The SZS Ontologies for Automated Reasoning Software

Geoff Sutcliffe
University of Miami

Abstract

This paper describes the SZS ontologies that provide status values for precisely describing what is known or has been established about logical data. The ontology values are useful for describing existing logical data, and for automated reasoning software to describe their input and output. Standards for presenting the ontology values are also provided.

1 Introduction

The real use of automated reasoning software - automated theorem proving (ATP) systems and other tools - is not as standalone software that a user invokes directly, but rather as embedded components of more complex reasoning systems. For one example, NASA's certifiable program synthesis system [6] embeds the SSCPA ATP system harness [20], the ATP systems E [13], SPASS [24], Vampire [12], and the GDV derivation verifier [17]. For another example, SRI's BioDeducta system [14] embeds the ATP system SNARK [15], and the BioBike integrated knowledge base and biocomputing platform [10]. In this embedded context automated reasoning software is typically treated as a black box with known processing capabilities. In order to use the software, the host system must know how to invoke the software, how to pass data into the software, and how to accept data produced by the software.

The data passed in to and out from automated reasoning software typically consists of *logical data*, e.g., formulae, derivations, interpretations, etc., and *status values* that describe what is known or has been established about the logical data, e.g., the nature of the formulae, their theoremhood or satisfiability, a reason why the software could not process the data, etc. For software that works with first-order logic, the de facto standard for expressing logical data is the TPTP language [19] (and it is expected that this will soon extend to higher-order logic [4]). The SZS ontologies that are linked to the TPTP are used by some automated reasoning software to express the status values. This paper describes the SZS ontologies and their use by automated reasoning software.

The status information output by current automated reasoning software varies widely in quantity, quality, and meaning. At the low end of the scale, for example, an ATP system might report only an assurance that the input problem's conjecture is a theorem of the axioms (the wonderful "`yes`" output). In some cases the claimed status is misleading, e.g., when a clause normal form refutation based ATP system claims that a first-order input problem consisting of axioms and a conjecture is "unsatisfiable", it typically means that the conjecture is a theorem of the axioms. At the high end of the scale, for example, a tool such as Infinox might report that a set of formulae does not have a finite model, or, for another example, a set of formulae might be tagged as representing a Herbrand interpretation. In order to seamlessly embed automated reasoning software in more complex reasoning systems, it is necessary to correctly and precisely specify status values for the input and output data. The SZS ontologies provide fine grained ontologies of status values that are suitable for this task.

The SZS *success ontology* provides status values to describe what is known or has been successfully established about the relationship between the axioms and conjecture in logical data. It is described in Section 2. The SZS *no-success ontology* provides status values to describe why a success ontology value has not been established. It is described in Section 3. The SZS *dataform ontology* provides status values

Rudnicki P, Sutcliffe G., Konev B., Schmidt R., Schulz S. (eds.);
Proceedings of the Combined KEAPPA - IWIL Workshops, pp. 38-49

to describe the nature of logical data. It is described in Section 4. All status values are expressed as "OneWord", and also have a three letter mnemonic. In addition to the ontologies themselves, standards for *presenting status values* have been specified. These are described in Section 5.

2 The SZS Success Ontology

The SZS success ontology was inspired by work done to establish communication protocols for systems on the MathWeb Software Bus [2, 25]. The ontology assumes that the logical data is a 2-tuple of the form $\langle Ax, C \rangle$, where Ax is a set (conjunction) of axioms and C is a conjecture formula. This is a common standard usage of ATP systems. If the input is not of the form $\langle Ax, C \rangle$, it is treated as a conjecture formula (even if it is a "set of axioms" from the user view point, e.g., a set of formulae all with the TPTP role `axiom`), and the 2-tuple is $\langle TRUE, C \rangle$. The success ontology values are based on the possible relationships between the sets of models of Ax and C. The ontology values can also be interpreted in terms of the formula $F \equiv Ax \Rightarrow C$. For example, the status `Theorem` means that the set of models of Ax is a (not necessarily strict) subset of the set of models of C, i.e., every model of Ax is a model of C. In this case F is valid.

Figure 1 shows the success ontology (many of the "OneWord" status values are abbreviated in the figure - see the list below for the official full "OneWord"s). The lines in the ontology can be followed up the hierarchy as *isa* links, e.g., an ETH *isa* EQV *isa* (SAT and a THM). Figure 2 shows the relationships between the model sets for some of the success ontology values. The outer grey ring contains all interpretations, the long dashed black ring contains the models of Ax, and the short dashed black ring contains the models of C.

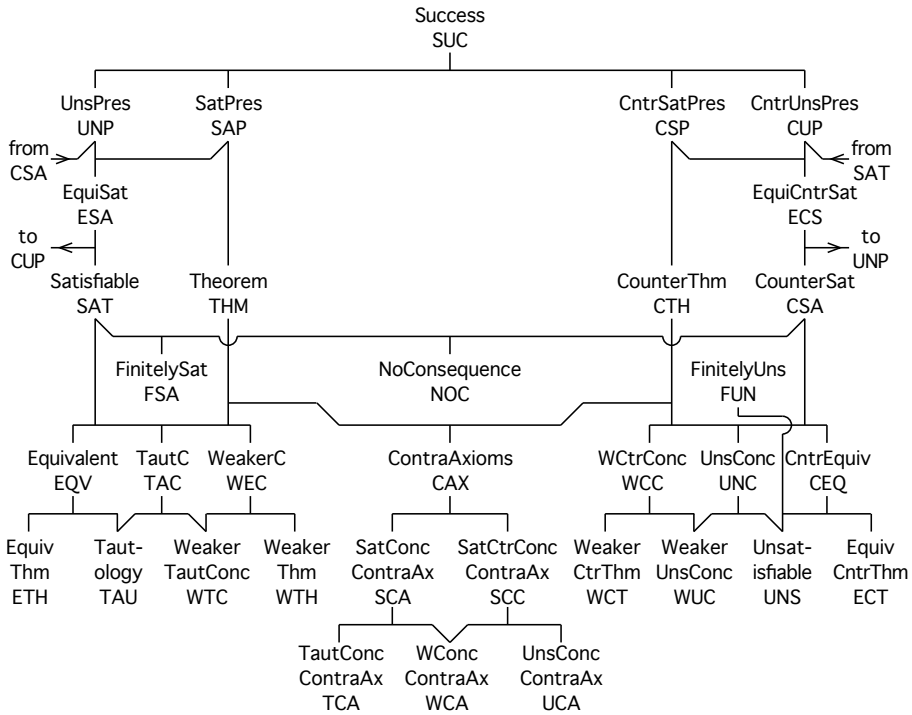

Figure 1: SZS Success Ontology

The meanings of the success ontology values are as follows. Associated with each status value are some possible dataforms that might be provided to justify the ontology value for given logical data - see Section 4.

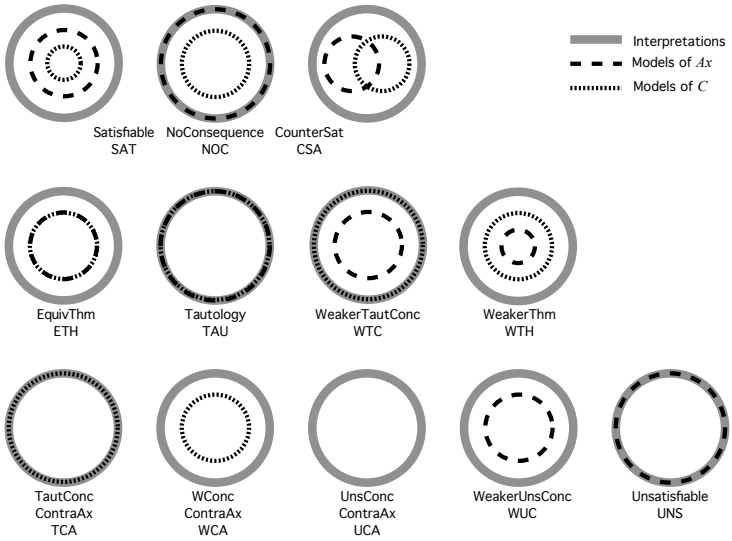

Figure 2: SZS Success Ontology Models

- Success (SUC): The logical data has been processed successfully.
- UnsatisfiabilityPreserving (UNP): If there does not exist a model of Ax then there does not exist a model of C, i.e., if Ax is unsatisfiable then C is unsatisfiable.
- SatisfiabilityPreserving (SAP): If there exists a model of Ax then there exists a model of C, i.e., if Ax is satisfiable then C is satisfiable. F is satisfiable.
- EquiSatisfiable (ESA): There exists a model of Ax iff there exists a model of C, i.e., Ax is (un)satisfiable iff C is (un)satisfiable.
- Satisfiable (SAT): Some interpretations are models of Ax, and some models of Ax are models of C. F is satisfiable, and $\neg F$ is not valid. Possible dataforms are Models of $Ax \wedge C$.
- FinitelySatisfiable (FSA): Some finite interpretations are finite models of Ax, and some finite models of Ax are finite models of C. F is satisfiable, and $\neg F$ is not valid. Possible dataforms are FiniteModels of $Ax \wedge C$.
- Theorem (THM): All models of Ax are models of C. F is valid, and C is a theorem of Ax. Possible dataforms are Proofs of C from Ax.
- Equivalent (EQV): Some interpretations are models of Ax, all models of Ax are models of C, and all models of C are models of Ax. F is valid, C is a theorem of Ax, and Ax is a theorem of C. Possible dataforms are Proofs of C from Ax and of Ax from C.
- TautologousConclusion (TAC): Some interpretations are models of Ax, and all interpretations are models of C. F is valid, and C is a tautology. Possible dataforms are Proofs of C.
- WeakerConclusion (WEC): Some interpretations are models of Ax, all models of Ax are models of C, and some models of C are not models of Ax. See Theorem and Satisfiable.
- EquivalentTheorem (ETH): Some, but not all, interpretations are models of Ax, all models of Ax are models of C, and all models of C are models of Ax. See Equivalent.
- Tautology (TAU): All interpretations are models of Ax, and all interpretations are models of C. F is valid, $\neg F$ is unsatisfiable, and C is a tautology. Possible dataforms are Proofs of Ax and of C.
- WeakerTautologousConclusion (WTC): Some, but not all, interpretations are models of Ax, and all interpretations are models of C. F is valid, and C is a tautology. See TautologousConclusion and WeakerConclusion.

- WeakerTheorem (WTH): Some interpretations are models of Ax, all models of Ax are models of C, some models of C are not models of Ax, and some interpretations are not models of C. See Theorem and Satisfiable.
- ContradictoryAxioms (CAX): No interpretations are models of Ax. F is valid, and anything is a theorem of Ax. Possible dataforms are Refutations of Ax.
- SatisfiableConclusionContradictoryAxioms (SCA): No interpretations are models of Ax, and some interpretations are models of C. See ContradictoryAxioms.
- TautologousConclusionContradictoryAxioms (TCA): No interpretations are models of Ax, and all interpretations are models of C. See TautologousConclusion and SatisfiableConclusionContradictoryAxioms.
- WeakerConclusionContradictoryAxioms (WCA): No interpretations are models of Ax, and some, but not all, interpretations are models of C. See SatisfiableConclusionContradictoryAxioms and SatisfiableCounterConclusionContradictoryAxioms.
- CounterUnsatisfiabilityPreserving (CUP): If there does not exist a model of Ax then there does not exist a model of $\neg C$, i.e., if Ax is unsatisfiable then $\neg C$ is unsatisfiable.
- CounterSatisfiabilityPreserving (CSP): If there exists a model of Ax then there exists a model of $\neg C$, i.e., if Ax is satisfiable then $\neg C$ is satisfiable.
- EquiCounterSatisfiable (ECS): There exists a model of Ax iff there exists a model of $\neg C$, i.e., Ax is (un)satisfiable iff $\neg C$ is (un)satisfiable.
- CounterSatisfiable (CSA): Some interpretations are models of Ax, and some models of Ax are models of $\neg C$. F is not valid, $\neg F$ is satisfiable, and C is not a theorem of Ax. Possible dataforms are Models of $Ax \wedge \neg C$.
- CounterTheorem (CTH): All models of Ax are models of $\neg C$. F is not valid, and $\neg C$ is a theorem of Ax. Possible dataforms are Proofs of $\neg C$ from Ax.
- CounterEquivalent (CEQ): Some interpretations are models of Ax, all models of Ax are models of $\neg C$, and all models of $\neg C$ are models of Ax. F is not valid, and $\neg C$ is a theorem of Ax. All interpretations are models of Ax xor of C. Possible dataforms are Proofs of $\neg C$ from Ax and of Ax from $\neg C$.
- UnsatisfiableConclusion (UNC): Some interpretations are models of Ax, and all interpretations are models of $\neg C$ (i.e., no interpretations are models of C). F is not valid, and $\neg C$ is a tautology. Possible dataforms are Proofs of $\neg C$.
- WeakerCounterConclusion (WCC): Some interpretations are models of Ax, and all models of Ax are models of $\neg C$, and some models of $\neg C$ are not models of Ax. See CounterTheorem and CounterSatisfiable.
- EquivalentCounterTheorem (ECT): Some, but not all, interpretations are models of Ax, all models of Ax are models of $\neg C$, and all models of $\neg C$ are models of Ax. See CounterEquivalent.
- FinitelyUnsatisfiable (FUN): All finite interpretations are finite models of Ax, and all finite interpretations are finite models of $\neg C$ (i.e., no finite interpretations are finite models of C).
- Unsatisfiable (UNS): All interpretations are models of Ax, and all interpretations are models of $\neg C$. (i.e., no interpretations are models of C). F is unsatisfiable, $\neg F$ is valid, and $\neg C$ is a tautology. Possible dataforms are Proofs of Ax and of C, and Refutations of F.
- WeakerUnsatisfiableConclusion (WUC): Some, but not all, interpretations are models of Ax, and all interpretations are models of $\neg C$. See Unsatisfiable and WeakerCounterConclusion.
- WeakerCounterTheorem (WCT): Some interpretations are models of Ax, all models of Ax are models of $\neg C$, some models of $\neg C$ are not models of Ax, and some interpretations are not models of $\neg C$. See CounterSatisfiable.

- SatisfiableCounterConclusionContradictoryAxioms (SCC): No interpretations are models of Ax, and some interpretations are models of $\neg C$. See ContradictoryAxioms.
- UnsatisfiableConclusionContradictoryAxioms (UCA): No interpretations are models of Ax, and all interpretations are models of $\neg C$ (i.e., no interpretations are models of C). See UnsatisfiableConclusion and SatisfiableCounterConclusionContradictoryAxioms.
- NoConsequence (NOC): Some interpretations are models of Ax, some models of Ax are models of C, and some models of Ax are models of $\neg C$. F is not valid, F is satisfiable, $\neg F$ is not valid, $\neg F$ is satisfiable, and C is not a theorem of Ax. Possible dataforms are pairs of models, one Model of $Ax \wedge C$ and one Model of $Ax \wedge \neg C$.

The success ontology is very fine grained, and has more status values than are commonly used by automated reasoning software, by ATP systems in particular. A suitable subset for practical uses of ATP systems is as follows:

- FOF problems with a conjecture - report Theorem or CounterSatisfiable.
- FOF problems without a conjecture - report Satisfiable or Unsatisfiable.
- CNF problems - report Satisfiable or Unsatisfiable.

2.1 Validation of the Success Ontology

Two steps have been taken towards formal validation of the success ontology. The first step was the enumeration of the possible relationships between the models of Ax and C (some of which are illustrated in Figure 2). This provided a basis for the ontology values, and a basis for the *isa* links. The second step[1] was to axiomatize the ontology and prove relevant properties. (The axiomatization implemented covers the "positive" part of the ontology regarding Ax and C, and just two commonly used values from the "negative" part regarding Ax and$\neg C$. It is expected that the results obtained will extend without difficulty to the full ontology.) The axiomatization encodes the relationship between the models of Ax and C for each ontology value, and, from that, relationships between the ontology values can be proven. Additionally, a finite model of the axioms was found, demonstrating the consistency of the axiomatization and hence the ontology.

The axiomatization is in first-order logic. As example, the axioms that describe the ESA, THM, and ETH values are given in Figure 3. Four relationships between pairs of ontology values were defined and axiomatized:

- α *isa* β, meaning that if $\langle Ax, C\rangle$ has the status α then it also has the status β. For example, WTH *isa* THM.
- α *nota* β, meaning that if $\langle Ax, C\rangle$ has the status α then it does not necessarily have the status β. For example, THM *nota* SAT (because SAT does not hold for the case of contradictory Ax).
- α *nevera* β, meaning that if $\langle Ax, C\rangle$ has the status α then it cannot have the status β. For example, SAT *nevera* CAX.
- α *xora* β, meaning that every $\langle Ax, C\rangle$ has the status α xor β. For example, THM *xora* CSA.

Additionally, axioms that deal with properties of formulae and models were provided. The relationships and properties axioms are given in Figure 3.

The axiomatization was shown to be consistent by generating a finite model using Paradox [5]. Some general properties of the relationships were proved using an ATP system (see below for a discussion of the ATP system used), e.g., that *isa* is a transitive relation, and that if α *isa* β and α *nota* γ then β *nota* γ. Next the relationships between all pairs of ontology values were investigated, using the ATP system

[1] Thanks to the reviewer of this paper whose comments instigated this step.

```
fof(esa,axiom,(
    ! [Ax,C] :
      ( ( ? [I1] : model(I1,Ax)
      <=> ? [I2] : model(I2,C) )
    <=> status(Ax,C,esa) ) )).

fof(thm,axiom,(
    ! [Ax,C] :
      ( ! [I1] :
          ( model(I1,Ax)
         => model(I1,C) )
    <=> status(Ax,C,thm) ) )).

fof(eth,axiom,(
    ! [Ax,C] :
      ( ( ? [I1] : model(I1,Ax)
        & ? [I2] : ~ model(I2,Ax)
        & ! [I1] :
            ( model(I1,Ax)
            <=> model(I1,C) ) )
      <=> status(Ax,C,eth) ) )).

fof(isa,axiom,(
    ! [S1,S2] :
      ( ! [Ax,C] :
          ( status(Ax,C,S1)
         => status(Ax,C,S2) )
    <=> isa(S1,S2) ) )).

fof(nota,axiom,(
    ! [S1,S2] :
      ( ? [Ax,C] :
          ( status(Ax,C,S1)
          & ~ status(Ax,C,S2) )
    <=> nota(S1,S2) ) )).

fof(nevera,axiom,(
    ! [S1,S2] :
      ( ! [Ax,C] :
          ( status(Ax,C,S1)
         => ~ status(Ax,C,S2) )
    <=> nevera(S1,S2) ) )).

fof(xora,axiom,(
    ! [S1,S2] :
      ( ! [Ax,C] :
          ( status(Ax,C,S1)
        <~> status(Ax,C,S2) )
    <=> xora(S1,S2) ) )).

fof(completeness,axiom,(
    ! [I,F] :
      ( model(I,F)
    <~> model(I,not(F)) ) )).

fof(not,axiom,(
    ! [I,F] :
      ( model(I,F)
    <=> ~ model(I,not(F)) ) )).

fof(tautology,axiom,(
    ? [F] : ! [I] : model(I,F) )).

fof(contradiction,axiom,(
    ? [F] : ! [I] : ~ model(I,F) )).

fof(sat_non_taut_pair,axiom,(
    ? [Ax,C] :
      ( ? [I1] :
          ( model(I1,Ax)
          & model(I1,C) )
      & ? [I2] :
          ( ~ model(I2,Ax)
          | ~ model(I2,C) ) ) )).

fof(satisfiable,axiom,(
    ? [F] :
      ( ? [I1] : model(I1,F)
      & ? [I2] : ~ model(I2,F) ) )).

fof(non_thm_spt,axiom,(
    ? [I1,Ax,C] :
      ( model(I1,Ax)
      & ~ model(I1,C)
      & ? [I2] : model(I2,C) ) )).
```

Figure 3: SZS Success Ontology Axioms

to prove the relationships from the axioms. The *isa* relationship was tested first, as if a pair of ontology values has the *isa* relationship they cannot have any of the other three relationships. For those pairs that were not proved to have the *isa* relationship, the *nevera* relationship was tested next. For those pairs that were proved to have the *nevera* relationship the *xora* relationship was tested, and for the other pairs the *nota* relationship was tested. Proving a *nota* relationship requires establishing the existence of formulae and models that deny the relationship. In the axiomatization five examples are provided, the `tautology`, `satisfiable`, `contradiction`, `non_thm_spt` and `sat_non_taut_pair` axioms above.

The results of the testing are shown in Table 1, where the vertical axis value has the shown relationship to the horizontal axis value. The *isa* relationship is denoted by $\Rightarrow$, *nota* by $\neg$, *nevera* by $\times$, and *xora* by $\oplus$. Sixty-eight pairs were proved to have the *isa* relationship, 89 to have the *nota* relationship, 179 to have the *nevera* (and not the *xora*) relationship, 4 to have the *xora* relationship, and for the remaining two pairs no relationship could be proved. The latter are the cases that WEC *nota* WTC and WEC *nota* TAC, which require exhibition of an $\langle Ax, C \rangle$ pair that has the WEC property but in which C is not a tautology. This could be done explicitly, along the lines of the `sat_non_taut_pair` axiom above, but that seemed like cheating. Some of the *nota* relationships may also be *nevera*, but could not be proved so.

As mentioned above, automated theorem proving was used to prove the relationships between the ontology values. At first, proofs were attempted using monolithic ATP systems such as EP, SPASS, and Vampire. The success rate was low, because the axiomatization forms a large theory - see [1]. Therefore the SRASS system [18] was used, and it was highly successful in identifying the necessary axioms for proving each conjecture, and subsequently obtaining either a proof using EP or an assurance of a proof using iProver [9]. In addition to SRASS, the MANSEX [22] and IDV [23] tools were used during the initial development of the axiomatization, to find the most obvious relationships and to analyze proofs.

	UNP	SAP	ESA	SAT	THM	EQV	TAC	WEC	ETH	TAU	WTC	WTH	CAX	SCA	TCA	WCA	CSA	UNS	NOC
UNP	●	¬	¬	¬	¬	¬	¬	¬	¬	¬	¬	¬	¬	⊕	×	×	¬	¬	¬
SAP	¬	●	¬	¬	¬	¬	¬	¬	¬	¬	¬	¬	¬	¬	¬	¬	¬	×	¬
ESA	⇒	⇒	●	¬	¬	¬	¬	¬	¬	¬	¬	¬	¬	×	×	×	¬	×	¬
SAT	⇒	⇒	⇒	●	¬	¬	¬	¬	¬	¬	¬	¬	×	×	×	×	¬	×	¬
THM	¬	⇒	¬	¬	●	¬	¬	¬	¬	¬	¬	¬	¬	¬	¬	¬	⊕	×	×
EQV	⇒	⇒	⇒	⇒	⇒	●	¬	×	¬	¬	×	×	×	×	×	×	×	×	×
TAC	⇒	⇒	⇒	⇒	⇒	¬	●	¬	×	¬	¬	×	×	×	×	×	×	×	×
WEC	⇒	⇒	⇒	⇒	⇒	×	¬	●	×	×	¬	¬	×	×	×	×	×	×	×
ETH	⇒	⇒	⇒	⇒	⇒	⇒	×	×	●	×	×	×	×	×	×	×	×	×	×
TAU	⇒	⇒	⇒	⇒	⇒	⇒	⇒	×	×	●	×	×	×	×	×	×	×	×	×
WTC	⇒	⇒	⇒	⇒	⇒	×	⇒	⇒	×	×	●	×	×	×	×	×	×	×	×
WTH	⇒	⇒	⇒	⇒	⇒	×	×	⇒	×	×	×	●	×	×	×	×	×	×	×
CAX	¬	⇒	¬	×	⇒	×	×	×	×	×	×	×	●	¬	¬	¬	×	×	×
SCA	⊕	⇒	×	×	⇒	×	×	×	×	×	×	×	⇒	●	¬	¬	×	×	×
TCA	×	⇒	×	×	⇒	¬	×	×	×	×	×	×	⇒	⇒	●	×	×	×	×
WCA	×	⇒	×	×	⇒	×	×	×	×	×	×	×	⇒	⇒	×	●	×	×	×
CSA	⇒	¬	¬	¬	⊕	×	×	×	×	×	×	×	×	×	×	×	●	¬	¬
UNS	⇒	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	⇒	●	×
NOC	⇒	⇒	⇒	⇒	×	×	×	×	×	×	×	×	×	×	×	×	⇒	×	●

Table 1: Success Ontology Relationships

All automated reasoning and proof processing was done on a computer with a Intel Xeon 2.80GHz CPU and 3GB memory, running the Linux 2.6 operating system, and with a 60s CPU time limit per proof attempt (on the entire SRASS process).

The formal analysis has had beneficial effects. Three new ontology values were added, three errors in the definitions of the ontology values were exposed and corrected, four incorrect *isa* links in the ontology were found and removed, and several unnoticed *isa* relationships were revealed and added. The *isa* links in Figure 1 correspond to those in Table 1.

3 The SZS No-Success Ontology

While it is always hoped that automated reasoning software will successfully process the logical data, and hence establish a success ontology value, in reality this often does not happen, for a variety of reasons. In order to understand and make productive use of a lack of success, e.g., [11, 8], it is necessary to precisely specify the reason for and nature of the lack of success. The SZS no-success ontology provides suitable status values for describing the reasons. Note that no-success is not the same as failure: failure means that the software has completed its attempt to process the logical data and could not establish a success ontology value. In contrast, no-success might be because the software is still running, or that it has not yet even started processing the logical data. Figure 4 shows the no-success ontology.

The meanings of the no-success ontology values are as follows:

- NoSuccess (NOS): The logical data has not been processed successfully (yet).
- Open (OPN): A success value has never been established.
- Unknown (UNK): Success value unknown, and no assumption has been made.

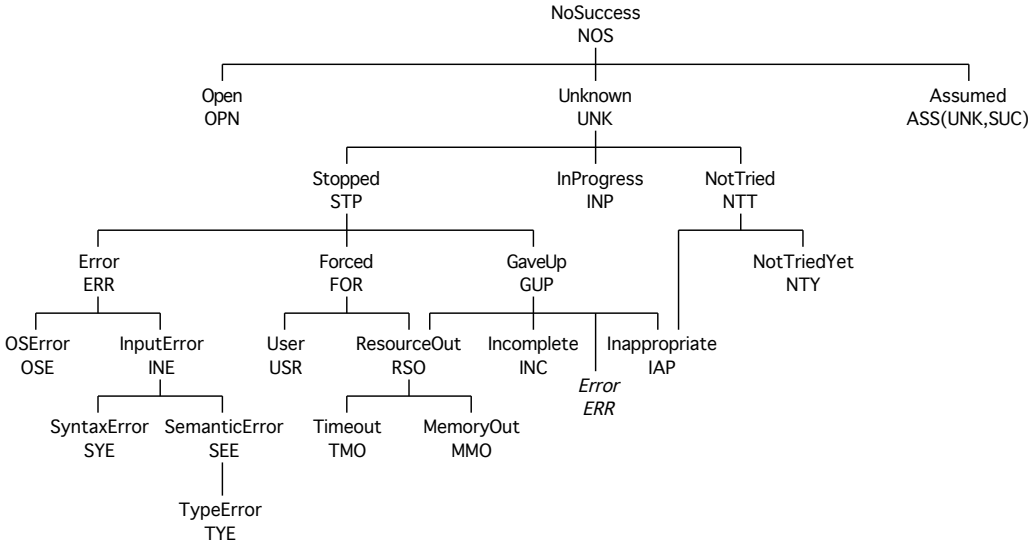

Figure 4: SZS No-Success Ontology

- Assumed (ASS(U,S)): The success ontology value S has been assumed because the actual value is unknown for the no-success ontology reason U. U is taken from the subontology starting at Unknown in the no-success ontology.
- Stopped (STP): Software attempted to process the data, and stopped without a success status.
- Error (ERR): Software stopped due to an error.
- OSError (OSE): Software stopped due to an operating system error.
- InputError (INE): Software stopped due to an input error.
- SyntaxError (SYE): Software stopped due to an input syntax error.
- SemanticError (SEE): Software stopped due to an input semantic error.
- TypeError (TYE): Software stopped due to an input type error (for typed logical data).
- Forced (FOR): Software was forced to stop by an external force.
- User (USR): Software was forced to stop by the user.
- ResourceOut (RSO): Software stopped because some resource ran out.
- Timeout (TMO): Software stopped because the CPU time limit ran out.
- MemoryOut (MMO): Software stopped because the memory limit ran out.
- GaveUp (GUP): Software gave up of its own accord.
- Incomplete (INC): Software gave up because it's incomplete.
- Inappropriate (IAP): Software gave up because it cannot process this type of data.
- InProgress (INP): Software is still running.
- NotTried (NTT): Software has not tried to process the data.
- NotTriedYet (NTY): Software has not tried to process the data yet, but might in the future.

The no-success ontology is very fine grained, and has more status values than are commonly used by automated reasoning software. A suitable subset for practical uses is as follows:

- The software stopped due to CPU limit - report Timeout.
- The software gave up due to incompleteness - report GaveUp.
- The software stopped due to an error - report Error.
- Any other cases - report Unknown.

4 The SZS Dataform Ontology

The success status values describe what is known or has been established about the relationship between the axioms and conjecture in logical data, but do not describe the form of logical data. The dataform ontology provides suitable values for describing the form of logical data. The dataform ontology values are commonly used to describe data provided to justify a success ontology value, e.g., if an ATP system reports the success ontology value `Theorem` it might output a proof to justify that. Figure 5 shows the dataform ontology.

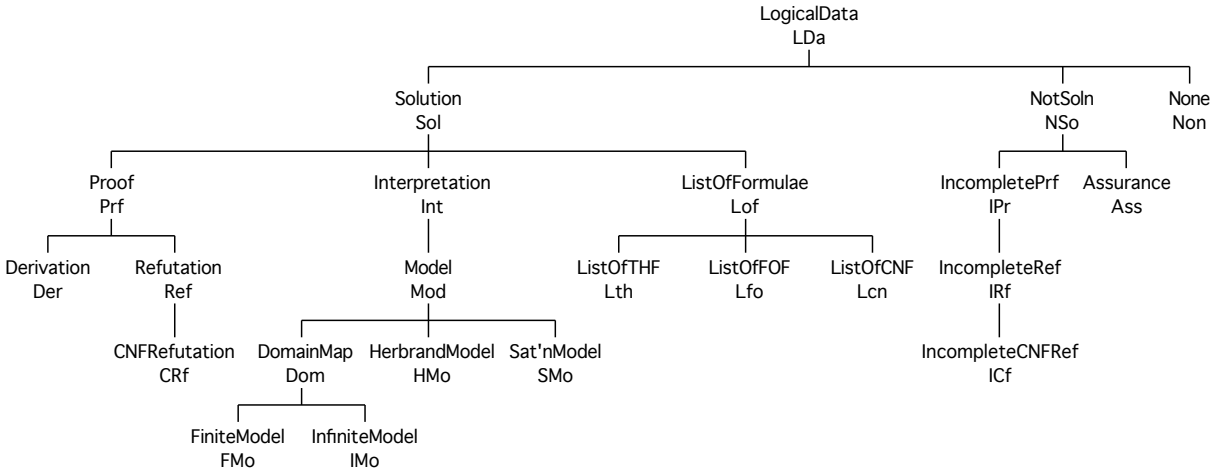

Figure 5: SZS Dataform Ontology

The meanings of the dataform ontology values are as follows:

- LogicalData (LDa): Logical data.
- Solution (Sln): A solution.
- Proof (Prf): A proof.
- Derivation (Der): A derivation (inference steps ending in the theorem, in the Hilbert style).
- Refutation (Ref): A refutation (starting with $Ax \cup \neg C$ and ending in FALSE).
- CNFRefutation (CRf): A refutation in clause normal form, including, for FOF Ax or C, the translation from FOF to CNF (without the FOF to CNF translation it is an IncompleteRefutation).
- Interpretation (Int): An interpretation.
- Model (Mod): A model.
- DomainMap (Dom): A model expressed as a domain, a mapping for functions, and a relation for predicates.
- FiniteModel (FMo): A domain map model with a finite domain.
- InfiniteModel (IMo): A domain map model with an infinite domain.
- HerbrandModel (HMo): A model expressed as a subset of the Herbrand base.
- SaturationModel (SMo): A model expressed as a saturating set of formulae.
- ListofFormulae (Lof): A list of formulae.
- ListofTHF (Lth): A list of THF formulae.
- ListofFOF (Lfo): A list of FOF formulae.
- ListofCNF (Lcn): A list of CNF formulae.
- IncompleteProof (IPr): A proof with part missing.
- IncompleteRefutation (IRf): A refutation with parts missing.

- IncompleteCNFRefutation (ICf): A CNF refutation with parts missing.
- Assurance (Ass): Only an assurance of the success ontology value.
- None (Non): Nothing.

The dataform ontology is very fine grained, and has more status values than are commonly used by automated reasoning software, by ATP systems in particular. A suitable subset for practical uses of ATP systems is as follows:

- A generic proof - report Proof.
- A refutation - report Refutation.
- A CNF refutation - report CNFRefutation.
- A generic model - report Model.
- A finite model - report FiniteModel.
- A Herbrand model - report HerbrandModel.
- A saturation model - report SaturationModel.

5 The SZS Presentation Standards

The SZS ontologies provide status values that precisely describe what is known or has been established about logical data. In order to make the use of the values easy in more complex reasoning systems, it is necessary to specify precisely how the values should be presented. This makes it easy for harness software to prepare input data and examine output data that contains ontology values, e.g., in practice, to `grep` the output from automated reasoning software for lines that provide status values.

Success and no-success ontology values should be presented in lines of the form

`% SZS status` *ontology_value* `for` *logical_data_identifier*

(The leading '%' makes the line into a TPTP language comment.) For example

```
% SZS status Unsatisfiable for SYN075+1
```

or

```
% SZS status GaveUp for SYN075+1
```

A success or no-success ontology value should be presented as early as possible, at least before any data output to justify the value. The justifying data should be delimited by lines of the form

`% SZS output start` *dataform_ontology_value* `for` *logical_data_identifier*

and

`% SZS output end` *dataform_ontology_value* `for` *logical_data_identifier*

For example

```
% SZS output start CNFRefutation for SYN075-1
    output_data
% SZS output end CNFRefutation for SYN075-1
```

All "SZS" lines can optionally have software specific information appended, separated by a `:`, i.e.,

`% SZS status` *ontology_value* `for` *logical_data_identifier* `:` *software_specific_information*

`% SZS output start` *dataform_ontology_value* `for` *logical_data_identifier* `:` *software_specific_info*

`% SZS output end` *dataform_ontology_value* `for` *logical_data_identifier* `:` *software_specific_info*

For example

```
% SZS status GaveUp for SYN075+1 : Could not complete CNF conversion
```

or

```
% SZS output end CNFRefutation for SYN075-1 : Completed in CNF conversion
```

6 Conclusion

This paper has presented the SZS ontologies of status values that are suitable for expressing precisely what is known or has been established about logical data. The ontologies can be used for existing logical data, e.g., they are used for the status of problems in the TPTP problem library [21] and solutions in the TSTP solution library [16], and can be used by automated reasoning software to describe their input and output. Already several ATP systems, e.g., Darwin [3], E, Metis [7], Paradox [5], use the SZS ontologies and the presentation standards, and this contributes to simplifying their embedding into more complex reasoning systems.

In addition to its use for reporting the overall status of a $\langle Ax, C \rangle$ 2-tuple, the SZS success ontology is used to report the status of individual inference steps in TPTP format derivations [19]. This is done in the "useful information" field of an `inference` record of an inferred formula. For example, in

```
cnf(58,plain,
    ( ~ hates(agatha,esk2_1(butler)) ),
    inference(spm,[status(thm)],[51,48])).
```

the status is Theorem (recorded as a lowercase acronym value `thm`), which indicates that the formulae is a theorem of it's two parent formulae `51` and `48`. The Theorem status is most common in derivations, but the SAP and ESA status values are also used quite often, e.g., for the formulae inferred by Skolemization and splitting steps. These status values can be used for semantic verification of the derivations, as is done by the GDV derivation verifier [17].

While the SZS ontologies are in use and have matured to some extent, it is not claimed that they are comprehensive and perfect. Developers and users of automated reasoning software are invited to provide feedback that might lead to improvements and increased usage. Already some users are working on success ontology values for results from computer algebra and other computational reasoning systems. In related work, SZS standards for returning answers from question-and-answer systems have been proposed.[2] It is hoped that over time, with increased usage, the ontologies will become battle hardened, and will be a core standard for automated reasoning.

References

[1] *Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories*, number 257 in CEUR Workshop Proceedings, 2007.

[2] A. Armando, M. Kohlhase, and S. Ranise. Communication Protocols for Mathematical Services based on KQML and OMRS. In M. Kerber and M. Kohlhase, editors, *Proceedings of the Calculemus Symposium 2000*, 2000.

[3] P. Baumgartner, A. Fuchs, and C. Tinelli. Darwin - A Theorem Prover for the Model Evolution Calculus. In G. Sutcliffe, S. Schulz, and T. Tammet, editors, *Proceedings of the Workshop on Empirically Successful First Order Reasoning, 2nd International Joint Conference on Automated Reasoning*, 2004.

[4] C. Benzmüller, F. Rabe, and G. Sutcliffe. THF0 - The Core TPTP Language for Classical Higher-Order Logic. In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, page Accepted, 2008.

[5] K. Claessen and N. Sorensson. New Techniques that Improve MACE-style Finite Model Finding. In P. Baumgartner and C. Fermueller, editors, *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*, 2003.

[6] E. Denney, B. Fischer, and J. Schumann. Using Automated Theorem Provers to Certify Auto-generated Aerospace Software. In M. Rusinowitch and D. Basin, editors, *Proceedings of the 2nd International Joint*

[2] See `http://www.tptp.org/TPTP/Proposals/AnswerExtraction.html`

Conference on Automated Reasoning, number 3097 in Lecture Notes in Artificial Intelligence, pages 198–212, 2004.

[7] J. Hurd. First-Order Proof Tactics in Higher-Order Logic Theorem Provers. In M. Archer, B. Di Vito, and C. Munoz, editors, *Proceedings of the 1st International Workshop on Design and Application of Strategies/-Tactics in Higher Order Logics*, number NASA/CP-2003-212448 in NASA Technical Reports, pages 56–68, 2003.

[8] A. Ireland and A. Bundy. Productive use of Failure in Inductive Proof. *Journal of Automated Reasoning*, 16(1-2):79–111, 1996.

[9] K. Korovin. iProver - An Instantiation-Based Theorem Prover for First-order Logic (System Description). In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 292–298, 2008.

[10] P. Massar, M. Travers, J. Elhai, and J. Shrager. BioLingua: A Programmable Knowledge Environment for Biologists. *Bioinformatics*, 21(2):199–207, 2005.

[11] R. Monroy, A. Bundy, and A. Ireland. Proof Plans for the Correction of False Conjectures. In F. Pfenning, editor, *Proceedings of the 5th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 822 in Lecture Notes in Artificial Intelligence, pages 178–189. Springer-Verlag, 1994.

[12] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

[13] S. Schulz. E: A Brainiac Theorem Prover. *AI Communications*, 15(2-3):111–126, 2002.

[14] J. Shrager, R. Waldinger, M. Stickel, and P. Massar. Deductive Biocomputing. *PLoS ONE*, 2(4), 2007.

[15] M.E. Stickel. SNARK - SRI's New Automated Reasoning Kit. http://www.ai.sri.com/ stickel/snark.html.

[16] G. Sutcliffe. The TSTP Solution Library. http://www.TPTP.org/TSTP.

[17] G. Sutcliffe. Semantic Derivation Verification. *International Journal on Artificial Intelligence Tools*, 15(6):1053–1070, 2006.

[18] G. Sutcliffe and Y. Puzis. SRASS - a Semantic Relevance Axiom Selection System. In F. Pfenning, editor, *Proceedings of the 21st International Conference on Automated Deduction*, number 4603 in Lecture Notes in Artificial Intelligence, pages 295–310. Springer-Verlag, 2007.

[19] G. Sutcliffe, S. Schulz, K. Claessen, and A. Van Gelder. Using the TPTP Language for Writing Derivations and Finite Interpretations. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, pages 67–81, 2006.

[20] G. Sutcliffe and D. Seyfang. Smart Selective Competition Parallelism ATP. In A. Kumar and I. Russell, editors, *Proceedings of the 12th International FLAIRS Conference*, pages 341–345. AAAI Press, 1999.

[21] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[22] G. Sutcliffe, A. Yerikalapudi, and S. Trac. Multiple Answer Extraction for Question Answering with Automated Theorem Proving Systems. Rejected from the 15th International Conference on Logic for Programming Artificial Intelligence and Reasoning, 2008.

[23] S. Trac, Y. Puzis, and G. Sutcliffe. An Interactive Derivation Viewer. In S. Autexier and C. Benzmüller, editors, *Proceedings of the 7th Workshop on User Interfaces for Theorem Provers, 3rd International Joint Conference on Automated Reasoning*, volume 174 of *Electronic Notes in Theoretical Computer Science*, pages 109–123, 2006.

[24] C. Weidenbach, R. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic. SPASS Version 3.0. In F. Pfenning, editor, *Proceedings of the 21st International Conference on Automated Deduction*, number 4603 in Lecture Notes in Artificial Intelligence, pages 514–520. Springer-Verlag, 2007.

[25] J. Zimmer and M. Kohlhase. System Description: The MathWeb Software Bus for Distributed Mathematical Reasoning. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, number 2392 in Lecture Notes in Artificial Intelligence, pages 139–143. Springer-Verlag, 2002.