarski* (*ε*) *L-fin* (*ε*) *L-dedekind* (*ε*)
by *unfold-locales*
qed

theorem (**in** *L-setext-empty-power-union-repl*)
dedekind-implies-fin: *L-dedekind* (*ε*) $\Longrightarrow$ *L-fin* (*ε*)
proof−
assume *L-dedekind* (*ε*)
then interpret *L-dedekind*.
interpret *L-setext-empty-power-union-repl-dedekind*
by *unfold-locales*
show *L-fin* (*ε*)
by (*simp add*: *L-fin-axioms*)
qed

Equivalence of finiteness principles, in sufficiently strong common context.

theorem (**in** *L-setext-empty-power-union-repl-reg*)
fin-implies-tarski: *L-fin* (*ε*) $\Longrightarrow$ *L-tarski* (*ε*) **and**
tarski-implies-fin: *L-tarski* (*ε*) $\Longrightarrow$ *L-fin* (*ε*) **and**
fin-implies-setind: *L-fin* (*ε*) $\Longrightarrow$ *L-setind* (*ε*) **and**
setind-implies-fin: *L-setind* (*ε*) $\Longrightarrow$ *L-fin* (*ε*) **and**
fin-implies-dedekind: *L-fin* (*ε*) $\Longrightarrow$ *L-dedekind* (*ε*)
proof−
assume *L-fin* (*ε*)
then interpret *L-fin* (*ε*).
interpret *L-setext-empty-power-union-repl-reg-fin* (*ε*)
by *unfold-locales*
interpret *L-setext-empty-setsuc-setind* (*ε*)
by *unfold-locales*
show *L-tarski* (*ε*)
by *unfold-locales*
show *L-dedekind* (*ε*)
by *unfold-locales*
show *L-setind* (*ε*)
by *unfold-locales*
next
assume *L-tarski* (*ε*)
then interpret *L-tarski* (*ε*).
interpret *L-setext-empty-power-sep-setsuc-tarski*
by *unfold-locales*
show *L-fin* (*ε*)
by *unfold-locales*
next
assume *L-setind* (*ε*)
then interpret *L-setind* (*ε*)
by *blast*
interpret *L-setsuc* (*ε*)
by *unfold-locales*

interpret *L-setext-empty-setsuc* (ε)
by *unfold-locales*
from *setind-implies-tarski*[*OF* ‹*L-setind* (ε)›]
interpret *L-tarski* (ε).
interpret *L-setext-empty-power-sep-setsuc-tarski* (ε)
by *unfold-locales*
show *L-fin* (ε)
by *unfold-locales*
qed

The validity of the regularity schema/epsilon induction is closely related to the existence of a transitive superset See also Proposition 5.4, Kaye and Wong [3]. There, the equivalence of *L-epsind* and *L-ts* is shown in the context of EST + regularity. Instead, we follow Sochor [7] and [8] in improving the claim by weakening both implications.

theorem (**in** *L-setext-sep-reg*) *ts-implies-epsind*:
L-ts (ε) $\Longrightarrow$ *L-epsind* (ε)
proof–
assume *L-ts* (ε)
then interpret *L-ts* (ε).
interpret *L-setext-sep-reg-ts*
by *unfold-locales*
interpret *L-regsch*
by *unfold-locales*
show *L-epsind* (ε)
using *L-epsind-def regsch-epsind* **by** *blast*
qed

theorem (**in** *L-setext-empty-union-repl-pair*) *epsind-implies-ts*:
L-epsind (ε) $\Longrightarrow$ *L-ts* (ε)
proof–
assume *L-epsind* (ε)
then interpret *L-epsind* (ε).
interpret *L-setext-empty-union-repl-pair-regsch*
by *unfold-locales*
show *L-ts* (ε)
using *L-ts-axioms*.
qed

Consequently, in a sufficiently strong context we can verify the equivalence of axioms A81 (regularity) + A82 (transitive superset) and A8 (schema of regularity) from Sochor [7], p. 709.

theorem (**in** *L-setext-empty-union-repl-pair*) *ts-reg-iff-regsch*:
L-ts (ε) $\wedge$ *L-reg* (ε) $\longleftrightarrow$ *L-regsch* (ε)
proof
assume *L-regsch* (ε)
then interpret *L-regsch* (ε).
have *L-reg* (ε)

by (*simp add*: *L-reg-axioms*)
have *L-ts* (ε)
by (*simp add*: *L-regsch-axioms epsind-regsch-iff epsind-implies-ts*)
then show *L-ts* (ε) $\wedge$ *L-reg* (ε)
using ‹*L-reg* (ε)› **by** *blast*
next
assume *L-ts* (ε) $\wedge$ *L-reg* (ε)
then interpret *L-ts* (ε)
by *simp*
interpret *L-reg* (ε)
using ‹*L-ts* (ε) $\wedge$ *L-reg* (ε)› **by** *simp*
interpret *L-setext-empty-repl*
by *unfold-locales*
interpret *L-sep* (ε)
using *repl-implies-sep* **by** (*simp add*: *L-sep-axioms*)
interpret *L-setext-sep-reg* (ε)
by *unfold-locales*
show *L-regsch* (ε)
using *ts-implies-epsind L-ts-axioms* **unfolding** *epsind-regsch-iff*.
qed

end

Chapter 2

Models and counter-models

theory *ZFfin-HF*
imports *HereditarilyFinite.Rank ZFfin*
begin

2.1 Hereditarily finite sets

We show that the hereditarily finite sets as implemented in *HereditarilyFinite.HF* are a model of ZFfin as implemented in *ZF-finite.ZFfin*

interpretation *zfhf*: *ZFfin* (∈)
  rewrites *zfhf.emptysetM* = *0* **and**
    zfhf.singletonM y = ⦃*y*⦄ **and**
    zfhf.setsucM x y = *x* ◁ *y*
proof−
  interpret *zfhf*: *L-setsuc* (∈)
    by *unfold-locales blast+*
  interpret *zfhf*: *L-empty* (∈)
    by *unfold-locales blast*
  interpret *zfhf*: *L-setext-empty* (∈)
    by *unfold-locales blast*
  show *zfhf-emp*: *zfhf.emptysetM* = *0*
      using *zfhf.empty-is-empty* **by** *auto*
  interpret *zfhf*: *L-setsuc* (∈)
    by *unfold-locales blast+*
  interpret *zfhf*: *L-empty* (∈)
    by *unfold-locales blast*
  interpret *zfhf*: *L-setext-empty-setsuc* (∈)
    by *unfold-locales*
  show *zfhf-sing*: *zfhf.singletonM y* = ⦃*y*⦄ **for** *y*
    using *zfhf.singleton-def'* **by** *blast*
  show *zfhf-suc*: *zfhf.setsucM x y* = *x* ◁ *y* **for** *x y*
    unfolding *zfhf.setext*[*of - x* ◁ *y*] *zfhf.setsuc-def'* **by** *auto*
  interpret *L-setind* (∈)
  proof

show *zfhf.SetFormulaPredicate P* $\Longrightarrow$ *P* (Ξ(*0* := *zfhf.emptysetM*)) $\longrightarrow$
($\forall$ *x y*. *P* (Ξ(*0* := *x*)) $\longrightarrow$ *P* (Ξ(*0* := *zfhf.setsucM x y*))) $\longrightarrow$ ($\forall$ *x*. *P* (Ξ(*0* := *x*))) **for** *P* Ξ
unfolding *zfhf-suc zfhf-emp* **using** *hf-induct*[*of* λ *a*. *P* (Ξ(*0*:=*a*))] **by** *blast*
qed
interpret *L-setext-empty-setsuc-setind* ($\in$)
by *unfold-locales*
interpret *L-epsind* ($\in$)
proof
fix Ξ :: *nat* $\Rightarrow$ *hf* **and** *Q*
from *Rank.hmem-induct*[*of* λ *x*. *Q*(Ξ(*0*:=*x*))]
show ($\forall$ *x*. ($\forall$ *y*. *y* $\in$ *x* $\longrightarrow$ *Q* (Ξ(*0* := *y*))) $\longrightarrow$ *Q* (Ξ(*0* := *x*))) $\longrightarrow$ ($\forall$ *x*. *Q* (Ξ(*0* := *x*)))
by *blast*
qed
show *ZFfin* ($\in$)
proof (*unfold-locales*, *unfold zfhf-emp zfhf-suc*)
fix Ξ :: *nat* $\Rightarrow$ *hf* **and** *P*
from *hf-induct*[*of* λ *x*. *P*(Ξ(*0*:=*x*))]
show $\nexists$ *x*. *0* $\in$ *x* $\wedge$ ($\forall$ *y*. *y* $\in$ *x* $\longrightarrow$ *y* $\triangleleft$ *y* $\in$ *x*)
using *fin zfhf-emp zfhf-suc* **by** *auto*
qed
qed

end

2.2 Permutation models

theory *Permutation-models*
imports *ZFfin*
begin

A useful tool of constructing models with specific properties is the Bernays-Rieger permutation method, which redefines the membership relation. We show that a permutation model obtained in this way preserves extensionality, empty set, powerset, union, and replacement. The method has been used, inter alia, in several works directly relevant to our formalization ([6], [11], [1], [4])

locale *permutation-model* = *L-setext-empty-power-union-repl* +
fixes *perm*
assumes
bij-perm: *bij* (*perm* :: $'a \Rightarrow 'a$) **and**
SR-fmem: *SetRelation* (λ *x y*. *x* ε *perm y*)

begin

abbreviation *perm-mem* (**infixr** ε^f *51*) **where**
perm-mem x y $\equiv$ *x* ε *perm y*

sublocale *L-setext-empty*
  by *unfold-locales*

interpretation *pm*: *L-setext perm-mem*
proof (*unfold-locales*, *rule iffI*, *blast*)
  show $x = y$ **if** $\forall z.\ z\ \varepsilon^f\ x = z\ \varepsilon^f\ y$ **for** $x\ y$
    using *that*[*folded setext*[*of perm x perm y*]] *bij-perm*
    by (*simp add*: *bij-betw-def inj-eq*)
qed

lemma *SFP-fmem*[*SFP-rules*]: *SetFormulaPredicate* ($\lambda\Xi.\ \Xi\ m\ \varepsilon^f\ \Xi\ n$)
 by (*rule transform-variables*[*OF SR-fmem*[*unfolded SetRelation-def*], *of id*(*0*:=*m*,*1*:=*n*), *simplified*])

lemma *SR-perm-eq*: *SetRelation* ($\lambda\ x\ y.\ perm\ x = y$)
  unfolding *SetRelation-def setext logsimps* **by** (*rule SFP-rules*)+

lemma *permSFP-SFP*: **assumes** *pm.SetFormulaPredicate P* **shows** *SetFormulaPredicate P*
proof (*rule pm.SetFormulaPredicate.induct*[*OF assms*])
  show *SetFormulaPredicate* ($\lambda\Xi.\ \Xi\ m\ \varepsilon^f\ \Xi\ n$) **for** $m\ n$
   using *transform-variables*[*OF SR-fmem*[*unfolded SetRelation-def*], *of id*(*0*:=-,*1*:=-)]
by *simp*
qed (*simp-all add*: *SFP-rules*)

lemma *perm-model*:
  shows *L-setext-empty-power-union-repl perm-mem*
proof
  write *pm.subsetM* (**infix** $\subseteq^f$ *51*)
  have *finv*: *perm* ((*inv perm*) *u*) = *u* **for** *u*
    using *bij-perm bij-inv-eq-iff* **by** *fast*
  have *finv′*: (*inv perm*) (*perm u*) = *u* **for** *u*
    using *bij-perm bij-inv-eq-iff* **by** *metis*
  have *inv-iff*: (*inv perm*) $x = y \longleftrightarrow$ *perm* $y = x$ **for** $x\ y$
    using *bij-perm finv finv′* **by** *auto*
  have *SR-f′*: *SetRelation* ($\lambda\ x\ y.$ (*inv perm*) $x = y$)
    using *SR-sym*[*OF SR-perm-eq*, *unfolded SR-perm-eq*] **unfolding** *inv-iff*.
  have *finv-unique*: $\forall u.\ \exists!v.\ v = inv\ perm\ u$
    using *bij-perm* **by** *blast*
  obtain *femp* **where** *femp-def*: *perm femp* $= \emptyset$
    using *bij-perm bij-pointE* **by** *metis*
  show $\exists x.\ \forall y.\ \neg\ y\ \varepsilon^f\ x$
    using *femp-def* **by** (*metis empty-is-empty*)
  have *fsubset*: $x \subseteq^f\ y \longleftrightarrow$ (*perm x*) $\subseteq_M$ (*perm y*) **for** $x\ y$
    unfolding *pm.subsetM-def subsetM-def*..
  show $\forall x.\ \exists y.\ \forall u.\ u\ \varepsilon^f\ y = u \subseteq^f\ x$
    unfolding *fsubset*
  proof

fix x
obtain y' **where** y': $\bigwedge u.\ u\ \varepsilon\ y' \longleftrightarrow u \subseteq_M (perm\ x)$
using *power*[*rule-format*, *of perm x*] **by** *blast*
obtain y **where** y: $\bigwedge u.\ (u\ \varepsilon\ y) = (\exists\, v.\ v\ \varepsilon\ y' \wedge inv\ perm\ v = u)$
using *replf*[*OF SR-f'*[*unfolded SetRelation-def*], *rule-format*, *of - y'*, *simplified*] **by** *blast*
have $\forall\, v.\ (v\ \varepsilon\ perm\ (inv\ perm\ y)) \longleftrightarrow perm\ v \subseteq_M perm\ x$
using *y y' bij-perm bij-inv-eq-iff* **by** *metis*
then show $\exists\, y.\ \forall\, v.\ (v\ \varepsilon\ perm\ y) = perm\ v \subseteq_M perm\ x$
by *blast*
qed
show $\forall\, x.\ \exists\, y.\ \forall\, v.\ v\ \varepsilon^f\ y = (\exists\, u.\ u\ \varepsilon^f\ x \wedge v\ \varepsilon^f\ u)$
proof
fix x
obtain x' **where** x': $\bigwedge u.\ (u\ \varepsilon\ x') = (\exists\, v.\ v\ \varepsilon\ perm\ x \wedge\ perm\ v = u)$
using *replf*[*OF SR-perm-eq*[*unfolded SetRelation-def*], *simplified*]
using *replf*[*OF SR-f'*[*unfolded SetRelation-def*], *rule-format*, *simplified*] **by** *blast*
obtain y **where** y: $\bigwedge v.\ v\ \varepsilon\ y \longleftrightarrow (\exists\ u.\ u\ \varepsilon\ x' \wedge v\ \varepsilon\ u)$
using *union*[*rule-format*, *of x'*] **by** *blast*
have $\forall\, v.\ (v\ \varepsilon\ perm\ (inv\ perm\ y)) \longleftrightarrow (\exists\, u.\ u\ \varepsilon\ perm\ x \wedge v\ \varepsilon\ perm\ u)$
using *y x' bij-perm finv* **by** *auto*
then show $\exists\, y.\ \forall\, v.\ (v\ \varepsilon\ perm\ y) = (\exists\, u.\ u\ \varepsilon\ perm\ x \wedge v\ \varepsilon\ perm\ u)$
by *blast*
qed
fix $P\ \Xi$
assume *pm.SetFormulaPredicate P*
show $(\forall\, u.\ \exists!v.\ P\ (\Xi(0 := u,\ 1 := v))) \longrightarrow (\forall\, x.\ \exists\, z.\ \forall\, v.\ v\ \varepsilon^f\ z = (\exists\, u.\ u\ \varepsilon^f\ x \wedge P\ (\Xi(0 := u,\ 1 := v))))$
proof (*rule impI*, *rule allI*)
assume $\forall\, u.\ \exists!v.\ P\ (\Xi(0 := u,\ 1 := v))$
fix x
have *SetFormulaPredicate P*
by (*simp add*: ‹*pm.SetFormulaPredicate P*› *permSFP-SFP*)
from *replf*[*rule-format*, *OF this* ‹$\forall\, u.\ \exists!v.\ P\ (\Xi(0 := u,\ 1 := v))$›[*rule-format*], *of perm x*]
obtain z **where** $\forall\, v.\ (v\ \varepsilon\ z) = (\exists\, u.\ u\ \varepsilon\ perm\ x \wedge P\ (\Xi(0 := u,\ 1 := v)))$
by *blast*
hence $\forall\, v.\ (v\ \varepsilon\ perm\ ((inv\ perm)\ z)) = (\exists\, u.\ u\ \varepsilon\ perm\ x \wedge P\ (\Xi(0 := u,\ 1 := v)))$
unfolding *finv*.
thus $\exists\, z.\ \forall\, v.\ (v\ \varepsilon\ perm\ z) = (\exists\, u.\ u\ \varepsilon\ perm\ x \wedge P\ (\Xi(0 := u,\ 1 := v)))$
by *blast*
qed
qed

end

end

2.3 Regularity

theory *Not-regular-model*
imports *Permutation-models*
begin

Axioms of extensionality, empty set, powerset, union, replacement and set induction are not sufficient to prove regularity, even in the presence of transitive superset axiom.

We prove this using permutation models. It is enough to transpose *0* and *1* in order to obtain a model which violates regularity, cf. [1, Theorem 13]

context *L-setext-empty-power-union-repl*

begin

definition *swap-zero-one* :: $'a \Rightarrow 'a$ **where**
  swap-zero-one $x = (\textit{if } x = \emptyset \textit{ then } \{\emptyset\}_M \textit{ else } (\textit{if } x = \{\emptyset\}_M \textit{ then } \emptyset \textit{ else } x))$

lemma *swap-zero-one-involution*[*simp*]: *swap-zero-one* (*swap-zero-one* x) = x
  by (*simp add*: *swap-zero-one-def*)

lemma *bij-swap-zero-one*: *bij swap-zero-one*
  by (*simp add*: *involuntory-imp-bij swap-zero-one-def*)

lemma *swap-zero-one-mem*: $a\ \varepsilon$ *swap-zero-one* $b \longleftrightarrow (b \neq \{\emptyset\}_M \wedge a\ \varepsilon\ b) \vee (a = \emptyset \wedge b = \emptyset)$
  unfolding *swap-zero-one-def* **by** (*simp add*: *singleton-def'*)

lemma *swap-zero-one-perm-model*: *permutation-model* (ε) *swap-zero-one*
proof
  show *SetRelation* ($\lambda x\ y.\ x\ \varepsilon$ *swap-zero-one* y)
    unfolding *SetRelation-def swap-zero-one-mem logsimps singleton-eq set-defs* **by** (*rule SFP-rules*)+
qed (*rule bij-swap-zero-one*)

interpretation *swap*: *permutation-model* (ε) *swap-zero-one*
  using *swap-zero-one-perm-model* **by** *blast*

notation *swap.perm-mem* (**infix** ‹ε^s› *50*)

interpretation *swap*: *L-setext-empty-power-union-repl* (ε^s)
  using *swap.perm-model*.

notation *swap.emptysetM* (‹$\emptyset^s$›)

notation *swap.setsucM* (‹$\mathfrak{suc}^s$›)

— $\{\emptyset\}$ is the empty set in the permuted model
lemma *one-swap*: $\neg\ (a\ \varepsilon^s\ \{\emptyset\}_M)$

unfolding *swap-zero-one-mem* **using** *singleton-def'* **by** *force*

lemma *swap-empty*: $\emptyset^s = \{\emptyset\}_M$
using *emptyset-def' swap.emptyset-def' swap-zero-one-mem* **by** *blast*

— $\emptyset$ is a singleton loop in the permuted model
lemma *zero-swap-mem-iff*: $a\ \varepsilon^s\ \emptyset \longleftrightarrow a = \emptyset$
unfolding *swap-zero-one-mem* **by** *auto*

Since $\emptyset\ \varepsilon^s\ \emptyset$, the permuted model is not regular.

theorem *not-reg-swap-zero-one*: $\neg$ (*L-reg* (ε^s))
unfolding *L-reg-def swap-zero-one-mem* **by** *force*

In order to show that the model with memebership ε^s satisfies *L-setind* (*L-ts* resp.) if the original model does, we first show how the modified successor behaves.

lemma *swap-mem-mem*: $\neg (x = \emptyset \wedge y = \emptyset) \Longrightarrow x\ \varepsilon^s\ y \Longrightarrow x\ \varepsilon\ y$
using *swap-zero-one-mem* **by** *auto*

lemma *swap-setsuc-1-y*: **assumes** $y \neq \emptyset$ **shows** $\mathfrak{suc}^s\ (\{\emptyset\}_M)\ y = \{y\}_M$
proof–
have *swap-zero-one* $(\{y\}_M) = \{y\}_M$
by (*metis assms empty-is-empty singleton-def' swap-zero-one-def*)
have $u\ \varepsilon^s\ \mathfrak{suc}^s\ (\{\emptyset\}_M)\ y \longleftrightarrow u = y$ **for** u
using *one-swap* **by** *auto*
hence *swap-zero-one* $(\mathfrak{suc}^s\ (\{\emptyset\}_M)\ y) = \{y\}_M$
unfolding *setext singleton-def'* **by** *fast*
from *arg-cong*[*OF this*, *of swap-zero-one*]
show $\mathfrak{suc}^s\ (\{\emptyset\}_M)\ y = \{y\}_M$
unfolding ‹*swap-zero-one* $(\{y\}_M) = \{y\}_M$› **by** *simp*
qed

lemma *swap-setsuc-0-y*: **assumes** $y \neq \emptyset$ **shows** $\mathfrak{suc}^s\ \emptyset\ y = \{\emptyset, y\}_M$
proof–
have $\{\emptyset, y\}_M \neq \emptyset \wedge \{\emptyset, y\}_M \neq \{\emptyset\}_M$
unfolding *setext* **unfolding** *pair-def' singleton-def'* **using** *assms* **by** *auto*
hence *swap-zero-one* $(\{\emptyset, y\}_M) = \{\emptyset, y\}_M$
by (*simp add*: *swap-zero-one-def*)
have $u\ \varepsilon^s\ \mathfrak{suc}^s\ \emptyset\ y \longleftrightarrow u = \emptyset \vee u = y$ **for** u
by (*simp add*: *zero-swap-mem-iff*)
hence *swap-zero-one* $(\mathfrak{suc}^s\ \emptyset\ y) = \{\emptyset, y\}_M$
unfolding *setext pair-def'* **by** *fast*
from *arg-cong*[*OF this*, *of swap-zero-one*]
show $\mathfrak{suc}^s\ \emptyset\ y = \{\emptyset, y\}_M$
unfolding ‹*swap-zero-one* $(\{\emptyset, y\}_M) = \{\emptyset, y\}_M$› **by** *simp*
qed

lemma *swap-setsuc-0-0*: $\mathfrak{suc}^s\ \emptyset\ \emptyset = \emptyset$
using *zero-swap-mem-iff* **by** *auto*

lemma *swap-setsuc-1-0*: $\mathfrak{suc}^s$ $(\{\emptyset\}_M)$ $\emptyset = \emptyset$
proof–
  have u ε^s $\mathfrak{suc}^s$ $(\{\emptyset\}_M)$ $\emptyset \longleftrightarrow u = \emptyset$ **for** u
    using *one-swap* **by** *auto*
  hence *swap-zero-one* $(\mathfrak{suc}^s$ $(\{\emptyset\}_M)$ $\emptyset) = \{\emptyset\}_M$
    unfolding *setext singleton-def′* **by** *blast*
  from *arg-cong*[*OF this*, *of swap-zero-one*]
  show $\mathfrak{suc}^s$ $(\{\emptyset\}_M)$ $\emptyset = \emptyset$
    by (*simp add*: *emptyset-def′ one-swap*)
qed

lemma *swap-setsuc-1-1*: $\mathfrak{suc}^s$ $(\{\emptyset\}_M)$ $(\{\emptyset\}_M) = \{\{\emptyset\}_M\}_M$
proof–
  have $\{\{\emptyset\}_M\}_M \neq \emptyset \wedge \{\{\emptyset\}_M\}_M \neq \{\emptyset\}_M$
    unfolding *setext*[*of* $\{\text{-}\}_M$] **by** (*metis empty-is-empty singleton-def′*)
  hence *swap-zero-one* $(\{\{\emptyset\}_M\}_M) = \{\{\emptyset\}_M\}_M$
    by (*simp add*: *swap-zero-one-def*)
  have u ε^s $\mathfrak{suc}^s$ $(\{\emptyset\}_M)$ $(\{\emptyset\}_M) \longleftrightarrow u = \{\emptyset\}_M$ **for** u
    using *one-swap* **by** *auto*
  hence *swap-zero-one* $(\mathfrak{suc}^s$ $(\{\emptyset\}_M)$ $(\{\emptyset\}_M)) = \{\{\emptyset\}_M\}_M$
    unfolding *setext singleton-def′* **by** *blast*
  from *arg-cong*[*OF this*, *of swap-zero-one*]
  show $\mathfrak{suc}^s$ $(\{\emptyset\}_M)$ $(\{\emptyset\}_M) = \{\{\emptyset\}_M\}_M$
    unfolding ‹*swap-zero-one* $(\{\{\emptyset\}_M\}_M) = \{\{\emptyset\}_M\}_M$› **by** *simp*
qed

lemma *setsuc-setsuc′*: **assumes** $x \neq \emptyset \wedge x \neq \{\emptyset\}_M$
  shows $\mathfrak{suc}^s$ x $y = \mathfrak{suc}$ x y
proof–
  have $\mathfrak{suc}$ x $y \neq \{\emptyset\}_M$ $\mathfrak{suc}$ x $y \neq \emptyset$
    by (*metis assms emptyset-def′ setsuc-def′ setsuc-triv singleton-def′*) (*use emptyset-def′* **in** *auto*)
  hence *aux*: z ε^s $\mathfrak{suc}$ x $y \longleftrightarrow z$ ε $\mathfrak{suc}$ x y z ε^s $x \longleftrightarrow z$ ε x **for** z
    by (*simp-all add*: *assms swap-zero-one-mem*)
  thus *?thesis*
   unfolding *swap.setext*[*of* - $\mathfrak{suc}$ x y] **unfolding** *aux swap.setsuc-def′*[*rule-format*] *setsuc-def′*[*rule-format*]
    by *blast*
qed

theorem *setind-swap-setind*: *L-setind* $(\varepsilon) \longrightarrow$ *L-setind* (ε^s)
proof
  assume *L-setind* (ε)
  then interpret *L-setind* (ε).
  show *L-setind* (ε^s)
  proof (*unfold-locales*, *rule impI*, *rule impI*, *rule allI*)
    fix $P :: (nat \Rightarrow 'a) \Rightarrow bool$ **and** w **and** Ξ
    let $?P = \lambda$ x. $P(\Xi(0{:=}\ x))$

assume *?P* $\emptyset^s$ **and** *step*: $\forall\, x\, y.\ ?P\ x \longrightarrow\ ?P\ (\mathfrak{suc}^s\ x\ y)$ **and** *SFP*: *swap.SetFormulaPredicate P*
hence *?P* $(\{\emptyset\}_M)$
using *swap-empty* **by** *auto*
have *aux*: $(\mathfrak{suc}^s\ (\{\emptyset\}_M)\ \emptyset) = \emptyset \longleftrightarrow (\mathfrak{suc}^s\ (\{\emptyset\}_M)\ \emptyset)\ \varepsilon^s\ (\mathfrak{suc}^s\ (\{\emptyset\}_M)\ \emptyset)$
using *swap.setsuc-def′ swap-zero-one-mem* **by** *blast*
have $\mathfrak{suc}^s\ (\{\emptyset\}_M)\ \emptyset = \emptyset$
by (*metis swap.setsuc-empty-sing swap.singleton-eq swap-empty zero-swap-mem-iff*)
hence *?P* $\emptyset$
using *step*[*rule-format*, *OF* ‹*?P* $(\{\emptyset\}_M)$›, *of* $\emptyset$] **by** *simp*
have *SetFormulaPredicate P*
by (*simp add*: *SFP swap.permSFP-SFP*)
show *?P w*
proof (*rule setind*[*OF* ‹*SetFormulaPredicate P*›, *of* Ξ, *rule-format*])
show *?P* $\emptyset$
by *fact*
next
fix *x y*
assume *?P x*
show *?P* $(\mathfrak{suc}\ x\ y)$
proof−
consider $x = \emptyset \wedge y = \emptyset \mid x = \emptyset \wedge y \neq \emptyset \mid x = \{\emptyset\}_M \wedge y = \emptyset \mid x = \{\emptyset\}_M \wedge y \neq \emptyset \mid x \neq \emptyset \wedge x \neq \{\emptyset\}_M$
by *blast*
then show *?thesis*
proof (*cases*)
assume $x = \emptyset \wedge y = \emptyset$
then show *?P* $(\mathfrak{suc}\ x\ y)$
using ‹*P* $(\Xi(0 := \{\emptyset\}_M))$› *setsuc-empty-sing* **by** *auto*
next
assume $x = \emptyset \wedge y \neq \emptyset$
then show *?P* $(\mathfrak{suc}\ x\ y)$
using *step*[*rule-format*, *OF* ‹*?P* $(\{\emptyset\}_M)$›, *of y*] *setsuc-empty-sing swap-setsuc-1-y* **by** *force*
next
assume $x = \{\emptyset\}_M \wedge y = \emptyset$
then show *?P* $(\mathfrak{suc}\ x\ y)$
using ‹*P* $(\Xi(0 := x))$› *singleton-def′* **by** *auto*
next
assume $x = \{\emptyset\}_M \wedge y \neq \emptyset$
hence $\mathfrak{suc}\ x\ y = \{\emptyset, y\}_M$
unfolding *setext setsuc-def′ singleton-def′ pair-def′* **by** *force*
hence $\mathfrak{suc}^s\ \emptyset\ y = \mathfrak{suc}\ x\ y$
by (*simp add*: *swap-setsuc-0-y* ‹$x = \{\emptyset\}_M \wedge y \neq \emptyset$›)
then show *?P* $(\mathfrak{suc}\ x\ y)$
using *step*[*rule-format*, *OF* ‹*?P* $\emptyset$›, *of y*] **by** *presburger*
next
assume $x \neq \emptyset \wedge x \neq \{\emptyset\}_M$
then show *?P* $(\mathfrak{suc}\ x\ y)$

using *setsuc-setsuc′ step*[*rule-format*, *OF* ‹*?P x*›] **by** *force*
qed
qed
qed
qed
qed

lemma *ts-swap-ts*: *L-ts* (ε) $\longrightarrow$ *L-ts* (ε^s)
proof
assume *L-ts* (ε)
then interpret *L-setext-sep-binunion-ts* (ε)
by (*simp add*: *L-binunion-axioms L-sep-axioms L-setext-axioms L-setext-sep-binunion-ts.intro*)

have *suc*: $x \cup_M \{\emptyset\}_M = \mathfrak{suc}\ x\ \emptyset$ **for** x
unfolding *binunion-def′ singleton-def′ setsuc-def′ setext* **by** *blast*
have *empts*: *least-tsM* ($\{\emptyset\}_M$) = $\{\emptyset\}_M$
unfolding *setext* **unfolding** *least-ts-def′ singleton-def′*
by (*metis subsetM-def transM-def set-two-mem powerset-def′ set-one-mem*)
show *L-ts* (ε^s)
proof (*unfold-locales*, *rule allI*, *unfold set-signature.transM-def set-signature.subsetM-def*)
fix x :: $'a$
consider $x = \emptyset \mid x = \{\emptyset\}_M \mid x \neq \emptyset \land x \neq \{\emptyset\}_M$
by *blast*
then show $\exists z.\ (\forall y.\ y\ \varepsilon^s\ z \longrightarrow (\forall za.\ za\ \varepsilon^s\ y \longrightarrow za\ \varepsilon^s\ z)) \land$
$(\forall za.\ za\ \varepsilon^s\ x \longrightarrow za\ \varepsilon^s\ z)$
proof (*cases*)
assume $x = \emptyset$
show *?thesis*
by (*rule exI*[*of* - $\emptyset$], *simp add*: ‹$x = \emptyset$› *swap-zero-one-mem*)
next
assume $x = \{\emptyset\}_M$
show *?thesis*
by (*rule exI*[*of* - $\{\emptyset\}_M$], *simp add*: ‹$x = \{\emptyset\}_M$› *swap-zero-one-mem*)
next
assume $x \neq \emptyset \land x \neq \{\emptyset\}_M$
let *?t* = *least-tsM* (*setsucM x* $\emptyset$)
have *t*: *?t* = (*least-tsM x*) $\cup_M \{\emptyset\}_M$
by (*metis empts leastTS-union suc*)
have *?t* $\neq \emptyset$
unfolding *t setext binunion-def′ singleton-def′* **by** *auto*
have *esimp*: $a\ \varepsilon^s$ *?t* $\longleftrightarrow a\ \varepsilon$ *least-tsM* $x \lor a = \emptyset$ **for** a
proof
assume $a\ \varepsilon^s$ *?t*
from *swap-mem-mem*[*OF* - *this*]
have $a\ \varepsilon$ *?t*
using ‹$x \neq \emptyset \land x \neq \{\emptyset\}_M$› ‹*?t* $\neq \emptyset$› **by** *blast*
show $a\ \varepsilon$ *least-tsM* $x \lor a = \emptyset$
proof (*cases* $a = \emptyset$)
assume $a \neq \emptyset$

have $a\ \varepsilon$ *least-tsM* x
unfolding *least-ts-def'*
proof (*rule ccontr*)
assume $\neg$ ($\forall$ v. *transM* $v \wedge x \subseteq_M v \longrightarrow a\ \varepsilon\ v$)
then obtain v **where** *transM* v $x \subseteq_M v$ $\neg$ $a\ \varepsilon\ v$
by *blast*
let *?v* = *setsucM* v $\emptyset$
have *transM ?v* $\mathfrak{suc}$ x $\emptyset \subseteq_M$ *?v*
using ‹*transM* v› ‹$x \subseteq_M v$› **unfolding** *transM-def subsetM-def setsuc-def'* **by** *force+*
hence $a\ \varepsilon$ *?v*
using ‹$a\ \varepsilon$ *least-tsM* ($\mathfrak{suc}$ x $\emptyset$)›[*unfolded least-ts-def'*] **by** *blast*
thus *False*
using ‹$\neg$ $a\ \varepsilon\ v$› ‹$a \neq \emptyset$› **unfolding** *setsuc-def'* **by** *blast*
qed
thus $a\ \varepsilon$ *least-tsM* $x \vee a = \emptyset$
by *blast*
qed *simp*
next
assume $a\ \varepsilon$ *least-tsM* $x \vee a = \emptyset$
hence $a\ \varepsilon$ *least-tsM* ($\mathfrak{suc}$ x $\emptyset$)
by (*simp add*: *binunion-def' singleton-def' t*)
have *least-tsM* ($\mathfrak{suc}$ x $\emptyset$) $\neq \{\emptyset\}_M$
proof
assume *least-tsM* ($\mathfrak{suc}$ x $\emptyset$) = $\{\emptyset\}_M$
have $x \subseteq_M$ *least-tsM* ($\mathfrak{suc}$ x $\emptyset$)
by (*simp add*: *least-ts-def' subsetM-def*)
show *False*
using ‹$x \neq \emptyset \wedge x \neq \{\emptyset\}_M$›
by (*metis* ‹*least-tsM* ($\mathfrak{suc}$ x $\emptyset$) = $\{\emptyset\}_M$› ‹$x \subseteq_M$ *least-tsM* ($\mathfrak{suc}$ x $\emptyset$)› *powerset-def' set-one-def set-two-mem*)
qed
then show $a\ \varepsilon^s$ *least-tsM* ($\mathfrak{suc}$ x $\emptyset$)
by (*simp add*: ‹$a\ \varepsilon$ *least-tsM* ($\mathfrak{suc}$ x $\emptyset$)› *swap-zero-one-mem*)
qed
show $\exists$ z. ($\forall$ y. $y\ \varepsilon^s\ z \longrightarrow$ ($\forall$ za. $za\ \varepsilon^s\ y \longrightarrow za\ \varepsilon^s\ z$)) $\wedge$
($\forall$ za. $za\ \varepsilon^s\ x \longrightarrow za\ \varepsilon^s\ z$)
proof (*rule exI*[*of - ?t*], *rule conjI*, *rule allI*, *rule impI*)
fix y **assume** $y\ \varepsilon^s$ *least-tsM* ($\mathfrak{suc}$ x $\emptyset$)
then show $\forall$ za. $za\ \varepsilon^s\ y \longrightarrow za\ \varepsilon^s$ *least-tsM* ($\mathfrak{suc}$ x $\emptyset$)
unfolding *esimp* **by** (*metis empty-mem-false least-ts-is-transitive set-signature.subsetM-def swap-zero-one-mem transM-def*)
next
show $\forall$ za. $za\ \varepsilon^s\ x \longrightarrow za\ \varepsilon^s$ *least-tsM* ($\mathfrak{suc}$ x $\emptyset$)
unfolding *esimp* **using** *least-ts-def' subsetM-def swap-mem-mem* **by** *blast*
qed
qed
qed
qed

end

theorem *not-reg-model*: **assumes** *L-setext-empty-power-union-repl* $(mem :: {}'a \Rightarrow {}'a \Rightarrow bool)$ *L-setind mem L-ts mem*
shows $\neg$ $(\forall\ (m :: {}'a \Rightarrow {}'a \Rightarrow bool)$. *L-setext-empty-power-union-repl m* $\wedge$ *L-setind m* $\wedge$ *L-ts m* $\longrightarrow$ *L-reg m*)
proof (*rule notI*)
assume *contr*: $\forall (m :: {}'a \Rightarrow {}'a \Rightarrow bool)$. *L-setext-empty-power-union-repl m* $\wedge$ *L-setind m* $\wedge$ *L-ts m* $\longrightarrow$ *L-reg m*
interpret *L-setext-empty-power-union-repl mem*
using *assms* **by** *simp*
interpret *L-setind mem*
using *assms* **by** *simp*
interpret *permutation-model mem swap-zero-one*
using *swap-zero-one-perm-model*.
have *L-setext-empty-power-union-repl perm-mem* $\wedge$ *L-setind perm-mem* $\wedge$ *L-ts perm-mem*
by (*simp add*: *perm-model setind-swap-setind ts-swap-ts* ‹*L-setind mem*› ‹*L-ts mem*›)
from *contr*[*rule-format*, *OF this*]
show *False*
using *not-reg-swap-zero-one* **by** *blast*
qed

end

2.4 Independence of transitive superset

theory *Not-ts-model*
imports *Permutation-models*
begin

We explore the model defined by the permutation transposing n and $\{n+1\}_M$ for each natural number n. The membership in the model is not well-founded since there is an infinite decreasing chain ... $2\ \varepsilon^f\ \ 1\ \varepsilon^f\ \ 0$.

Despite that if the native memebership satisfies regularity and finiteness, the permuted model also satisfies regularity and finiteness.

Consequently, 0 (the empty set) has not transitive superset, since it sould be infinite if it existed. As a corollary, the permutation model does not satisfy schema of regularity (or, equivalently, epsilon induction) since it entails transitive superset axiom.

The construction is given in Rieger in [6] and also in [4] who underscore that the model is recursive. The statement that regularity schema does not follow from regularity was also proved in [11].

context *L-setext-empty-power-union-repl-reg*

begin

fun *regperm* :: $'a \Rightarrow 'a$ **where**
regperm a = (*if natM a then* $\{setsucM\ a\ a\}_M$ *else*
if ($\exists$ *b. natM b* $\wedge$ *a* = $\{setsucM\ b\ b\}_M$) *then THE b. a* = $\{setsucM\ b\ b\}_M$ *else a*)

lemma *two-natM*: *natM* (*setsucM* $(\{\emptyset\}_M)(\{\emptyset\}_M)$)
using *nat-suc-nat*[*OF one-natM*]**.**

lemma *suc-sing-not-nat*: $\neg$ *natM* ($\{setsucM\ b\ b\}_M$)
proof
assume *natM* ($\{setsucM\ b\ b\}_M$)
hence *ordM* ($\{setsucM\ b\ b\}_M$)
unfolding *natM-def* **using** *natM-D* **by** *blast*
from *ordM-D1*[*OF this*, *of setsucM b b*]
have *b* = *setsucM b b*
unfolding *singleton-def'* *subsetM-def* **by** *force*
moreover have *b* ε *setsucM b b*
unfolding *set-defs* **by** *blast*
ultimately show *False*
using *mem-not-refl*[*of b*] **by** *simp*
qed

lemma *regperm-image-nat*: **assumes** *natM a*
shows *regperm a* = $\{setsucM\ a\ a\}_M$
using *assms* **by** *simp*

lemma *regperm-image-plus*: **assumes** *natM b*
shows *regperm* ($\{setsucM\ b\ b\}_M$) = *b*
proof–
have *aux*: (*THE c.* ($\{setsucM\ b\ b\}_M$) = ($\{setsucM\ c\ c\}_M$)) = *regperm* ($\{setsucM\ b\ b\}_M$)
using ‹*natM b*› *suc-sing-not-nat* **by** *fastforce*
have *aux'*: $\{setsucM\ b\ b\}_M = \{setsucM\ x\ x\}_M \Longrightarrow x = b$ **for** *x*
using *suc-unique*[*of x b*] **unfolding** *setext singleton-def'* **by** *blast*
show *?thesis*
using *the-equality*[*of* λ *c.* $\{setsucM\ b\ b\}_M = \{setsucM\ c\ c\}_M$, *OF refl aux'*]
unfolding *aux* **by** *blast*
qed

lemma *regperm-image-else*: **assumes** $\neg$ *natM a* $\nexists$ *b. natM b* $\wedge$ *a* = $\{setsucM\ b\ b\}_M$
shows *regperm a* = *a*
using *assms* **by** *force*

lemma *regperm-bij*: *bij regperm*
proof (*rule involuntory-imp-bij*)
fix *a*

show *regperm* (*regperm a*) = *a*
by (*cases natM a*, *use regperm-image-nat regperm-image-plus* **in** *force*)
(*cases* $\exists$ *b*. *natM b* $\wedge$ *a* = $\{setsucM\ b\ b\}_M$, *use regperm-image-nat regperm-image-plus* **in** *metis*, *auto*)
qed

lemma *SR-regperm-eq*: *SetRelation* (λ *x y*. *regperm x* = *y*)
proof–
let *?Def* = λ *x y*. (*natM x* $\wedge$ *y* = $\{setsucM\ x\ x\}_M$) $\vee$ ($\neg$ *natM x* $\wedge$ ($\exists$ *b*. *natM b* $\wedge$ *x* = $\{setsucM\ b\ b\}_M$ $\wedge$ *y* = *b*))
$\vee$ ($\neg$ *natM x* $\wedge$ ($\nexists$ *b*. *natM b* $\wedge$ *x* = $\{setsucM\ b\ b\}_M$) $\wedge$ *y* = *x*)
have *iff*: *?Def x y* $\longleftrightarrow$ *regperm x* = *y* **for** *x y*
proof–
consider *natM x* | $\neg$ *natM x* $\wedge$ ($\exists$ *b*. *natM b* $\wedge$ *x* = $\{setsucM\ b\ b\}_M$) |
$\neg$ *natM x* $\wedge$ ($\nexists$ *b*. *natM b* $\wedge$ *x* = $\{setsucM\ b\ b\}_M$) **by** *blast*
then show *?thesis*
by (*cases*, *force*) (*use regperm-image-plus* **in** *blast*, *use regperm-image-else* **in** *auto*)
qed
have *SetRelation ?Def*
unfolding *SetRelation-def logsimps set-defs* **by** (*rule SFP-rules*)+
thus *?thesis*
unfolding *iff*.
qed

lemma *SR-regperm-mem*: *SetRelation* (λ *x y*. *x* ε *regperm y*)
proof–
have *SetRelation* (λ *x y*. $\exists$ *z*. *x* ε *z* $\wedge$ *regperm y* = *z*)
unfolding *SetRelation-def logsimps*
by (*rule SFP-rules*)+
(*use transform-variables*[*OF SR-regperm-eq*[*unfolded SetRelation-def*], *of id*(*0*:=*1*, *1*:=-)] **in** *simp*)
thus *?thesis*
by *metis*
qed

end

context *L-setext-empty-power-union-repl-reg-fin*

begin

interpretation *perm*: *permutation-model* (ε) *regperm*
by *unfold-locales* (*use regperm-bij SR-regperm-mem* **in** *blast*)+

— membership in the model
abbreviation *fmem* (**infix** ε^f *51*) **where**
fmem x y $\equiv$ *perm.perm-mem x y*

interpretation *fm*: *L-setext-empty-power-union-repl fmem*
using *perm.perm-model*.

lemma *regperm-mem-nat*: **assumes** *natM a*
shows $u\ \varepsilon^f\ a \longleftrightarrow u = (setsucM\ a\ a)$
unfolding *regperm-image-nat*[*OF assms*] *singleton-def′*..

— First meta-theorem : not used in the main meta-theorem below.
lemma *fmem-not-wf*: $\neg\ (wfp\ (\varepsilon^f))$
proof–
let $?B = \{x\ .\ natM\ x\}$
have $?B \neq \{\}$
using *Collect-empty-eq one-natM* **by** *metis*
moreover have $z \in\ ?B \Longrightarrow (setsucM\ z\ z)\ \varepsilon^f\ z$ **for** z
unfolding *mem-Collect-eq* **using** *regperm-image-nat* **unfolding** *setext singleton-def′* **by** *blast*
moreover have $z \in\ ?B \Longrightarrow (setsucM\ z\ z) \in\ ?B$ **for** z
unfolding *mem-Collect-eq* **using** *nat-suc-nat*.
ultimately show *?thesis*
unfolding *wfp-iff-ex-minimal* **by** *metis*
qed

The main meta-theorem. The axiom of the transitive superset does not hold in the model since the transitive closure of $\emptyset$ would be an infinite set.

theorem *regperm-not-ts*: $\neg\ L\text{-}ts\ (\varepsilon^f)$
proof
assume $L\text{-}ts\ (\varepsilon^f)$
then interpret $L\text{-}ts\ (\varepsilon^f)$.
— take the transitive closure of $\emptyset$
interpret $L\text{-}setext\text{-}sep\text{-}ts\ (\varepsilon^f)$
by *unfold-locales*
obtain t **where** $fm.subsetM\ \emptyset\ t\ fm.transM\ t$ **and** *least*: $\forall u.\ u\ \varepsilon^f\ t \longleftrightarrow (\forall s.\ fm.transM\ s \wedge\ fm.subsetM\ \emptyset\ s \longrightarrow u\ \varepsilon^f\ s)$
using *least-ts-is-transitive*[*of* $\emptyset$] *least-ts-def′*[*of* - $\emptyset$] *fm.subsetM-def* **by** *force*
have *trans-t*: $z\ \varepsilon^f\ t$ **if** $y\ \varepsilon^f\ t\ z\ \varepsilon^f\ y$ **for** $y\ z$
using ‹*fm.transM t*› *that* **unfolding** *fm.transM-def fm.subsetM-def* **by** *blast*
— t contains an infinite ε^f chain of natural numbers
have *suc-t*: $setsucM\ u\ u\ \ \varepsilon^f\ t$ **if** $natM\ u\ u\ \varepsilon^f\ t$ **for** u
using *trans-t*[*OF* ‹$u\ \varepsilon^f\ t$›] **using** *regperm-mem-nat*[*OF* ‹*natM u*›] **by** *blast*
— we show that ε^f coincides with ε in t
have *one-in-t*: $\{\emptyset\}_M\ \varepsilon^f\ t$
using ‹*fm.subsetM* $\emptyset$ *t*›**unfolding** *fm.subsetM-def* **using** *regperm-mem-nat*[*OF emp-natM*]
setsuc-empty-sing **by** *fastforce*
have *two-in-t*: $setsucM\ (\{\emptyset\}_M)\ (\{\emptyset\}_M)\ \varepsilon^f\ t$
using *one-in-t suc-t*[*OF one-natM*] **by** *blast*
have *tmem*: $u\ \varepsilon^f\ t \longleftrightarrow u\ \varepsilon\ t$ **for** u
proof–
have $\neg\ natM\ t$

proof
assume *natM t*
have *iff*: $u\ \varepsilon^f\ t \longleftrightarrow u = setsucM\ t\ t$ **for** *u*
unfolding *regperm-image-nat*[*OF* ‹*natM t*›] *singleton-def′*..
have $\neg\ \{\emptyset\}_M\ \varepsilon\ setsucM\ (\{\emptyset\}_M)\ (\{\emptyset\}_M)$
unfolding *two-in-t*[*unfolded iff*, *folded one-in-t*[*unfolded iff*]]
by (*rule mem-not-refl*)
then show *False*
unfolding *setsuc-def′* **by** *blast*
qed
have $\nexists\ b.\ \ natM\ b \wedge t = \{setsucM\ b\ b\}_M$
proof
assume $\exists\, b.\ natM\ b \wedge t = \{setsucM\ b\ b\}_M$
then obtain *b* **where** $natM\ b\ t = \{setsucM\ b\ b\}_M$
by *blast*
hence *iff*: $u\ \varepsilon^f\ t \longleftrightarrow u\ \varepsilon\ b$ **for** *u*
using *regperm-image-plus* **by** *auto*
have $b \neq \emptyset$
using *iff one-in-t* **by** *auto*
hence $\emptyset\ \varepsilon\ b$
using ‹*natM b*› **by** (*simp add*: *empty-mem-ord natM-def*)
have $setsucM\ y\ y\ \varepsilon\ b$ **if** $y\ \varepsilon\ b$ **for** *y*
using *suc-t*[*unfolded iff*, *OF - that*] *nat-mem-nat*[*OF - that*] ‹*natM b*› **by** *blast*
then show *False*
using ‹$\emptyset\ \varepsilon\ b$› *fin* **by** *blast*
qed
show *?thesis*
by (*simp add*: ‹$\nexists\, b.\ natM\ b \wedge t = \{setsucM\ b\ b\}_M$› ‹$\neg$ *natM t*›)
qed
— We get a contradiction with the finiteness axiom since the subset of t consisting of natural numbers is an inductive set.
thm *notE*[*OF fin*[*unfolded not-ex*, *rule-format*]]
show *False*
proof (*rule notE*[*OF fin*[*unfolded not-ex*, *rule-format*]], *rule conjI*)
define *tnatM* **where** $tnatM = separationM\ t\ natM$
have *tnat*: $u\ \varepsilon\ tnatM \longleftrightarrow natM\ u \wedge u\ \varepsilon\ t$ **for** *u*
using *separ-def-SP tnatM-def* **by** *auto*
show $\emptyset\ \varepsilon\ setsucM\ tnatM\ \emptyset$
unfolding *set-defs* **by** *simp*
show $\forall\, y.\ y\ \varepsilon\ setsucM\ tnatM\ \emptyset \longrightarrow setsucM\ y\ y\ \varepsilon\ setsucM\ tnatM\ \emptyset$
proof (*rule allI*, *rule impI*)
fix *y*
assume $y\ \varepsilon\ setsucM\ tnatM\ \emptyset$
then consider $y = \emptyset \mid y\ \varepsilon\ tnatM$
unfolding *set-defs* **by** *blast*
then show $setsucM\ y\ y\ \varepsilon\ setsucM\ tnatM\ \emptyset$
proof (*cases*)
assume $y = \emptyset$

then show *?thesis*
using *one-in-t*[*unfolded tmem*] *setsuc-empty-sing tnat* **unfolding** *bin-union-def′* **by** *simp*
next
assume $y\ \varepsilon\ tnatM$
then show *?thesis*
unfolding *tnat binunion-def′ setsuc-def′* **using** *suc-t*[*unfolded tmem*, *of y*] *nat-suc-nat*[*of y*] **by** *blast*
qed
qed
qed
qed

— It follows that the scheme of regularity does not hold in the model, since it entails transitive supersets.
theorem *regperm-not-regsch*: $\neg$ *L-regsch* (ε^f)
using *regperm-not-ts epsind-regsch-iff fm.epsind-implies-ts* **by** *blast*

lemma *regperm-reg*:
shows *L-setext-empty-power-union-repl-reg* (ε^f)
proof (*unfold-locales*, *rule allI*, *rule impI*)
fix a
assume $\exists y.\ y\ \varepsilon^f\ a$
show $\exists y.\ y\ \varepsilon^f\ a \wedge (\forall v.\ \neg\ (v\ \varepsilon^f\ y \wedge v\ \varepsilon^f\ a))$
proof−
consider $\exists\ n.\ natM\ n \wedge n\ \varepsilon^f\ a \mid (\nexists\ n.\ natM\ n \wedge n\ \varepsilon^f\ a) \wedge (\exists\ m.\ natM\ m \wedge (\{setsucM\ m\ m\}_M)\ \varepsilon^f\ a) \mid$
$(\nexists\ n.\ natM\ n \wedge n\ \varepsilon^f\ a) \wedge (\nexists\ m.\ natM\ m \wedge (\{setsucM\ m\ m\}_M)\ \varepsilon^f\ a)$
by *blast*
then show *?thesis*
proof (*cases*)
assume $\exists\ n.\ natM\ n \wedge n\ \varepsilon^f\ a$
then obtain k **where** $natM\ k\ k\ \varepsilon^f\ a$ **and** $\bigwedge k'.\ natM\ k' \Longrightarrow k'\ \varepsilon^f\ a \Longrightarrow \neg\ k \subset_M k'$
using *max-mem*[*OF - - SP-nat*, *of - regperm a*] **by** *fast*
then have *max-k*: $\bigwedge k'.\ natM\ k' \Longrightarrow k'\ \varepsilon^f\ a \Longrightarrow k' \neq k \Longrightarrow k'\ \varepsilon\ k$
unfolding *proper-subset-def′* **using** *natM-D ordM-D1 ordM-total natM-def* **by** *metis*
show $\exists y.\ y\ \varepsilon^f\ a \wedge (\forall v.\ \neg\ (v\ \varepsilon^f\ y \wedge v\ \varepsilon^f\ a))$
proof (*rule exI*[*of - k*], *rule conjI*, *fact*)
show $\forall v.\ \neg\ (v\ \varepsilon^f\ k \wedge v\ \varepsilon^f\ a)$
proof (*rule allI*, *rule notI*)
fix v
assume $v\ \varepsilon^f\ k \wedge v\ \varepsilon^f\ a$
hence $v\ \varepsilon^f\ a\ v\ \varepsilon^f\ k\ v = setsucM\ k\ k$
unfolding *regperm-image-nat*[*OF ‹natM k›*] *singleton-def′* **by** *blast+*
show *False*
using *max-k*[*of v*, *OF - ‹v* ε^f *a›*, *unfolded ‹v = setsucM k k›*]
‹natM k› mem-antisym nat-suc-nat setsuc-def′ natM-def **by** *blast*

qed
qed
next
assume $(\nexists\, n.\ natM\ n \wedge n\ \varepsilon^f\ a) \wedge (\exists\, m.\ natM\ m \wedge \{setsucM\ m\ m\}_M\ \varepsilon^f\ a)$
then obtain m **where** $natM\ m\ \{setsucM\ m\ m\}_M\ \varepsilon^f\ a\ \nexists\, n.\ natM\ n \wedge n\ \varepsilon^f\ a$
by *blast*
show $\exists\, y.\ y\ \varepsilon^f\ a \wedge (\forall\, v.\ \neg\ (v\ \varepsilon^f\ y \wedge v\ \varepsilon^f\ a))$
proof (*rule exI*[*of* - $\{setsucM\ m\ m\}_M$], *rule conjI*, *fact*)
show $\forall\, v.\ \neg\ (v\ \varepsilon^f\ \{setsucM\ m\ m\}_M \wedge v\ \varepsilon^f\ a)$
unfolding *regperm-image-plus*[*OF* ‹*natM m*›]
using ‹$\nexists\, n.\ natM\ n \wedge n\ \varepsilon^f\ a$› ‹*natM m*› *nat-mem-nat* **by** *blast*
qed
next
assume *case3*: $(\nexists\, n.\ natM\ n \wedge n\ \varepsilon^f\ a) \wedge (\nexists\, m.\ natM\ m \wedge \{setsucM\ m\ m\}_M\ \varepsilon^f\ a)$
hence $\neg\ natM\ a$
using *nat-suc-nat*[*of a*]
singleton-def' **by** *force*
have $\nexists\ b.\ natM\ b \wedge a = \{setsucM\ b\ b\}_M$
using *case3* ‹$\exists\, y.\ y\ \varepsilon^f\ a$› *nat-mem-nat regperm-image-plus* **by** *metis*
hence *a-mem*: $u\ \varepsilon^f\ a \longleftrightarrow u\ \varepsilon\ a$ **for** u
using ‹$\neg\ natM\ a$› **by** *auto*
have $a \neq \emptyset$
using ‹$\neg\ natM\ a$› *emp-natM* **by** *blast*
then obtain b **where** $b\ \varepsilon\ a\ \forall\ v.\ \neg\ (v\ \varepsilon\ b \wedge v\ \varepsilon\ a)$
using *reg* **by** (*metis min-subset-ex*)
have $\neg\ natM\ b\ \nexists\ m.\ natM\ m \wedge b = \{setsucM\ m\ m\}_M$
using ‹$b\ \varepsilon\ a$› *case3* **unfolding** *a-mem* **by** *blast*+
hence *b-mem*: $u\ \varepsilon^f\ b \longleftrightarrow u\ \varepsilon\ b$ **for** u
by *auto*
show $\exists\, y.\ y\ \varepsilon^f\ a \wedge (\forall\, v.\ \neg\ (v\ \varepsilon^f\ y \wedge v\ \varepsilon^f\ a))$
by (*rule exI*[*of* - *b*], *rule conjI*, *unfold a-mem b-mem*) *fact*+
qed
qed
qed

lemma *perm-empty*: $fm.emptysetM = \{\{\emptyset\}_M\}_M$
proof (*rule sym*, *unfold fm.emptyset-def'*)
have $regperm\ (\{\{\emptyset\}_M\}_M) = \emptyset$
using *regperm-image-plus*[*OF emp-natM*] **unfolding** *setsuc-empty-sing*.
show $\forall\, u.\ \neg\ u\ \varepsilon\ (regperm\ (\{\{\emptyset\}_M\}_M))$
unfolding ‹$regperm\ (\{\{\emptyset\}_M\}_M) = \emptyset$› **by** (*fact empty-is-empty*)
qed

lemma *perm-one*: $fm.binunionM\ fm.emptysetM\ (fm.singletonM\ fm.emptysetM) = \{\{\{\emptyset\}_M\}_M\}_M$
proof–
have *L*: $z\ \varepsilon^f\ fm.binunionM\ fm.emptysetM\ (fm.singletonM\ fm.emptysetM) \longleftrightarrow z = fm.emptysetM$ **for** z

unfolding *fm.setext fm.binunion-def′ fm.singleton-def′*
using *fm.nemp-setI fm.setsuc-def′* **by** *blast*
have $z\ \varepsilon^f\ \{\{\{\emptyset\}_M\}_M\}_M \longleftrightarrow z\ \varepsilon\ \{\{\{\emptyset\}_M\}_M\}_M$ **for** z
using *singleton-def′ binunion-emp nat-mem-nat nat-suc-nat*
regperm-image-else[*of* $\{\{\{\emptyset\}_M\}_M\}_M$] *suc-sing-not-nat*[*of* $\emptyset$] **unfolding**
setsuc-empty-sing **by** *metis*
hence *R*: $z\ \varepsilon^f\ \{\{\{\emptyset\}_M\}_M\}_M \longleftrightarrow z = fm.emptysetM$ **for** z
unfolding *perm-empty singleton-def′*.
show *?thesis*
unfolding *fm.setext L R* **by** *blast*
qed

lemma *perm-else-setsuc-else*: **assumes** $\neg$ *natM u* $\nexists$ *b. natM b* $\wedge$ *u* = $\{$*setsucM b b*$\}_M$
shows $\neg$ *natM* (*setsucM u u*) $\nexists$ *b. natM b* $\wedge$ *setsucM u u* = $\{$*setsucM b b*$\}_M$
using *assms* **by** (*meson nat-mem-nat setsuc-def′*) (*metis mem-not-refl singleton-def′ setsuc-def′*)

lemma *perm-setsuc*: **assumes** $\neg$ *natM u* $\nexists$ *b. natM b* $\wedge$ *u* = $\{$*setsucM b b*$\}_M$
shows *fm.setsucM u u* = *setsucM u u*
unfolding *fm.setext* **unfolding** *fm.binunion-def′ regperm-image-else*[*OF perm-else-setsuc-else*[*OF assms*]]
regperm-image-else[*OF assms*] *binunion-def′ singleton-def′ fm.singleton-def′ setsuc-def′*
using ‹*regperm u* = *u*› *fm.setsuc-def′* **by** *auto*

lemma *perm-fin*: *L-fin* (ε^f)
proof(*unfold-locales*, *rule notI*)
assume $\exists x$. *fm.emptysetM* ε^f *x* $\wedge$ ($\forall y$. *y* ε^f *x* $\longrightarrow$ *fm.setsucM y y* ε^f *x*)
then obtain *x* **where** *xf*: *fm.emptysetM* ε^f *x* $\forall y$. *y* ε^f *x* $\longrightarrow$ *fm.setsucM y y* ε^f *x*
by *blast*
hence *mem0*: $\{\{\emptyset\}_M\}_M\ \varepsilon^f\ x$
using *perm-empty* **by** *force*
from *xf*(*2*)[*rule-format*, *OF xf*(*1*), *unfolded perm-one*]
have *mem1*: $\{\{\{\emptyset\}_M\}_M\}_M\ \varepsilon^f\ x$
using *fm.binunion-emp fm.setsuc-empty-sing perm-one* **by** *force*
hence *not1*: $\neg$ *natM x*
using *mem0* **by** (*metis mem-neq-singleton regperm-mem-nat*)
have *not2*: $\nexists$ *b. natM b* $\wedge$ *x* = $\{$*setsucM b b*$\}_M$
using *mem0 mem1*
using *binunion-emp nat-mem-nat regperm-image-plus suc-sing-not-nat setsuc-empty-sing* **by** *metis*
from *regperm-image-else*[*OF not1 not2*]
have *x*: $\{\{\emptyset\}_M\}_M\ \varepsilon\ x\ \forall y.\ y\ \varepsilon\ x \longrightarrow$ *fm.setsucM y y* ε *x*
using *xf perm-empty* **by** *force+*
let *?P* = λ *u*. ($\neg$ *natM u* $\wedge$ ($\nexists$ *b. natM b* $\wedge$ (*u* = $\{$*setsucM b b*$\}_M$)))
have *sp*: *SetProperty ?P*
unfolding *SetProperty-def set-defs logsimps* **by** (*rule SFP-rules*)+

from *sep-SP*[*OF this*, *rule-format*, *of x*]
obtain x' **where** x': $\forall u.\ (u\ \varepsilon\ x') = (u\ \varepsilon\ x \wedge \neg\ natM\ u \wedge (\nexists b.\ natM\ b \wedge u = \{setsucM\ b\ b\}_M))$
by *blast*
have $\{\{\{\emptyset\}_M\}_M\}_M\ \varepsilon\ x'$
using *mem1* **unfolding** x'[*rule-format*] ‹*regperm x = x*›
using *binunion-emp nat-mem-nat nat-suc-nat singleton-def′ suc-sing-not-nat setsuc-empty-sing* **by** *metis*
hence $x' \neq \emptyset$
by *force*
have x'*-setsuc*: *setsucM u u* $\varepsilon\ x'$ **if** $u\ \varepsilon\ x'$ **for** *u*
unfolding x'[*rule-format*]
proof(*unfold conj-assoc*[*symmetric*], *rule conjI*, *rule conjI*)
note *u* = ‹$u\ \varepsilon\ x'$›[*unfolded* x'[*rule-format*]]
have *red*: *fm.setsucM u u* = *setsucM u u*
by (*rule perm-setsuc*)
(*use* ‹$u\ \varepsilon\ x'$›[*unfolded* x'[*rule-format*]] **in** *blast*)+
show *setsucM u u* ε *x*
by (*rule* *x*(*2*)[*rule-format*, *of u*, *unfolded red*]) (*simp add*: *u*)
show $\neg\ natM\ (setsucM\ u\ u)\ \nexists b.\ natM\ b \wedge setsucM\ u\ u = \{setsucM\ b\ b\}_M$
using *u perm-else-setsuc-else* **by** *blast*+
qed
with *max-subset-ex*[*OF* ‹$x' \neq \emptyset$›]
show *False*
using *suc-subset* **by** *blast*
qed

interpretation *find*: *L-setind* (ε^f)
using *L-setext-empty-power-union-repl-reg.fin-implies-setind regperm-reg perm-fin*
by *blast*

end

Summary of results. We have shown that if a type admits a membership relation satisfying certain axioms of finite sets (namely axioms of ZF where inf is replaced by fin), it does not follow that the membership on the type satisfies the existence of transitive supersets or the regularity schema.

theorem *not-reg-setind-implies-regsch-ts*:
assumes *L-setext-empty-power-union-repl-reg-fin* $(m :: 'a \Rightarrow 'a \Rightarrow bool)$
shows $\neg\ (\forall\ (mem :: 'a \Rightarrow 'a \Rightarrow bool).$ *L-setext-empty-power-union-repl-reg mem* $\wedge$ *L-setind mem* $\longrightarrow$ *L-regsch mem* $\vee$ *L-ts mem*)
proof (*rule notI*)
assume *contr*: $\forall (mem :: 'a \Rightarrow 'a \Rightarrow bool).$ *L-setext-empty-power-union-repl-reg mem* $\wedge$ *L-setind mem* $\longrightarrow$ *L-regsch mem* $\vee$ *L-ts mem*
interpret *L-setext-empty-power-union-repl-reg-fin m*
using *assms*.
have *L1*: *L-setind fmem*
using *L-setext-empty-power-union-repl-reg.fin-implies-setind regperm-reg perm-fin*
by *blast*

have *L2*: *L-setext-empty-power-union-repl-reg fmem*
using *L-setext-empty-power-union-repl-def L-setext-empty-power-union-repl-reg-def regperm-reg*
by *blast*
have *L3*: *L-regsch fmem* $\vee$ *L-ts fmem*
using *L1 L2 contr* **by** *blast*
interpret *i2*: *L-setext-empty-power-union-repl-reg fmem*
using *L2*.
show *False*
using *L3 regperm-not-regsch regperm-not-ts* **by** *blast*
qed

end

References

[1] S. Baratella and R. Ferro. A theory of sets with the negation of the axiom of infinity. *Mathematical Logic Quarterly*, 39:338–352, 1993.

[2] L. Běhounek. Nezávislost axiomů ve dvou axiomatikách teorie konečných množin. Ročníková práce (equivalent of bachelor thesis), 1998.

[3] R. Kaye and T. L. Wong. On interpretations of arithmetic and set theory. *Notre Dame Journal of Formal Logic*, 48(4):497–510, 2007.

[4] A. Mancini and D. Zambella. A note on recursive models of set theories. *Notre Dame Journal of Formal Logic*, 42(2):109–115, 2001.

[5] P. Pudlák and A. Sochor. Models of the Alternative Set Theory. *The Journal of Symbolic Logic*, 49(2):570–585, 1984.

[6] L. S. Rieger. A contribution to Gödel's axiomatic set theory, I. *Czechoslovak Mathematical Journal*, 3:323–357, 1957.

[7] A. Sochor. Metamathematics of the alternative set theory I. *Commentationes Mathematicae Universitatis Carolinae*, 20(4):697–722, 1979.

[8] A. Sochor. Metamathematics of the alternative set theory II. *Commentationes Mathematicae Universitatis Carolinae*, 23(1):55–79, 1982.

[9] A. Sochor. Metamathematics of the alternative set theory III. *Commentationes Mathematicae Universitatis Carolinae*, 24(1):137–154, 1983.

[10] P. Vopěnka. *Mathematics in the Alternative Set Theory.* Teubner, Leipzig, 1979.

[11] P. Vopěnka and P. Hájek. Über die Gültigkeit des Fundierungsaxioms in speziellen Systemen der Mengentheorie. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:235–241, 1963.