The SZS Ontologies

Johannes Schuster Geoff Sutcliffe

November 12, 2026

Abstract

The SZS success ontology [2, 1] is formalised: each of its forty-five model theoretic values is a condition on the models of an axiom set and a conjecture, and the counter values are obtained from the positive ones by a mirror theorem. Entailment between values is decided by reducing the two model sets to one of 81 finite shapes; the decision procedure is proved sound and complete. The isa hierarchy is the transitive reduction of the entailment order, and the BNF of the success ontology is generated from it. Each edge of the hierarchy is exported as a TPTP problem in TF0 and TH0 [4, 3], with a proof that the problem states the corresponding entailment.

Contents

1 Status values **2**
- 1.1 The success ontology vocabulary . . . 3
- 1.2 Spellings . . . 4
- 1.3 The mirror involution . . . 6
- 1.4 Model theoretic values . . . 8

2 Semantics **8**
- 2.1 The semantic locale . . . 9
- 2.2 The four relationships . . . 11
- 2.3 The populated locale . . . 12
- 2.4 Signature extension . . . 13
- 2.5 A concrete instance . . . 14

3 Shapes **15**
- 3.1 Regions and shapes . . . 15
- 3.2 Shape level abbreviations . . . 17
- 3.3 The status conditions on shapes . . . 17
- 3.4 Realizable shapes and the decision procedure . . . 18
- 3.5 The bridge to the model theory . . . 20
- 3.6 Soundness and completeness of the check . . . 22

3.7 Sanity checks . . . 26
3.8 Invariance under signature extension . . . 27

4 Hierarchy **29**
4.1 The derived order . . . 29
4.2 Transitive reduction . . . 30
4.3 The full hierarchy . . . 31

5 BNF **32**
5.1 The no-success ontology . . . 32
5.2 The dataform ontology . . . 34
5.3 Structural checks . . . 36
5.4 Layout . . . 39
5.5 The grammar . . . 40
5.6 Export . . . 44

6 TPTP problems **45**
6.1 Formulas . . . 45
6.2 The status conditions . . . 46
6.3 Edges as problems . . . 47
6.4 Printing . . . 50

7 Relations **53**
7.1 The relations . . . 53
7.2 Pair lists . . . 53
7.3 The strongest relation of a pair . . . 54
7.4 Text . . . 55

8 Export **56**

A The generated BNF **57**

theory *SZS-Status*
imports *Main*
begin

1 Status values

The values of the SZS success ontology, their OneWord names used in the % *Status* : header field and in % *SZS status* output lines, their mnemonics, the lower case form used in a TPTP *status*(...) annotation, and the mirror involution. The type is *success*; the no-success and dataform ontologies are separate types in SZS_BNF. The ontologies are published at https://szs.tptp.org.

1.1 The success ontology vocabulary

Fifty-three values, as at https://szs.tptp.org/success. The counter values are listed in the order induced by *mirror* on the positive values.

```
datatype success =
  — frame
    SUC | SSU
  — positive semantic values
  | UNP | SAP | TAP | ESA | ETA | SAT | FSA | THM | FTH | STH
  | EQV | TAC | WEC | ETH | TAU | FTT | WTC | WTH | MEX | NOC
  — contradictory axioms
  | CAX | SCA | TCA | WCA
  — counter semantic values
  | CUP | CSP | CTP | ECS | ECA | CSA | FCS | CTH | FCT | SCT
  | CEQ | UNC | WCC | ECT | UNS | FUN | WUC | WCT | CMX | SCC | UCA
  — type checking and verification
  | TSU | TCP | TCC | VSU | VSG | VSB

definition all-success :: success list where
  all-success =
    [SUC, SSU,
     UNP, SAP, TAP, ESA, ETA, SAT, FSA, THM, FTH, STH,
     EQV, TAC, WEC, ETH, TAU, FTT, WTC, WTH, MEX, NOC,
     CAX, SCA, TCA, WCA,
     CUP, CSP, CTP, ECS, ECA, CSA, FCS, CTH, FCT, SCT,
     CEQ, UNC, WCC, ECT, UNS, FUN, WUC, WCT, CMX, SCC, UCA,
     TSU, TCP, TCC, VSU, VSG, VSB]

lemma all-success-UNIV [simp]: set all-success = UNIV
proof −
  have s ∈ set all-success for s
    by (cases s) (simp-all add: all-success-def)
  then show ?thesis by auto
qed

lemma distinct-all-success: distinct all-success
  by eval

lemma length-all-success: length all-success = 53
  by (simp add: all-success-def)

lemma finite-success [simp]: finite (UNIV :: success set)
  using all-success-UNIV by (metis List.finite-set)

lemma card-success: card (UNIV :: success set) = 53
  using all-success-UNIV distinct-all-success length-all-success
  by (metis distinct-card)
```

1.2 Spellings

1.2.1 OneWord names

The OneWord is the form used in the % *Status* : header field of TPTP problems and in % *SZS status* lines.

primrec *one-word* :: *success* ⇒ *string* **where**
  one-word SUC = *''Success''*
| *one-word SSU* = *''SemanticSuccess''*
| *one-word UNP* = *''UnsatisfiabilityPreserving''*
| *one-word SAP* = *''SatisfiabilityPreserving''*
| *one-word TAP* = *''TautologyPreserving''*
| *one-word ESA* = *''EquiSatisfiable''*
| *one-word ETA* = *''EquiTautologous''*
| *one-word SAT* = *''Satisfiable''*
| *one-word FSA* = *''FinitelySatisfiable''*
| *one-word THM* = *''Theorem''*
| *one-word FTH* = *''FiniteTheorem''*
| *one-word STH* = *''SatisfiableAxiomsTheorem''*
| *one-word EQV* = *''Equivalent''*
| *one-word TAC* = *''TautologousConclusion''*
| *one-word WEC* = *''WeakerConclusion''*
| *one-word ETH* = *''EquivalentTheorem''*
| *one-word TAU* = *''Tautology''*
| *one-word FTT* = *''FiniteTautology''*
| *one-word WTC* = *''WeakerTautologousConclusion''*
| *one-word WTH* = *''WeakerTheorem''*
| *one-word MEX* = *''ModelExtending''*
| *one-word NOC* = *''NoConsequence''*
| *one-word CAX* = *''ContradictoryAxioms''*
| *one-word SCA* = *''SatisfiableConclusionContradictoryAxioms''*
| *one-word TCA* = *''TautologousConclusionContradictoryAxioms''*
| *one-word WCA* = *''WeakerConclusionContradictoryAxioms''*
| *one-word CUP* = *''CounterUnsatisfiabilityPreserving''*
| *one-word CSP* = *''CounterSatisfiabilityPreserving''*
| *one-word CTP* = *''CounterTautologyPreserving''*
| *one-word ECS* = *''EquiCounterSatisfiable''*
| *one-word ECA* = *''EquiCounterTautologous''*
| *one-word CSA* = *''CounterSatisfiable''*
| *one-word FCS* = *''FinitelyCounterSatisfiable''*
| *one-word CTH* = *''CounterTheorem''*
| *one-word FCT* = *''FiniteCounterTheorem''*
| *one-word SCT* = *''SatisfiableAxiomsCounterTheorem''*
| *one-word CEQ* = *''CounterEquivalent''*
| *one-word UNC* = *''UnsatisfiableConclusion''*
| *one-word WCC* = *''WeakerCounterConclusion''*
| *one-word ECT* = *''EquivalentCounterTheorem''*
| *one-word UNS* = *''Unsatisfiable''*
| *one-word FUN* = *''FinitelyUnsatisfiable''*

```
| one-word WUC = ''WeakerUnsatisfiableConclusion''
| one-word WCT = ''WeakerCounterTheorem''
| one-word CMX = ''CounterModelExtending''
| one-word SCC = ''SatisfiableCounterConclusionContradictoryAxioms''
| one-word UCA = ''UnsatisfiableConclusionContradictoryAxioms''
| one-word TSU = ''TypeCheckSuccess''
| one-word TCP = ''TypeCheckPartial''
| one-word TCC = ''TypeCheckedComplete''
| one-word VSU = ''VerifySuccess''
| one-word VSG = ''VerifiedGood''
| one-word VSB = ''VerifiedBad''
```

1.2.2 Mnemonics

The mnemonics are the constructor names. The function exists because the dataform ontology, whose mnemonics are mixed case, is accessed through the same interface.

1.2.3 The status(...) form

In a problem header the OneWord is used; in a TPTP inference record a value is written as its lower case mnemonic. Lower casing is injective on the success ontology and not on the union of the three ontologies, where SAT is Satisfiable and Sat is Saturation.

```
definition lower :: char ⇒ char where
  lower c =
    (let n = (of-char c :: nat)
     in if 65 ≤ n ∧ n ≤ 90 then char-of (n + 32) else c)

definition status-value :: success ⇒ string where
  status-value s = map lower (mnemonic s)
```

1.2.4 Injectivity

```
lemma distinct-one-words: distinct (map one-word all-success)
  by eval

lemma distinct-mnemonics: distinct (map mnemonic all-success)
  by eval

lemma distinct-status-values: distinct (map status-value all-success)
  by eval

lemma inj-one-word: inj one-word
proof (rule injI)
  fix s t :: success
  assume one-word s = one-word t
  moreover have inj-on one-word (set all-success)
```

```
    using distinct-one-words by (simp add: distinct-map)
  ultimately show s = t
    by (metis UNIV-I all-success-UNIV inj-onD)
qed

lemma inj-mnemonic: inj mnemonic
proof (rule injI)
  fix s t :: success
  assume mnemonic s = mnemonic t
  moreover have inj-on mnemonic (set all-success)
    using distinct-mnemonics by (simp add: distinct-map)
  ultimately show s = t
    by (metis UNIV-I all-success-UNIV inj-onD)
qed

lemma inj-status-value: inj status-value
proof (rule injI)
  fix s t :: success
  assume status-value s = status-value t
  moreover have inj-on status-value (set all-success)
    using distinct-status-values by (simp add: distinct-map)
  ultimately show s = t
    by (metis UNIV-I all-success-UNIV inj-onD)
qed
```

1.3 The mirror involution

mirror s says about $\langle Ax, \neg C\rangle$ what *s* says about $\langle Ax,\ C\rangle$. Twenty-one pairs are exchanged and eleven values are fixed: the frame, the type checking and verification values, CAX, NOC and WCA. SZS_Semantics proves *holds* (*mirror s*) *A C* = *holds s A* (− *C*).

```
primrec mirror :: success ⇒ success where
  mirror SUC = SUC
| mirror SSU = SSU
| mirror UNP = CUP
| mirror SAP = CSP
| mirror TAP = CTP
| mirror ESA = ECS
| mirror ETA = ECA
| mirror SAT = CSA
| mirror FSA = FCS
| mirror THM = CTH
| mirror FTH = FCT
| mirror STH = SCT
| mirror EQV = CEQ
| mirror TAC = UNC
| mirror WEC = WCC
| mirror ETH = ECT
```

| *mirror TAU = UNS*
| *mirror FTT = FUN*
| *mirror WTC = WUC*
| *mirror WTH = WCT*
| *mirror MEX = CMX*
| *mirror NOC = NOC*
| *mirror CAX = CAX*
| *mirror SCA = SCC*
| *mirror TCA = UCA*
| *mirror WCA = WCA*
| *mirror CUP = UNP*
| *mirror CSP = SAP*
| *mirror CTP = TAP*
| *mirror ECS = ESA*
| *mirror ECA = ETA*
| *mirror CSA = SAT*
| *mirror FCS = FSA*
| *mirror CTH = THM*
| *mirror FCT = FTH*
| *mirror SCT = STH*
| *mirror CEQ = EQV*
| *mirror UNC = TAC*
| *mirror WCC = WEC*
| *mirror ECT = ETH*
| *mirror UNS = TAU*
| *mirror FUN = FTT*
| *mirror WUC = WTC*
| *mirror WCT = WTH*
| *mirror CMX = MEX*
| *mirror SCC = SCA*
| *mirror UCA = TCA*
| *mirror TSU = TSU*
| *mirror TCP = TCP*
| *mirror TCC = TCC*
| *mirror VSU = VSU*
| *mirror VSG = VSG*
| *mirror VSB = VSB*

lemma *mirror-involution* [*simp*]: *mirror* (*mirror s*) = *s*
by (*cases s*) *simp-all*

lemma *bij-mirror*: *bij mirror*
by (*rule o-bij* [**where** *g* = *mirror*]) (*simp-all add*: *fun-eq-iff comp-def*)

lemma *surj-mirror*: *surj mirror*
using *bij-mirror* **by** (*rule bij-is-surj*)

lemma *inj-mirror*: *inj mirror*
using *bij-mirror* **by** (*rule bij-is-inj*)

definition *mirror-fixed-points* :: *success set* **where**
mirror-fixed-points = {*SUC*, *SSU*, *NOC*, *CAX*, *WCA*, *TSU*, *TCP*, *TCC*, *VSU*, *VSG*, *VSB*}

lemma *mirror-fixed-iff*: *mirror s* = *s* ⟷ *s* ∈ *mirror-fixed-points*
by (*cases s*) (*simp-all add*: *mirror-fixed-points-def*)

lemma *card-mirror-fixed-points*: *card mirror-fixed-points* = *11*
by (*simp add*: *mirror-fixed-points-def*)

1.4 Model theoretic values

Success, SemanticSuccess and the six type checking and verification values carry no condition on models. Under *holds* they are identically true, so SZS_Hierarchy excludes them from the derived order. Forty-five values remain.

definition *model-theoretic* :: *success* ⇒ *bool* **where**
model-theoretic s ⟷ *s* ∉ {*SUC*, *SSU*, *TSU*, *TCP*, *TCC*, *VSU*, *VSG*, *VSB*}

definition *model-theoretic-values* :: *success list* **where**
model-theoretic-values = *filter model-theoretic all-success*

lemma *length-model-theoretic-values*: *length model-theoretic-values* = *45*
by *eval*

lemma *model-theoretic-mirror* [*simp*]: *model-theoretic* (*mirror s*) = *model-theoretic s*
by (*cases s*) (*simp-all add*: *model-theoretic-def*)

end

theory *SZS-Semantics*
imports *SZS-Status*
begin

2 Semantics

Every semantic value is a condition on the set A of models of Ax, the set C of models of C and the set *Fin* of finite interpretations. The models of $\neg\ C$ are $-\ C$, and *holds-mirror* derives the counter values from the positive ones.

2.1 The semantic locale

$'i$ is an uninterpreted type of interpretations and *Fin* a parameter. *szs-base* makes no assumption on *Fin*; *holds* and the mirror theorem hold for every choice.

locale *szs-base* =
  fixes *Fin* :: $'i$ *set*
begin

2.1.1 The status conditions

primrec *holds* :: *success* ⇒ $'i$ *set* ⇒ $'i$ *set* ⇒ *bool* **where**
  holds SUC A C = *True*
| *holds SSU A C* = *True*
| *holds UNP A C* = (*A* = {} ⟶ *C* = {})
| *holds SAP A C* = (*A* ≠ {} ⟶ *C* ≠ {})
| *holds TAP A C* = (*A* = *UNIV* ⟶ *C* = *UNIV*)
| *holds ESA A C* = ((*A* = {}) = (*C* = {}))
| *holds ETA A C* = ((*A* = *UNIV*) = (*C* = *UNIV*))
| *holds SAT A C* = (*A* ≠ {} ∧ *A* ∩ *C* ≠ {})
| *holds FSA A C* = (*A* ∩ *Fin* ≠ {} ∧ *A* ∩ *C* ∩ *Fin* ≠ {})
| *holds THM A C* = (*A* ⊆ *C*)
| *holds FTH A C* = (*A* ∩ *Fin* ⊆ *C*)
| *holds STH A C* = (*A* ≠ {} ∧ *A* ⊆ *C*)
| *holds EQV A C* = (*A* ≠ {} ∧ *A* = *C*)
| *holds TAC A C* = (*A* ≠ {} ∧ *C* = *UNIV*)
| *holds WEC A C* = (*A* ≠ {} ∧ *A* ⊆ *C* ∧ *C* ∩ − *A* ≠ {})
| *holds ETH A C* = (*A* ≠ {} ∧ *A* ≠ *UNIV* ∧ *A* = *C*)
| *holds TAU A C* = (*A* = *UNIV* ∧ *C* = *UNIV*)
| *holds FTT A C* = (*Fin* ⊆ *A* ∧ *Fin* ⊆ *C*)
| *holds WTC A C* = (*A* ≠ {} ∧ *A* ≠ *UNIV* ∧ *C* = *UNIV*)
| *holds WTH A C* = (*A* ≠ {} ∧ *A* ⊆ *C* ∧ *C* ∩ − *A* ≠ {} ∧ *C* ≠ *UNIV*)
| *holds MEX A C* = (*A* ≠ {} ∧ *C* ≠ {} ∧ *C* ⊆ *A*)
| *holds NOC A C* = (*A* ≠ {} ∧ *A* ∩ *C* ≠ {} ∧ *A* ∩ − *C* ≠ {})
| *holds CAX A C* = (*A* = {})
| *holds SCA A C* = (*A* = {} ∧ *C* ≠ {})
| *holds TCA A C* = (*A* = {} ∧ *C* = *UNIV*)
| *holds WCA A C* = (*A* = {} ∧ *C* ≠ {} ∧ − *C* ≠ {})
| *holds CUP A C* = (*A* = {} ⟶ − *C* = {})
| *holds CSP A C* = (*A* ≠ {} ⟶ − *C* ≠ {})
| *holds CTP A C* = (*A* = *UNIV* ⟶ − *C* = *UNIV*)
| *holds ECS A C* = ((*A* = {}) = (− *C* = {}))
| *holds ECA A C* = ((*A* = *UNIV*) = (− *C* = *UNIV*))
| *holds CSA A C* = (*A* ≠ {} ∧ *A* ∩ − *C* ≠ {})
| *holds FCS A C* = (*A* ∩ *Fin* ≠ {} ∧ *A* ∩ − *C* ∩ *Fin* ≠ {})
| *holds CTH A C* = (*A* ⊆ − *C*)
| *holds FCT A C* = (*A* ∩ *Fin* ⊆ − *C*)
| *holds SCT A C* = (*A* ≠ {} ∧ *A* ⊆ − *C*)
| *holds CEQ A C* = (*A* ≠ {} ∧ *A* = − *C*)

| *holds UNC A C* = $(A \neq \{\} \wedge - C = UNIV)$
| *holds WCC A C* = $(A \neq \{\} \wedge A \subseteq - C \wedge - C \cap - A \neq \{\})$
| *holds ECT A C* = $(A \neq \{\} \wedge A \neq UNIV \wedge A = - C)$
| *holds UNS A C* = $(A = UNIV \wedge - C = UNIV)$
| *holds FUN A C* = $(Fin \subseteq A \wedge Fin \subseteq - C)$
| *holds WUC A C* = $(A \neq \{\} \wedge A \neq UNIV \wedge - C = UNIV)$
| *holds WCT A C* = $(A \neq \{\} \wedge A \subseteq - C \wedge - C \cap - A \neq \{\} \wedge - C \neq UNIV)$
| *holds CMX A C* = $(A \neq \{\} \wedge - C \neq \{\} \wedge - C \subseteq A)$
| *holds SCC A C* = $(A = \{\} \wedge - C \neq \{\})$
| *holds UCA A C* = $(A = \{\} \wedge - C = UNIV)$
| *holds TSU A C* = *True*
| *holds TCP A C* = *True*
| *holds TCC A C* = *True*
| *holds VSU A C* = *True*
| *holds VSG A C* = *True*
| *holds VSB A C* = *True*

2.1.2 The mirror theorem

theorem *holds-mirror*: *holds* (*mirror s*) *A C* = *holds s A* (− *C*)
by (*cases s*) *auto*

2.1.3 Redundant conjuncts

SAT, FSA and NOC, and their mirrors, contain a conjunct implied by the rest. The definitions follow the wording of the SZS page; the short forms are proved here.

lemma *holds-SAT-iff*: *holds SAT A C* = $(A \cap C \neq \{\})$
by *auto*

lemma *holds-CSA-iff*: *holds CSA A C* = $(A \cap - C \neq \{\})$
by *auto*

lemma *holds-FSA-iff*: *holds FSA A C* = $(A \cap C \cap Fin \neq \{\})$
by *auto*

lemma *holds-FCS-iff*: *holds FCS A C* = $(A \cap - C \cap Fin \neq \{\})$
by *auto*

lemma *holds-NOC-iff*: *holds NOC A C* = (*holds SAT A C* ∧ *holds CSA A C*)
by *auto*

A finite interpretation that is a model of C is a finite model of C, so the qualification on the right of FiniteTheorem is vacuous.

lemma *holds-FTH-iff*: *holds FTH A C* = $(A \cap Fin \subseteq C \cap Fin)$
by *auto*

lemma *holds-FCT-iff*: *holds FCT A C* = $(A \cap Fin \subseteq - C \cap Fin)$
by *auto*

2.1.4 Values with no model theoretic content

lemma *holds-trivial*: $\neg$ *model-theoretic s* $\Longrightarrow$ *holds s A C*
by (*cases s*) (*simp-all add*: *model-theoretic-def*)

lemma *holds-SUC*: *holds SUC A C*
by *simp*

lemma *holds-SSU*: *holds SSU A C*
by *simp*

2.2 The four relationships

isa is entailment, *nota* its negation, *nevera* incompatibility, *xora* exhaustive incompatibility and *mighta* joint satisfiability.

definition *isa* :: *success* $\Rightarrow$ *success* $\Rightarrow$ *bool* **where**
isa s t $\equiv$ ($\forall$ *A C*. *holds s A C* $\longrightarrow$ *holds t A C*)

definition *nota* :: *success* $\Rightarrow$ *success* $\Rightarrow$ *bool* **where**
nota s t $\equiv$ ($\exists$ *A C*. *holds s A C* $\wedge$ $\neg$ *holds t A C*)

definition *nevera* :: *success* $\Rightarrow$ *success* $\Rightarrow$ *bool* **where**
nevera s t $\equiv$ ($\forall$ *A C*. *holds s A C* $\longrightarrow$ $\neg$ *holds t A C*)

definition *xora* :: *success* $\Rightarrow$ *success* $\Rightarrow$ *bool* **where**
xora s t $\equiv$ ($\forall$ *A C*. *holds s A C* $\neq$ *holds t A C*)

definition *mighta* :: *success* $\Rightarrow$ *success* $\Rightarrow$ *bool* **where**
mighta s t $\equiv$ ($\exists$ *A C*. *holds s A C* $\wedge$ *holds t A C*)

lemma *nota-iff*: *nota s t* $\longleftrightarrow$ $\neg$ *isa s t*
unfolding *nota-def isa-def* **by** *blast*

lemma *mighta-iff*: *mighta s t* $\longleftrightarrow$ $\neg$ *nevera s t*
unfolding *mighta-def nevera-def* **by** *blast*

lemma *xora-imp-nevera*: *xora s t* $\Longrightarrow$ *nevera s t*
unfolding *xora-def nevera-def* **by** *blast*

lemma *nevera-sym*: *nevera s t* $\Longrightarrow$ *nevera t s*
unfolding *nevera-def* **by** *blast*

lemma *xora-sym*: *xora s t* $\Longrightarrow$ *xora t s*
unfolding *xora-def* **by** *blast*

lemma *mighta-sym*: *mighta s t* $\Longrightarrow$ *mighta t s*
unfolding *mighta-def* **by** *blast*

lemma *isa-refl* [*simp*]: *isa s s*

unfolding *isa-def* **by** *blast*

lemma *isa-trans*: *isa s t* $\Longrightarrow$ *isa t u* $\Longrightarrow$ *isa s u*
unfolding *isa-def* **by** *blast*

lemma *isa-antisym-holds*: *isa s t* $\Longrightarrow$ *isa t s* $\Longrightarrow$ *holds s A C* = *holds t A C*
unfolding *isa-def* **by** *blast*

isa is a preorder. The eight values outside *model-theoretic* are equivalent to each other.

lemma *isa-trivial*:
assumes ¬ *model-theoretic s* **and** ¬ *model-theoretic t*
shows *isa s t*
unfolding *isa-def* **using** *holds-trivial* [*OF assms*(*2*)] **by** *blast*

lemma *isa-into-trivial*: ¬ *model-theoretic t* $\Longrightarrow$ *isa s t*
unfolding *isa-def* **using** *holds-trivial* **by** *blast*

corollary *isa-SUC* [*simp*]: *isa s SUC*
unfolding *isa-def* **by** *simp*

2.2.1 Mirror compatibility

theorem *isa-mirror*: *isa s t* $\Longrightarrow$ *isa* (*mirror s*) (*mirror t*)
unfolding *isa-def* **by** (*metis holds-mirror*)

corollary *isa-mirror-iff*: *isa* (*mirror s*) (*mirror t*) $\longleftrightarrow$ *isa s t*
by (*metis isa-mirror mirror-involution*)

corollary *nota-mirror-iff*: *nota* (*mirror s*) (*mirror t*) $\longleftrightarrow$ *nota s t*
by (*simp add*: *nota-iff isa-mirror-iff*)

theorem *nevera-mirror-iff*: *nevera* (*mirror s*) (*mirror t*) $\longleftrightarrow$ *nevera s t*
unfolding *nevera-def* **by** (*metis holds-mirror mirror-involution*)

theorem *xora-mirror-iff*: *xora* (*mirror s*) (*mirror t*) $\longleftrightarrow$ *xora s t*
unfolding *xora-def* **by** (*metis holds-mirror mirror-involution*)

theorem *mighta-mirror-iff*: *mighta* (*mirror s*) (*mirror t*) $\longleftrightarrow$ *mighta s t*
by (*simp add*: *mighta-iff nevera-mirror-iff*)

end

2.3 The populated locale

The refutations need infinitely many finite and infinitely many infinite interpretations. Under these assumptions every shape of SZS_Shapes has a witness.

locale *szs* = *szs-base Fin* **for** *Fin* :: *'i set* +
  assumes *Fin-infinite*: *infinite Fin*
    and *coFin-infinite*: *infinite* (− *Fin*)
begin

lemma *Fin-nonempty* [*simp*]: *Fin* ≠ {}
  using *Fin-infinite* **by** *auto*

lemma *coFin-nonempty* [*simp*]: − *Fin* ≠ {}
  using *coFin-infinite* **by** *auto*

lemma *Fin-neq-UNIV* [*simp*]: *Fin* ≠ *UNIV*
  using *coFin-nonempty* **by** *blast*

lemma *UNIV-nonempty* [*simp*]: (*UNIV* :: *'i set*) ≠ {}
  using *Fin-nonempty* **by** *auto*

end

2.4 Signature extension

C may be stated over an extension of the signature of Ax. ModelExtending is meant for Skolemisation: C is over a conservative extension of the signature of Ax, and every model of C is a conservative extension of a model of Ax. *r* maps an interpretation of the extended signature to its reduct. *r* is assumed surjective, and *Fin-j* is the preimage of *Fin*, so an expansion has the domain of its reduct. The second assumption fails for Henkin frames in which a new type contributes to the structure.

locale *szs-ext* = *szs Fin* **for** *Fin* :: *'i set* +
  fixes *r* :: *'j* ⇒ *'i*
  assumes *surj-r*: *surj r*
begin

definition *Fin-j* :: *'j set* **where**
  Fin-j = *r* −‘ *Fin*

definition *mex-sig* :: *'i set* ⇒ *'j set* ⇒ *bool* **where**
  mex-sig A C ⟷ *A* ≠ {} ∧ *C* ≠ {} ∧ *r* ‘ *C* ⊆ *A*

definition *cmx-sig* :: *'i set* ⇒ *'j set* ⇒ *bool* **where**
  cmx-sig A C ⟷ *mex-sig A* (− *C*)

2.4.1 Preimages

lemma *vimage-empty-iff* [*simp*]: *r* −‘ *S* = {} ⟷ *S* = {}
  using *surj-r* **by** (*simp add*: *surj-vimage-empty*)

lemma *image-vimage-r* [*simp*]: *r* ‘ (*r* −‘ *S*) = *S*

using *surj-r* **by** (*auto simp*: *surj-def*)

lemma *image-subset-vimage*: r ‘ $C \subseteq A \longleftrightarrow C \subseteq r -‘ A$
by *auto*

lemma *vimage-infinite*:
assumes *infinite* S **shows** *infinite* $(r -‘ S)$
by (*metis assms finite-imageI image-vimage-r*)

lemma *szs-Fin-j*: *szs Fin-j*
proof *unfold-locales*
show *infinite Fin-j*
unfolding *Fin-j-def* **by** (*rule vimage-infinite* [*OF Fin-infinite*])
have $-$ *Fin-j* $= r -‘ (-$ *Fin*$)$ **by** (*auto simp*: *Fin-j-def*)
then show *infinite* $(-$ *Fin-j*$)$
using *vimage-infinite* [*OF coFin-infinite*] **by** *simp*
qed

2.4.2 Factorisation

lemma *mex-sig-iff*: *mex-sig A C* $\longleftrightarrow$ *szs-base.holds Fin-j MEX* $(r -‘ A)$ *C*
by (*simp add*: *mex-sig-def szs-base.holds.simps image-subset-vimage*)

lemma *cmx-sig-iff*: *cmx-sig A C* $\longleftrightarrow$ *szs-base.holds Fin-j CMX* $(r -‘ A)$ *C*
by (*simp add*: *cmx-sig-def mex-sig-def szs-base.holds.simps image-subset-vimage*)

end

2.5 A concrete instance

The naturals, with the even numbers as the finite interpretations. The instance shows the locale inhabited and gives *nitpick* and *quickcheck* something to run on.

definition *Evens* :: *nat set* **where**
Evens $= \{n.$ *even* $n\}$

lemma *infinite-Evens*: *infinite Evens*
proof $-$
have *range* $(\lambda n.\ 2 * n) \subseteq$ *Evens* **by** (*auto simp*: *Evens-def*)
moreover have *infinite* (*range* $(\lambda n ::$ *nat*. $2 * n)$)
by (*simp add*: *finite-image-iff inj-on-def*)
ultimately show *?thesis* **using** *finite-subset* **by** *blast*
qed

lemma *infinite-coEvens*: *infinite* $(-$ *Evens*$)$
proof $-$
have *range* $(\lambda n.\ 2 * n + 1) \subseteq -$ *Evens* **by** (*auto simp*: *Evens-def*)
moreover have *infinite* (*range* $(\lambda n ::$ *nat*. $2 * n + 1)$)
by (*simp add*: *finite-image-iff inj-on-def*)

ultimately show *?thesis* **using** *finite-subset* **by** *blast*
qed

interpretation *szs-nat*: *szs Evens*
by *unfold-locales* (*rule infinite-Evens*, *rule infinite-coEvens*)

end

theory *SZS-Shapes*
imports *SZS-Semantics*
begin

3 Shapes

Every condition in SZS_Semantics depends on A and C only through the regions $A \cap C$, $A - C$, $C - A$ and $-(A \cup C)$, and on *Fin* only through whether a region contains a finite interpretation. A shape records for each region whether it is empty, nonempty without finite members, or has a finite member. There are 81 shapes, 65 of them realizable, and entailment between values is decided over them. The check *isa-chk* is executable and independent of the interpretation type; the locale supplies the bridge to the sets. Figure 1 shows the four regions and one shape.

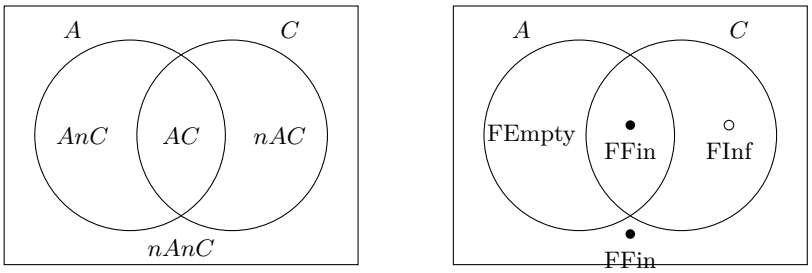


Figure 1: Left: the four regions of a pair A, C of model sets. Right: the shape `mk_shape FFin FEmpty FInf FFin`; a filled dot marks a region with a finite interpretation, an open dot a nonempty region without one. On it THM, FTH, STH, WEC, WTH, SAT and FSA hold; EQV does not, since nAC is nonempty, and FTT does not, since $nAnC$ has a finite member.

3.1 Regions and shapes

datatype *region* = *AC* | *AnC* | *nAC* | *nAnC*

definition *all-regions* :: *region list* **where**
all-regions = [*AC*, *AnC*, *nAC*, *nAnC*]

lemma *all-regions-UNIV* [*simp*]: *set all-regions* = *UNIV*
proof −
  have *r* ∈ *set all-regions* **for** *r*
    by (*cases r*) (*simp-all add*: *all-regions-def*)
  then show *?thesis* **by** *auto*
qed

A region is *FEmpty*, *FInf* (nonempty, no finite interpretation) or *FFin* (contains a finite interpretation).

datatype *fill* = *FEmpty* | *FInf* | *FFin*

definition *all-fills* :: *fill list* **where**
  all-fills = [*FEmpty*, *FInf*, *FFin*]

lemma *all-fills-UNIV* [*simp*]: *set all-fills* = *UNIV*
proof −
  have *a* ∈ *set all-fills* **for** *a*
    by (*cases a*) (*simp-all add*: *all-fills-def*)
  then show *?thesis* **by** *auto*
qed

type-synonym *shape* = *region* ⇒ *fill*

definition *mk-shape* :: *fill* ⇒ *fill* ⇒ *fill* ⇒ *fill* ⇒ *shape* **where**
  mk-shape a b c d =
    (λ*r*. *case r of AC* ⇒ *a* | *AnC* ⇒ *b* | *nAC* ⇒ *c* | *nAnC* ⇒ *d*)

lemma *mk-shape-simps* [*simp*]:
  mk-shape a b c d AC = *a*
  mk-shape a b c d AnC = *b*
  mk-shape a b c d nAC = *c*
  mk-shape a b c d nAnC = *d*
  by (*simp-all add*: *mk-shape-def*)

lemma *shape-expand*: σ = *mk-shape* (σ *AC*) (σ *AnC*) (σ *nAC*) (σ *nAnC*)
proof (*rule ext*)
  fix *r* **show** σ *r* = *mk-shape* (σ *AC*) (σ *AnC*) (σ *nAC*) (σ *nAnC*) *r*
    by (*cases r*) *simp-all*
qed

definition *quads* :: (*fill* × *fill* × *fill* × *fill*) *list* **where**
  quads = [(*a*, *b*, *c*, *d*). *a* ← *all-fills*, *b* ← *all-fills*, *c* ← *all-fills*, *d* ← *all-fills*]

lemma *quads-complete*: (*a*, *b*, *c*, *d*) ∈ *set quads*
  by (*cases a*; *cases b*; *cases c*; *cases d*) (*simp-all add*: *quads-def all-fills-def*)

lemma *length-quads*: *length quads* = *81*
  by (*simp add*: *quads-def all-fills-def*)

3.2 Shape level abbreviations

The conditions that recur in *holds*, on shapes. Each is related to the sets by a lemma in the locale below.

definition *sA-empty* :: *shape* ⇒ *bool* **where**
sA-empty σ ⟷ σ *AC* = *FEmpty* ∧ σ *AnC* = *FEmpty*

definition *sC-empty* :: *shape* ⇒ *bool* **where**
sC-empty σ ⟷ σ *AC* = *FEmpty* ∧ σ *nAC* = *FEmpty*

definition *sA-univ* :: *shape* ⇒ *bool* **where**
sA-univ σ ⟷ σ *nAC* = *FEmpty* ∧ σ *nAnC* = *FEmpty*

definition *sC-univ* :: *shape* ⇒ *bool* **where**
sC-univ σ ⟷ σ *AnC* = *FEmpty* ∧ σ *nAnC* = *FEmpty*

definition *sA-fin* :: *shape* ⇒ *bool* **where**
sA-fin σ ⟷ σ *AC* = *FFin* ∨ σ *AnC* = *FFin*

definition *sFin-sub-A* :: *shape* ⇒ *bool* **where**
sFin-sub-A σ ⟷ σ *nAC* ≠ *FFin* ∧ σ *nAnC* ≠ *FFin*

definition *sFin-sub-C* :: *shape* ⇒ *bool* **where**
sFin-sub-C σ ⟷ σ *AnC* ≠ *FFin* ∧ σ *nAnC* ≠ *FFin*

definition *sFin-sub-nC* :: *shape* ⇒ *bool* **where**
sFin-sub-nC σ ⟷ σ *AC* ≠ *FFin* ∧ σ *nAC* ≠ *FFin*

3.3 The status conditions on shapes

One equation per value, in the order of *holds*. − *C* = {} is *sC-univ*, − *C* = *UNIV* is *sC-empty*, − *C* ⊆ *A* is σ *nAnC* = *FEmpty*, and − *C* − *A* is the region *nAnC*.

primrec *holds-shape* :: *success* ⇒ *shape* ⇒ *bool* **where**
holds-shape SUC σ = *True*
| *holds-shape SSU* σ = *True*
| *holds-shape UNP* σ = (*sA-empty* σ ⟶ *sC-empty* σ)
| *holds-shape SAP* σ = (¬ *sA-empty* σ ⟶ ¬ *sC-empty* σ)
| *holds-shape TAP* σ = (*sA-univ* σ ⟶ *sC-univ* σ)
| *holds-shape ESA* σ = (*sA-empty* σ = *sC-empty* σ)
| *holds-shape ETA* σ = (*sA-univ* σ = *sC-univ* σ)
| *holds-shape SAT* σ = (¬ *sA-empty* σ ∧ σ *AC* ≠ *FEmpty*)
| *holds-shape FSA* σ = (*sA-fin* σ ∧ σ *AC* = *FFin*)
| *holds-shape THM* σ = (σ *AnC* = *FEmpty*)
| *holds-shape FTH* σ = (σ *AnC* ≠ *FFin*)
| *holds-shape STH* σ = (¬ *sA-empty* σ ∧ σ *AnC* = *FEmpty*)
| *holds-shape EQV* σ = (¬ *sA-empty* σ ∧ σ *AnC* = *FEmpty* ∧ σ *nAC* = *FEmpty*)
| *holds-shape TAC* σ = (¬ *sA-empty* σ ∧ *sC-univ* σ)

$|$ *holds-shape WEC* $\sigma = (\neg$ *sA-empty* $\sigma \wedge \sigma$ *AnC* $=$ *FEmpty* $\wedge \sigma$ *nAC* $\neq$ *FEmpty*)
$|$ *holds-shape ETH* $\sigma = (\neg$ *sA-empty* $\sigma \wedge \neg$ *sA-univ* $\sigma \wedge \sigma$ *AnC* $=$ *FEmpty* $\wedge \sigma$ *nAC* $=$ *FEmpty*)
$|$ *holds-shape TAU* $\sigma = ($*sA-univ* $\sigma \wedge$ *sC-univ* $\sigma)$
$|$ *holds-shape FTT* $\sigma = ($*sFin-sub-A* $\sigma \wedge$ *sFin-sub-C* $\sigma)$
$|$ *holds-shape WTC* $\sigma = (\neg$ *sA-empty* $\sigma \wedge \neg$ *sA-univ* $\sigma \wedge$ *sC-univ* $\sigma)$
$|$ *holds-shape WTH* $\sigma =$
$(\neg$ *sA-empty* $\sigma \wedge \sigma$ *AnC* $=$ *FEmpty* $\wedge \sigma$ *nAC* $\neq$ *FEmpty* $\wedge \neg$ *sC-univ* $\sigma)$
$|$ *holds-shape MEX* $\sigma = (\neg$ *sA-empty* $\sigma \wedge \neg$ *sC-empty* $\sigma \wedge \sigma$ *nAC* $=$ *FEmpty*)
$|$ *holds-shape NOC* $\sigma = (\sigma$ *AC* $\neq$ *FEmpty* $\wedge \sigma$ *AnC* $\neq$ *FEmpty*)
$|$ *holds-shape CAX* $\sigma =$ *sA-empty* σ
$|$ *holds-shape SCA* $\sigma = ($*sA-empty* $\sigma \wedge \neg$ *sC-empty* $\sigma)$
$|$ *holds-shape TCA* $\sigma = ($*sA-empty* $\sigma \wedge$ *sC-univ* $\sigma)$
$|$ *holds-shape WCA* $\sigma = ($*sA-empty* $\sigma \wedge \neg$ *sC-empty* $\sigma \wedge \neg$ *sC-univ* $\sigma)$
$|$ *holds-shape CUP* $\sigma = ($*sA-empty* $\sigma \longrightarrow$ *sC-univ* $\sigma)$
$|$ *holds-shape CSP* $\sigma = (\neg$ *sA-empty* $\sigma \longrightarrow \neg$ *sC-univ* $\sigma)$
$|$ *holds-shape CTP* $\sigma = ($*sA-univ* $\sigma \longrightarrow$ *sC-empty* $\sigma)$
$|$ *holds-shape ECS* $\sigma = ($*sA-empty* $\sigma =$ *sC-univ* $\sigma)$
$|$ *holds-shape ECA* $\sigma = ($*sA-univ* $\sigma =$ *sC-empty* $\sigma)$
$|$ *holds-shape CSA* $\sigma = (\neg$ *sA-empty* $\sigma \wedge \sigma$ *AnC* $\neq$ *FEmpty*)
$|$ *holds-shape FCS* $\sigma = ($*sA-fin* $\sigma \wedge \sigma$ *AnC* $=$ *FFin*)
$|$ *holds-shape CTH* $\sigma = (\sigma$ *AC* $=$ *FEmpty*)
$|$ *holds-shape FCT* $\sigma = (\sigma$ *AC* $\neq$ *FFin*)
$|$ *holds-shape SCT* $\sigma = (\neg$ *sA-empty* $\sigma \wedge \sigma$ *AC* $=$ *FEmpty*)
$|$ *holds-shape CEQ* $\sigma = (\neg$ *sA-empty* $\sigma \wedge \sigma$ *AC* $=$ *FEmpty* $\wedge \sigma$ *nAnC* $=$ *FEmpty*)
$|$ *holds-shape UNC* $\sigma = (\neg$ *sA-empty* $\sigma \wedge$ *sC-empty* $\sigma)$
$|$ *holds-shape WCC* $\sigma = (\neg$ *sA-empty* $\sigma \wedge \sigma$ *AC* $=$ *FEmpty* $\wedge \sigma$ *nAnC* $\neq$ *FEmpty*)
$|$ *holds-shape ECT* $\sigma = (\neg$ *sA-empty* $\sigma \wedge \neg$ *sA-univ* $\sigma \wedge \sigma$ *AC* $=$ *FEmpty* $\wedge \sigma$ *nAnC* $=$ *FEmpty*)
$|$ *holds-shape UNS* $\sigma = ($*sA-univ* $\sigma \wedge$ *sC-empty* $\sigma)$
$|$ *holds-shape FUN* $\sigma = ($*sFin-sub-A* $\sigma \wedge$ *sFin-sub-nC* $\sigma)$
$|$ *holds-shape WUC* $\sigma = (\neg$ *sA-empty* $\sigma \wedge \neg$ *sA-univ* $\sigma \wedge$ *sC-empty* $\sigma)$
$|$ *holds-shape WCT* $\sigma =$
$(\neg$ *sA-empty* $\sigma \wedge \sigma$ *AC* $=$ *FEmpty* $\wedge \sigma$ *nAnC* $\neq$ *FEmpty* $\wedge \neg$ *sC-empty* $\sigma)$
$|$ *holds-shape CMX* $\sigma = (\neg$ *sA-empty* $\sigma \wedge \neg$ *sC-univ* $\sigma \wedge \sigma$ *nAnC* $=$ *FEmpty*)
$|$ *holds-shape SCC* $\sigma = ($*sA-empty* $\sigma \wedge \neg$ *sC-univ* $\sigma)$
$|$ *holds-shape UCA* $\sigma = ($*sA-empty* $\sigma \wedge$ *sC-empty* $\sigma)$
$|$ *holds-shape TSU* $\sigma =$ *True*
$|$ *holds-shape TCP* $\sigma =$ *True*
$|$ *holds-shape TCC* $\sigma =$ *True*
$|$ *holds-shape VSU* $\sigma =$ *True*
$|$ *holds-shape VSG* $\sigma =$ *True*
$|$ *holds-shape VSB* $\sigma =$ *True*

3.4 Realizable shapes and the decision procedure

A shape is realizable when some region has a finite member. The all-empty shape needs an empty universe, and a shape without *FFin* region needs no finite interpretation to exist; the *szs* locale excludes both. 65 of the 81

shapes are realizable.

definition *realizable* :: *shape* ⇒ *bool* **where**
realizable σ ⟷ *list-ex* (λ*r*. *σ r* = *FFin*) *all-regions*

lemma *realizable-iff*: *realizable σ* ⟷ (∃ *r*. *σ r* = *FFin*)
by (*auto simp*: *realizable-def list-ex-iff*)

definition *isa-chk* :: *success* ⇒ *success* ⇒ *bool* **where**
isa-chk s t ≡
list-all (λ(*a*, *b*, *c*, *d*).
let σ = *mk-shape a b c d*
in realizable σ ⟶ *holds-shape s σ* ⟶ *holds-shape t σ*)
quads

definition *nevera-chk* :: *success* ⇒ *success* ⇒ *bool* **where**
nevera-chk s t ≡
list-all (λ(*a*, *b*, *c*, *d*).
let σ = *mk-shape a b c d*
in realizable σ ⟶ *holds-shape s σ* ⟶ ¬ *holds-shape t σ*)
quads

definition *xora-chk* :: *success* ⇒ *success* ⇒ *bool* **where**
xora-chk s t ≡
list-all (λ(*a*, *b*, *c*, *d*).
let σ = *mk-shape a b c d*
in realizable σ ⟶ *holds-shape s σ* ≠ *holds-shape t σ*)
quads

definition *nota-chk* :: *success* ⇒ *success* ⇒ *bool* **where**
nota-chk s t ≡ ¬ *isa-chk s t*

definition *mighta-chk* :: *success* ⇒ *success* ⇒ *bool* **where**
mighta-chk s t ≡ ¬ *nevera-chk s t*

lemma *isa-chk-iff*:
isa-chk s t ⟷ (∀ *σ*. *realizable σ* ⟶ *holds-shape s σ* ⟶ *holds-shape t σ*)
proof
assume ∗: *isa-chk s t*
show ∀ *σ*. *realizable σ* ⟶ *holds-shape s σ* ⟶ *holds-shape t σ*
proof (*intro allI impI*)
fix *σ* **assume** *realizable σ* **and** *holds-shape s σ*
have (*σ AC*, *σ AnC*, *σ nAC*, *σ nAnC*) ∈ *set quads* **by** (*rule quads-complete*)
with ∗ **have** *realizable σ* ⟶ *holds-shape s σ* ⟶ *holds-shape t σ*
unfolding *isa-chk-def Let-def*
by (*metis* (*mono-tags*, *lifting*) *Ball-set-list-all case-prodD shape-expand*)
with ‹*realizable σ*› ‹*holds-shape s σ*› **show** *holds-shape t σ* **by** *blast*
qed
next
assume ∀ *σ*. *realizable σ* ⟶ *holds-shape s σ* ⟶ *holds-shape t σ*

then show *isa-chk s t*
unfolding *isa-chk-def Let-def* **by** (*simp add*: *list-all-iff split-beta*)
qed

lemma *nevera-chk-iff*:
nevera-chk s t $\longleftrightarrow$ ($\forall \sigma$. *realizable* $\sigma \longrightarrow$ *holds-shape s* $\sigma \longrightarrow \neg$ *holds-shape t* σ)
proof
assume $*$: *nevera-chk s t*
show $\forall \sigma$. *realizable* $\sigma \longrightarrow$ *holds-shape s* $\sigma \longrightarrow \neg$ *holds-shape t* σ
proof (*intro allI impI*)
fix σ **assume** *realizable* σ **and** *holds-shape s* σ
have (σ *AC*, σ *AnC*, σ *nAC*, σ *nAnC*) $\in$ *set quads* **by** (*rule quads-complete*)
with $*$ **have** *realizable* $\sigma \longrightarrow$ *holds-shape s* $\sigma \longrightarrow \neg$ *holds-shape t* σ
unfolding *nevera-chk-def Let-def*
by (*metis* (*mono-tags*, *lifting*) *Ball-set-list-all case-prodD shape-expand*)
with ‹*realizable* σ› ‹*holds-shape s* σ› **show** $\neg$ *holds-shape t* σ **by** *blast*
qed
next
assume $\forall \sigma$. *realizable* $\sigma \longrightarrow$ *holds-shape s* $\sigma \longrightarrow \neg$ *holds-shape t* σ
then show *nevera-chk s t*
unfolding *nevera-chk-def Let-def* **by** (*simp add*: *list-all-iff split-beta*)
qed

lemma *xora-chk-iff*:
xora-chk s t $\longleftrightarrow$ ($\forall \sigma$. *realizable* $\sigma \longrightarrow$ *holds-shape s* $\sigma \neq$ *holds-shape t* σ)
proof
assume $*$: *xora-chk s t*
show $\forall \sigma$. *realizable* $\sigma \longrightarrow$ *holds-shape s* $\sigma \neq$ *holds-shape t* σ
proof (*intro allI impI*)
fix σ **assume** *realizable* σ
have (σ *AC*, σ *AnC*, σ *nAC*, σ *nAnC*) $\in$ *set quads* **by** (*rule quads-complete*)
with $*$ **have** *realizable* $\sigma \longrightarrow$ *holds-shape s* $\sigma \neq$ *holds-shape t* σ
unfolding *xora-chk-def Let-def*
by (*metis* (*mono-tags*, *lifting*) *Ball-set-list-all case-prodD shape-expand*)
with ‹*realizable* σ› **show** *holds-shape s* $\sigma \neq$ *holds-shape t* σ **by** *blast*
qed
next
assume $\forall \sigma$. *realizable* $\sigma \longrightarrow$ *holds-shape s* $\sigma \neq$ *holds-shape t* σ
then show *xora-chk s t*
unfolding *xora-chk-def Let-def* **by** (*simp add*: *list-all-iff split-beta*)
qed

3.5 The bridge to the model theory

definition *reg* :: *region* $\Rightarrow$ *'i set* $\Rightarrow$ *'i set* $\Rightarrow$ *'i set* **where**
reg r A C =
(*case r of AC* $\Rightarrow$ *A* $\cap$ *C* | *AnC* $\Rightarrow$ *A* $-$ *C* | *nAC* $\Rightarrow$ *C* $-$ *A* | *nAnC* $\Rightarrow$ $-$ (*A* $\cup$ *C*))

lemma *reg-simps* [*simp*]:
reg AC A C = *A* ∩ *C*
reg AnC A C = *A* − *C*
reg nAC A C = *C* − *A*
reg nAnC A C = − (*A* ∪ *C*)
by (*simp-all add*: *reg-def*)

lemma *reg-cover*: ∃ *r*. *x* ∈ *reg r A C*
by (*cases x* ∈ *A*; *cases x* ∈ *C*) (*auto intro*: *exI* [*of* - *AC*] *exI* [*of* - *AnC*]
exI [*of* - *nAC*] *exI* [*of* - *nAnC*])

context *szs-base*
begin

definition *shape-of* :: *'i set* ⇒ *'i set* ⇒ *shape* **where**
shape-of A C =
(λ*r*. *if reg r A C* = {} *then FEmpty*
else if reg r A C ∩ *Fin* = {} *then FInf else FFin*)

lemma *shape-of-FEmpty* [*simp*]: *shape-of A C r* = *FEmpty* ⟷ *reg r A C* = {}
by (*auto simp*: *shape-of-def*)

lemma *shape-of-FFin* [*simp*]: *shape-of A C r* = *FFin* ⟷ *reg r A C* ∩ *Fin* ≠ {}
by (*auto simp*: *shape-of-def*)

lemma *shape-of-neq-FFin*: *shape-of A C r* ≠ *FFin* ⟷ *reg r A C* ∩ *Fin* = {}
by (*simp add*: *shape-of-def*)

3.5.1 The abbreviations against the sets

lemma *sA-empty-shape* [*simp*]: *sA-empty* (*shape-of A C*) ⟷ *A* = {}
by (*auto simp*: *sA-empty-def*)

lemma *sC-empty-shape* [*simp*]: *sC-empty* (*shape-of A C*) ⟷ *C* = {}
by (*auto simp*: *sC-empty-def*)

lemma *sA-univ-shape* [*simp*]: *sA-univ* (*shape-of A C*) ⟷ *A* = *UNIV*
by (*auto simp*: *sA-univ-def*)

lemma *sC-univ-shape* [*simp*]: *sC-univ* (*shape-of A C*) ⟷ *C* = *UNIV*
by (*auto simp*: *sC-univ-def*)

lemma *sA-fin-shape* [*simp*]: *sA-fin* (*shape-of A C*) ⟷ *A* ∩ *Fin* ≠ {}
by (*auto simp*: *sA-fin-def*)

lemma *sFin-sub-A-shape* [*simp*]: *sFin-sub-A* (*shape-of A C*) ⟷ *Fin* ⊆ *A*
by (*auto simp*: *sFin-sub-A-def shape-of-neq-FFin*)

lemma *sFin-sub-C-shape* [*simp*]: *sFin-sub-C* (*shape-of A C*) ⟷ *Fin* ⊆ *C*

by (*auto simp*: *sFin-sub-C-def shape-of-neq-FFin*)

lemma *sFin-sub-nC-shape* [*simp*]: *sFin-sub-nC* (*shape-of A C*) $\longleftrightarrow$ *Fin* $\subseteq -$ *C*
by (*auto simp*: *sFin-sub-nC-def shape-of-neq-FFin*)

3.5.2 The bridge lemma

theorem *holds-shape-of*: *holds s A C* $\longleftrightarrow$ *holds-shape s* (*shape-of A C*)
by (*cases s*) (*auto simp*: *shape-of-neq-FFin*)

end

3.6 Soundness and completeness of the check

context *szs*
begin

lemma *realizable-shape-of*: *realizable* (*shape-of A C*)
proof −
obtain *x* **where** *x*: *x* $\in$ *Fin* **using** *Fin-nonempty* **by** *blast*
obtain *r* **where** *x* $\in$ *reg r A C* **using** *reg-cover* **by** *metis*
with *x* **have** *reg r A C* $\cap$ *Fin* $\neq$ {} **by** *blast*
then have *shape-of A C r* = *FFin* **by** *simp*
then show *?thesis* **by** (*auto simp*: *realizable-iff*)
qed

Every realizable shape is the shape of some $\langle Ax,\ C\rangle$. The witness takes one interpretation per nonempty region, finite for *FFin* and infinite for *FInf*, sends the remaining interpretations to a designated *FFin* region, and reads A and C off the partition.

lemma *realizable-witness*:
assumes *realizable* σ
shows $\exists$ *A C*. *shape-of A C* = σ
proof −
obtain *r0* **where** *r0*: σ *r0* = *FFin* **using** *assms* **by** (*auto simp*: *realizable-iff*)

obtain *f* :: *nat* $\Rightarrow$ *'i* **where** *f*: *inj f range f* $\subseteq$ *Fin*
using *Fin-infinite infinite-countable-subset* **by** *blast*
obtain *g* :: *nat* $\Rightarrow$ *'i* **where** *g*: *inj g range g* $\subseteq -$ *Fin*
using *coFin-infinite infinite-countable-subset* **by** *blast*

define *idx* :: *region* $\Rightarrow$ *nat* **where**
idx r = (*case r of AC* $\Rightarrow$ *0* | *AnC* $\Rightarrow$ *1* | *nAC* $\Rightarrow$ *2* | *nAnC* $\Rightarrow$ *3*) **for** *r*
have *idx-inj*: *inj idx*
by (*rule injI*) (*case-tac x*; *case-tac y*; *simp add*: *idx-def*)

define *w* :: *region* $\Rightarrow$ *'i* **where**
w r = (*if* σ *r* = *FFin then f* (*idx r*) *else g* (*idx r*)) **for** *r*
have *w-Fin*: σ *r* = *FFin* $\Longrightarrow$ *w r* $\in$ *Fin* **for** *r*

using f **by** (*auto simp*: *w-def*)
have *w-coFin*: $\sigma\ r \neq FFin \Longrightarrow w\ r \notin Fin$ **for** r
using g **by** (*auto simp*: *w-def*)
have *w-inj*: *inj w*
proof (*rule injI*)
fix $r\ r'$ **assume** $w\ r = w\ r'$
moreover have $\sigma\ r = FFin \longleftrightarrow \sigma\ r' = FFin$
using ‹$w\ r = w\ r'$› *w-Fin w-coFin* **by** *metis*
ultimately have $idx\ r = idx\ r'$
using $f\ g$ **by** (*cases* $\sigma\ r = FFin$) (*auto simp*: *w-def dest*: *injD*)
then show $r = r'$ **using** *idx-inj* **by** (*auto dest*: *injD*)
qed

define W :: *region set* **where** $W = \{r.\ r \neq r0 \wedge \sigma\ r \neq FEmpty\}$
have *W-fin*: *finite W*
by (*metis List.finite-set all-regions-UNIV rev-finite-subset top-greatest*)

define *pick* :: $'i \Rightarrow$ *region* **where**
pick x = (*if* $\exists r \in W.\ x = w\ r$ *then THE* $r.\ r \in W \wedge x = w\ r$ *else r0*) **for** x
have *pick-w*: $r \in W \Longrightarrow pick\ (w\ r) = r$ **for** r
proof −
assume *r*: $r \in W$
have (*THE* $r'.\ r' \in W \wedge w\ r = w\ r'$) $= r$
by (*rule the-equality*) (*use r w-inj* **in** ‹*auto dest*: *injD*›)
with r **show** *?thesis* **by** (*auto simp*: *pick-def*)
qed
have *pick-r0*: $\forall r \in W.\ x \neq w\ r \Longrightarrow pick\ x = r0$ **for** x
by (*simp add*: *pick-def*)
have *pick-in-W*: *pick* $x \neq r0 \Longrightarrow pick\ x \in W \wedge x = w\ (pick\ x)$ **for** x
proof −
assume *pick* $x \neq r0$
then obtain r **where** *r*: $r \in W\ x = w\ r$
using *pick-r0* **by** *auto*
with *pick-w* **show** *?thesis* **by** *auto*
qed

define S :: *region* $\Rightarrow$ $'i$ *set* **where** $S\ r = \{x.\ pick\ x = r\}$ **for** r
have *S-disj*: $r \neq r' \Longrightarrow S\ r \cap S\ r' = \{\}$ **for** $r\ r'$ **by** (*auto simp*: *S-def*)
have *S-cover*: $(\bigcup r.\ S\ r) = UNIV$ **by** (*auto simp*: *S-def*)

have *S-empty*: $\sigma\ r = FEmpty \Longrightarrow S\ r = \{\}$ **for** r
using *r0 pick-in-W W-def S-def* **by** *force*
have *S-witness*: $r \in W \Longrightarrow S\ r = \{w\ r\}$ **for** r
using *pick-w pick-in-W W-def S-def* **by** *force*
have *S-r0-big*: $-\ S\ r0 \subseteq w\ `\ W$
using *pick-in-W* **by** (*auto simp*: *S-def*)

define A **where** $A = S\ AC \cup S\ AnC$
define C **where** $C = S\ AC \cup S\ nAC$

```
have reg-S: reg r A C = S r for r
proof (cases r)
  case AC show ?thesis using AC S-disj by (auto simp: A-def C-def)
next
  case AnC show ?thesis using AnC S-disj by (auto simp: A-def C-def)
next
  case nAC show ?thesis using nAC S-disj by (auto simp: A-def C-def)
next
  case nAnC
  have x ∈ - (A ∪ C) ⟷ x ∈ S nAnC for x
    using S-cover S-disj S-def
    by (smt (verit, del-insts) A-def C-def Compl-iff Un-iff
        mem-Collect-eq region.distinct(11,5,9)
        region.exhaust)
  with nAnC show ?thesis by auto
qed

have shape-of A C r = σ r for r
proof (cases σ r)
  case FEmpty
  then show ?thesis using S-empty reg-S by simp
next
  case FInf
  then have r ∈ W using r0 by (auto simp: W-def)
  then have reg r A C = {w r} using reg-S S-witness by simp
  moreover have w r ∉ Fin using FInf w-coFin by simp
  ultimately show ?thesis using FInf by (simp add: shape-of-def)
next
  case FFin
  show ?thesis
  proof (cases r = r0)
    case False
    then have r ∈ W using FFin by (auto simp: W-def)
    then have reg r A C = {w r} using reg-S S-witness by simp
    moreover have w r ∈ Fin using FFin w-Fin by simp
    ultimately show ?thesis using FFin by simp
  next
    case True
    have finite (w ‘ W) using W-fin by simp
    then have ¬ Fin ⊆ w ‘ W using Fin-infinite by (metis finite-subset)
    then obtain x where x ∈ Fin x ∉ w ‘ W by blast
    then have x ∈ S r0 using S-r0-big by blast
    then have reg r0 A C ∩ Fin ≠ {} using reg-S ‹x ∈ Fin› by auto
    then show ?thesis using True FFin by simp
  qed
qed
then have shape-of A C = σ by (rule ext)
then show ?thesis by blast
```

qed

3.6.1 The main equivalences

theorem *isa-iff-chk*: *isa s t* $\longleftrightarrow$ *isa-chk s t*
proof
assume *isa s t*
show *isa-chk s t*
unfolding *isa-chk-iff*
proof (*intro allI impI*)
fix σ **assume** *realizable* σ **and** *holds-shape s* σ
then obtain *A C* **where** *AC*: *shape-of A C* $= \sigma$ **using** *realizable-witness* **by** *blast*
with ‹*holds-shape s* σ› **have** *holds s A C* **by** (*simp add*: *holds-shape-of*)
with ‹*isa s t*› **have** *holds t A C* **unfolding** *isa-def* **by** *blast*
with *AC* **show** *holds-shape t* σ **by** (*simp add*: *holds-shape-of*)
qed
next
assume *isa-chk s t*
show *isa s t*
unfolding *isa-def*
proof (*intro allI impI*)
fix *A C* **assume** *holds s A C*
then have *holds-shape s* (*shape-of A C*) **by** (*simp add*: *holds-shape-of*)
with ‹*isa-chk s t*› *realizable-shape-of*
have *holds-shape t* (*shape-of A C*) **unfolding** *isa-chk-iff* **by** *blast*
then show *holds t A C* **by** (*simp add*: *holds-shape-of*)
qed
qed

theorem *nevera-iff-chk*: *nevera s t* $\longleftrightarrow$ *nevera-chk s t*
proof
assume *nevera s t*
show *nevera-chk s t*
unfolding *nevera-chk-iff*
proof (*intro allI impI*)
fix σ **assume** *realizable* σ **and** *holds-shape s* σ
then obtain *A C* **where** *AC*: *shape-of A C* $= \sigma$ **using** *realizable-witness* **by** *blast*
with ‹*holds-shape s* σ› **have** *holds s A C* **by** (*simp add*: *holds-shape-of*)
with ‹*nevera s t*› **have** $\neg$ *holds t A C* **unfolding** *nevera-def* **by** *blast*
with *AC* **show** $\neg$ *holds-shape t* σ **by** (*simp add*: *holds-shape-of*)
qed
next
assume *nevera-chk s t*
show *nevera s t*
unfolding *nevera-def*
proof (*intro allI impI*)
fix *A C* **assume** *holds s A C*

```
    then have holds-shape s (shape-of A C) by (simp add: holds-shape-of)
    with ‹nevera-chk s t› realizable-shape-of
    have ¬ holds-shape t (shape-of A C) unfolding nevera-chk-iff by blast
    then show ¬ holds t A C by (simp add: holds-shape-of)
  qed
qed

theorem xora-iff-chk: xora s t ⟷ xora-chk s t
proof
  assume xora s t
  show xora-chk s t
    unfolding xora-chk-iff
  proof (intro allI impI)
    fix σ assume realizable σ
    then obtain A C where AC: shape-of A C = σ using realizable-witness by
blast
    from ‹xora s t› have holds s A C ≠ holds t A C unfolding xora-def by blast
   with AC show holds-shape s σ ≠ holds-shape t σ by (simp add: holds-shape-of)
  qed
next
  assume xora-chk s t
  show xora s t
    unfolding xora-def
  proof (intro allI)
    fix A C
    have holds-shape s (shape-of A C) ≠ holds-shape t (shape-of A C)
      using ‹xora-chk s t› realizable-shape-of unfolding xora-chk-iff by blast
    then show holds s A C ≠ holds t A C by (simp add: holds-shape-of)
  qed
qed

theorem nota-iff-chk: nota s t ⟷ nota-chk s t
  by (simp add: nota-iff nota-chk-def isa-iff-chk)

theorem mighta-iff-chk: mighta s t ⟷ mighta-chk s t
  by (simp add: mighta-iff mighta-chk-def nevera-iff-chk)

end
```

3.7 Sanity checks

The size of the search space and the mirror symmetry of *isa-chk*. *isa-iff-chk* transfers both to *isa* in every instance of *szs*.

```
lemma card-realizable-shapes:
  length (filter (λ(a, b, c, d). realizable (mk-shape a b c d)) quads) = 65
  by eval
```

Mirror symmetry of the check, computed over 53×53 pairs.

```
lemma isa-chk-mirror:
```

list-all (λ*s*. *list-all* (λ*t*. *isa-chk* (*mirror s*) (*mirror t*) = *isa-chk s t*) *all-success*) *all-success*
by *eval*

3.8 Invariance under signature extension

The shape of $\langle Ax,\ C\rangle$ is the same over the signature of Ax and over any extension of it. With *holds-shape-of* this gives *holds-vimage*, the invariance of all conditions, and *mex-sig-iff* identifies the MEX entry of *holds* with the two-signature condition.

context *szs-ext*
begin

lemma *reg-vimage*: *reg q* (*r* −‘ *A*) (*r* −‘ *C*) = *r* −‘ *reg q A C*
by (*cases q*) *auto*

lemma *shape-of-vimage*:
szs-base.shape-of Fin-j (*r* −‘ *A*) (*r* −‘ *C*) = *shape-of A C*
proof (*rule ext*)
fix *q* :: *region*
have (*reg q* (*r* −‘ *A*) (*r* −‘ *C*) = {}) = (*reg q A C* = {})
by (*simp add*: *reg-vimage*)
moreover have (*reg q* (*r* −‘ *A*) (*r* −‘ *C*) ∩ *Fin-j* = {}) = (*reg q A C* ∩ *Fin* = {})
by (*simp add*: *reg-vimage Fin-j-def flip*: *vimage-Int*)
ultimately show *szs-base.shape-of Fin-j* (*r* −‘ *A*) (*r* −‘ *C*) *q* = *shape-of A C q*
by (*simp add*: *szs-base.shape-of-def shape-of-def*)
qed

theorem *holds-vimage*: *szs-base.holds Fin-j s* (*r* −‘ *A*) (*r* −‘ *C*) = *holds s A C*
by (*simp add*: *szs-base.holds-shape-of shape-of-vimage holds-shape-of*)

definition *isa-ext* :: *success* ⇒ *success* ⇒ *bool* **where**
isa-ext s t ≡
(∀ *A C*. *szs-base.holds Fin-j s* (*r* −‘ *A*) *C* ⟶ *szs-base.holds Fin-j t* (*r* −‘ *A*) *C*)

theorem *isa-ext-iff-chk*: *isa-ext s t* ⟷ *isa-chk s t*
proof
assume *ext*: *isa-ext s t*
show *isa-chk s t*
unfolding *isa-chk-iff*
proof (*intro allI impI*)
fix σ **assume** *realizable* σ **and** *holds-shape s* σ
then obtain *A C* **where** *AC*: *shape-of A C* = σ
using *realizable-witness* **by** *blast*
have *lift*: *szs-base.shape-of Fin-j* (*r* −‘ *A*) (*r* −‘ *C*) = σ
by (*simp add*: *shape-of-vimage AC*)

```
    from ‹holds-shape s σ› lift
    have szs-base.holds Fin-j s (r −‘ A) (r −‘ C)
      by (simp add: szs-base.holds-shape-of)
    with ext have szs-base.holds Fin-j t (r −‘ A) (r −‘ C)
      unfolding isa-ext-def by blast
    with lift show holds-shape t σ
      by (simp add: szs-base.holds-shape-of)
  qed
next
  assume chk: isa-chk s t
  show isa-ext s t
    unfolding isa-ext-def
  proof (intro allI impI)
    fix A :: 'i set and C :: 'j set
    assume szs-base.holds Fin-j s (r −‘ A) C
    then have holds-shape s (szs-base.shape-of Fin-j (r −‘ A) C)
      by (simp add: szs-base.holds-shape-of)
    moreover have realizable (szs-base.shape-of Fin-j (r −‘ A) C)
      by (rule szs.realizable-shape-of [OF szs-Fin-j])
    ultimately have holds-shape t (szs-base.shape-of Fin-j (r −‘ A) C)
      using chk unfolding isa-chk-iff by blast
    then show szs-base.holds Fin-j t (r −‘ A) C
      by (simp add: szs-base.holds-shape-of)
  qed
qed

theorem mex-sig-entails-iff:
  (∀ A C. mex-sig A C ⟶ szs-base.holds Fin-j t (r −‘ A) C) ⟷ isa-chk MEX t
proof −
  have (∀ A C. mex-sig A C ⟶ szs-base.holds Fin-j t (r −‘ A) C) = isa-ext MEX
t
    by (simp add: isa-ext-def mex-sig-iff)
  then show ?thesis by (simp add: isa-ext-iff-chk)
qed

theorem entails-mex-sig-iff:
  (∀ A C. szs-base.holds Fin-j s (r −‘ A) C ⟶ mex-sig A C) ⟷ isa-chk s MEX
proof −
  have (∀ A C. szs-base.holds Fin-j s (r −‘ A) C ⟶ mex-sig A C) = isa-ext s
MEX
    by (simp add: isa-ext-def mex-sig-iff)
  then show ?thesis by (simp add: isa-ext-iff-chk)
qed

theorem cmx-sig-entails-iff:
  (∀ A C. cmx-sig A C ⟶ szs-base.holds Fin-j t (r −‘ A) C) ⟷ isa-chk CMX t
proof −
  have (∀ A C. cmx-sig A C ⟶ szs-base.holds Fin-j t (r −‘ A) C) = isa-ext CMX
t
```

by (*simp add*: *isa-ext-def cmx-sig-iff*)
then show *?thesis* **by** (*simp add*: *isa-ext-iff-chk*)
qed

theorem *entails-cmx-sig-iff*:
(∀ *A C*. *szs-base.holds Fin-j s* (*r* −‘ *A*) *C* ⟶ *cmx-sig A C*) ⟷ *isa-chk s CMX*
proof −
have (∀ *A C*. *szs-base.holds Fin-j s* (*r* −‘ *A*) *C* ⟶ *cmx-sig A C*) = *isa-ext s CMX*
by (*simp add*: *isa-ext-def cmx-sig-iff*)
then show *?thesis* **by** (*simp add*: *isa-ext-iff-chk*)
qed

end

end

theory *SZS-Hierarchy*
imports *SZS-Shapes*
begin

4 Hierarchy

The entailment order over the forty-five model theoretic values is computed from SZS_Shapes and reduced to its transitive reduction. Fifteen stipulated edges for the frame and the two non model theoretic subontologies are added. SZS_BNF generates the grammar from the result.

4.1 The derived order

entails is strict entailment restricted to the model theoretic values. Pairs (*s*, *t*) are oriented child before parent: *s* entails *t*.

definition *entails* :: (*success* × *success*) *list* **where**
entails =
filter (λ(*s*, *t*). *s* ≠ *t* ∧ *isa-chk s t*)
(*List.product model-theoretic-values model-theoretic-values*)

definition *above* :: *success* ⇒ *success list* **where**
above s = *filter* (λ*t*. (*s*, *t*) ∈ *set entails*) *model-theoretic-values*

definition *below* :: *success* ⇒ *success list* **where**
below t = *filter* (λ*s*. (*s*, *t*) ∈ *set entails*) *model-theoretic-values*

4.1.1 Well-formedness

entails is a strict partial order. Antisymmetry says that distinct values have distinct meanings; without it the transitive reduction is not unique.

lemma *entails-irrefl*: $(s,\ s) \notin$ *set entails*
by (*simp add*: *entails-def*)

lemma *entails-isa-chk*: $(s,\ t) \in$ *set entails* $\Longrightarrow$ *isa-chk s t*
by (*auto simp*: *entails-def*)

lemma *entails-model-theoretic*:
$(s,\ t) \in$ *set entails* $\Longrightarrow$ *model-theoretic s* $\wedge$ *model-theoretic t*
by (*auto simp*: *entails-def model-theoretic-values-def*)

lemma *entails-antisym*:
list-all $(\lambda(s,\ t).\ (t,\ s) \notin$ *set entails*$)$ *entails*
by *eval*

lemma *entails-trans*:
list-all $(\lambda(s,\ t).$ *list-all* $(\lambda u.\ (t,\ u) \in$ *set entails* $\longrightarrow (s,\ u) \in$ *set entails*$)$ *model-theoretic-values*$)$
entails
by *eval*

Mirror closure, checked directly.

lemma *entails-mirror*:
list-all $(\lambda(s,\ t).\ ($*mirror s*, *mirror t*$) \in$ *set entails*$)$ *entails*
by *eval*

4.2 Transitive reduction

covers s t holds when s entails t with nothing strictly between. For a finite strict partial order the cover relation is the unique transitive reduction.

definition *covers* :: (*success* $\times$ *success*) *list* **where**
covers =
filter $(\lambda(s,\ t).$
$\neg$ *list-ex* $(\lambda u.\ (s,\ u) \in$ *set entails* $\wedge\ (u,\ t) \in$ *set entails*$)$
model-theoretic-values$)$
entails

lemma *covers-subset*: *set covers* $\subseteq$ *set entails*
by (*auto simp*: *covers-def*)

lemma *covers-mirror*:
list-all $(\lambda(s,\ t).\ ($*mirror s*, *mirror t*$) \in$ *set covers*$)$ *covers*
by *eval*

The reduction generates the order back, checked by evaluation over the finite carrier.

lemma *covers-generates*:
list-all (λ(*s*, *t*).
(*s*, *t*) ∈ *set covers* ∨
list-ex (λ*u*. (*s*, *u*) ∈ *set covers* ∧ (*u*, *t*) ∈ *set entails*) *model-theoretic-values*)
entails
by *eval*

4.3 The full hierarchy

4.3.1 Maximal model theoretic values

The values with no proper consequence among the forty-five. They hang directly below SemanticSuccess.

definition *maximal-values* :: *success list* **where**
maximal-values = *filter* (λ*t*. *above t* = []) *model-theoretic-values*

lemma *maximal-values-eval*: *maximal-values* = [*UNP*, *SAP*, *TAP*, *FTH*, *CUP*, *CSP*, *CTP*, *FCT*]
by *eval*

lemma *maximal-values-mirror*:
list-all (λ*s*. *mirror s* ∈ *set maximal-values*) *maximal-values*
by *eval*

4.3.2 Children of every value

definition *cover-children* :: *success* ⇒ *success list* **where**
cover-children t = *filter* (λ*s*. (*s*, *t*) ∈ *set covers*) *model-theoretic-values*

The stipulated edges. Success splits into the three subontologies, SemanticSuccess covers the derived order, and the type checking and verification subontologies are chains and pairs.

fun *bnf-children* :: *success* ⇒ *success list* **where**
bnf-children SUC = [*SSU*, *TSU*, *VSU*]
| *bnf-children SSU* = *maximal-values*
| *bnf-children TSU* = [*TCP*]
| *bnf-children TCP* = [*TCC*]
| *bnf-children TCC* = []
| *bnf-children VSU* = [*VSG*, *VSB*]
| *bnf-children VSG* = []
| *bnf-children VSB* = []
| *bnf-children t* = *cover-children t*

definition *bnf-edges* :: (*success* × *success*) *list* **where**
bnf-edges = [(*s*, *t*). *t* ← *all-success*, *s* ← *bnf-children t*]

lemma *bnf-edges-cover-all*:
list-all (λ*s*. *s* = *SUC* ∨ *list-ex* (λ*t*. (*s*, *t*) ∈ *set bnf-edges*) *all-success*) *all-success*
by *eval*

lemma *bnf-acyclic*:
list-all $(\lambda(s,\ t).\ s \neq t \wedge (t,\ s) \notin set\ bnf\text{-}edges)$ *bnf-edges*
by *eval*

end

theory *SZS-BNF*
imports *SZS-Hierarchy*
begin

5 BNF

The BNF of the three SZS ontologies. The success ontology comes from SZS_Hierarchy as *bnf-edges*; the no-success and dataform ontologies are transcribed from https://szs.tptp.org/nosuccess and https://szs.tptp.org/dataform. Each ontology is checked to have one root, no cycle, every value reachable, no implied edge and distinct spellings. The grammar has one rule per value with children. It is rendered as text and exported from the session; the BNF in the repository is checked against it by SZS_BNF_Check.

5.1 The no-success ontology

5.1.1 Vocabulary

The values of https://szs.tptp.org/nosuccess.

datatype *nosuccess* =
  NOS
| *UNK* | *STP* | *INP* | *NTT* | *NTY*
| *ERR* | *FOR* | *GUP*
| *OSE* | *INE* | *SYE* | *SEE* | *TYE* | *USM* | *USE*
| *USR* | *RSO* | *TMO* | *CTO* | *WTO* | *MMO* | *INO*
| *INC* | *IAP* | *ICT*
| *ASS* | *OPN* | *NVE* | *FVE*

definition *all-nosuccess* :: *nosuccess list* **where**
all-nosuccess =
[*NOS*,
 UNK, *STP*, *INP*, *NTT*, *NTY*,
 ERR, *FOR*, *GUP*,
 OSE, *INE*, *SYE*, *SEE*, *TYE*, *USM*, *USE*,
 USR, *RSO*, *TMO*, *CTO*, *WTO*, *MMO*, *INO*,
 INC, *IAP*, *ICT*,
 ASS, *OPN*, *NVE*, *FVE*]

lemma *all-nosuccess-UNIV* [*simp*]: *set all-nosuccess* = *UNIV*

proof −
have $s \in$ *set all-nosuccess* **for** *s*
by (*cases s*) (*simp-all add*: *all-nosuccess-def*)
then show *?thesis* **by** *auto*
qed

lemma *distinct-all-nosuccess*: *distinct all-nosuccess*
by *eval*

lemma *length-all-nosuccess*: *length all-nosuccess* = *30*
by (*simp add*: *all-nosuccess-def*)

primrec *ns-one-word* :: *nosuccess* ⇒ *string* **where**
ns-one-word NOS = *''NoSuccess''*
| *ns-one-word UNK* = *''Unknown''*
| *ns-one-word STP* = *''Stopped''*
| *ns-one-word INP* = *''InProgress''*
| *ns-one-word NTT* = *''NotTried''*
| *ns-one-word NTY* = *''NotTriedYet''*
| *ns-one-word ERR* = *''Error''*
| *ns-one-word FOR* = *''Forced''*
| *ns-one-word GUP* = *''GaveUp''*
| *ns-one-word OSE* = *''OSError''*
| *ns-one-word INE* = *''InputError''*
| *ns-one-word SYE* = *''SyntaxError''*
| *ns-one-word SEE* = *''SemanticError''*
| *ns-one-word TYE* = *''TypeError''*
| *ns-one-word USM* = *''Unsemantic''*
| *ns-one-word USE* = *''UsageError''*
| *ns-one-word USR* = *''User''*
| *ns-one-word RSO* = *''ResourceOut''*
| *ns-one-word TMO* = *''Timeout''*
| *ns-one-word CTO* = *''CPUTimeout''*
| *ns-one-word WTO* = *''WCTimeout''*
| *ns-one-word MMO* = *''MemoryOut''*
| *ns-one-word INO* = *''InstructionsOut''*
| *ns-one-word INC* = *''Incomplete''*
| *ns-one-word IAP* = *''Inappropriate''*
| *ns-one-word ICT* = *''Incorrect''*
| *ns-one-word ASS* = *''Assumed''*
| *ns-one-word OPN* = *''Open''*
| *ns-one-word NVE* = *''NotVerified''*
| *ns-one-word FVE* = *''FailedVerified''*

5.1.2 The stipulated order

Child before parent.

definition *nosuccess-isa* :: (*nosuccess* × *nosuccess*) *list* **where**

nosuccess-isa =
[(*OPN*, *NOS*), (*NVE*, *NOS*), (*ASS*, *NOS*), (*UNK*, *NOS*), (*ICT*, *NOS*),
(*FVE*, *NVE*),
(*STP*, *UNK*), (*INP*, *UNK*), (*NTT*, *UNK*),
(*NTY*, *NTT*), (*IAP*, *NTT*),
(*FOR*, *STP*), (*GUP*, *STP*),
(*INE*, *ERR*), (*OSE*, *ERR*),
(*SYE*, *INE*), (*SEE*, *INE*), (*USE*, *INE*),
(*TYE*, *SEE*), (*USM*, *SEE*),
(*USR*, *FOR*), (*RSO*, *FOR*),
(*TMO*, *RSO*), (*MMO*, *RSO*), (*INO*, *RSO*),
(*CTO*, *TMO*), (*WTO*, *TMO*),
(*RSO*, *GUP*), (*INC*, *GUP*), (*ERR*, *GUP*), (*IAP*, *GUP*)]

lemma *length-nosuccess-isa*: *length nosuccess-isa* = *31*
by (*simp add*: *nosuccess-isa-def*)

5.2 The dataform ontology

5.2.1 Vocabulary

The values of https://szs.tptp.org/dataform. The constructors are the mnemonics. Interpretation is *Itp*, since *Int* is taken.

datatype *dataform* =
Dat
| *LDa* | *NLd*
| *NSo* | *Sln*
| *Ass* | *IPr* | *InI*
| *Prf* | *Itp* | *Lof*
| *Ref* | *Der* | *CRf*
| *Mod*
| *DIn* | *DMo* | *FIn* | *FMo* | *IIn* | *IMo*
| *HIn* | *HMo* | *FHi* | *FHm* | *Sat*
| *Com* | *FTx* | *Ver*
| *Non*

definition *all-dataform* :: *dataform list* **where**
all-dataform =
[*Dat*,
LDa, *NLd*,
NSo, *Sln*,
Ass, *IPr*, *InI*,
Prf, *Itp*, *Lof*,
Ref, *Der*, *CRf*,
Mod,
DIn, *DMo*, *FIn*, *FMo*, *IIn*, *IMo*,
HIn, *HMo*, *FHi*, *FHm*, *Sat*,
Com, *FTx*, *Ver*,
Non]

lemma *all-dataform-UNIV* [*simp*]: *set all-dataform = UNIV*
proof −
  have *d ∈ set all-dataform* **for** *d*
    by (*cases d*) (*simp-all add*: *all-dataform-def*)
  then show *?thesis* **by** *auto*
qed

lemma *distinct-all-dataform*: *distinct all-dataform*
  by *eval*

lemma *length-all-dataform*: *length all-dataform = 30*
  by (*simp add*: *all-dataform-def*)

primrec *df-one-word* :: *dataform ⇒ string* **where**
  df-one-word Dat = ''Data''
| *df-one-word LDa = ''LogicalData''*
| *df-one-word NLd = ''NonLogicalData''*
| *df-one-word NSo = ''NotASolution''*
| *df-one-word Sln = ''Solution''*
| *df-one-word Ass = ''Assurance''*
| *df-one-word IPr = ''IncompleteProof''*
| *df-one-word InI = ''IncompleteInterpretation''*
| *df-one-word Prf = ''Proof''*
| *df-one-word Itp = ''Interpretation''*
| *df-one-word Lof = ''ListOfFormulae''*
| *df-one-word Ref = ''Refutation''*
| *df-one-word Der = ''Derivation''*
| *df-one-word CRf = ''CNFRefutation''*
| *df-one-word Mod = ''Model''*
| *df-one-word DIn = ''DomainInterpretation''*
| *df-one-word DMo = ''DomainModel''*
| *df-one-word FIn = ''FiniteInterpretation''*
| *df-one-word FMo = ''FiniteModel''*
| *df-one-word IIn = ''InfiniteInterpretation''*
| *df-one-word IMo = ''InfiniteModel''*
| *df-one-word HIn = ''HerbrandInterpretation''*
| *df-one-word HMo = ''HerbrandModel''*
| *df-one-word FHi = ''FormulaHerbrandInterpretation''*
| *df-one-word FHm = ''FormulaHerbrandModel''*
| *df-one-word Sat = ''Saturation''*
| *df-one-word Com = ''Comment''*
| *df-one-word FTx = ''FreeText''*
| *df-one-word Ver = ''Verification''*
| *df-one-word Non = ''None''*

5.2.2 The stipulated order

definition *dataform-isa* :: (*dataform* × *dataform*) *list* **where**
dataform-isa =
[(*Non*, *Dat*), (*LDa*, *Dat*), (*NLd*, *Dat*),
(*NSo*, *LDa*), (*Sln*, *LDa*),
(*Ass*, *NSo*), (*IPr*, *NSo*), (*InI*, *NSo*),
(*Prf*, *Sln*), (*Itp*, *Sln*), (*Lof*, *Sln*),
(*Ref*, *Prf*), (*Der*, *Prf*), (*CRf*, *Ref*),
(*Mod*, *Itp*),
(*DIn*, *Itp*), (*DMo*, *Mod*), (*DMo*, *DIn*),
(*FIn*, *DIn*), (*FMo*, *DMo*), (*FMo*, *FIn*),
(*IIn*, *DIn*), (*IMo*, *DMo*), (*IMo*, *IIn*),
(*HIn*, *Itp*), (*HIn*, *Lof*), (*HMo*, *Mod*), (*HMo*, *HIn*),
(*FHi*, *HIn*), (*FHm*, *HMo*), (*FHm*, *FHi*),
(*Sat*, *FHm*),
(*Com*, *NLd*), (*FTx*, *NLd*), (*Ver*, *FTx*)]

lemma *length-dataform-isa*: *length dataform-isa* = *35*
by (*simp add*: *dataform-isa-def*)

5.3 Structural checks

5.3.1 Graph operations

definition *nodes* :: ('*a*::*equal* × '*a*) *list* ⇒ '*a list* **where**
nodes E = *remdups* (*map fst E* @ *map snd E*)

definition *parents* :: ('*a*::*equal* × '*a*) *list* ⇒ '*a* ⇒ '*a list* **where**
parents E x = *map snd* (*filter* (λ*e*. *fst e* = *x*) *E*)

definition *children* :: ('*a*::*equal* × '*a*) *list* ⇒ '*a* ⇒ '*a list* **where**
children E x = *map fst* (*filter* (λ*e*. *snd e* = *x*) *E*)

fun *close* :: *nat* ⇒ ('*a*::*equal* ⇒ '*a list*) ⇒ '*a list* ⇒ '*a list* **where**
close 0 f xs = *xs*
| *close* (*Suc k*) *f xs* =
(*let ys* = *remdups* (*xs* @ *concat* (*map f xs*))
in if length ys = *length xs then xs else close k f ys*)

definition *ancestors* :: ('*a*::*equal* × '*a*) *list* ⇒ '*a* ⇒ '*a list* **where**
ancestors E x = *close* (*length* (*nodes E*)) (*parents E*) (*parents E x*)

definition *descendants* :: ('*a*::*equal* × '*a*) *list* ⇒ '*a* ⇒ '*a list* **where**
descendants E x = *close* (*length* (*nodes E*)) (*children E*) (*children E x*)

definition *roots* :: ('*a*::*equal* × '*a*) *list* ⇒ '*a list* **where**
roots E = *filter* (λ*x*. *parents E x* = []) (*nodes E*)

definition *acyclic-edges* :: ('*a*::*equal* × '*a*) *list* ⇒ *bool* **where**

acyclic-edges E = *list-all* (λ*x*. *x* ∉ *set* (*ancestors E x*)) (*nodes E*)

definition *rooted* :: ('*a*::*equal* × '*a*) *list* ⇒ '*a* ⇒ *bool* **where**
rooted E r = *list-all* (λ*x*. *x* = *r* ∨ *r* ∈ *set* (*ancestors E x*)) (*nodes E*)

definition *implied* :: ('*a*::*equal* × '*a*) *list* ⇒ ('*a* × '*a*) *list* **where**
implied E =
filter (λ(*c*, *p*). *list-ex* (λ*q*. *q* ≠ *p* ∧ *p* ∈ *set* (*ancestors E q*)) (*parents E c*)) *E*

The number of paths from the root to a value is the number of derivations of its word. *derivations E r x* = *0* means that the grammar does not generate *x*, *1* < that it generates *x* in more than one way.

fun *path-count* :: *nat* ⇒ ('*a*::*equal* × '*a*) *list* ⇒ '*a* ⇒ '*a* ⇒ *nat* **where**
path-count 0 E r x = (*if x* = *r then 1 else 0*)
| *path-count* (*Suc k*) *E r x* =
(*if x* = *r then 1 else sum-list* (*map* (*path-count k E r*) (*parents E x*)))

definition *derivations* :: ('*a*::*equal* × '*a*) *list* ⇒ '*a* ⇒ '*a* ⇒ *nat* **where**
derivations E r x = *path-count* (*length* (*nodes E*)) *E r x*

definition *generates* :: ('*a*::*equal* × '*a*) *list* ⇒ '*a* ⇒ '*a list* ⇒ *bool* **where**
generates E r xs = *list-all* (λ*x*. *0* < *derivations E r x*) *xs*

definition *ambiguous* :: ('*a*::*equal* × '*a*) *list* ⇒ '*a* ⇒ '*a list* ⇒ '*a list* **where**
ambiguous E r xs = *filter* (λ*x*. *1* < *derivations E r x*) *xs*

5.3.2 The success ontology

bnf-acyclic in SZS_Hierarchy excludes two-cycles; *acyclic-edges* excludes cycles of any length.

lemma *nodes-bnf-edges*: *length* (*nodes bnf-edges*) = *53*
by *eval*

lemma *nodes-bnf-edges-complete*:
list-all (λ*s*. *s* ∈ *set* (*nodes bnf-edges*)) *all-success*
by *eval*

lemma *roots-bnf-edges*: *roots bnf-edges* = [*SUC*]
by *eval*

lemma *acyclic-bnf-edges*: *acyclic-edges bnf-edges*
by *eval*

lemma *rooted-bnf-edges*: *rooted bnf-edges SUC*
by *eval*

lemma *implied-bnf-edges*: *implied bnf-edges* = []
by *eval*

5.3.3 The no-success ontology

lemma *nodes-nosuccess-isa*: *length* (*nodes nosuccess-isa*) = *30*
by *eval*

lemma *nodes-nosuccess-complete*:
list-all (λ*s*. *s* ∈ *set* (*nodes nosuccess-isa*)) *all-nosuccess*
by *eval*

lemma *roots-nosuccess-isa*: *roots nosuccess-isa* = [*NOS*]
by *eval*

lemma *acyclic-nosuccess-isa*: *acyclic-edges nosuccess-isa*
by *eval*

lemma *rooted-nosuccess-isa*: *rooted nosuccess-isa NOS*
by *eval*

5.3.4 The dataform ontology

lemma *nodes-dataform-isa*: *length* (*nodes dataform-isa*) = *30*
by *eval*

lemma *nodes-dataform-complete*:
list-all (λ*d*. *d* ∈ *set* (*nodes dataform-isa*)) *all-dataform*
by *eval*

lemma *roots-dataform-isa*: *roots dataform-isa* = [*Dat*]
by *eval*

lemma *acyclic-dataform-isa*: *acyclic-edges dataform-isa*
by *eval*

lemma *rooted-dataform-isa*: *rooted dataform-isa Dat*
by *eval*

lemma *implied-dataform-isa*: *implied dataform-isa* = []
by *eval*

5.3.5 Spellings across the three ontologies

definition *all-one-words* :: *string list* **where**
all-one-words =
map one-word all-success @ *map ns-one-word all-nosuccess* @
map df-one-word all-dataform

definition *all-mnemonics* :: *string list* **where**
all-mnemonics =
map mnemonic all-success @ *map ns-mnemonic all-nosuccess* @
map df-mnemonic all-dataform

lemma *length-all-one-words*: *length all-one-words = 113*
by *eval*

lemma *distinct-all-one-words*: *distinct all-one-words*
by *eval*

lemma *distinct-all-mnemonics*: *distinct all-mnemonics*
by *eval*

Lower casing is not injective on the union: Satisfiable and Saturation both give *sat*, Assumed and Assurance both give *ass*. Only the success ontology is written in lower case.

definition *lowered-collisions* :: *string list* **where**
lowered-collisions =
remdups (*filter* (λ*w*. *1* < *length* (*filter* (λ*v*. *v* = *w*) (*map* (*map lower*) *all-mnemonics*)))
(*map* (*map lower*) *all-mnemonics*))

lemma *lowered-collisions-eval*: *lowered-collisions* = [*''ass''*, *''sat''*]
by *eval*

lemma *distinct-status-values-success*: *distinct* (*map status-value all-success*)
by (*rule distinct-status-values*)

5.4 Layout

definition *sp* :: *char* **where** *sp* = *CHR 0x20*

definition *nl* :: *string* **where** *nl* = [*CHR 0x0A*]

definition *dq* :: *string* **where** *dq* = [*CHR 0x22*]

definition *unlines* :: *string list* ⇒ *string* **where**
unlines ls = *concat* (*map* (λ*l*. *l* @ *nl*) *ls*)

definition *pad* :: *nat* ⇒ *string* ⇒ *string* **where**
pad n s = *s* @ *replicate* (*n* − *length s*) *sp*

definition *lhs-col* :: *nat* **where** *lhs-col* = *24*

definition *line-width* :: *nat* **where** *line-width* = *79*

fun *pack* :: *nat* ⇒ *string* ⇒ *string* ⇒ *string list* ⇒ *string list* **where**
pack w ind cur [] = [*cur*]
| *pack w ind cur* (*a* # *as*) =
(*if length cur* + *3* + *length a* ≤ *w* − *2*
then pack w ind (*cur* @ *'' | ''* @ *a*) *as*
else (*cur* @ *'' |''*) # *pack w ind* (*ind* @ *a*) *as*)

definition *bnf-rule* :: *string* ⇒ *string list* ⇒ *string list* **where**
bnf-rule lhs alts =
(*case alts of*
[] ⇒ []
| *a* # *as* ⇒
pack line-width (*replicate* (*lhs-col* + 5) *sp*)
(*pad lhs-col lhs* @ '' ::= '' @ *a*) *as*)

definition *comment* :: *string* ⇒ *string list* **where**
comment s = ['%----'' @ *s*]

definition *banner* :: *string list* **where**
banner = [''%'' @ *replicate* 78 (*CHR 0x2D*)]

5.5 The grammar

semantic is the success ontology without the type checking and verification values, which cannot occur in an inference record.

definition *semantic* :: *success* ⇒ *bool* **where**
semantic s ⟷ *s* ∉ {*TSU*, *TCP*, *TCC*, *VSU*, *VSG*, *VSB*}

definition *inference-status-values* :: *success list* **where**
inference-status-values = *filter semantic all-success*

lemma *length-inference-status-values*: *length inference-status-values* = 47
by *eval*

5.5.1 Rules from the hierarchy

The left hand side of a rule is the OneWord in angle brackets, the first alternative the OneWord as a terminal, the remaining alternatives the children. A leaf gets <*X*> ::= *X*. *override* supplies the alternatives of Assumed, which takes arguments.

type-synonym *production* = *string* × *string list*

definition *node-ref* :: ('*a*::*equal* ⇒ *string*) ⇒ '*a* ⇒ *string* **where**
node-ref word x = ''<'' @ *word x* @ ''>''

definition *rules-of* ::
('*a*::*equal* ⇒ *string*) ⇒ ('*a* ⇒ *string list option*) ⇒ ('*a* × '*a*) *list* ⇒ '*a list*
⇒ *production list* **where**
rules-of word override E xs =
map (λ*x*. (*node-ref word x*,
case override x of
Some alts ⇒ *alts*
| *None* ⇒ *word x* # *map* (*node-ref word*) (*children E x*)))
xs

definition *bnf-lines-of* :: *production list* ⇒ *string list* **where**
bnf-lines-of rs = *concat* (*map* (λ*r*. *bnf-rule* (*fst r*) (*snd r*)) *rs*)

definition *no-override* :: *'a* ⇒ *string list option* **where**
no-override x = *None*

definition *assumed-override* :: *nosuccess* ⇒ *string list option* **where**
assumed-override s =
(*if s* = *ASS then Some* [*''Assumed(<Unknown>,<Success>)''*] *else None*)

definition *success-rules* :: *production list* **where**
success-rules = *rules-of one-word no-override bnf-edges all-success*

definition *nosuccess-rules* :: *production list* **where**
nosuccess-rules = *rules-of ns-one-word assumed-override nosuccess-isa all-nosuccess*

definition *dataform-rules* :: *production list* **where**
dataform-rules = *rules-of df-one-word no-override dataform-isa all-dataform*

5.5.2 The argument of Assumed

definition *unknown-values* :: *nosuccess list* **where**
unknown-values =
filter (λ*s*. *s* = *UNK* ∨ *s* ∈ *set* (*descendants nosuccess-isa UNK*)) *all-nosuccess*

lemma *length-unknown-values*: *length unknown-values* = *24*
by *eval*

lemma *ASS-not-unknown*: *ASS* ∉ *set unknown-values*
by *eval*

5.5.3 The document

definition *section-comment* :: *string* ⇒ *string list* **where**
section-comment s = [] # *comment s*

definition *szs-rule* :: *production* **where**
szs-rule = (*''<SZS>''*, [*''<Success>''*, *''<NoSuccess>''*, *''<Data>''*])

definition *inference-status-rule* :: *production* **where**
inference-status-rule =
(*''<inference-status-value>''*, *map status-value inference-status-values*)

definition *all-rules* :: *production list* **where**
all-rules =
szs-rule # *success-rules* @ *nosuccess-rules* @ *dataform-rules* @
[*inference-status-rule*]

definition *szs-bnf-lines* :: *string list* **where**

```
  szs-bnf-lines =
    banner @
    comment ''The SZS ontologies, generated from SZS-BNF.thy'' @
    section-comment ''A status value, as used in a % SZS status line'' @
    bnf-lines-of [szs-rule] @
    section-comment ''The success ontology'' @
    bnf-lines-of success-rules @
    section-comment ''The no−success ontology'' @
    bnf-lines-of nosuccess-rules @
    section-comment ''The dataform ontology, as used in a % SZS output line'' @
    bnf-lines-of dataform-rules @
     section-comment ''A success ontology value inside a TPTP inference record''
@
    bnf-lines-of [inference-status-rule] @
    banner
```

5.5.4 The rule set is well formed

```
definition nonterminal-alts :: production list ⇒ string list where
  nonterminal-alts rs =
    remdups (filter (λa. a ≠ [] ∧ hd a = CHR 0x3C) (concat (map snd rs)))

definition terminal-discipline ::
  ('a::equal ⇒ string) ⇒ 'a list ⇒ production list ⇒ bool where
  terminal-discipline word xs rs =
    list-all (λx. list-all (λr. word x ∈ set (snd r) ⟶ fst r = node-ref word x) rs)
          xs

lemma distinct-rule-names: distinct (map fst all-rules)
  by eval

lemma count-all-rules: length all-rules = 115
  by eval

lemma count-success-rules: length success-rules = 53
  by eval

lemma count-nosuccess-rules: length nosuccess-rules = 30
  by eval

lemma count-dataform-rules: length dataform-rules = 30
  by eval

lemma terminal-discipline-success: terminal-discipline one-word all-success all-rules
  by eval

lemma terminal-discipline-nosuccess: terminal-discipline ns-one-word all-nosuccess
all-rules
  by eval
```

lemma *terminal-discipline-dataform*: *terminal-discipline df-one-word all-dataform all-rules*
by *eval*

lemma *alts-have-rules*:
list-all (λ*a*. *a* ∈ *set* (*map fst all-rules*)) (*nonterminal-alts all-rules*)
by *eval*

lemma *assumed-arguments-have-rules*:
''<Unknown>'' ∈ *set* (*map fst all-rules*) ∧ *''<Success>''* ∈ *set* (*map fst all-rules*)
by *eval*

5.5.5 What the grammar generates

Every value is generated. The nonterminals are the OneWords, which are distinct across the ontologies by *distinct-all-one-words*.

lemma *generates-success*: *generates bnf-edges SUC all-success*
by *eval*

lemma *generates-nosuccess*: *generates nosuccess-isa NOS all-nosuccess*
by *eval*

lemma *generates-dataform*: *generates dataform-isa Dat all-dataform*
by *eval*

A value with two parents is generated by two rules, so the grammar is ambiguous: Tautology is below Equivalent, TautologousConclusion and FiniteTautology. The counts are recorded; pruning would discard isa edges.

definition *ambiguous-success* :: *success list* **where**
ambiguous-success = *ambiguous bnf-edges SUC all-success*

definition *ambiguous-nosuccess* :: *nosuccess list* **where**
ambiguous-nosuccess = *ambiguous nosuccess-isa NOS all-nosuccess*

definition *ambiguous-dataform* :: *dataform list* **where**
ambiguous-dataform = *ambiguous dataform-isa Dat all-dataform*

lemma *length-ambiguous-success*: *length ambiguous-success* = *35*
by *eval*

lemma *length-ambiguous-nosuccess*: *length ambiguous-nosuccess* = *7*
by *eval*

lemma *length-ambiguous-dataform*: *length ambiguous-dataform* = *8*
by *eval*

lemma *derivations-TAU*: *derivations bnf-edges SUC TAU* = *20*

by *eval*

lemma *derivations-UNS*: *derivations bnf-edges SUC UNS = 20*
by *eval*

Fifteen success values are leaves; the other thirty-eight get a rule.

5.6 Export

The document is exported from the generated SML, where a *String.literal* is a native string. A hyperlinked rendering of the BNF is at https://szs.tptp.org/bnf.

definition *szs-ontology-bnf* :: *String.literal* **where**
szs-ontology-bnf = *String.implode* (*unlines szs-bnf-lines*)

code-reflect *SZS-Code*
functions *szs-ontology-bnf*

ML ‹*writeln SZS-Code.szs-ontology-bnf*›

@{*szs-bnf*} typesets the grammar; the appendix of the document uses it.

ML ‹
val - =
Theory.setup
(*Document-Output.antiquotation-raw* ***binding*** ‹*szs-bnf*› (*Scan.succeed* ())
(*fn - => fn () =>*
Latex.string
({*scriptsize**begin*{*verbatim*}*n* ^*SZS-Code.szs-ontology-bnf* ^*end*{*verbatim*}})))
›

The document is a session export; nothing is written to the file system. `isabelle export -d . -x '*:SZSOntology.bnf' SZS_Ontologies` writes it to `export/`.

ML ‹
fun export name s =
Export.export ***theory*** (*Path.binding0* (*Path.basic name*)) (*XML.blob* [*s*]);

val - = export SZSOntology.bnf SZS-Code.szs-ontology-bnf;
›

writeln shows the document with its line breaks.

end

theory *SZS-TPTP*
imports *SZS-BNF*
begin

6 TPTP problems

The status conditions as first-order formulas over *a*, *c* and *fin*, and the derived edges as TPTP problems. *cond-of-holds* relates the formula of a value to *holds*, and *edge-problem-isa* the problem of an edge to *isa*. The printers are unverified.

6.1 Formulas

datatype *pred* = *PA* | *PC* | *PFin*

datatype *fo* =
  Top
| *Atom pred nat*
| *Neg fo*
| *Conj fo fo*
| *Disj fo fo*
| *Imp fo fo*
| *Iff fo fo*
| *All fo*
| *Ex fo*

Binders are de Bruijn indices over one sort.

primrec *sel* :: *pred* ⇒ *'i set* ⇒ *'i set* ⇒ *'i set* ⇒ *'i set* **where**
  sel PA A C Fin = *A*
| *sel PC A C Fin* = *C*
| *sel PFin A C Fin* = *Fin*

primrec *fo-sem* :: *'i set* ⇒ *'i set* ⇒ *'i set* ⇒ *'i list* ⇒ *fo* ⇒ *bool* **where**
  fo-sem A C Fin env Top = *True*
| *fo-sem A C Fin env* (*Atom p n*) = (*env* ! *n* ∈ *sel p A C Fin*)
| *fo-sem A C Fin env* (*Neg φ*) = (¬ *fo-sem A C Fin env φ*)
| *fo-sem A C Fin env* (*Conj φ ψ*) = (*fo-sem A C Fin env φ* ∧ *fo-sem A C Fin env ψ*)
| *fo-sem A C Fin env* (*Disj φ ψ*) = (*fo-sem A C Fin env φ* ∨ *fo-sem A C Fin env ψ*)
| *fo-sem A C Fin env* (*Imp φ ψ*) = (*fo-sem A C Fin env φ* ⟶ *fo-sem A C Fin env ψ*)
| *fo-sem A C Fin env* (*Iff φ ψ*) = (*fo-sem A C Fin env φ* ⟷ *fo-sem A C Fin env ψ*)
| *fo-sem A C Fin env* (*All φ*) = (∀ *x*. *fo-sem A C Fin* (*x* # *env*) *φ*)
| *fo-sem A C Fin env* (*Ex φ*) = (∃ *x*. *fo-sem A C Fin* (*x* # *env*) *φ*)

abbreviation *a* :: *fo* **where** *a* ≡ *Atom PA 0*
abbreviation *c* :: *fo* **where** *c* ≡ *Atom PC 0*
abbreviation *fin* :: *fo* **where** *fin* ≡ *Atom PFin 0*

6.2 The status conditions

One closed formula per value, transcribed from *holds*: $X = \{\}$ is *Neg* (*Ex* x), $X \neq \{\}$ is *Ex* x, $X =$ *UNIV* is *All* x, $X \neq$ *UNIV* is *Ex* (*Neg* x), and $-C$ is *Neg* c. Values without model theoretic content get *Top*.

primrec *cond-of* :: *success* ⇒ *fo* **where**
cond-of SUC = *Top*
| *cond-of SSU* = *Top*
| *cond-of UNP* = *Imp* (*Neg* (*Ex a*)) (*Neg* (*Ex c*))
| *cond-of SAP* = *Imp* (*Ex a*) (*Ex c*)
| *cond-of TAP* = *Imp* (*All a*) (*All c*)
| *cond-of ESA* = *Iff* (*Neg* (*Ex a*)) (*Neg* (*Ex c*))
| *cond-of ETA* = *Iff* (*All a*) (*All c*)
| *cond-of SAT* = *Conj* (*Ex a*) (*Ex* (*Conj a c*))
| *cond-of FSA* = *Conj* (*Ex* (*Conj a fin*)) (*Ex* (*Conj* (*Conj a c*) *fin*))
| *cond-of THM* = *All* (*Imp a c*)
| *cond-of FTH* = *All* (*Imp* (*Conj a fin*) *c*)
| *cond-of STH* = *Conj* (*Ex a*) (*All* (*Imp a c*))
| *cond-of EQV* = *Conj* (*Ex a*) (*All* (*Iff a c*))
| *cond-of TAC* = *Conj* (*Ex a*) (*All c*)
| *cond-of WEC* = *Conj* (*Conj* (*Ex a*) (*All* (*Imp a c*))) (*Ex* (*Conj c* (*Neg a*)))
| *cond-of ETH* = *Conj* (*Conj* (*Ex a*) (*Ex* (*Neg a*))) (*All* (*Iff a c*))
| *cond-of TAU* = *Conj* (*All a*) (*All c*)
| *cond-of FTT* = *Conj* (*All* (*Imp fin a*)) (*All* (*Imp fin c*))
| *cond-of WTC* = *Conj* (*Conj* (*Ex a*) (*Ex* (*Neg a*))) (*All c*)
| *cond-of WTH* = *Conj* (*Conj* (*Conj* (*Ex a*) (*All* (*Imp a c*))) (*Ex* (*Conj c* (*Neg a*)))) (*Ex* (*Neg c*))
| *cond-of MEX* = *Conj* (*Conj* (*Ex a*) (*Ex c*)) (*All* (*Imp c a*))
| *cond-of NOC* = *Conj* (*Ex* (*Conj a c*)) (*Ex* (*Conj a* (*Neg c*)))
| *cond-of CAX* = *Neg* (*Ex a*)
| *cond-of SCA* = *Conj* (*Neg* (*Ex a*)) (*Ex c*)
| *cond-of TCA* = *Conj* (*Neg* (*Ex a*)) (*All c*)
| *cond-of WCA* = *Conj* (*Conj* (*Neg* (*Ex a*)) (*Ex c*)) (*Ex* (*Neg c*))
| *cond-of CUP* = *Imp* (*Neg* (*Ex a*)) (*Neg* (*Ex* (*Neg c*)))
| *cond-of CSP* = *Imp* (*Ex a*) (*Ex* (*Neg c*))
| *cond-of CTP* = *Imp* (*All a*) (*All* (*Neg c*))
| *cond-of ECS* = *Iff* (*Neg* (*Ex a*)) (*Neg* (*Ex* (*Neg c*)))
| *cond-of ECA* = *Iff* (*All a*) (*All* (*Neg c*))
| *cond-of CSA* = *Conj* (*Ex a*) (*Ex* (*Conj a* (*Neg c*)))
| *cond-of FCS* = *Conj* (*Ex* (*Conj a fin*)) (*Ex* (*Conj* (*Conj a* (*Neg c*)) *fin*))
| *cond-of CTH* = *All* (*Imp a* (*Neg c*))
| *cond-of FCT* = *All* (*Imp* (*Conj a fin*) (*Neg c*))
| *cond-of SCT* = *Conj* (*Ex a*) (*All* (*Imp a* (*Neg c*)))
| *cond-of CEQ* = *Conj* (*Ex a*) (*All* (*Iff a* (*Neg c*)))
| *cond-of UNC* = *Conj* (*Ex a*) (*All* (*Neg c*))
| *cond-of WCC* = *Conj* (*Conj* (*Ex a*) (*All* (*Imp a* (*Neg c*)))) (*Ex* (*Conj* (*Neg c*) (*Neg a*)))
| *cond-of ECT* = *Conj* (*Conj* (*Ex a*) (*Ex* (*Neg a*))) (*All* (*Iff a* (*Neg c*)))
| *cond-of UNS* = *Conj* (*All a*) (*All* (*Neg c*))

| *cond-of FUN* = *Conj* (*All* (*Imp fin a*)) (*All* (*Imp fin* (*Neg c*)))
| *cond-of WUC* = *Conj* (*Conj* (*Ex a*) (*Ex* (*Neg a*))) (*All* (*Neg c*))
| *cond-of WCT* = *Conj* (*Conj* (*Conj* (*Ex a*) (*All* (*Imp a* (*Neg c*)))) (*Ex* (*Conj* (*Neg c*) (*Neg a*)))) (*Ex c*)
| *cond-of CMX* = *Conj* (*Conj* (*Ex a*) (*Ex* (*Neg c*))) (*All* (*Imp* (*Neg c*) *a*))
| *cond-of SCC* = *Conj* (*Neg* (*Ex a*)) (*Ex* (*Neg c*))
| *cond-of UCA* = *Conj* (*Neg* (*Ex a*)) (*All* (*Neg c*))
| *cond-of TSU* = *Top*
| *cond-of TCP* = *Top*
| *cond-of TCC* = *Top*
| *cond-of VSU* = *Top*
| *cond-of VSG* = *Top*
| *cond-of VSB* = *Top*

context *szs-base*
begin

theorem *cond-of-holds*: *fo-sem A C Fin* [] (*cond-of s*) = *holds s A C*
  by (*cases s*; *simp add*: *set-eq-iff subset-iff del*: *all-not-in-conv*; *blast?*)

end

6.2.1 Mirror

fo-mirror exchanges *c* and *Neg c*. NOC and WCA are self-mirror with a *c* conjunct; their images have the conjuncts swapped.

fun *fo-mirror* :: *fo* ⇒ *fo* **where**
  fo-mirror (*Atom PC n*) = *Neg* (*Atom PC n*)
| *fo-mirror* (*Neg* (*Atom PC n*)) = *Atom PC n*
| *fo-mirror* (*Neg* φ) = *Neg* (*fo-mirror* φ)
| *fo-mirror* (*Conj* φ ψ) = *Conj* (*fo-mirror* φ) (*fo-mirror* ψ)
| *fo-mirror* (*Disj* φ ψ) = *Disj* (*fo-mirror* φ) (*fo-mirror* ψ)
| *fo-mirror* (*Imp* φ ψ) = *Imp* (*fo-mirror* φ) (*fo-mirror* ψ)
| *fo-mirror* (*Iff* φ ψ) = *Iff* (*fo-mirror* φ) (*fo-mirror* ψ)
| *fo-mirror* (*All* φ) = *All* (*fo-mirror* φ)
| *fo-mirror* (*Ex* φ) = *Ex* (*fo-mirror* φ)
| *fo-mirror* φ = φ

lemma *fo-sem-mirror*: *fo-sem A C Fin env* (*fo-mirror* φ) = *fo-sem A* (− *C*) *Fin env* φ
  by (*induction* φ *arbitrary*: *env rule*: *fo-mirror.induct*) *simp-all*

lemma *cond-of-mirror*:
  s ≠ *NOC* ⟹ *s* ≠ *WCA* ⟹ *cond-of* (*mirror s*) = *fo-mirror* (*cond-of s*)
  by (*cases s*) *simp-all*

6.3 Edges as problems

definition *edge-problem* :: *success* ⇒ *success* ⇒ *fo* **where**

edge-problem s t = Imp (cond-of s) (cond-of t)

definition *fin-nonempty* :: *fo* **where**
fin-nonempty = Ex fin

context *szs-base*
begin

theorem *edge-problem-isa*:
(∀ *A C*. *fo-sem A C Fin* [] (*edge-problem s t*)) ⟷ *isa s t*
by (*simp add*: *edge-problem-def isa-def cond-of-holds*)

lemma *fin-nonempty-sem*: *fo-sem A C Fin* [] *fin-nonempty* ⟷ *Fin* ≠ {}
by (*auto simp*: *fin-nonempty-def*)

The direction of *isa-iff-chk* the problems need, under the one assumption they use.

lemma *realizable-shape-of-nonempty*:
assumes *Fin* ≠ {} **shows** *realizable* (*shape-of A C*)
proof −
obtain *x* **where** *x*: *x* ∈ *Fin* **using** *assms* **by** *blast*
obtain *r* **where** *x* ∈ *reg r A C* **using** *reg-cover* **by** *metis*
with *x* **have** *reg r A C* ∩ *Fin* ≠ {} **by** *blast*
then have *shape-of A C r = FFin* **by** *simp*
then show *?thesis* **by** (*auto simp*: *realizable-iff*)
qed

theorem *isa-chk-isa*: *Fin* ≠ {} ⟹ *isa-chk s t* ⟹ *isa s t*
unfolding *isa-def*
proof (*intro allI impI*)
fix *A C* **assume** *Fin* ≠ {} **and** *isa-chk s t* **and** *holds s A C*
then have *holds-shape s* (*shape-of A C*) **by** (*simp add*: *holds-shape-of*)
with ‹*isa-chk s t*› *realizable-shape-of-nonempty* [*OF* ‹*Fin* ≠ {}›]
have *holds-shape t* (*shape-of A C*) **unfolding** *isa-chk-iff* **by** *blast*
then show *holds t A C* **by** (*simp add*: *holds-shape-of*)
qed

theorem *covers-valid*:
(*s*, *t*) ∈ *set covers* ⟹ *Fin* ≠ {} ⟹ *fo-sem A C Fin* [] (*edge-problem s t*)
using *covers-subset entails-isa-chk isa-chk-isa edge-problem-isa* **by** *blast*

end

6.3.1 Which edges need the assumption

isa-chk-base checks all shapes with a nonempty universe, so it decides *isa* for every *Fin*. Two edges fail it.

definition *inhabited* :: *shape* ⇒ *bool* **where**

inhabited $\sigma \longleftrightarrow$ *list-ex* ($\lambda r.\ \sigma\ r \neq$ *FEmpty*) *all-regions*

definition *isa-chk-base* :: *success* $\Rightarrow$ *success* $\Rightarrow$ *bool* **where**
isa-chk-base s t $\equiv$
list-all ($\lambda(a,\ b,\ c,\ d)$.
let σ = *mk-shape a b c d*
in inhabited $\sigma \longrightarrow$ *holds-shape s* $\sigma \longrightarrow$ *holds-shape t* σ)
quads

lemma *isa-chk-base-iff*:
isa-chk-base s t $\longleftrightarrow$ ($\forall\sigma$. *inhabited* $\sigma \longrightarrow$ *holds-shape s* $\sigma \longrightarrow$ *holds-shape t* σ)
proof
assume $*$: *isa-chk-base s t*
show $\forall\sigma$. *inhabited* $\sigma \longrightarrow$ *holds-shape s* $\sigma \longrightarrow$ *holds-shape t* σ
proof (*intro allI impI*)
fix σ **assume** *inhabited* σ **and** *holds-shape s* σ
have (σ *AC*, σ *AnC*, σ *nAC*, σ *nAnC*) $\in$ *set quads* **by** (*rule quads-complete*)
with $*$ **have** *inhabited* $\sigma \longrightarrow$ *holds-shape s* $\sigma \longrightarrow$ *holds-shape t* σ
unfolding *isa-chk-base-def Let-def*
by (*metis* (*mono-tags*, *lifting*) *Ball-set-list-all case-prodD shape-expand*)
with ‹*inhabited* σ› ‹*holds-shape s* σ› **show** *holds-shape t* σ **by** *blast*
qed
next
assume $\forall\sigma$. *inhabited* $\sigma \longrightarrow$ *holds-shape s* $\sigma \longrightarrow$ *holds-shape t* σ
then show *isa-chk-base s t*
unfolding *isa-chk-base-def Let-def* **by** (*simp add*: *list-all-iff split-beta*)
qed

context *szs-base*
begin

lemma *inhabited-shape-of*: *inhabited* (*shape-of A C*)
proof −
obtain *r* **where** (*undefined* :: *'i*) $\in$ *reg r A C* **using** *reg-cover* **by** *metis*
then have *shape-of A C r* $\neq$ *FEmpty* **by** *auto*
then show *?thesis* **by** (*auto simp*: *inhabited-def list-ex-iff*)
qed

theorem *isa-chk-base-isa*: *isa-chk-base s t* $\Longrightarrow$ *isa s t*
unfolding *isa-def*
proof (*intro allI impI*)
fix *A C* **assume** *isa-chk-base s t* **and** *holds s A C*
then have *holds-shape s* (*shape-of A C*) **by** (*simp add*: *holds-shape-of*)
with ‹*isa-chk-base s t*› *inhabited-shape-of*
have *holds-shape t* (*shape-of A C*) **unfolding** *isa-chk-base-iff* **by** *blast*
then show *holds t A C* **by** (*simp add*: *holds-shape-of*)
qed

end

lemma *covers-needing-fin*:
filter (λ(*s*, *t*). ¬ *isa-chk-base s t*) *covers* = [(*FTT*, *FSA*), (*FUN*, *FCS*)]
by *eval*

6.4 Printing

Variables are *I1*, *I2*, ... by binder depth, as in the KRS problems of the TPTP library. Binary formulas are parenthesised.

fun *nat-string* :: *nat* ⇒ *string* **where**
nat-string n =
(*if n* < *10 then* [*char-of* (*48* + *n*)]
else nat-string (*n div 10*) @ [*char-of* (*48* + *n mod 10*)])

definition *var-name* :: *nat* ⇒ *string* **where**
var-name d = ″*I*″ @ *nat-string d*

fun *print-fo* :: (*pred* ⇒ *string* ⇒ *string*) ⇒ *string* ⇒ *nat* ⇒ *fo* ⇒ *string* **where**
print-fo app srt d Top = ″$*true*″
| *print-fo app srt d* (*Atom p n*) = *app p* (*var-name* (*d* − *n*))
| *print-fo app srt d* (*Neg* φ) = ″~ ″ @ *print-fo app srt d* φ
| *print-fo app srt d* (*Conj* φ ψ) =
″(″ @ *print-fo app srt d* φ @ ″ & ″ @ *print-fo app srt d* ψ @ ″)″
| *print-fo app srt d* (*Disj* φ ψ) =
″(″ @ *print-fo app srt d* φ @ ″ | ″ @ *print-fo app srt d* ψ @ ″)″
| *print-fo app srt d* (*Imp* φ ψ) =
″(″ @ *print-fo app srt d* φ @ ″ => ″ @ *print-fo app srt d* ψ @ ″)″
| *print-fo app srt d* (*Iff* φ ψ) =
″(″ @ *print-fo app srt d* φ @ ″ <=> ″ @ *print-fo app srt d* ψ @ ″)″
| *print-fo app srt d* (*All* φ) =
″(! [″ @ *var-name* (*d* + *1*) @ ″: ″ @ *srt* @ ″] : ″ @ *print-fo app srt* (*d* + *1*) φ @ ″)″
| *print-fo app srt d* (*Ex* φ) =
″(? [″ @ *var-name* (*d* + *1*) @ ″: ″ @ *srt* @ ″] : ″ @ *print-fo app srt* (*d* + *1*) φ @ ″)″

definition *tf0-app* :: *pred* ⇒ *string* ⇒ *string* **where**
tf0-app p v = (*case p of PA* ⇒ ″*a*″ | *PC* ⇒ ″*c*″ | *PFin* ⇒ ″*fin*″) @ ″(″ @ *v* @ ″)″

definition *th0-app* :: *pred* ⇒ *string* ⇒ *string* **where**
th0-app p v = ″(″ @ (*case p of PA* ⇒ ″*A*″ | *PC* ⇒ ″*C*″ | *PFin* ⇒ ″*fin*″) @ ″ @ ″ @ *v* @ ″)″

definition *tf0* :: *fo* ⇒ *string* **where**
tf0 = *print-fo tf0-app* ″*interp*″ *0*

definition *th0* :: *fo* ⇒ *string* **where**
th0 = *print-fo th0-app* ″$*i*″ *0*

6.4.1 Problem files

definition *problem-name* :: *success* ⇒ *success* ⇒ *string* **where**
problem-name s t = *''SZS-''* @ *mnemonic s* @ *''-isa-''* @ *mnemonic t*

definition *formula-name* :: *success* ⇒ *success* ⇒ *string* **where**
formula-name s t = *''isa-''* @ *status-value s* @ *''-''* @ *status-value t*

definition *hrule* :: *string* **where**
hrule = *''%''* @ *replicate 78* (*CHR ''−''*)

definition *header* :: *string* ⇒ *success* ⇒ *success* ⇒ *string list* **where**
header lang s t =
[*hrule*,
''% File : '' @ *problem-name s t* @ *'' : SZS Ontologies v1.0.0.''*,
''% Domain : Knowledge Representation'',
''% Problem : '' @ *mnemonic s* @ *'' isa ''* @ *mnemonic t*,
''% Version : Especial.'',
''% English : '' @ *one-word s* @ *'' isa ''* @ *one-word t* @ *'' in the SZS success ontology.''*,
''% Refs : [Sut08] Sutcliffe (2008), The SZS Ontologies for Automated Rea'',
''% : [SS26] Schuster & Sutcliffe (2026), The SZS Ontologies.'',
''% Source : [SS26]'',
''% Names :'',
''% Status : Theorem'',
''% Rating : ?'',
''% SPC : '' @ *lang* @ *''-THM-NEQ-NAR''*,
''% Comments : Generated by SZS-Export of the AFP entry SZS-Ontologies from the'',
''% cover relation of SZS-Hierarchy. The conjecture is the formula'',
''% edge-problem '' @ *mnemonic s* @ *'' ''* @ *mnemonic t* @ *'' of SZS-TPTP, which by edge-problem-isa''*,
''% and cond-of-holds states isa '' @ *mnemonic s* @ *'' ''* @ *mnemonic t* @ *''; the axiom is the''*,
''% assumption of covers-valid.'',
hrule]

definition *tf0-problem* :: *success* ⇒ *success* ⇒ *string* **where**
tf0-problem s t = *unlines* (*header ''TF0'' s t* @
[*''tff(interp-type,type,''*,
'' interp: $*tType*).''*,
'''',
''tff(a-type,type,'',
'' a: *interp* > $*o*).''*,
'''',
''tff(c-type,type,'',
'' c: *interp* > $*o*).''*,
'''',
''tff(fin-type,type,'',

```
    ''    fin: interp > $o ).'',
    '''',
    ''%----There is a finite interpretation'',
    ''tff(fin-nonempty,axiom,'',
    ''    '' @ tf0 fin-nonempty @ '' ).'',
    '''',
    ''%----'' @ one-word s @ '' isa '' @ one-word t,
    ''tff('' @ formula-name s t @ '',conjecture,'',
    ''    '' @ tf0 (edge-problem s t) @ '' ).'',
    '''',
    hrule])

definition th0-problem :: success ⇒ success ⇒ string where
  th0-problem s t = unlines (header ''TH0'' s t @
    [''thf(fin-type,type,'',
    ''    fin: $i > $o ).'',
    '''',
    ''%----There is a finite interpretation'',
    ''thf(fin-nonempty,axiom,'',
    ''    '' @ th0 fin-nonempty @ '' ).'',
    '''',
    ''%----'' @ one-word s @ '' isa '' @ one-word t,
    ''thf('' @ formula-name s t @ '',conjecture,'',
    ''    ! [A: $i > $o,C: $i > $o] :'',
    ''      '' @ th0 (edge-problem s t) @ '' ).'',
    '''',
    hrule])

definition tf0-problems :: (String.literal × String.literal) list where
  tf0-problems =
    [(String.implode (''TF0/'' @ problem-name s t @ ''.p''), String.implode (tf0-problem
s t)).
      (s, t) ← covers]

definition th0-problems :: (String.literal × String.literal) list where
  th0-problems =
    [(String.implode (''TH0/'' @ problem-name s t @ ''.p''), String.implode (th0-problem
s t)).
      (s, t) ← covers]

lemma length-tf0-problems: length tf0-problems = 76
  by eval

end


theory SZS-Relations
  imports SZS-BNF
begin
```

7 Relations

The five relations of SZS_Semantics, decided by the checks of SZS_Shapes over the model theoretic values and rendered as text for export. Pairs are ordered, exclude the diagonal, and are listed in the order of *all-success*. Every statement is relative to the *szs* locale.

7.1 The relations

datatype *relation* = *Isa* | *Nota* | *Nevera* | *Xora* | *Mighta*

definition *all-relations* :: *relation list* **where**
  all-relations = [*Isa*, *Nota*, *Nevera*, *Xora*, *Mighta*]

primrec *rel-name* :: *relation* ⇒ *string* **where**
  rel-name Isa = ″*isa*″
| *rel-name Nota* = ″*nota*″
| *rel-name Nevera* = ″*nevera*″
| *rel-name Xora* = ″*xora*″
| *rel-name Mighta* = ″*mighta*″

primrec *rel-chk* :: *relation* ⇒ *success* ⇒ *success* ⇒ *bool* **where**
  rel-chk Isa s t = *isa-chk s t*
| *rel-chk Nota s t* = *nota-chk s t*
| *rel-chk Nevera s t* = *nevera-chk s t*
| *rel-chk Xora s t* = *xora-chk s t*
| *rel-chk Mighta s t* = *mighta-chk s t*

7.2 Pair lists

definition *ordered-pairs* :: (*success* × *success*) *list* **where**
  ordered-pairs =
    filter (λ(*s*, *t*). *s* ≠ *t*)
      (*List.product model-theoretic-values model-theoretic-values*)

definition *rel-pairs* :: *relation* ⇒ (*success* × *success*) *list* **where**
  rel-pairs r = *filter* (λ(*s*, *t*). *rel-chk r s t*) *ordered-pairs*

lemma *length-ordered-pairs*: *length ordered-pairs* = *1980*
  by *eval*

lemma *rel-pairs-Isa*: *rel-pairs Isa* = *entails*
  by *eval*

lemma *rel-pairs-Nota*:
  rel-pairs Nota = *filter* (λ*p*. *p* ∉ *set* (*rel-pairs Isa*)) *ordered-pairs*
  by *eval*

lemma *rel-pairs-Mighta*:

rel-pairs Mighta = *filter* (λ*p*. *p* ∉ *set* (*rel-pairs Nevera*)) *ordered-pairs*
by *eval*

lemma *rel-pairs-Xora-Nevera*:
list-all (λ*p*. *p* ∈ *set* (*rel-pairs Nevera*)) (*rel-pairs Xora*)
by *eval*

7.2.1 Symmetry and mirror closure

lemma *rel-pairs-sym*:
list-all (λ*r*. *list-all* (λ(*s*, *t*). (*t*, *s*) ∈ *set* (*rel-pairs r*)) (*rel-pairs r*))
[*Nevera*, *Xora*, *Mighta*]
by *eval*

lemma *rel-pairs-mirror*:
list-all (λ*r*. *list-all* (λ(*s*, *t*). (*mirror s*, *mirror t*) ∈ *set* (*rel-pairs r*)) (*rel-pairs r*))
all-relations
by *eval*

7.2.2 Sizes

lemma *length-rel-pairs*:
map (λ*r*. *length* (*rel-pairs r*)) *all-relations* = [*325*, *1655*, *796*, *16*, *1184*]
by *eval*

7.3 The strongest relation of a pair

Every model theoretic value holds on some realizable shape, so *isa* and *nevera* exclude each other and the cases below are disjoint.

lemma *model-theoretic-satisfiable*:
list-all (λ*s*. *mighta-chk s s*) *model-theoretic-values*
by *eval*

fun *pairs-of* :: *'a list* ⇒ (*'a* × *'a*) *list* **where**
pairs-of [] = []
| *pairs-of* (*x* # *xs*) = *map* (*Pair x*) *xs* @ *pairs-of xs*

definition *unordered-pairs* :: (*success* × *success*) *list* **where**
unordered-pairs = *pairs-of model-theoretic-values*

lemma *length-unordered-pairs*: *length unordered-pairs* = *990*
by *eval*

definition *strongest* :: *success* ⇒ *success* ⇒ *relation* × *success* × *success* **where**
strongest s t =
(*if isa-chk s t then* (*Isa*, *s*, *t*)
else if isa-chk t s then (*Isa*, *t*, *s*)
else if xora-chk s t then (*Xora*, *s*, *t*)
else if nevera-chk s t then (*Nevera*, *s*, *t*)

else (*Mighta*, *s*, *t*))

definition *classified-pairs* :: (*relation* × *success* × *success*) *list* **where**
classified-pairs = [*strongest s t*. (*s*, *t*) ← *unordered-pairs*]

lemma *length-classified-pairs*:
map (λ*r*. *length* (*filter* (λ(*r'*, *s*, *t*). *r'* = *r*) *classified-pairs*)) [*Isa*, *Xora*, *Nevera*, *Mighta*]
= [*325*, *8*, *390*, *267*]
by *eval*

lemma *classified-pairs-Xora*:
filter (λ(*r*, *s*, *t*). *r* = *Xora*) *classified-pairs* =
[(*Xora*, *UNP*, *SCA*), (*Xora*, *SAP*, *UNC*), (*Xora*, *SAT*, *CTH*), (*Xora*, *FSA*, *FCT*),
(*Xora*, *THM*, *CSA*), (*Xora*, *FTH*, *FCS*), (*Xora*, *TAC*, *CSP*), (*Xora*, *CUP*, *SCC*)]
by *eval*

lemma *strongest-SAT-CSA*: *strongest SAT CSA* = (*Mighta*, *SAT*, *CSA*)
by *eval*

7.4 Text

definition *pair-line* :: *relation* ⇒ *success* ⇒ *success* ⇒ *string* **where**
pair-line r s t = *mnemonic s* @ *'' ''* @ *rel-name r* @ *'' ''* @ *mnemonic t*

definition *rel-header* :: *relation* ⇒ *string list* **where**
rel-header r =
[*''% ''* @ *rel-name r* @ *'' on the 45 model theoretic values of the SZS success ontology.''*,
''% Generated by SZS-Export of the AFP entry SZS-Ontologies from rel-pairs '' @ *rel-name r*,
''% of SZS-Relations: *the ordered pairs* (*s*, *t*) *with s* ~= *t and ''* @ *rel-name r* @ *''-chk s t,''*,
''% in the order of all-success. Relative to the szs locale of SZS-Semantics.'']

definition *rel-text* :: *relation* ⇒ *string* **where**
rel-text r = *unlines* (*rel-header r* @ *map* (λ(*s*, *t*). *pair-line r s t*) (*rel-pairs r*))

definition *pairs-header* :: *string list* **where**
pairs-header =
[*''% The 990 unordered pairs of model theoretic values of the SZS success ontology,''*,
''% each with the strongest relation of SZS-Semantics that holds: *isa* (*in the''*,
''% direction given), *else xora*, *else nevera*, *else mighta.''*,
''% Generated by SZS-Export of the AFP entry SZS-Ontologies from classified-pairs'',
''% of SZS-Relations. Relative to the szs locale of SZS-Semantics.'']

definition *pairs-text* :: *string* **where**
pairs-text = *unlines* (*pairs-header* @ *map* (λ(*r*, *s*, *t*). *pair-line r s t*) *classified-pairs*)

definition *relation-files* :: (*String.literal* × *String.literal*) *list* **where**
relation-files =
[(*String.implode* (''*Relations*/'' @ *rel-name r* @ ''*.txt*''), *String.implode* (*rel-text r*)).
r ← *all-relations*]
@ [(*String.implode* ''*Relations*/*pairs.txt*'', *String.implode pairs-text*)]

lemma *length-relation-files*: *length relation-files* = *6*
by *eval*

end

theory *SZS-Export*
imports *SZS-TPTP SZS-Relations*
begin

8 Export

The 76 edges of *covers* as TF0 and TH0 problems, exported under `TF0/SZS_<s>_isa_<t>.p` and `TH0/`, and the five relations of SZS_Relations with the classified pairs, exported under `Relations/<relation>.txt` and `Relations/pairs.txt`. `isabelle export -d . -x 'SZS_Ontologies.SZS_Export:T*/*' -p 1 -O Problems SZS_Ontologies` and `isabelle export -d . -x 'SZS_Ontologies.SZS_Export:Relati -p 1 -O . SZS_Ontologies` write the files. SZS_BNF exports the grammar as `SZSOntology.bnf`; its hyperlinked rendering is at https://szs.tptp.org/bnf.

code-reflect *SZS-Export-Code*
functions *tf0-problems th0-problems relation-files*

ML ‹
fun export (*name*, *s*) =
Export.export ***theory*** (*Path.binding0* (*Path.explode name*)) (*XML.blob* [*s*]);

val - = List.app export SZS-Export-Code.tf0-problems;
val - = List.app export SZS-Export-Code.th0-problems;
val - = List.app export SZS-Export-Code.relation-files;
›

A The generated BNF

The grammar generated by SZS_BNF; a hyperlinked rendering is at https://szs.tptp.org/bnf.

```
%------------------------------------------------------------------------------
%----The SZS ontologies, generated from SZS_BNF.thy

%----A status value, as used in a % SZS status line
<SZS>                    ::= <Success> | <NoSuccess> | <Data>

%----The success ontology
<Success>                ::= Success | <SemanticSuccess> | <TypeCheckSuccess> |
                             <VerifySuccess>
<SemanticSuccess>        ::= SemanticSuccess | <UnsatisfiabilityPreserving> |
                             <SatisfiabilityPreserving> |
                             <TautologyPreserving> | <FiniteTheorem> |
                             <CounterUnsatisfiabilityPreserving> |
                             <CounterSatisfiabilityPreserving> |
                             <CounterTautologyPreserving> |
                             <FiniteCounterTheorem>
<UnsatisfiabilityPreserving> ::= UnsatisfiabilityPreserving |
                             <EquiSatisfiable> | <CounterSatisfiable>
<SatisfiabilityPreserving> ::= SatisfiabilityPreserving | <EquiSatisfiable> |
                             <Theorem>
<TautologyPreserving>    ::= TautologyPreserving | <EquiTautologous> |
                             <Theorem>
<EquiSatisfiable>        ::= EquiSatisfiable | <Satisfiable> |
                             <EquivalentCounterTheorem> |
                             <WeakerCounterTheorem> |
                             <UnsatisfiableConclusionContradictoryAxioms>
<EquiTautologous>        ::= EquiTautologous | <Equivalent> | <WeakerTheorem> |
                             <WeakerCounterConclusion> |
                             <EquivalentCounterTheorem> |
                             <SatisfiableCounterConclusionContradictoryAxioms>
<Satisfiable>            ::= Satisfiable | <FinitelySatisfiable> |
                             <SatisfiableAxiomsTheorem> | <ModelExtending> |
                             <NoConsequence>
<FinitelySatisfiable>    ::= FinitelySatisfiable | <FiniteTautology>
<Theorem>                ::= Theorem | <SatisfiableAxiomsTheorem> |
                             <ContradictoryAxioms>
<FiniteTheorem>          ::= FiniteTheorem | <Theorem> | <FiniteTautology>
<SatisfiableAxiomsTheorem> ::= SatisfiableAxiomsTheorem | <Equivalent> |
                             <TautologousConclusion> | <WeakerConclusion>
<Equivalent>             ::= Equivalent | <EquivalentTheorem> | <Tautology>
<TautologousConclusion>  ::= TautologousConclusion | <Tautology> |
                             <WeakerTautologousConclusion>
<WeakerConclusion>       ::= WeakerConclusion | <WeakerTautologousConclusion> |
                             <WeakerTheorem>
<EquivalentTheorem>      ::= EquivalentTheorem
<Tautology>              ::= Tautology
<FiniteTautology>        ::= FiniteTautology | <Tautology>
<WeakerTautologousConclusion> ::= WeakerTautologousConclusion
<WeakerTheorem>          ::= WeakerTheorem
<ModelExtending>         ::= ModelExtending | <Equivalent>
<NoConsequence>          ::= NoConsequence
<ContradictoryAxioms>    ::= ContradictoryAxioms |
                             <SatisfiableConclusionContradictoryAxioms> |
                             <SatisfiableCounterConclusionContradictoryAxioms>
<SatisfiableConclusionContradictoryAxioms> ::= SatisfiableConclusionContradictoryAxioms |
```

```
                            <TautologousConclusionContradictoryAxioms> |
                            <WeakerConclusionContradictoryAxioms>
<TautologousConclusionContradictoryAxioms> ::= TautologousConclusionContradictoryAxioms
<WeakerConclusionContradictoryAxioms> ::= WeakerConclusionContradictoryAxioms
<CounterUnsatisfiabilityPreserving> ::= CounterUnsatisfiabilityPreserving |
                            <Satisfiable> | <EquiCounterSatisfiable>
<CounterSatisfiabilityPreserving> ::= CounterSatisfiabilityPreserving |
                            <EquiCounterSatisfiable> | <CounterTheorem>
<CounterTautologyPreserving> ::= CounterTautologyPreserving |
                            <EquiCounterTautologous> | <CounterTheorem>
<EquiCounterSatisfiable> ::= EquiCounterSatisfiable | <EquivalentTheorem> |
                            <WeakerTheorem> |
                            <TautologousConclusionContradictoryAxioms> |
                            <CounterSatisfiable>
<EquiCounterTautologous> ::= EquiCounterTautologous | <WeakerConclusion> |
                            <EquivalentTheorem> |
                            <SatisfiableConclusionContradictoryAxioms> |
                            <CounterEquivalent> | <WeakerCounterTheorem>
<CounterSatisfiable>     ::= CounterSatisfiable | <NoConsequence> |
                            <FinitelyCounterSatisfiable> |
                            <SatisfiableAxiomsCounterTheorem> |
                            <CounterModelExtending>
<FinitelyCounterSatisfiable> ::= FinitelyCounterSatisfiable |
                            <FinitelyUnsatisfiable>
<CounterTheorem>         ::= CounterTheorem | <ContradictoryAxioms> |
                            <SatisfiableAxiomsCounterTheorem>
<FiniteCounterTheorem>   ::= FiniteCounterTheorem | <CounterTheorem> |
                            <FinitelyUnsatisfiable>
<SatisfiableAxiomsCounterTheorem> ::= SatisfiableAxiomsCounterTheorem |
                            <CounterEquivalent> | <UnsatisfiableConclusion> |
                            <WeakerCounterConclusion>
<CounterEquivalent>      ::= CounterEquivalent | <EquivalentCounterTheorem> |
                            <Unsatisfiable>
<UnsatisfiableConclusion> ::= UnsatisfiableConclusion | <Unsatisfiable> |
                            <WeakerUnsatisfiableConclusion>
<WeakerCounterConclusion> ::= WeakerCounterConclusion |
                            <WeakerUnsatisfiableConclusion> |
                            <WeakerCounterTheorem>
<EquivalentCounterTheorem> ::= EquivalentCounterTheorem
<Unsatisfiable>          ::= Unsatisfiable
<FinitelyUnsatisfiable>  ::= FinitelyUnsatisfiable | <Unsatisfiable>
<WeakerUnsatisfiableConclusion> ::= WeakerUnsatisfiableConclusion
<WeakerCounterTheorem>   ::= WeakerCounterTheorem
<CounterModelExtending>  ::= CounterModelExtending | <CounterEquivalent>
<SatisfiableCounterConclusionContradictoryAxioms> ::= SatisfiableCounterConclusionContradictoryAxioms |
                            <WeakerConclusionContradictoryAxioms> |
                            <UnsatisfiableConclusionContradictoryAxioms>
<UnsatisfiableConclusionContradictoryAxioms> ::= UnsatisfiableConclusionContradictoryAxioms
<TypeCheckSuccess>       ::= TypeCheckSuccess | <TypeCheckPartial>
<TypeCheckPartial>       ::= TypeCheckPartial | <TypeCheckedComplete>
<TypeCheckedComplete>    ::= TypeCheckedComplete
<VerifySuccess>          ::= VerifySuccess | <VerifiedGood> | <VerifiedBad>
<VerifiedGood>           ::= VerifiedGood
<VerifiedBad>            ::= VerifiedBad

%----The no-success ontology
<NoSuccess>              ::= NoSuccess | <Open> | <NotVerified> | <Assumed> |
                            <Unknown> | <Incorrect>
<Unknown>                ::= Unknown | <Stopped> | <InProgress> | <NotTried>
<Stopped>                ::= Stopped | <Forced> | <GaveUp>
<InProgress>             ::= InProgress
<NotTried>               ::= NotTried | <NotTriedYet> | <Inappropriate>
```

```
<NotTriedYet>            ::= NotTriedYet
<Error>                  ::= Error | <InputError> | <OSError>
<Forced>                 ::= Forced | <User> | <ResourceOut>
<GaveUp>                 ::= GaveUp | <ResourceOut> | <Incomplete> | <Error> |
                             <Inappropriate>
<OSError>                ::= OSError
<InputError>             ::= InputError | <SyntaxError> | <SemanticError> |
                             <UsageError>
<SyntaxError>            ::= SyntaxError
<SemanticError>          ::= SemanticError | <TypeError> | <Unsemantic>
<TypeError>              ::= TypeError
<Unsemantic>             ::= Unsemantic
<UsageError>             ::= UsageError
<User>                   ::= User
<ResourceOut>            ::= ResourceOut | <Timeout> | <MemoryOut> |
                             <InstructionsOut>
<Timeout>                ::= Timeout | <CPUTimeout> | <WCTimeout>
<CPUTimeout>             ::= CPUTimeout
<WCTimeout>              ::= WCTimeout
<MemoryOut>              ::= MemoryOut
<InstructionsOut>        ::= InstructionsOut
<Incomplete>             ::= Incomplete
<Inappropriate>          ::= Inappropriate
<Incorrect>              ::= Incorrect
<Assumed>                ::= Assumed(<Unknown>,<Success>)
<Open>                   ::= Open
<NotVerified>            ::= NotVerified | <FailedVerified>
<FailedVerified>         ::= FailedVerified

%----The dataform ontology, as used in a % SZS output line
<Data>                   ::= Data | <None> | <LogicalData> | <NonLogicalData>
<LogicalData>            ::= LogicalData | <NotASolution> | <Solution>
<NonLogicalData>         ::= NonLogicalData | <Comment> | <FreeText>
<NotASolution>           ::= NotASolution | <Assurance> | <IncompleteProof> |
                             <IncompleteInterpretation>
<Solution>               ::= Solution | <Proof> | <Interpretation> |
                             <ListOfFormulae>
<Assurance>              ::= Assurance
<IncompleteProof>        ::= IncompleteProof
<IncompleteInterpretation> ::= IncompleteInterpretation
<Proof>                  ::= Proof | <Refutation> | <Derivation>
<Interpretation>         ::= Interpretation | <Model> |
                             <DomainInterpretation> |
                             <HerbrandInterpretation>
<ListOfFormulae>         ::= ListOfFormulae | <HerbrandInterpretation>
<Refutation>             ::= Refutation | <CNFRefutation>
<Derivation>             ::= Derivation
<CNFRefutation>          ::= CNFRefutation
<Model>                  ::= Model | <DomainModel> | <HerbrandModel>
<DomainInterpretation>   ::= DomainInterpretation | <DomainModel> |
                             <FiniteInterpretation> |
                             <InfiniteInterpretation>
<DomainModel>            ::= DomainModel | <FiniteModel> | <InfiniteModel>
<FiniteInterpretation>   ::= FiniteInterpretation | <FiniteModel>
<FiniteModel>            ::= FiniteModel
<InfiniteInterpretation> ::= InfiniteInterpretation | <InfiniteModel>
<InfiniteModel>          ::= InfiniteModel
<HerbrandInterpretation> ::= HerbrandInterpretation | <HerbrandModel> |
                             <FormulaHerbrandInterpretation>
<HerbrandModel>          ::= HerbrandModel | <FormulaHerbrandModel>
<FormulaHerbrandInterpretation> ::= FormulaHerbrandInterpretation |
                             <FormulaHerbrandModel>
```

```
<FormulaHerbrandModel>   ::= FormulaHerbrandModel | <Saturation>
<Saturation>             ::= Saturation
<Comment>                ::= Comment
<FreeText>               ::= FreeText | <Verification>
<Verification>           ::= Verification
<None>                   ::= None

%----A success ontology value inside a TPTP inference record
<inference_status_value> ::= suc | ssu | unp | sap | tap | esa | eta | sat |
                             fsa | thm | fth | sth | eqv | tac | wec | eth |
                             tau | ftt | wtc | wth | mex | noc | cax | sca |
                             tca | wca | cup | csp | ctp | ecs | eca | csa |
                             fcs | cth | fct | sct | ceq | unc | wcc | ect |
                             uns | fun | wuc | wct | cmx | scc | uca
%--------------------------------------------------------------------------
```

end

References

[1] G. Sutcliffe. The SZS ontologies. https://szs.tptp.org. Accessed 9 November 2026.

[2] G. Sutcliffe. The SZS ontologies for automated reasoning software. In P. Rudnicki, G. Sutcliffe, B. Konev, R. A. Schmidt, and S. Schulz, editors, *Proceedings of the LPAR 2008 Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and the 7th International Workshop on the Implementation of Logics*, volume 418 of *CEUR Workshop Proceedings*, pages 38–49, 2008.

[3] G. Sutcliffe and C. Benzmüller. Automated reasoning in higher-order logic using the TPTP THF infrastructure. *Journal of Formalized Reasoning*, 3(1):1–27, 2010.

[4] G. Sutcliffe, S. Schulz, K. Claessen, and P. Baumgartner. The TPTP typed first-order form with arithmetic. In N. Bjørner and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-18)*, volume 7180 of *Lecture Notes in Computer Science*, pages 406–419. Springer, 2012.