Axiomatic theory of hereditarily finite sets and its fragments.

Štěpán Holub Zuzana Haniková

July 5, 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 Beranys-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

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*]: $(\exists\, w.\ (w = x \vee w = y) \wedge P\ u\ w) \longleftrightarrow (P\ u\ x \vee P\ u\ y)$
⟨*proof*⟩
lemma *abstract-foundation-iff*: $((\exists\, x.\ P\ x) \longrightarrow (\exists\, x.\ B\ x \wedge P\ x)) \longleftrightarrow (\forall\ x.\ B\ x \longrightarrow \neg\ P\ x) \longrightarrow (\forall\ x.\ \neg\ P\ x)$
⟨*proof*⟩

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* $\lambda\ x.\ \neg\ P\ x\ \lambda\ x.\ (\forall\ y.\ y\ \varepsilon\ x \longrightarrow (P\ y))$, *unfolded not-not*, *symmetric*]

thm *abstract-foundation-iff*[*of* $\lambda\ y.\ P\ y\ \lambda\ x.\ (\forall\ y.\ y\ \varepsilon\ x \longrightarrow \neg\ (P\ y))$, *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 $\equiv (\forall\ z.\ z\ \varepsilon\ x \longrightarrow z\ \varepsilon\ y)$

lemma *subsetM-refl*[*simp*]: $x \subseteq_M x$
⟨*proof*⟩

lemma *subsetM-trans*[*simp*]: **assumes** $x \subseteq_M y$ $y \subseteq_M z$ **shows** $x \subseteq_M z$
⟨*proof*⟩

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

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

lemma *proper-subset-def'*: $x \subset_M y \longleftrightarrow x \subseteq_M y \wedge x \neq y$
⟨*proof*⟩

lemma *proper-subsetI*: $x \subseteq_M y \Longrightarrow x \neq y \Longrightarrow x \subset_M y$
⟨*proof*⟩

lemma *proper-subsetM-irrefl*[*simp*]: $\neg\ x \subset_M x$
⟨*proof*⟩

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

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* . $(\forall\ u\ .\ (u\ \varepsilon\ z \longleftrightarrow Q\ u))$

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

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

definition *powersetM* :: $'a \Rightarrow 'a$ ($\mathfrak{P}$) **where**

powersetM x ≡ *collectM* (λ *u*. *u* ⊆$_M$ *x*)

definition *binunionM* :: *'a* ⇒ *'a* ⇒ *'a* (**infixl** ∪$_M$ *55*) **where**
binunionM x y ≡ *collectM* (λ *u*. *u* ε *x* ∨ *u* ε *y*)

definition *intersectionM* :: *'a* ⇒ *'a* ⇒ *'a* (- ∩$_M$ - [*55*,*54*] *60*) **where**
intersectionM x y ≡ *collectM* (λ *u*. *u* ε *x* ∧ *u* ε *y*)

lemma *intersection-symm*: *x* ∩$_M$ *y* = *y* ∩$_M$ *x*
⟨*proof*⟩

definition *unionM* :: *'a* ⇒ *'a* (⋃$_M$) **where**
unionM x ≡ *collectM* (λ *u*. (∃ *v*. *v* ε *x* ∧ *u* ε *v*))

definition *differenceM* :: *'a* ⇒ *'a* ⇒ *'a* (- \$_M$ -) **where**
differenceM x y ≡ *collectM* (λ *u*. *u* ε *x* ∧ ¬ *u* ε *y*)

definition *singletonM* :: *'a* ⇒ *'a* ({-}$_M$ *70*) **where**
singletonM x ≡ *collectM* (λ *u*. *u* = *x*)

definition *pairM* :: *'a* ⇒ *'a* ⇒ *'a* ({-,-}$_M$ [*51*,*51*] *60*) **where**
pairM x y ≡ *collectM* (λ *u*. *u* = *x* ∨ *u* = *y*)

— Set successor. Also known as adjunction.
definition *setsucM* :: *'a* ⇒ *'a* ⇒ *'a* (𝔰𝔲𝔠) **where**
setsucM x y ≡ *collectM* (λ *u*. *u* ε *x* ∨ *u* = *y*)

definition *replaceM* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *'a* ⇒ *'a* **where**
replaceM P x ≡ *collectM* (λ *u*. ∃ *v*. *v* ε *x* ∧ *P v u*)

definition *leastM* :: (*'a* ⇒ *bool*) ⇒ *'a* (⋂$_M$) **where**
leastM Q ≡ *collectM* (λ *u*. ∀ *y*. *Q y* ⟶ *u* ε *y*)

Least transitive superset

definition *least-tsM* :: *'a* ⇒ *'a* **where**
least-tsM x = ⋂$_M$ (λ *y*. *transM y* ∧ *x* ⊆$_M$ *y*)

definition *ordered-pairM* :: *'a* ⇒ *'a* ⇒ *'a* (⟨-,-⟩ [*51*,*51*] *60*) **where**
ordered-pairM a b ≡ {{*a*}$_M$,{*a*,*b*}$_M$}$_M$

definition *relationM* :: *'a* ⇒ *bool* **where**
relationM x ≡ ∀ *v*. *v* ε *x* ⟶ (∃ *a b*. *v* = ⟨*a*,*b*⟩)

definition *rel-inverseM* :: *'a* ⇒ *'a* **where**
rel-inverseM r ≡ *collectM* (λ *u*. ∃ *a b*. ⟨*a*,*b*⟩ ε *r* ∧ *u* = ⟨*b*,*a*⟩)

definition *functionM* :: *'a* ⇒ *bool* **where**
functionM x ≡ *relationM x* ∧ (∀ *a b c*. ⟨*a*,*b*⟩ ε *x* ∧ ⟨*a*,*c*⟩ ε *x* ⟶ *b* = *c*)

lemma *funD*: *functionM f* $\Longrightarrow \langle a,b\rangle\ \varepsilon\ f \Longrightarrow \langle a,c\rangle\ \varepsilon\ f \Longrightarrow\ b = c$
⟨*proof*⟩

definition *domM* :: $'a \Rightarrow 'a$ **where**
domM r ≡ *collectM* $(\lambda\ u.\ \exists\ v.\ \langle u,v\rangle\ \varepsilon\ r)$

definition *rngM* :: $'a \Rightarrow 'a$ **where**
rngM r ≡ *collectM* $(\lambda\ u.\ \exists\ v.\ \langle v,u\rangle\ \varepsilon\ r)$

definition *one-oneM* :: $'a \Rightarrow$ *bool* **where**
one-oneM f ≡ *functionM f* ∧ $(\forall\ a\ b\ c.\ \langle b,a\rangle\ \varepsilon\ f \wedge \langle c,a\rangle\ \varepsilon\ f \longrightarrow b = c)$

lemma *one-one-inj*: *one-oneM f* $\Longrightarrow \langle b,a\rangle\ \varepsilon\ f \Longrightarrow \langle c,a\rangle\ \varepsilon\ f \Longrightarrow\ b = c$
⟨*proof*⟩

lemma *one-oneI*: **assumes** $\forall\ v.\ v\ \varepsilon\ f \longrightarrow (\exists\ a\ b.\ v = \langle a,b\rangle)$ $(\forall\ a\ b\ c.\ \langle b,a\rangle\ \varepsilon\ f \wedge \langle c,a\rangle\ \varepsilon\ f \longrightarrow b = c)$ $(\forall\ a\ b\ c.\ \langle a,b\rangle\ \varepsilon\ f \wedge \langle a,c\rangle\ \varepsilon\ f \longrightarrow b = c)$
shows *one-oneM f*
⟨*proof*⟩

lemma *one-oneD1*: *one-oneM f* $\Longrightarrow \forall\ v.\ v\ \varepsilon\ f \longrightarrow (\exists\ a\ b.\ v = \langle a,b\rangle)$
⟨*proof*⟩

lemma *one-oneD2*: *one-oneM f* $\Longrightarrow (\forall\ a\ b\ c.\ \langle b,a\rangle\ \varepsilon\ f \wedge \langle c,a\rangle\ \varepsilon\ f \longrightarrow b = c)$
⟨*proof*⟩

lemma *one-oneD3*: *one-oneM f* $\Longrightarrow (\forall\ a\ b\ c.\ \langle a,b\rangle\ \varepsilon\ f \wedge \langle a,c\rangle\ \varepsilon\ f \longrightarrow b = c)$
⟨*proof*⟩

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

definition *cartesian-productM* :: $'a \Rightarrow 'a \Rightarrow 'a$ $(\text{-} \times_M \text{-}\ [51,51]\ 60)$ **where**
cartesian-productM x y ≡ *collectM* $(\lambda\ u.\ \exists\ v_1\ v_2.\ v_1\ \varepsilon\ x \wedge v_2\ \varepsilon\ y \wedge u = \langle v_1,v_2\rangle)$

definition *composeM* :: $'a \Rightarrow 'a \Rightarrow 'a$ $(\text{-} \circ_M \text{-}\ [51,51]\ 60)$ **where**
f $\circ_M$ *g* = *collectM* $(\lambda\ u.\ \exists\ a\ b\ c.\ \langle a,b\rangle\ \varepsilon\ g \wedge \langle b,c\rangle\ \varepsilon\ f \wedge u = \langle a,c\rangle)$

definition *tarski-fin* :: $'a \Rightarrow$ *bool* **where**
tarski-fin x ≡ $\forall\ y.\ (\forall\ z.\ z\ \varepsilon\ y \longrightarrow z \subseteq_M x) \longrightarrow (\exists\ z.\ z\ \varepsilon\ y) \longrightarrow (\exists\ z.\ z\ \varepsilon\ y \wedge \neg(\exists\ w.\ w\ \varepsilon\ y \wedge z \subset_M w))$

lemma *tarski-subset-tarski*: **assumes** *tarski-fin x y* $\subseteq_M$ *x* **shows** *tarski-fin y*
⟨*proof*⟩

definition *dedekind-fin* :: $'a \Rightarrow$ *bool* **where**
dedekind-fin x ≡ $\forall\ y.\ (y \subset_M x \longrightarrow \neg\ x \approx_M y)$

— *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
⟨*proof*⟩

lemma *ordM-regular*[*simp*]: *ordM* $x \Longrightarrow$ *regular* x
⟨*proof*⟩

lemma *ordM-D1*: *ordM* $x \Longrightarrow y\ \varepsilon\ x \Longrightarrow y \subseteq_M x$
⟨*proof*⟩

lemma *ordM-trans*: **assumes** *ordM* v $w\ \varepsilon\ u\ u\ \varepsilon\ v$ **shows** $w\ \varepsilon\ v$
⟨*proof*⟩

lemma *ordM-D2*: *ordM* $x \Longrightarrow y\ \varepsilon\ x \Longrightarrow z\ \varepsilon\ x \Longrightarrow y\ \varepsilon\ z \vee y = z \vee z\ \varepsilon\ y$
⟨*proof*⟩

definition *is-sucM* :: $'a \Rightarrow$ *bool* **where**
is-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-sucM* $v)$

lemma *natM-I*: *ordM* $x \Longrightarrow$ *is-sucM* $x \Longrightarrow (\bigwedge v.\ \exists\ u.\ u\ \varepsilon\ v \Longrightarrow v\ \varepsilon\ x \Longrightarrow$ *is-sucM* $v) \Longrightarrow$ *natM* x
⟨*proof*⟩

lemma *nat-is-suc*: *natM* $x \Longrightarrow \exists\ u.\ u\ \varepsilon\ x \Longrightarrow$ *is-sucM* x
⟨*proof*⟩

lemma *nat-mem-is-suc*: *natM* $x \Longrightarrow v\ \varepsilon\ x \Longrightarrow \exists\ u.\ u\ \varepsilon\ v \Longrightarrow$ *is-sucM* v
⟨*proof*⟩

lemma *natM-D*: *natM* $x \Longrightarrow$ *ordM* x
⟨*proof*⟩

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* (λ Ξ. (Ξ *m*) ε (Ξ *n*)) — formula $x_m \in x_n$
| *SFP-eq*: $\bigwedge$ *m n. SetFormulaPredicate* (λ Ξ. (Ξ *m*) = (Ξ *n*)) — formula $x_m = x_n$
| *SFP-neg*: *SetFormulaPredicate P* $\Longrightarrow$ *SetFormulaPredicate* (λ Ξ. $\neg$ *P* Ξ) — formula $\neg \varphi$
| *SFP-disj*: *SetFormulaPredicate P* $\Longrightarrow$ *SetFormulaPredicate Q*
$\Longrightarrow$ *SetFormulaPredicate* (λ Ξ. *P* Ξ $\vee$ *Q* Ξ) — formula $\varphi \vee \psi$
| *SFP-all*: $\bigwedge$ *n. SetFormulaPredicate P* $\Longrightarrow$ *SetFormulaPredicate* (λ Ξ. $\forall$ *a. P* (Ξ(*n*:=*a*)))
— formula $\forall$ x_n. φ

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.* $\forall$ Ξ Ξ'. ($\forall$ *i* < *m.* Ξ *i* = Ξ' *i*) $\longrightarrow$ *P*(Ξ) = *P* (Ξ')
⟨*proof*⟩
lemma *transform-variables*: **assumes** *SetFormulaPredicate P*
shows *SetFormulaPredicate* (λ Ξ. *P* (λ *n.* Ξ(*f n*)))
⟨*proof*⟩

lemma *update-variable*[*intro*]: **assumes** *SetFormulaPredicate P*
shows *SetFormulaPredicate* (λ Ξ. *P*(Ξ(*n*:= Ξ *m*)))
⟨*proof*⟩

lemma *swap-variables*: **assumes** *SetFormulaPredicate P*
shows *SetFormulaPredicate* (λ Ξ. *P* (Ξ(*k* := Ξ *l*, *l* := Ξ *k*)))
⟨*proof*⟩

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*))
⟨*proof*⟩

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)))
⟨*proof*⟩
lemma *reduce-and* [*logsimps*]: (λ Ξ. (*P* Ξ) $\wedge$ (*Q* Ξ)) = (λ Ξ. ($\neg$ (($\neg$ (*P* Ξ)) $\vee$ ($\neg$ (*Q* Ξ)))))
⟨*proof*⟩
lemma *reduce-imp*[*logsimps*]: (λ Ξ. (*P* Ξ) $\longrightarrow$ (*Q* Ξ)) = (λ Ξ. ($\neg$ (*P* Ξ) $\vee$ (*Q* Ξ)))
⟨*proof*⟩
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)))
⟨*proof*⟩

lemma *SFP-const*[*intro*,*simp*]: *SetFormulaPredicate* ($\lambda\Xi$. *b*)
⟨*proof*⟩

lemma *SFP-ex*: **assumes** *SetFormulaPredicate P* **shows** *SetFormulaPredicate* (λ Ξ. ($\exists$ *z*. *P* (Ξ (*m* := *z*))))
⟨*proof*⟩

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*⟩

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* (λ Ξ. *P* (Ξ *0*) (Ξ *1*))

A set-theoretically definable property is defined similarly.

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

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

lemma *SR-neg*[*simp*, *intro*]: *SetRelation P* $\Longrightarrow$ *SetRelation* (λ *x y*. $\neg$ *P x y*)
⟨*proof*⟩

lemma *SR-sym*[*simp*, *intro*]: **assumes** *SetRelation P* **shows** *SetRelation* (λ *x y*. *P y x*)
⟨*proof*⟩

lemma *SR-neq*[*simp*]: *SetRelation* ($\neq$)
⟨*proof*⟩

lemma *SP-const*[*simp*]: *SetProperty* (λ *x*. *b*)
⟨*proof*⟩

lemma *SP-neg*[*simp*, *intro*]: *SetProperty P* $\Longrightarrow$ *SetProperty* (λ *x*. $\neg$ *P x*)
⟨*proof*⟩

lemma *SP-tarski*[*simp*]: *SetProperty tarski-fin*
⟨*proof*⟩

lemma *SP-setsuc*[*simp*]: *SetProperty is-sucM*
⟨*proof*⟩

lemma *SP-nat*[*simp*]: *SetProperty natM*
⟨*proof*⟩

lemma *empty-mem-ord'*: **assumes** *ordM x* $\exists$ *z*. *z* ε *x*
shows $\exists$ *emp*. ($\forall$ *u*. $\neg$ *u* ε *emp*) $\wedge$ *emp* ε *x*
⟨*proof*⟩

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*: $\neg$ ($\exists$ *x*. $\emptyset$ ε *x* $\wedge$ ($\forall$ *y*. *y* ε *x* $\longrightarrow$ *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*: $\bigwedge$ *P*. *SetFormulaPredicate P* $\Longrightarrow$

$P(\Xi(0{:=}\ \emptyset)) \longrightarrow (\forall\ x\ y.\ P(\Xi(0{:=}\ x)) \longrightarrow P\ (\Xi(0{:=}\ setsucM\ x\ y))) \longrightarrow (\forall\ x.\ P(\Xi(0{:=}\ x)))$
begin

lemma *setind-SP*: **assumes** *SetProperty P* **and** $P\ \emptyset$ **and** *step*: $\forall\ x\ y.\ P\ x \longrightarrow P\ (setsucM\ x\ y)$
shows $P\ x$
⟨*proof*⟩

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*⟩

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 $x \in \{u.\ P\ u\} \longleftrightarrow P\ x$
⟨*proof*⟩

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*⟩

lemma *collect-def-ex*: — A formulation useful in proofs
assumes $\exists\ w.\ \forall\ u\ .\ (u\ \varepsilon\ w \longleftrightarrow Q\ u)$
shows $u\ \varepsilon\ collectM\ Q \longleftrightarrow Q\ u$
⟨*proof*⟩

lemma *proper-subset-diff*[*simp*] : $y \subset_M x \Longrightarrow \exists\ z.\ z\ \varepsilon\ x \wedge \neg\ z\ \varepsilon\ y$
⟨*proof*⟩

lemma *subsetM-antisym*: $y \subseteq_M x \Longrightarrow x \subseteq_M y \Longrightarrow x = y$
⟨*proof*⟩

lemma *proper-subsetM-trans*[*simp*]: **assumes** $x \subset_M y$ $y \subset_M z$ **shows** $x \subset_M z$
⟨*proof*⟩

lemma *emptyset-mem-ord*: **assumes** *ordM* x $\exists\ z.\ z \in x$ **shows** $\emptyset \in x$
⟨*proof*⟩

end

1.3.3 Empty set

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

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*]: $\forall\ y.\ (\neg\ y \in \emptyset)$
⟨*proof*⟩

lemma *emptyset-def'*[*set-defs*]: $x = \emptyset \longleftrightarrow (\forall\ u.\ \neg\ u \in x)$
⟨*proof*⟩

lemma *empty-mem-false* [*set-defs*]: $u \in \emptyset \longleftrightarrow \mathit{False}$
⟨*proof*⟩

lemma *setsuc-empty-sing*: $\mathit{setsucM}\ \emptyset\ x = \{x\}_M$
⟨*proof*⟩

lemma *SFP-empty-n* [*simp*]: *SetFormulaPredicate* $(\lambda\Xi.\ \Xi\ n = \emptyset)$
⟨*proof*⟩

lemma *SP-empty*[*simp*]: *SetProperty* $(\lambda x.\ x = \emptyset)$
⟨*proof*⟩

lemma *SFP-empty-n'* [*simp*]: *SetFormulaPredicate* $(\lambda\Xi.\ \forall u.\ \neg\ u \in \Xi\ n)$
⟨*proof*⟩

lemma *empty-dedekind*[*simp*]: **shows** *dedekind-fin* $\emptyset$
⟨*proof*⟩

lemma *subset-of-empty*[*simp*]: $u \subseteq_M \emptyset \longleftrightarrow u = \emptyset$
⟨*proof*⟩

lemma *empty-tarski*[*simp*]: **shows** *tarski-fin* $\emptyset$
⟨*proof*⟩

lemma *nemp-setI*[*intro*]: $x \ \varepsilon \ y \Longrightarrow y \neq \emptyset$
⟨*proof*⟩

lemma *nemp-setI-ex*: $\exists \ x. \ x \ \varepsilon \ y \Longrightarrow y \neq \emptyset$
⟨*proof*⟩

lemma *empty-is-subset*: $\emptyset \subseteq_M A$
⟨*proof*⟩

lemma *empty-regular*[*simp*]: *regular* $\emptyset$
⟨*proof*⟩

lemma *empty-trans*[*simp*]: *transM* $\emptyset$
⟨*proof*⟩

lemma *emp-natM*[*simp*]: *natM* $\emptyset$
⟨*proof*⟩

lemma *binunion-emp*[*simp*]: $\emptyset \cup_M \ t = t \ t \cup_M \emptyset = t$
⟨*proof*⟩

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*: $\forall \ x \ y. \ \exists \ z. \ \forall \ v. \ v \ \varepsilon \ z \longleftrightarrow v = x \vee v = y$

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

begin

lemma *singleton-def'*[*set-defs*]: $u \ \varepsilon \ \{y\}_M \longleftrightarrow \ u = y$
⟨*proof*⟩

lemma *singleton-eq*[*set-defs*]: $u = \{y\}_M \longleftrightarrow \ (\forall \ v. \ v \ \varepsilon \ u \longleftrightarrow v = y)$
⟨*proof*⟩

lemma *pair-def'*[*set-defs*]: $u \ \varepsilon \ \{x,y\}_M \longleftrightarrow u = x \vee u = y$
⟨*proof*⟩

lemma *pair-eq* [*set-defs*]: $w = \{x,y\}_M \longleftrightarrow (\forall \ u. \ (u \ \varepsilon \ w) \longleftrightarrow \ u = x \vee u = y)$

⟨*proof*⟩

lemma *regular-not-self-mem*: **assumes** *regular x* **shows** ¬ *x ε x*
⟨*proof*⟩

lemma *ordered-pair-unique*[*simp*]: ⟨*u*,*v*⟩ = ⟨*u′*,*v′*⟩ ⟷ *u* = *u′* ∧ *v* = *v′*
⟨*proof*⟩

lemma *sing-eq-iff*: {*a*}$_M$ = {*b*}$_M$ ⟷ *a* = *b*
⟨*proof*⟩

lemma *sing-eq-pair-iff*: {*a*}$_M$ = {*b*,*c*}$_M$ ⟷ *a* = *b* ∧ *b* = *c*
⟨*proof*⟩

lemma *sing-eq-pair-iff′*: {*b*,*c*}$_M$ = {*a*}$_M$ ⟷ *a* = *b* ∧ *b* = *c*
⟨*proof*⟩

lemma *SFP-mem-ordered-pair*[*SFP-rules*]: *SetFormulaPredicate* (λ Ξ. Ξ *k ε* ⟨Ξ *l*, Ξ *m*⟩)
⟨*proof*⟩

lemma *SFP-eq-ordered-pair* [*SFP-rules*]: *SetFormulaPredicate* (λ Ξ. Ξ *k* = ⟨Ξ *l*,Ξ *m*⟩)
⟨*proof*⟩

lemma *SFP-ordered-pair-mem*[*SFP-rules*]: *SetFormulaPredicate* (λΞ. ⟨Ξ *k*,Ξ *l*⟩ *ε* Ξ *m*)
⟨*proof*⟩

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*: ∀ *x y*. ∃ *z*. ∀ *u*. *u ε z* ⟷ *u ε x* ∨ *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 ε setsucM x y* ⟷ (*u ε x* ∨ *u* = *y*)
⟨*proof*⟩

lemma *setsuc-triv*[*simp*]: $y \ \varepsilon \ x \Longrightarrow setsucM \ x \ y = x$
⟨*proof*⟩

lemma *is-suc-def'*: $is\text{-}sucM \ x \longleftrightarrow (\exists \ y. \ x = setsucM \ y \ y)$
⟨*proof*⟩

lemma *trans-suc-trans*: $transM \ \ x \Longrightarrow transM \ (setsucM \ x \ x)$
⟨*proof*⟩

lemma *epschain-suc-epschain*: $epschain \ \ x \Longrightarrow epschain \ (setsucM \ x \ x)$
⟨*proof*⟩

lemma *ord-pred-fun*: **assumes** $ordM \ x \ ordM \ y \ setsucM \ x \ x = setsucM \ y \ y$
shows $x = y$
⟨*proof*⟩

lemma *SFP-setsuc-mem*[*SFP-rules*]: $SetFormulaPredicate \ (\lambda\Xi. \ \Xi \ k \ \varepsilon \ setsucM \ (\Xi \ l) \ (\Xi \ m))$
⟨*proof*⟩

lemma *SFP-setsuc-eq*[*SFP-rules*]: $SetFormulaPredicate \ (\lambda\Xi. \ \Xi \ k = setsucM \ (\Xi \ l) \ (\Xi \ m))$
⟨*proof*⟩

end

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

begin

sublocale *L-pair*
⟨*proof*⟩

lemma *singleton-setsuc*: **shows** $\exists \ y. \ \forall \ u. \ u \ \varepsilon \ y \longleftrightarrow u = z$
⟨*proof*⟩

lemma *pair-setsuc*: **shows** $\exists \ y. \ \forall \ u. \ u \ \varepsilon \ y \longleftrightarrow u = z_1 \vee u = z_2$
⟨*proof*⟩

lemma *triple-setsuc*: **shows** $\exists \ y. \ \forall \ u. \ u \ \varepsilon \ y \longleftrightarrow u = z_1 \vee u = z_2 \vee u = z_3$
⟨*proof*⟩

lemma *regular-antisym-mem*: **assumes** $regular \ x \ z_1 \ \varepsilon \ x \ z_2 \ \varepsilon \ x$ **shows** $\neg \ (z_1 \ \varepsilon \ z_2 \wedge z_2 \ \varepsilon \ z_1)$
⟨*proof*⟩

lemma *regular-antisym2-mem*: **assumes** $regular \ x \ z_1 \ \varepsilon \ x \ z_2 \ \varepsilon \ x \ z_3 \ \varepsilon \ x$ **shows** $\neg$

$(z_1 \ \varepsilon \ z_2 \ \wedge \ z_2 \ \varepsilon \ z_3 \ \wedge \ z_3 \ \varepsilon \ z_1)$
⟨*proof*⟩

lemma *ord-mem-ord*: **assumes** *ordM v u* ε *v* **shows** *ordM u*
⟨*proof*⟩

end

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

begin

sublocale *L-setext-empty*
⟨*proof*⟩

sublocale *L-setext-setsuc*
⟨*proof*⟩

sublocale *L-empty-setsuc*
⟨*proof*⟩

sublocale *L-setext-pair*
⟨*proof*⟩

lemma *empty-mem-ord*: **assumes** *ordM x x* $\neq \emptyset$ **shows** $\emptyset \ \varepsilon \ x$
⟨*proof*⟩

lemma *SP-injective* [*simp*]: *SetProperty* ($\lambda x. \ \forall \, a \ b \ c. \ \langle b,a\rangle \ \varepsilon \ x \ \wedge \ \langle c,a\rangle \ \varepsilon \ x \longrightarrow b = c$)
⟨*proof*⟩

lemma *SP-unique-image* [*simp*]: *SetProperty* ($\lambda x. \ \forall \, a \ b \ c. \ \langle a,b\rangle \ \varepsilon \ x \ \wedge \ \langle a,c\rangle \ \varepsilon \ x \longrightarrow b = c$)
⟨*proof*⟩

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*⟩

lemma *SP-function*[*simp*]: *SetProperty* ($\lambda \ x. \ functionM \ x$)
⟨*proof*⟩

lemma *SP-one-one*[*simp*]: *SetProperty* ($\lambda \ x. \ one\text{-}oneM \ x$)
⟨*proof*⟩

lemma *dom-SR* [*simp*]: *SetRelation* ($\lambda x \ u. \ \exists \, w. \ x = \langle u,w\rangle$)
⟨*proof*⟩

lemma *rng-SR* [*simp*]: *SetRelation* ($\lambda x \ u. \ \exists \, w. \ x = \langle w,u\rangle$)

⟨*proof*⟩

end

locale *L-binunion* = *set-signature* +
  assumes *binunion*: ∀ *x y*. ∃ *z*. ∀ *v*. *v* ε *z* ⟷ (*v* ε *x* ∨ *v* ε *y*)

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

begin

lemma *binunion-def'*[*set-defs*]: *u* ε *x* ∪$_M$ *y* ⟷ *u* ε *x* ∨ *u* ε *y*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

sublocale *L-setext-pair*
  ⟨*proof*⟩

sublocale *L-setsuc*
⟨*proof*⟩

sublocale *L-setext-setsuc*
  ⟨*proof*⟩

lemma *SFP-ordered-pair-setsuc-eq*[*SFP-rules*]: *SetFormulaPredicate* (λΞ. Ξ *k* = ⟨Ξ *l*, *setsucM* (Ξ *m*) (Ξ *n*)⟩)
⟨*proof*⟩

end

1.3.6 Regularity

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

begin

lemma *any-reg*[*simp*]: *regular x*

⟨*proof*⟩

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 $(\varepsilon) \longleftrightarrow (\forall\ x.\ regular\ x)$
⟨*proof*⟩

Schema of regularity

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

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*⟩

lemma *regsch-epsind*:
SetFormulaPredicate $Q \Longrightarrow (\forall\ x.\ (\forall\, y.\ (y\ \varepsilon\ x \longrightarrow Q\ (\Xi\ (0{:=}y)))) \longrightarrow\ Q\ (\Xi\ (0{:=}x))) \longrightarrow\ (\forall\ x.\ Q\ (\Xi\ (0{:=}x)))$
⟨*proof*⟩

lemma *regsch-mem-not-refl*: $\neg\ u\ \varepsilon\ u$
⟨*proof*⟩

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*⟩

sublocale *L-regsch*
⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *sep-var*: **assumes** *SetFormulaPredicate P* **shows** $\forall\, x.\ \exists\ z.\ \forall\ u.\ (u\ \varepsilon\ z \longleftrightarrow u\ \varepsilon\ x \wedge P\ (\Xi(n{:=}\ u)))$
  ⟨*proof*⟩

lemma *sep-SP*: **assumes** *SetProperty P* **shows** $\forall\, x.\ \exists\ z.\ \forall\ u.\ (u\ \varepsilon\ z \longleftrightarrow\ u\ \varepsilon\ x \wedge P\ u)$
  ⟨*proof*⟩

lemma *sep-SR*: **assumes** *SetRelation P* **shows** $\forall\, x.\ \exists\ z.\ \forall\ u.\ (u\ \varepsilon\ z \longleftrightarrow\ u\ \varepsilon\ x \wedge P\ u\ r)$
  ⟨*proof*⟩

lemma *singleton-sep*: **assumes** $z\ \varepsilon\ x$
  shows $\exists\ y.\ \forall\ u.\ u\ \varepsilon\ y \longleftrightarrow u = z$
  ⟨*proof*⟩

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*⟩

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*⟩

lemma *regular-not-self-mem-sep*: **assumes** *regular x* **shows** $\neg\ x\ \varepsilon\ x$
⟨*proof*⟩

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*⟩

lemma *regular-antisym2-mem-sep*: **assumes** *regular* $x\ y_1\ \varepsilon\ x\ y_2\ \varepsilon\ x\ y_3\ \varepsilon\ x$ **shows** $\neg\ (y_1\ \varepsilon\ y_2 \wedge y_2\ \varepsilon\ y_3 \wedge y_3\ \varepsilon\ y_1)$
⟨*proof*⟩

lemma *ord-mem-ord-sep*: **assumes** *ordM v u* ε *v* **shows** *ordM u*

⟨*proof*⟩

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*
⟨*proof*⟩

lemma *separ-def-SFP*: **assumes** *SetFormulaPredicate P*
shows $u\ \varepsilon\ separationM\ y\ (\lambda\ x.\ P(\Xi(n{:=}x))) \longleftrightarrow (u\ \varepsilon\ y \wedge (P\ (\Xi(n{:=}\ u))))$
⟨*proof*⟩

lemma *separ-def-SP*: **assumes** *SetProperty P*
shows $u\ \varepsilon\ separationM\ y\ (\lambda\ x.\ P\ x) \longleftrightarrow (u\ \varepsilon\ y \wedge P\ u)$
⟨*proof*⟩

lemma *separ-def-SR*: **assumes** *SetRelation P*
shows $u\ \varepsilon\ separationM\ y\ (\lambda\ x.\ P\ x\ r) \longleftrightarrow (u\ \varepsilon\ y \wedge P\ u\ r)$
⟨*proof*⟩

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*⟩

lemma *least-def-ex*:
$\exists\ w.\ Q\ (\Xi(n := w)) \Longrightarrow SetFormulaPredicate\ Q \Longrightarrow$
$u\ \varepsilon \bigcap_M\ (\lambda\ x.\ Q\ (\Xi(n{:=}x))) \longleftrightarrow (\forall\ y.\ Q\ (\Xi(n{:=}y)) \longrightarrow u\ \varepsilon\ y)$
⟨*proof*⟩

lemma *intersection-def'*[*set-defs*]: $z\ \varepsilon\ x \cap_M\ y \longleftrightarrow z\ \varepsilon\ x \wedge z\ \varepsilon\ y$
⟨*proof*⟩

lemma *intersect-subset*: $x \cap_M\ y \subseteq_M\ x\ x \cap_M\ y \subseteq_M\ y$
⟨*proof*⟩

lemma *intersect-regular*: **assumes** *regular x regular y*
shows *regular* $(x \cap_M\ y)$
⟨*proof*⟩

lemma *intersect-transM*: **assumes** *transM x transM y*
shows *transM* $(x \cap_M\ y)$
⟨*proof*⟩

lemma *intersect-epschain*: **assumes** *epschain x epschain y*

shows *epschain* ($x \cap_M y$)
⟨*proof*⟩

lemma *intersect-ordM*: **assumes** *ordM x ordM y*
shows *ordM* ($x \cap_M y$)
⟨*proof*⟩

lemma *ordM-subset-mem*: **assumes** *ordM x ordM y* $y \subset_M x$
shows $y \in x$
⟨*proof*⟩

lemma *ordM-total*: **assumes** *ordM x ordM y*
shows $x \in y \vee y \in x \vee x = y$
⟨*proof*⟩

lemma *nat-mem-nat*: **assumes** *natM v u* $\in$ *v* **shows** *natM u*
⟨*proof*⟩

lemma *set-of-ords-regular*: **assumes** $\forall\ y.\ y \in s \longrightarrow$ *ordM y*
shows *regular s*
⟨*proof*⟩

lemma *intersect-natM*: **assumes** *natM x natM y*
shows *natM* ($x \cap_M y$)
⟨*proof*⟩

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, $u \in c_m$ 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*
⟨*proof*⟩

sublocale *L-setext-sep*
⟨*proof*⟩

lemma *regular-setsuc*: **assumes** *transM x regular x regular y*

shows *regular* (*setsucM x y*)
⟨*proof*⟩

lemma *ord-suc-ord*: **assumes** *ordM x* **shows** *ordM* (*setsucM x x*)
⟨*proof*⟩

lemma *nat-suc-nat*: **assumes** *natM x* **shows** *natM* (*setsucM x x*)
⟨*proof*⟩

lemma *one-natM*[*simp*]: *natM* ($\{\emptyset\}_M$)
⟨*proof*⟩

lemma *ord-mem-suc*: **assumes** *ordM x* **and** $y \in x$
shows $setsucM\ y\ y \in x \vee setsucM\ y\ y = x$
⟨*proof*⟩

lemma *ord-limit-or-suc*: **assumes** *ordM x*
shows $is\text{-}sucM\ x \vee (\forall\ u.\ u \in x \longrightarrow setsucM\ u\ u \in x)$
⟨*proof*⟩

lemma assumes *regular x regular y*
shows *regular* (*setsucM x y*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *least-ts-def'* : $u \in least\text{-}tsM\ x \longleftrightarrow (\forall\ v.\ transM\ v \wedge x \subseteq_M v \longrightarrow u \in v)$
(**is** $u \in least\text{-}tsM\ x \longleftrightarrow (\forall\ v.\ ?Q\ v \longrightarrow u \in v)$)
⟨*proof*⟩

lemma *least-ts-is-transitive*: *transM* (*least-tsM x*)
⟨*proof*⟩

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*
⟨*proof*⟩

The schema of regularity follows from regularity and transitive superset.

sublocale *L-regsch*
⟨*proof*⟩

end

1.3.9 Replacement

locale *L-repl* = *set-signature* +
assumes *replf*: *SetFormulaPredicate P* ⟹ (∀ *u*. ∃! *v*. *P* (Ξ(*0*:=*u*, *1*:= *v*))) ⟶ (∀ *x*. ∃ *z*. ∀ *v*. *v* ε *z* ⟷ (∃ *u*. *u* ε *x* ∧ *P* (Ξ(*0*:=*u*, *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*: ∀ *u v w*. *P* (Ξ(*0*:=*u*, *1*:= *v*)) ∧ *P* (Ξ(*0*:=*u*, *1*:= *w*)) ⟶ *v* = *w*
shows ∀ *x*. ∃ *z*. (∀ *v*. *v* ε *z* ⟷ (∃ *u*. *u* ε *x* ∧ *P* (Ξ(*0*:=*u*, *1*:= *v*))))
⟨*proof*⟩

lemma *replp-vars*: **assumes** *SetFormulaPredicate P* **and** *func*: ∀ *u v w*. *P* (Ξ(*k*:=*u*, *l*:= *v*)) ∧ *P* (Ξ(*k*:=*u*, *l*:= *w*)) ⟶ *v* = *w*
shows ∀ *x*. ∃ *z*. (∀ *v*. *v* ε *z* ⟷ (∃ *u*. *u* ε *x* ∧ *P* (Ξ(*k*:=*u*, *l*:= *v*))))
⟨*proof*⟩

lemma *repl-SR*: **assumes** *SetRelation P* ∀ *u v w*. *P u v* ∧ *P u w* ⟶ *v* = *w*
shows ∀ *x*. ∃ *z*. (∀ *v*. *v* ε *z* ⟷ (∃ *u*. *u* ε *x* ∧ *P u v*))
⟨*proof*⟩

lemma *replace-def′*:
assumes *SetFormulaPredicate P* ∀ *u v w*. *P*(Ξ(*k*:=*u*, *l*:= *v*)) ∧ *P*(Ξ(*k*:=*u*, *l*:= *w*)) ⟶ *v* = *w*
shows ∀ *v*. *v* ε *replaceM* (λ *u v*. *P*(Ξ(*k*:=*u*, *l*:= *v*))) *x* ⟷ (∃ *u*. *u* ε *x* ∧ (*P*(Ξ(*k*:=*u*, *l*:= *v*))))

⟨*proof*⟩

Separation can be obtained from replacement.

sublocale *L-setext-sep*
⟨*proof*⟩

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*: ¬ (∀ *mem*. *L-repl mem* ∧ *L-setext mem* ⟶ *L-sep mem*)
⟨*proof*⟩

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

begin

sublocale *L-setext-empty-setsuc*
⟨*proof*⟩

sublocale *L-setext-empty-repl*
⟨*proof*⟩

lemma *tarski-setsuc-tarski*: **assumes** *tarski-fin x* **shows** *tarski-fin* (*setsucM x y*)
⟨*proof*⟩

end

1.3.10 Powerset

locale *L-power* = *set-signature* +
assumes
power: ∀ *x*. ∃ *y*. (∀ *u*. *u* ε *y* ⟷ *u* ⊆$_M$ *x*)

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

begin

sublocale *L-setext-empty*
⟨*proof*⟩

lemma *powerset-def'*[*set-defs*]: *u* ε (𝔓 *x*) ⟷ *u* ⊆$_M$ *x*
⟨*proof*⟩

lemma ∅ ε 𝔓 ∅
⟨*proof*⟩

lemma *set-one-mem* [*simp*]: $u \ \varepsilon \ \mathfrak{P} \ \emptyset \longleftrightarrow u = \emptyset$
⟨*proof*⟩

lemma *set-one-def*: $\mathfrak{P} \ \emptyset = \{\emptyset\}_M$
⟨*proof*⟩

lemma *set-two-mem*: $u \ \varepsilon \ \mathfrak{P} \ (\mathfrak{P} \ \emptyset) \longleftrightarrow u = \emptyset \vee u = \{\emptyset\}_M$
⟨*proof*⟩

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*
⟨*proof*⟩

sublocale *L-setext-empty-power*
⟨*proof*⟩

sublocale *L-setext-empty-setsuc*
⟨*proof*⟩

sublocale *L-setext-empty-setsuc-repl*
⟨*proof*⟩

end

1.3.11 Union

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

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

begin

lemma *union-def'*[*set-defs*]: $u \ \varepsilon \bigcup_M x \longleftrightarrow (\exists \ v. \ v \ \varepsilon \ x \wedge u \ \varepsilon \ v)$
⟨*proof*⟩

lemma *union-trans-trans*: **assumes** $\forall \ v. \ v \ \varepsilon \ x \longrightarrow$ *transM* v
shows *transM* $(\bigcup_M x)$
⟨*proof*⟩

end

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

begin

sublocale *L-setext-binunion*
⟨*proof*⟩

sublocale *L-setext-sep-ts*
⟨*proof*⟩

lemma *trans-union*: *transM u* ⟹ *transM v* ⟹ *transM* (*u* ∪$_M$ *v*)
⟨*proof*⟩

lemma *leastTS-union*: *least-tsM* (*u* ∪$_M$ *v*) = *least-tsM u* ∪$_M$ *least-tsM v*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

sublocale *L-setext-empty-repl*
⟨*proof*⟩

sublocale *L-setext-union*
⟨*proof*⟩

lemma *transitive-closure-ex*:
shows $\forall\ x.\ \exists\ z.\ x \subseteq_M z \wedge transM\ z \wedge (\forall\ u.\ u \in z \longleftrightarrow (\forall\ t.\ x \subseteq_M t \wedge transM\ t \longrightarrow u \in t))$
(**is** $\forall\ x.\ \exists\ z.\ ?TC\ x\ z$ **is** $\forall\ x.\ ?hasTC\ x$)
⟨*proof*⟩

sublocale *L-setext-sep-reg-ts*
⟨*proof*⟩

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*
⟨*proof*⟩

sublocale *L-setext-setsuc-sep*
⟨*proof*⟩

sublocale *L-setext-union*
⟨*proof*⟩

sublocale *L-setext-empty-union-repl-pair*
⟨*proof*⟩

lemma *ordered-pair-mem*: **assumes** $v \in x\ v' \in y$ **shows** $\langle v,v'\rangle \in \mathfrak{P}\ (\mathfrak{P}\ (x \cup_M y))$
⟨*proof*⟩

lemma *ordered-pair-mem-union*: **assumes** $\langle u,v\rangle \in r$ **shows** $u \in \bigcup_M\ (\bigcup_M\ r)\ v \in \bigcup_M\ (\bigcup_M\ r)$
⟨*proof*⟩

lemma *car-prod-ex*: $\exists\ z.\ \forall\ u.\ u \in z \longleftrightarrow (\exists\ v\ v'.\ v \in x \wedge v' \in y \wedge u = \langle v,v'\rangle)$
⟨*proof*⟩

lemma *car-prod-def'*: $u \in x \times_M y \longleftrightarrow (\exists\ v\ v'.\ \ v \in x \wedge v' \in y \wedge\ \ u = \langle v,v'\rangle)$
⟨*proof*⟩

lemma *rel-inv-def'*: $u \in rel\text{-}inverseM\ r \longleftrightarrow (\exists\ a\ b.\ \langle a,b\rangle \in r \wedge u = \langle b,a\rangle)$
⟨*proof*⟩

lemma *dom-def'*: $u \in domM\ r \longleftrightarrow (\exists\, v.\ \langle u,v\rangle \in r)$

⟨*proof*⟩

lemma *rng-def'*: $u \in rngM\ r \longleftrightarrow (\exists\, v.\ \langle v,u\rangle \in r)$
⟨*proof*⟩

lemma *SFP-dom*[*simp*, *intro*]: *SetFormulaPredicate* $(\lambda\ \Xi.\ \Xi\ k = domM\ (\Xi\ l))$
⟨*proof*⟩

lemma *SFP-rng*[*simp*, *intro*]: *SetFormulaPredicate* $(\lambda\ \Xi.\ \Xi\ k = rngM\ (\Xi\ l))$
⟨*proof*⟩

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

lemma *SetRelation* $(\lambda x\ y.\ \exists f.\ one\text{-}oneM\ f \wedge x = domM\ f \wedge y = rngM\ f)$
(**is** *SetRelation* $(\lambda\ x\ y.\ (\exists\ f.\ ?P\ x\ y\ f))$)
⟨*proof*⟩

lemma *SR-equiv*[*simp*]: *SetRelation* $(\lambda\ x\ y.\ x \approx_M y)$
⟨*proof*⟩

lemma *SP-dedekind*[*simp*]: *SetProperty dedekind-fin*
⟨*proof*⟩

lemma *SR-card*: *SetRelation cardinality*
⟨*proof*⟩

lemma *rel-inv-dom-rng*: $domM\ (rel\text{-}inverseM\ r) = rngM\ r\ rngM\ (rel\text{-}inverseM\ r) = domM\ r$
⟨*proof*⟩

lemma *rel-inv-rel*: $relationM\ f \Longrightarrow relationM\ (rel\text{-}inverseM\ f)$
⟨*proof*⟩

lemma *one-one-inv*: $one\text{-}oneM\ f \Longrightarrow one\text{-}oneM\ (rel\text{-}inverseM\ f)$
⟨*proof*⟩

lemma *dedekind-setsuc-dedekind*: **assumes** *dedekind-fin x* **shows** *dedekind-fin* $(setsucM\ x\ t)$
⟨*proof*⟩

lemma *set-equivalent-sym*: $x \approx_M y \longleftrightarrow y \approx_M x$
⟨*proof*⟩

lemma *compose-def'*: $u \in f \circ_M g \longleftrightarrow (\exists\, a\ b\ c.\ \langle a,b\rangle \in g \wedge \langle b,c\rangle \in f \wedge u = \langle a,c\rangle)$
⟨*proof*⟩

lemma *rel-comp*: **assumes** *relationM f relationM g* **shows** $relationM\ (f \circ_M g)$
⟨*proof*⟩

lemma *fun-comp*: **assumes** *functionM f functionM g* **shows** *functionM* $(f \circ_M g)$
⟨*proof*⟩

lemma *one-one-comp*: **assumes** *one-oneM f one-oneM g* **shows** *one-oneM* $(f \circ_M g)$
⟨*proof*⟩

lemma *set-equivalent-trans*: **assumes** $x \approx_M y$ $y \approx_M z$ **shows** $x \approx_M z$
⟨*proof*⟩

lemma *union-of-ords-regular*: **assumes** $\forall\ y.\ y\ \varepsilon\ s \longrightarrow ordM\ y$
shows *regular* $(\bigcup_M s)$
⟨*proof*⟩

lemma *union-of-ords-ord*: **assumes** $\forall\ y.\ y\ \varepsilon\ s \longrightarrow ordM\ y$
shows *ordM* $(\bigcup_M s)$
⟨*proof*⟩

lemma *union-ord*: **assumes** *ordM x* **shows** *ordM* $(\bigcup_M x)$
⟨*proof*⟩

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*⟩

lemma *limit-ord*: **assumes** $\forall\ u.\ u\ \varepsilon\ s \longrightarrow ordM\ u \wedge setsucM\ u\ u\ \varepsilon\ s$
shows $\neg$ *is-sucM* $(\bigcup_M s)$
⟨*proof*⟩

lemma *not-limit-ord*: **assumes** *ordM x is-sucM x*
shows $x =$ *setsucM* $(\bigcup_M x)$ $(\bigcup_M x)$
⟨*proof*⟩

lemma *natM-fin*: **assumes** $s \subseteq_M x\ s \neq \emptyset\ \forall\ u.\ u\ \varepsilon\ s \longrightarrow setsucM\ u\ u\ \varepsilon\ s$
shows $\neg$ *natM x*
⟨*proof*⟩

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*⟩
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 $P\ x$
⟨*proof*⟩

theorem *nat-tarski-fin*: **assumes** *natM x* **shows** *tarski-fin x*
⟨*proof*⟩

lemma *nat-dedekind-finite*: **assumes** *natM x* **shows** *dedekind-fin x*
⟨*proof*⟩

lemma *card-emp*: *cardinality* ∅ ∅
⟨*proof*⟩

lemma *nat-equiv-unique*: **assumes** *natM x natM y x* ≈$_M$ *y*
shows $x = y$
⟨*proof*⟩

lemma *card-fun*: **assumes** *cardinality x n cardinality x m* **shows** $n = m$
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *mem-not-refl*: ¬ *x* ε *x*
⟨*proof*⟩

lemma *mem-antisym*: ¬ (*u* ε *v* ∧ *v* ε *u*)
⟨*proof*⟩

lemma *suc-unique*: **assumes** *setsucM c c* = *setsucM d d*
shows $c = d$
⟨*proof*⟩

lemma *mem-neq-singleton*: $x \neq \{x\}_M$
⟨*proof*⟩

lemma *suc-subset*[*simp*]: *z* ⊂$_M$ *setsucM z z*
⟨*proof*⟩

lemma *card-setsuc*: **assumes** ¬ *y* ε *x cardinality x m* **shows** *cardinality* (*setsucM x y*) (*setsucM m m*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *binunion-ex*:
shows ∃ *z*. (∀ *u*. *u* ε *z* ⟷ *u* ε *x* ∨ *u* ε *x'*) (**is** *?P x x'*)
⟨*proof*⟩

sublocale *L-union*
⟨*proof*⟩

sublocale *L-repl*
⟨*proof*⟩

sublocale *L-setext-empty-repl*
⟨*proof*⟩

lemma *setsuc-separ*: **assumes** *y* ε *u*
shows *u* = *setsucM* (*separationM u* (λ *x*. *x* ≠ *y*)) *y*
⟨*proof*⟩

lemma *setsuc-subset-setind*: *u* ⊆$_M$ *setsucM x y* ⟷ *u* ⊆$_M$ *x* ∨ (∃ *u'*. *u'* ⊆$_M$ *x* ∧ *u* = *setsucM u' y*)
⟨*proof*⟩

sublocale *L-power*
⟨*proof*⟩

sublocale *L-setext-empty-power-union-repl*
⟨*proof*⟩

lemma *min-subset-ex*: **assumes** *u* ≠ ∅ **shows** ∃ *z*. *z* ε *u* ∧ (∄ *w*. *w* ε *u* ∧ *w* ⊂$_M$ *z*)
⟨*proof*⟩

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* ≠ ∅ **shows** ∃ *z*. *z* ε *u* ∧ (∄ *w*. *w* ε *u* ∧ *z* ⊂$_M$

w)
⟨$proof$⟩

lemma *max-mem*: **assumes** $P\ n\ n\ \varepsilon\ x$ *SetProperty* P
shows $\exists\ m.\ m\ \varepsilon\ x \wedge P\ m \wedge (\forall\ k.\ k\ \varepsilon\ x \wedge P\ k \longrightarrow \neg\ m \subset_M k)$
⟨$proof$⟩

sublocale *L-tarski*
⟨$proof$⟩

sublocale *L-dedekind*
⟨$proof$⟩

lemma *fun-images*: **assumes** *SetFormulaPredicate* P **and** $\forall\ u.\ (\exists!\ v.\ P(\Xi(0{:=}u,1{:=}v)))$
shows $\exists\ z.\ (\forall\ v.\ v\ \varepsilon\ z \longleftrightarrow (\exists\ u.\ u\ \varepsilon\ x \wedge P(\Xi(0{:=}u,1{:=}v))))$
⟨$proof$⟩

sublocale *L-fin*
⟨$proof$⟩

theorem *ord-is-suc*: **assumes** *ordM* x **shows** $x = \emptyset \vee$ *is-sucM* x
⟨$proof$⟩

theorem *ord-iff-nat*: *ordM* $x \longleftrightarrow$ *natM* x
⟨$proof$⟩

lemma *ord-iff-empty-or-suc*: *ordM* $x \longleftrightarrow x = \emptyset \vee (\exists\ y.$ *ordM* $y \wedge x =$ *setsucM* y y)
⟨$proof$⟩

lemma *union-of-ords-mem*: **assumes** $\forall\ y.\ y\ \varepsilon\ s \longrightarrow$ *ordM* $y\ s \neq \emptyset$
shows $\bigcup_M\ s\ \varepsilon\ s$
⟨$proof$⟩

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*
⟨$proof$⟩

sublocale *L-setind*
⟨$proof$⟩

lemma *cardinality-ex*: $\exists!\ n.\ natM\ n \wedge x \approx_M n$
⟨*proof*⟩

sublocale *L-setext-empty-setsuc-setind*
⟨*proof*⟩

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*
⟨*proof*⟩

sublocale *L-fin*
⟨*proof*⟩

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*
⟨*proof*⟩

sublocale *L-setext-empty-setsuc*
⟨*proof*⟩

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

sublocale *L-setind*
⟨*proof*⟩

sublocale *L-setext-empty-setsuc-setind*
⟨*proof*⟩

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

sublocale *L-repl*
⟨*proof*⟩

sublocale *L-union*
⟨*proof*⟩

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* ⟹
P(Ξ(*0*:= ∅)) ⟶ (∀ *x y*. *P*(Ξ(*0*:= *x*)) ∧ *P*(Ξ(*0*:= *y*)) ⟶ *P*(Ξ(*0*:= *setsucM x y*))) ⟶ (∀ *x*. *P*(Ξ(*0*:= *x*)))

begin

lemma *setindregsch-sp*: **assumes** *SetProperty P P* ∅ ∀ *x y*. *P x* ∧ *P y* ⟶ *P* (*setsucM x y*)
shows ∀ *x*. *P x*
⟨*proof*⟩

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*⟩

sublocale *L-setind*
⟨*proof*⟩

end

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

begin

sublocale *L-setext-empty-setsuc-setind*
⟨*proof*⟩

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*⟩

lemma *epsind-from-setindregsch*: **assumes** *sfp*: *SetFormulaPredicate P* **and** *indp*: (∀ *x*. (∀ *y*. (*y* ε *x* ⟶ *P*(Ξ(*0*:=*y*)))) ⟶ *P*(Ξ(*0*:=*x*)))

shows $\forall\ x.\ P(\Xi(0{:=}x))$
⟨*proof*⟩

sublocale *L-epsind*
⟨*proof*⟩

sublocale *L-setext-empty-union-repl-pair-regsch*
⟨*proof*⟩

end

1.3.17 Summary of dependencies

theorem *epsind-regsch-iff*:
L-epsind mem $\longleftrightarrow$ *L-regsch mem*
⟨*proof*⟩

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*
⟨*proof*⟩
sublocale *L-regsch*
⟨*proof*⟩
sublocale *L-reg*
⟨*proof*⟩
sublocale *L-setsuc*
⟨*proof*⟩
sublocale *L-sep*
⟨*proof*⟩
sublocale *L-setind*
⟨*proof*⟩
sublocale *L-dedekind*
⟨*proof*⟩
sublocale *L-tarski*
⟨*proof*⟩
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*⟩

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*⟩

end

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

sublocale *ASTset* ⊆ *ZFfin*
⟨*proof*⟩
sublocale *ZFfin* ⊆ *ASTset*
⟨*proof*⟩

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*⟩

theorem *zffin-ast*: *ZFfin mem* ⟷ *ASTset mem*
⟨*proof*⟩

Ambivalence of the separation schema and the replacement schema.

theorem *repl-implies-sep*:
shows *L-setext-empty-repl mem* ⟹ *L-sep mem*
⟨*proof*⟩

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

theorem *sep-implies-repl*:
shows *L-setext-empty-power-sep-setsuc-tarski mem* ⟹ *L-repl mem*
⟨*proof*⟩

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*) ⟹ *L-tarski* (*ε*) **and**
setind-implies-fin-by-setsuc: *L-setind* (*ε*) ⟹ *L-fin* (*ε*) **and**
setind-implies-dedekind-by-setsuc: *L-setind* (*ε*) ⟹ *L-dedekind* (*ε*)
⟨*proof*⟩

theorem (**in** *L-setext-empty-power-union-repl*)
dedekind-implies-fin: *L-dedekind* (*ε*) ⟹ *L-fin* (*ε*)
⟨*proof*⟩

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*⟩

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*⟩

theorem (**in** *L-setext-empty-union-repl-pair*) *epsind-implies-ts*:
L-epsind (ε) $\Longrightarrow$ *L-ts* (ε)
⟨*proof*⟩

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*⟩

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*⟩

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 ⇒ '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*
⟨*proof*⟩

interpretation *pm*: *L-setext perm-mem*
⟨*proof*⟩

lemma *SFP-fmem*[*SFP-rules*]: *SetFormulaPredicate* ($\lambda\Xi$. Ξ *m* ε^f Ξ *n*)
⟨*proof*⟩

lemma *SR-perm-eq*: *SetRelation* (λ *x y. perm x* = *y*)
⟨*proof*⟩

lemma *permSFP-SFP*: **assumes** *pm.SetFormulaPredicate P* **shows** *SetFormulaPredicate P*
⟨*proof*⟩

lemma *perm-model*:
shows *L-setext-empty-power-union-repl perm-mem*
⟨*proof*⟩

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 = (*if x* = $\emptyset$ *then* $\{\emptyset\}_M$ *else* (*if x* = $\{\emptyset\}_M$ *then* $\emptyset$ *else x*))

lemma *swap-zero-one-involution*[*simp*]: *swap-zero-one* (*swap-zero-one x*) = *x*
⟨*proof*⟩

lemma *bij-swap-zero-one*: *bij swap-zero-one*
⟨*proof*⟩

lemma *swap-zero-one-mem*: *a* ε *swap-zero-one b* ⟷ (*b* ≠ $\{\emptyset\}_M$ ∧ *a* ε *b*) ∨ (*a* = $\emptyset$ ∧ *b* = $\emptyset$)
⟨*proof*⟩

lemma *swap-zero-one-perm-model*: *permutation-model* (ε) *swap-zero-one*
⟨*proof*⟩

interpretation *swap*: *permutation-model* (ε) *swap-zero-one*
⟨*proof*⟩

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

interpretation *swap*: *L-setext-empty-power-union-repl* (ε^s)
⟨*proof*⟩

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

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

— $\{\emptyset\}$ is the empty set in the permuted model
lemma *one-swap*: ¬ (*a* ε^s $\{\emptyset\}_M$)
⟨*proof*⟩

lemma *swap-empty*: $\emptyset^s$ = $\{\emptyset\}_M$
⟨*proof*⟩
lemma *zero-swap-mem-iff*: *a* ε^s $\emptyset$ ⟷ *a* = $\emptyset$
⟨*proof*⟩

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

theorem *not-reg-swap-zero-one*: ¬ (*L-reg* (ε^s))
⟨*proof*⟩

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*: ¬ (*x* = $\emptyset$ ∧ *y* = $\emptyset$) ⟹ *x* ε^s *y* ⟹ *x* ε *y*
⟨*proof*⟩

lemma *swap-setsuc-1-y*: **assumes** *y* ≠ $\emptyset$ **shows** $\mathfrak{suc}^s$ ($\{\emptyset\}_M$) *y* = $\{y\}_M$
⟨*proof*⟩

lemma *swap-setsuc-0-y*: **assumes** *y* ≠ $\emptyset$ **shows** $\mathfrak{suc}^s$ $\emptyset$ *y* = $\{\emptyset,y\}_M$

⟨*proof*⟩

lemma *swap-setsuc-0-0*: $\mathfrak{suc}^s\ \emptyset\ \emptyset = \emptyset$
⟨*proof*⟩

lemma *swap-setsuc-1-0*: $\mathfrak{suc}^s\ (\{\emptyset\}_M)\ \emptyset = \emptyset$
⟨*proof*⟩

lemma *swap-setsuc-1-1*: $\mathfrak{suc}^s\ (\{\emptyset\}_M)\ (\{\emptyset\}_M) = \{\{\emptyset\}_M\}_M$
⟨*proof*⟩

lemma *setsuc-setsuc'*: **assumes** $x \neq \emptyset \wedge x \neq \{\emptyset\}_M$
shows $\mathfrak{suc}^s\ x\ y = \mathfrak{suc}\ x\ y$
⟨*proof*⟩

theorem *setind-swap-setind*: *L-setind* $(\varepsilon) \longrightarrow$ *L-setind* (ε^s)
⟨*proof*⟩

lemma *ts-swap-ts*: *L-ts* $(\varepsilon) \longrightarrow$ *L-ts* (ε^s)
⟨*proof*⟩

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*⟩

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* ε^f *1* ε^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* ($\{\emptyset\}_M$)($\{\emptyset\}_M$))
  ⟨*proof*⟩

lemma *suc-sing-not-nat*: $\neg$ *natM* ($\{setsucM\ b\ b\}_M$)
⟨*proof*⟩

lemma *regperm-image-nat*: **assumes** *natM a*
  shows *regperm a* = $\{setsucM\ a\ a\}_M$
  ⟨*proof*⟩

lemma *regperm-image-plus*: **assumes** *natM b*
  shows *regperm* ($\{setsucM\ b\ b\}_M$) = *b*
⟨*proof*⟩

lemma *regperm-image-else*: **assumes** $\neg$ *natM a* $\nexists$ *b. natM b* $\wedge$ *a* = $\{setsucM\ b\ b\}_M$
  shows *regperm a* = *a*
  ⟨*proof*⟩

lemma *regperm-bij*: *bij regperm*
⟨*proof*⟩

lemma *SR-regperm-eq*: *SetRelation* (λ *x y. regperm x* = *y*)
⟨*proof*⟩

lemma *SR-regperm-mem*: *SetRelation* (λ *x y. x* ε *regperm y*)
⟨*proof*⟩

end

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

begin

interpretation *perm*: *permutation-model* (ε) *regperm*
  ⟨*proof*⟩
abbreviation *fmem* (**infix** ε^f *51*) **where**
  fmem x y ≡ *perm.perm-mem x y*

interpretation *fm*: *L-setext-empty-power-union-repl fmem*
⟨*proof*⟩

lemma *regperm-mem-nat*: **assumes** *natM a*
shows $u\ \varepsilon^f\ a \longleftrightarrow u = (setsucM\ a\ a)$
⟨*proof*⟩
lemma *fmem-not-wf*: ¬ (*wfp* (ε^f))
⟨*proof*⟩

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

theorem *regperm-not-ts*: ¬ *L-ts* (ε^f)
⟨*proof*⟩
theorem *regperm-not-regsch*: ¬ *L-regsch* (ε^f)
⟨*proof*⟩

lemma *regperm-reg*:
shows *L-setext-empty-power-union-repl-reg* (ε^f)
⟨*proof*⟩

lemma *perm-empty*: $fm.emptysetM = \{\{\emptyset\}_M\}_M$
⟨*proof*⟩

lemma *perm-one*: $fm.binunionM\ fm.emptysetM\ (fm.singletonM\ fm.emptysetM) = \{\{\{\emptyset\}_M\}_M\}_M$
⟨*proof*⟩

lemma *perm-else-setsuc-else*: **assumes** ¬ $natM\ u \nexists\ b.\ natM\ b \wedge u = \{setsucM\ b\ b\}_M$
shows ¬ $natM\ (setsucM\ u\ u) \nexists\ b.\ natM\ b \wedge setsucM\ u\ u = \{setsucM\ b\ b\}_M$
⟨*proof*⟩

lemma *perm-setsuc*: **assumes** ¬ $natM\ u \nexists\ b.\ natM\ b \wedge u = \{setsucM\ b\ b\}_M$
shows $fm.setsucM\ u\ u = setsucM\ u\ u$
⟨*proof*⟩

lemma *perm-fin*: *L-fin* (ε^f)
⟨*proof*⟩

interpretation *find*: *L-setind* (ε^f)
⟨*proof*⟩

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*⟩

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.