Detecting Inconsistencies in Large First-Order Knowledge Bases

Stephan Schulz[1(✉)], Geoff Sutcliffe[2(✉)], Josef Urban[3(✉)], and Adam Pease[4(✉)]

[1] DHBW Stuttgart, Stuttgart, Germany
schulz@eprover.org
[2] University of Miami, Coral Gables, USA
geoff@cs.miami.edu
[3] Czech Technical University in Prague, Prague, Czech Republic
josef.urban@gmail.com
[4] Articulate Software, San Francisco, USA
apease@articulatesoftware.com

Abstract. Large formalizations carry the risk of inconsistency, and hence may lead to instances of spurious reasoning. This paper describes a new approach and tool that automatically probes large first-order axiomatizations for inconsistency, by selecting subsets of the axioms centered on certain function and predicate symbols, and handling the subsets to a first-order theorem prover to test for unsatisfiability. The tool has been applied to several large axiomatizations, inconsistencies have been found, inconsistent cores extracted, and semi-automatic analysis of the inconsistent cores has helped to pinpoint the axioms that appear to be the underlying cause of inconsistency.

1 Introduction

Automated Theorem Proving (ATP) is concerned with the development and use of computer programs that automate sound reasoning: the derivation of conclusions that follow inevitably from facts. The dual discipline, automated model finding, develops computer programs that establish that a set of statements is consistent. These capabilities lie at the heart of many important computational tasks, e.g., formal methods for software and hardware design and verification, [11,37], reasoning in meta-physics [4,46], solving hard problems in mathematics, [19,24], and inference for the semantic web [13]. The use of automated reasoning systems (theorem proving and model finding) requires a user to (rather precisely) describe the domain of application as a set of *axioms*. For theorem proving, an ATP system is then used to prove that a *conjecture* is a *theorem* of the axioms, and hopefully produce a proof. For model finding, a model finding system is used to demonstrate the *consistency* of the axioms, and hopefully produce a model of the axioms. Automated *theorem proving* in classical logic relies on the axioms being consistent, for otherwise all conjectures are theorems of the axiomatization.

J. Urban—Supported by the ERC Consolidator grant no. 649043 *AI4REASON*.

L. de Moura (Ed.): CADE 2017, LNAI 10395, pp. 310–325, 2017.
DOI: 10.1007/978-3-319-63046-5_19

The direct method of showing that a set of axioms is consistent is to use a model finder on the axioms. However, for large axiom sets this approach becomes rather difficult (or impossible), because most large axiomatizations have large or infinite models.

In the last 10 to 15 years there has been an increased, and increasingly successful, use of automated reasoning in "large theories", i.e., domain descriptions that have many symbols, and many axioms of which typically only a few are required for the proof of a theorem. Examples include commonsense and ontological knowledge bases such as Cyc [23,33] and the SUMO (Suggested Upper Merged Ontology) family of ontologies [26,29,31], large mathematical formalizations such as Mizar [42,44], Flyspeck [17] and Isabelle's *Archive of Formal Proofs* [20], and encodings of biological domains [10]. Such large axiomatizations always carry the risk of inconsistency - either because of mistakes in the original formulation, or because of errors encoding the original formulation into logic. Advances in ATP systems have revealed these inconsistencies while finding proofs of theorems [38], thus stimulating efforts to check such axiomatizations for consistency (for which model finding is largely unsuccessful, as noted above), and to pinpoint and fix inconsistencies.

This paper proposes a new approach and tool for automatically probing large first-order axiomatizations for inconsistencies. If inconsistencies are found, a small inconsistent core can typically be presented, which makes it easy to identify errors and to repair the axiomatization. This paper describes the idea and implementation of the method, and demonstrates how effective it can be on some existing large first-order axiomatizations.

2 Automated Reasoning in Large Theories

A common, almost necessary, part of reasoning over large axiomatizations is focussing on axioms that are likely to be relevant to proving a given conjecture. Typically, only a few axioms are needed for a proof. The irrelevant axioms increase the search space, often to the extent that it is impossible to find a proof. Axiom selection techniques address this problem [22,41]. A number of strong axiom selection methods for assisting formal mathematics [7] are based on various ways of learning from a large body of previous proofs [1,2,16,45]. In this work we are however to a large extent interested in common-sense knowledge bases, which mainly consist just of definitions and axioms. Below we therefore focus on *heuristic* rather than learning selection methods.

Relevance pruning tries to identify all formulae that are potentially relevant to proving a conjecture (or set of conjectures). Two formulae are relevant with respect to each other if they share a function or predicate symbol. Relevance pruning selects all formulae in the reflexive transitive closure of the conjecture with respect to the relevancy relation. Unrestricted relevance pruning maintains completeness in the non-equational case. A weakness is that the closure is often still very large.

A symbolic heuristic approach is the MePo families of filters [25] developed in the context of Sledgehammer [27], Isabelle's interface to ATP systems. The MePo

filters represent formulae (and formula sets) as vectors of symbol occurrences, and iteratively select formulae with vectors similar to the vector of the already selected set. The process is again seeded with the conjecture.

Maybe the currently most widely used family of algorithms is derived from Hoder's SInE (SUMO Inference Engine) [15]). SInE can be seen as a heuristic variant of relevancy pruning. It is based on the idea of a "defines" relation between symbols and formulae. As for the previous approaches, SInE starts with a conjecture (and its symbols) and tries to add formulae until a fixpoint is reached in which all symbols in the set of selected formulae are defined. SInE assumes that rare symbols are typically defined in terms of more common symbols. Thus the "defines" relation associates a symbol with the formulae in which the symbol is the rarest symbol. Various implementations allow for slightly weaker constraints on this condition, or support early termination after a given number of formulae has been added or a maximum level of definitions has been followed. Variants of SInE have been implemented in, e.g., Vampire [21] and E [35,36], and are also the basis for this work.

3 Automatic Inconsistency Probing

In previous work, we have employed a method where we simply iterate through the set of axioms in the SUMO theory, testing them one at a time for inconsistency with the growing knowledge base [30]. The method starts from an empty knowledge base, and iteratively adds each axiom not proven to lead to a contradiction. However, this process can fail to find existing inconsistencies as soon as the knowledge base becomes complex enough that not every prover run terminates with a satisfiable/unsatisfiable result. Ideally, we would have a method fast enough to be run every time a new axiom is added to the theory.

The core idea of our new approach is to automatically extract subsets of an axiomatization, and test each for unsatisfiability. If any subset is unsatisfiable, so is the full axiomatization. The inconsistent core of such a subset can be extracted from the proof of unsatisfiability, and analyzed to pinpoint the cause of the inconsistency.

To extract a potentially unsatisfiable subset of axioms, a symbol is selected as the *seed symbol*. The seed symbol is used to select a set of seed formulae (possibly a single formula). The seed formulae are used as pseudo-goals for a SInE filter that recursively extracts definitions related to the seed formulae until the definitional closure is reached or one of the hard bounds (number of formulae or depth of definition chain) is reached. The resulting set of axioms is handed to a refutation-based ATP system that tries to show the set to be unsatisfiable. The overall architecture and data flow of the system is depicted in Fig. 1.

This general approach has a number of choice points: How to select the seed symbol, how to use the seed symbol to select seed formulae, which SInE filters to use, and how to parameterize the ATP system with respect to search strategy and resource limits.

The easiest way to select seed symbols is to try all the symbols in the axiomatization. For large axiomatizations with a big signature this leads to very many

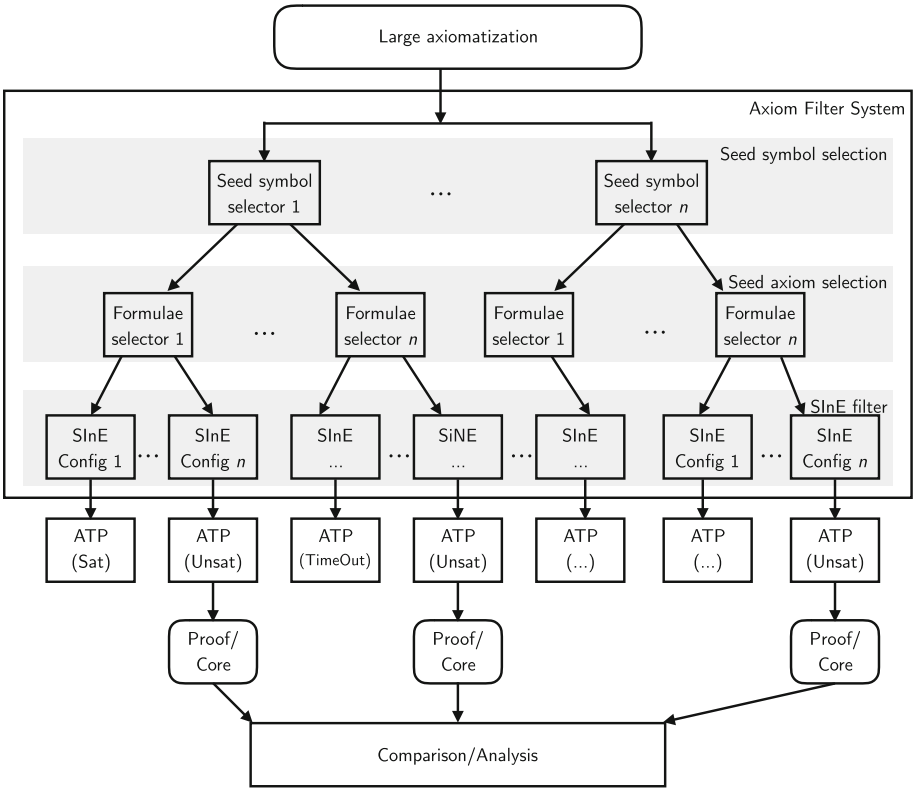

Fig. 1. Data flow and architecture

subsets to be tested for unsatisfiability, and hence is computationally challenging, although in many cases not prohibitively so. To limit the number of subsets, restrictions can be imposed on the type and number of seed symbols. First, the class of symbols considered as seeds can be restricted to predicate symbols only, proper function symbols (excluding constants) only, or constants only, or any combination of the three. At least in commonsense scenarios, a very large proportion of symbols are constants, and our experiments have shown that excluding constants does not seem to reduce the number of inconsistencies found.

Secondly, a subset of the eligible seed symbols can be selected, according to a desired number of seed symbols and a selection criterion. We implemented a variety of relatively simple methods to get an impression of the spectrum of behaviours. Here, the first approach is to pick *rare symbols*, i.e. symbols that do not occur in many axioms, as seed. This is based on the assumption that more specialized parts of the ontology will typically be less exercised than more general parts, and are hence more likely to contain hidden bugs. The opposite approach, picking the most *frequent symbols* as seeds, is based on the idea that

they tend to bring together different, possibly conflicting parts of the ontology. The last method, picking *random symbols*, acts as a control.

Given a set of seed symbols, three different methods have been tested for selecting seed formulae:

- *Use of the most diverse axiom* selects a single seed formula with the largest number of different function and predicate symbols among all that contain the given seed symbol. Ties are broken by selecting the first candidate formula with maximal diversity.
- *Use of the largest axiom* picks the syntactically largest formula that contains the seed symbol, i.e. the formula with the most nodes in its tree representation. Ties are again broken in favor of the first formulae found.
- *Use of all* axioms that contain the seed symbol.

After selection of the seed formulae, a number of different SInE variants are used as filters. Each of the SInE filters produces one subset to be tested for unsatisfiability for each input set. The resulting files are handed to the theorem prover (usually in some batch processing configuration), which tries to prove them with a short time limit (3 s to 30 s).

3.1 Implementation

The extraction of subsets to test for unsatisfiability has been implemented in `e_axfilter`, a component of the E system distribution. It implements a version of SInE that efficiently applies multiple filters to its input, amortizing the cost of parsing, preprocessing and indexing. Code to select seed symbols and seed formulae was added, as described in the previous section.

In the experiments, SInE filters that have previously proven their worth with respect to conventional theorem proving were used. Of these, 9 different SInE configurations are applicable to this setting (which does not distinguish between the proper conjecture and additional local hypotheses, as there is no proper conjecture).

E 2.0pre12 was used as the default ATP system to check the extracted formula sets for unsatisfiability. The prover was configured to run in *automatic mode*, but without engaging its built-in SInE selection. Postprocessing and analysis of the unsatisfiable cores was done with simple shell scripts and some manual processing.

A cleaned up distribution of the current state of the system is available at http://eprover.eu/E-eu/AxProbing.html.

4 Experimental Results

4.1 SUMO Results

We have first applied our system to the TPTP v6.4.0 [39] axiom file `CSR003+2.ax`, which contains a first-order translation [28] of the 2010 release

of SUMO, MILO (the *MId-Level Ontology*) and 30 domain ontologies. The file contains 55588 formulae, which use 1291 predicates, 291 non-constant function symbols, and 32838 constants. It was originally believed to be consistent, but results from the CASC-J6 ATP system competition revealed that the axioms were inconsistent [38], and further inconsistencies have been found since then. In each case corrections were made to make the theory consistent again, at least as far as was known at the time. The experiments for this paper revealed more inconsistencies. We explored the axiomatization, confirmed the inconsistency, and identified at least one common root cause.

The seed symbols were selected in each of the three ways ... randomly, the most frequently occurring, and the least frequently occurring. For each way, 600 symbols were selected ... 200 predicate symbols, 200 (non-constant) function symbols, and 200 constants. We ran E with a time limit of 3 seconds on current-generation hardware (2.6 GHz Intel Core processors, no memory limit, automatic mode) to determine the status of each probe.

Table 1 summarizes the properties of the generated files. At this stage, a number of interesting and maybe unintuitive observations can be made. First, we were surprised by the fact that less frequent symbols seem to generate larger probes - the naive assumption being that the larger set of seed axioms in the "use all applicable formulas" setting would bias the size up. However, the effect can be explained by looking at SInE's "defined" relation, which assumes that rare symbols are defined in terms of more common ones. Very specialized (rare) symbols have more levels of definitions to traverse until the fix-point is reached. The second observation is that success of the ATP - both for satisfiable and unsatisfiable probes - strongly correlates with the average size of the problem, with most successes for the smaller probes based on more common symbols.

Table 1. Properties of generated SUMO subsets

Seed symbols selection method	# formulae				ATP status		
	Min	Med	Avg	Max	SAT	UNS	TMO
Random	1	6855	1001	20001	1577	19	14604
Least frequent symbols	8	7963	3430	20001	965	11	15224
Most frequent symbols	1	5303	501	20001	4024	623	11553

Columns show the minimum number of axioms for probes in the corresponding category, the median size, the average size, the maximum size, and the number of probes shown satisfiable, unsatisfiable, or running into the time limit.

Table 2 provides an overview of the results. For each seed symbol selection method it provides the number of subsets that were found to be unsatisfiable, the number of distinct unsatisfiable cores, the number of distinct Predicate/Function/Constant seeds that led to an unsatisfiable subset, and the number of distinct seed symbols leading to the Diverse/Largest/All seed formula selection method producing an unsatisfiable probe.

Table 2. SUMO experimental results

Seed symbols selection method	# of UNS	# distinct	# by seed type			# by axiom select.		
	Subsets	UNS cores	P	F	C	D	L	A
Random	19	15	6	3	0	1	1	9
Least frequent symbols	11	9	2	4	0	1	1	6
Most frequent symbols	623	43	78	3	0	79	79	81
All together	653	67	86	9	0	81	81	95

The results indicate that (for at least this axiomatization) using the most frequently occurring predicate symbols as seed symbols is the most effective. We can also observe that the "Use all formulas with the seed symbol" method subsumes the other approaches - every symbol that was successful with one of the other seed axiom selection method also produced at least one unsatisfiable probe with that method.

Automated analysis of the unsatisfiable cores reveals that the following axiom is present in all of them:

```
fof(kb_SUMO_32603,axiom,(
    ! [V__C2,V__U,V__C1] :
      ( V__U = s__UnionFn(V__C1,V__C2)
    <=> ! [V__I1,V__I2,V__I3] :
          ( ( s__instance(V__C1,s__SetOrClass)
            & s__instance(V__U,s__SetOrClass)
            & s__instance(V__C2,s__SetOrClass) )
         => ( ( s__instance(V__I1,V__C1)
              & s__instance(V__I2,V__C2)
              & s__instance(V__I3,V__U) )
           => ( s__instance(V__I1,V__U)
              & s__instance(V__I2,V__U)
              & ( s__instance(V__I3,V__C1)
                | s__instance(V__I3,V__C2) ) ) ) ) ) )).
```

The axiom (incorrectly) defines the union function `s__UnionFn`, but mistakenly uses the `s__instance` predicate instead of the `s__member` predicate in the consequent of the outer implication on the right-hand side of the equivalence. If we remove the offending axiom, no more inconsistencies are found and SUMO is consistent to the best of our knowledge.

With this method for debugging SUMO we now have, for large logical theories, something approaching common practice in software engineering on non-trivial systems, of adding to a software system and then going through one or more cycles of validation and correction. Hopefully, further research will continue to yield methods that improve the completeness of the debugging process,

Table 3. Probe status by SInE configuration

SInE configuration	Min	Med	Avg	Max	SAT	UNS	TMO
gf120_h_gu_R02_F100_L20000	1	385	314	928	1109	241	4050
gf120_h_gu_RUU_F100_L00100	1	79	101	101	1258	0	4142
gf120_h_gu_RUU_F100_L00500	1	322	501	501	814	241	4345
gf120_h_gu_RUU_F100_L01000	1	601	978	1001	790	158	4452
gf150_h_gu_RUU_F100_L20000	8	6283	10075	10625	603	0	4797
gf200_h_gu_R03_F100_L20000	5	1590	1805	4003	833	13	4554
gf200_h_gu_RUU_F100_L20000	8	13211	17703	17966	516	0	4884
gf500_h_gu_R04_F100_L20000	15	11789	18156	20001	439	0	4961
gf600_h_gu_R05_F100_L20000	28	13472	20001	20001	204	0	5196

and improve its speed, so that it becomes possible to check every new axiom at the time it is authored for whether it introduces an inconsistency.

We can also analyze which of the 9 different SInE configurations have contributed most to finding inconsistent probes. Table 3 summarizes the result. We can see that only 4 of the 9 filter configurations generate at least one probe that is unsatisfiable. We can again confirm that the overall ATP success rate seems to strongly correspond to the median (and average) problem size, i.e. the larger the problems are, the more likely the ATP is to time out. On the other hand, for finding inconsistencies, we are only interested in UNS (unsatisfiable) results. There, over-pruning (as likely the case for gf120_h_gu_RUU_F100_L00100, which has a very small hard size limit and selects at most 100 formulae to complement the seed formulae) is also a risk.

4.2 OpenCyc Results

Experiments similar to those performed on the SUMO axiom set were performed on an export of a fragment of the OpenCyc knowledge base [33], in the TPTP v6.4.0 axiom file `CSR002+4.ax`. Two inconsistencies were found.

The first unsatisfiable subset was generated by selecting the 200 least frequently occurring predicate symbols as seed symbols, and selecting all axioms with those seed symbols as seed formulae. The offending unsatisfiable core of seven axioms contains the following two axioms, which confuse temporal objects with geographical subregions - clearly a mistake in the underlying Cyc axiomatization.

```
fof(ax4_357170,axiom,(
    ! [X] :
      ( temporalstufftype(X)
     => geographicalsubregiontypes(X,X) ) )).

fof(ax4_231810,axiom,(
    ! [ARG1,ARG2] :
```

```
 ( geographicalsubregiontypes(ARG1,ARG2)
=> temporalstufftype(ARG1) ) )).
```

The second unsatisfiable subset was generated by selecting 400 random predicate symbols as seed symbols, and selecting all axioms with those seed symbols as seed formulae. The offending unsatisfiable core of 20 axioms contains two axioms that mix temporal objects with physical parts, but in this case it is not obvious that they are the direct cause of the inconsistency.

4.3 Mizar Results

For the Mizar experiments the MPTP translation [42] of the Mizar Mathematical Library (MML) [12] to TPTP was used. More precisely, version 4.181.1147[1] of the MML, which has been used for the so far most extensive ATP and premise-selection experiments over Mizar [18], was used. These previous experiments took several weeks of real-time computation on a 64-core AMD server. Several hundred combinations of premise selection methods and theorem provers were tried. Neither a contradiction, nor a proof that would be illegal with respect to the Mizar system was found. This makes it very unlikely to find a contradiction in our current experiments.

The axiomatization file `statements`[2] that was used for generating subtheories contains 146700 top-level Mizar lemmas, definitions, scheme instances, type-system formulae, and other formulae encoding the Mizar built-in knowledge. The formulae come from 1153 Mizar articles[3], containing 17355 function and constant symbols (including 3428 numbers), and 3689 predicate symbols[4].

Mizar Unsampled. Initially, all problems were generated using both predicate and function symbols and without subsampling. This takes about 15 hours using a single CPU and produces 286614 files taking up about 700GB. Then we run E in auto mode for 10 seconds on each problem. This takes 18 hours of real time using 50 CPUs in parallel. As expected, no problems are found to be Unsatisfiable.

Mizar Sampled. In the next version subsampling with `m200` (picking the 200 most frequently occurring symbols as seeds) was used, limiting the seed symbols only to predicates (based on the SUMO experiments) and seed method `dl` (using the most diverse formula as a seed formula, and using the syntactically largest formula as a seed formula). This generates in 23 min 3600 problems taking up 11GB. Since this is much fewer problems, we can run E with higher time limit (30 seconds). This takes 45 min of real time when using 50 CPUs in parallel. Even with this higher time limit, no problems are found to be Unsatisfiable.

[1] http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/.
[2] http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/MPTP2/statements.
[3] http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/.
[4] http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/MPTP2/symbols.

Mizar Sampled with Some Omitted Type Guards. In the last Mizar experiment, a frequent error in formal mathematical developments was emulated by omitting an "obvious" assumption from a lemma. In particular, non-emptiness (and thus also non-zero) assumption was omitted from the toplevel Mizar statements. There are over 6000 affected statements.

Model finders such as Nitpick [8] and Nunchaku [34] can be tried in proof assistants to quickly find such errors. Such methods have however limited use when working over foundations such as set theory, where the underlying models (if any) are infinitary. Finding a contradiction caused by the too strongly stated lemma – using the methods developed here – is an interesting alternative.

Another scenario which actually occurred in various moments of the history of building translations between ITPs and ATPs is that such typing assumptions are sometimes omitted due to various corner cases in the ITP-ATP translation modules. The methods developed here can be used as a global debugging tool when creating such translations. Since it was known beforehand how to correct all the corrupted statements, it was also possible to fully automate and observe the process of gradually finding them with our tools, interleaved with (automated) correction and re-formulation. This provides an empirical evaluation of the strength of the tools.

To generate the problems, the corrupted `statements` file was used, and again subsampling was used to generate 3600 problems. As before, E was run with a 30 s time limit and 50-fold parallelization. E found 2312 of the 3600 problems to be unsatisfiable. E quickly finds very simple refutations (226 that use just three formulae), but also more complicated ones: 347 of the refutations use 10 or more formulae. All these 2312 refutations are due to only 8 corrupted formulae. The one that occurs most frequently (in 1227 of the refutations) is the corrupted Mizar typing statement `cc1_ami_3`:[5]

```
fof(cc1_ami_3,axiom,(
    ! [X1] :
      ( v7_ordinal1(X1)
     => ( ~ v1_xboole_0(X1)
        & v7_ordinal1(X1)
        & ~ v1_setfam_1(X1) ) ) )).
```

This claims that every natural number is non-empty and has an empty element. This is of course false for zero, which is modelled as the empty set in set theory.

After repairing the eight corrupted formulae that were found automatically in the 3600 generated problems, E was run again. This time 853 problems are found to be unsatisfiable, and 18 new corrupted formulae involved in the refutations were found. It is clear that the loop finds the most obvious offending formulae early, and proceeds to find more and more complicated proofs of the contradiction. After obtaining a fixpoint, the system becomes "effectively consistent" wrt. to our tool, however it is still possible to see if there are problematic formulae left.

[5] http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/ami_3.html#CC1.

Table 4. 15 iterations of the contradiction-finding and axiom-correcting loop run on the 3600 problems constructed from the Mizar data with some type guards omitted.

Loop iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bad formulae found	8	18	10	8	6	1	1	0	0	0	1	0	0	0	0
Unsat. problems found	2312	853	637	238	102	2	2	0	0	0	1	0	0	0	0
Average proof time	10.2	8.5	13.7	21.2	22.7	21.3	27.7				29.5				

This automated contradiction-finding and axiom-correcting loop is implemented in about 30 lines of Perl.[6] Note however that this can be easily automated only because the correct versions of all the intentionally modified facts are known beforehand. In general, the axiom-correcting step is nontrivial. The loop was run over the 3600 problems for 15 iterations, using a 30 s time limit for a problem and 50-fold parallelization. The 15 iterations thus took about 8 hours of real time and 400 hours of CPU time.

It takes 7 iterations to reach a state in which no more contradictions are found within the time limit. Many problematic formulae are however still left: 3590 of the 3600 problems still contain at least one of them after the 7th iteration. Letting the loop run further shows that adding more time will very likely discover further contradictions: in the 11th iteration, one more contradiction is found taking 29.5 s of CPU time. In total, the loop discovered 53 problematic formulae. The iterations are shown in Table 4.

A brief experiment was done with a different ATP system, thus offering a different notion of "effective consistency". Instead of E, Vampire 4.1 was used, again with a 30 s time limit on all the problems from the 15th iteration (none refutable by E). Vampire finds a contradiction in 297 of the problems, detecting 15 more problematic formulae. A fixpoint for Vampire is reached after 5 iterations and discovering in total 29 more problematic formulae.

The remaining number of problematic formulae in the 3600 repaired problems is however still high. 3590 of the repaired problems contain at least one of the formulae, 3188 different problematic formulae occur there, and the total number of occurrences of the formulae in all the problems is 1228432 (2.85% of all the 43049126 formulae there). For comparison, before starting the repairing loop, this number was 1378711 (3.2%), i.e., 11% of such occurrences were automatically removed.

5 Future Work

This paper has laid the groundwork to automatically search for inconsistencies in large knowledge bases. There is a number of possible extensions.

First, there are many ways to experiment with the filtering. We have so far only used E's existing SInE implementation and its existing filter configurations. One approach would be to try new filter configurations close to the parameter

[6] https://github.com/JUrban/MPTP2/blob/master/Ebot/loop.pl.

space demarcated by the so far most successful filters, using parameter-searching systems such as BliStr [43] on a large set of problems. This applies not just to the SInE filters, but to the whole interplay between seed symbol selection, seed axiom selection and SInE filtering. Again, we could try to narrow down the probes to a subset with a higher per-probe success rates, thus identifying a similar number of inconsistencies with less computational effort. In this context, we can also explore additional seed selection preferences (i.e. only use symbols with certain arities, or symbols with certain minimum frequency) and seed axiom selection methods (use the smallest axiom, use the least diverse axiom, etc.).

Second, we could add more axiom selection methods. For example, a very different *semantic* heuristic selection method is available in SRASS [40] and MaLARea [45]. It interleaves model finding for the axioms selected so far with adding a (most relevant) axiom that is false in the model found so far. When the loop stops (typically because no more models can be found), the axiom selection is given to an ATP. In situations when a large number of proofs is available (or generated e.g. by theory exploration), the current methods that produce the seeding formula could also be followed by axiom selection based on some of the many machine-learning methods developed recently.

Third, it would be instructive to apply our approach to additional large axiomatizations, both in mathematical domains (e.g. a first-order translation of the HOL Light Flyspeck corpus [14]), but also in more application-oriented domains, where an interesting example would be SnowMed CT [3]. There may be unsound-but-efficient encodings (historically used e.g. for Isabelle's first-order translation), that might keep most of their efficiency after automatically removing the worst sources of unsoundness using a similar process as described for Mizar in Sect. 4.3. This process – namely removal of type guards – will also apply to SUMO, since it uses a method for automatically adding explicit types to axioms that lack such expressions in their originally authored forms.

Finally, we could experiment with different sound encodings. In some early work, different methods for canonicalization of the knowledge base have lead to the surfacing of different possible inconsistencies. We expect to integrate these approaches and see if they yield more and different results. Our approach could be also extended to work directly with more expressive logics, e.g. the higher order logics used in systems like Isabelle, or the original higher order formulation of SUMO in SUO-KIF or its translation to THF [6], either by translation of higher-order theories to first order, or by direct use of higher-order provers like Leo-II [5] or Sattalax [9].

6 Conclusion

For any large theory under active development there is a process that combines the addition of new axioms to extend the coverage of the theory, with the use of tools that aim to ensure desired properties of the axiomatization. The tool presented here is one more in that armory, helping to ensure that the axiomatization is consistent. Both the SUMO and OpenCyc knowledge bases have,

on-and-off, been considered to be consistent, particularly those parts that lie at their core. The discovery of inconsistencies might thus come as a surprise to users, particularly when the axiomatizations have been used productively in applications without revealing the contradictions. While the use of tools such as the one described in this paper can be very helpful in finding inconsistencies, it is important to note that such tools (including this one) are incomplete – they do not ensure that the axiomatization is consistent.

Therefore it remains important to guard against conclusions that have been derived because of inconsistency. Simply checking that the conjecture is part of its proof can guard against this. One can also employ more refined approaches based on paraconsistent logics [32].

As might be obvious from the differences in the experiments described in Sects. 4.1 to 4.3, the tool has been used independently, and successfully, in three quite distinct efforts. This shows the flexibility of the tool, and that different combinations of choices can be effective in different circumstances.

References

1. Alama, J., Heskes, T., Kühlwein, D., Tsivtsivadze, E., Urban, J.: Premise selection for mathematics by corpus analysis and kernel methods. J. Autom. Reasoning **52**(2), 191–213 (2014)
2. Alemi, A.A., Chollet, F., Eén, N., Irving, G., Szegedy, C., Urban, J.: DeepMath - deep sequence models for premise selection. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 5–10, 2016, Barcelona, Spain, pp. 2235–2243 (2016). http://papers.nips.cc/paper/6280-deepmath-deep-sequence-models-for-premise-selection
3. Benson, T.: Principles of Health Interoperability: SNOMED CT, HL7 and FHIR (Health Information Technology Standards). Springer, London (2016)
4. Benzmüller, C., Woltzenlogel Paleo, B.: Automating gödel's ontological proof of god's existence with higher-order automated theorem provers. In: Schaub, T. (ed.) Proceedings of the 21st European Conference on Artificial Intelligence, pp. 93–98 (2014)
5. Benzmüller, C., Paulson, L.C., Theiss, F., Fietzke, A.: LEO-II - a cooperative automatic theorem prover for classical higher-order logic (System Description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 162–170. Springer, Heidelberg (2008). doi:10.1007/978-3-540-71070-7_14
6. Benzmller, C., Pease, A.: Higher-order aspects and context in SUMO. In: Jos Lehmann, I.J.V., Bundy, A. (eds.) Special issue on Reasoning with context in the Semantic Web, vol. 12–13. Science, Services and Agents on the World Wide Web (2012)
7. Blanchette, J.C., Kaliszyk, C., Paulson, L.C., Urban, J.: Hammering towards QED. J. Formalized Reasoning **9**(1), 101–148 (2016). http://dx.doi.org/10.6092/issn.1972-5787/4593
8. Blanchette, J.C., Nipkow, T.: Nitpick: a counterexample generator for higher-order logic based on a relational model finder. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010. LNCS, vol. 6172, pp. 131–146. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14052-5_11

9. Brown, C.E.: Reducing higher-order theorem proving to a sequence of SAT problems. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS (LNAI), vol. 6803, pp. 147–161. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22438-6_13
10. Chaudhri, V., Inclezan, D.: Representing states in a biology textbook. In: Leora Morgenstern, L., Patkos, T., Sloan, R. (eds.) Proceedings of 12th International Symposium on Logical Formalizations of Commonsense Reasoning. AAAI Press (2015)
11. Chaudhuri, S., Farzan, A. (eds.): Proceedings of the 28th International Conference on Computer Aided Verification. LNCS, vol. 9779–9780. Springer, Heidelberg (2016)
12. Grabowski, A., Korniłowicz, A., Naumowicz, A.: Mizar in a nutshell. J. Formalized Reasoning **3**(2), 153–245 (2010)
13. Groth, P., Simperi, E., Gray, A., Sabou, M., Krötzsch, M., Lecue, F., Flöck, F., Gil, Y. (eds.): Proceedings of the 15th International Semantic Web Conference. LNCS, vol. 9981–9982. Springer, Heidelberg (2016)
14. Hales, T., Adams, M., Bauer, G., Dang, D.T., Harrison, J., Hoang, T.L., Kaliszyk, C., Magron, V., McLaughlin, S., Nguyen, T.T., et al.: A formal proof of the Kepler conjecture. arXiv preprint (2015). arXiv:1501.02155
15. Hoder, K., Voronkov, A.: Sine qua non for large theory reasoning. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS (LNAI), vol. 6803, pp. 299–314. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22438-6_23
16. Kaliszyk, C., Urban, J.: Stronger automation for Flyspeck by feature weighting and strategy evolution. In: Blanchette, J.C., Urban, J. (eds.) PxTP 2013. EPiC Series, vol. 14, pp. 87–95. EasyChair (2013)
17. Kaliszyk, C., Urban, J.: Learning-assisted automated reasoning with Flyspeck. J. Autom. Reasoning **53**(2), 173–213 (2014)
18. Kaliszyk, C., Urban, J.: MizAR 40 for Mizar 40. J. Autom. Reasoning **55**(3), 245–256 (2015). http://dx.doi.org/10.1007/s10817-015-9330-8
19. Kinyon, M., Veroff, R., Vojtěchovský, P.: Loops with abelian inner mapping groups: an application of automated deduction. In: Bonacina, M.P., Stickel, M.E. (eds.) Automated Reasoning and Mathematics. LNCS, vol. 7788, pp. 151–164. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36675-8_8
20. Klein, G., Nipkow, T., Paulson, L.: The archive of formal proofs (2010). https://www.isa-afp.org/
21. Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 1–35. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39799-8_1
22. Kühlwein, D., Laarhoven, T., Tsivtsivadze, E., Urban, J., Heskes, T.: Overview and evaluation of premise selection techniques for large theory mathematics. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 378–392. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31365-3_30
23. Lenat, D.: CYC: a large-scale investment in knowledge infrastructure. Commun. ACM **38**(11), 35–38 (1995)
24. McCune, W.: Solution of the robbins problem. J. Autom. Reasoning **19**(3), 263–276 (1997)
25. Meng, J., Paulson, L.C.: Lightweight relevance filtering for machine-generated resolution problems. J. Appl. Logics **7**(1), 41–57 (2009)
26. Niles, I., Pease, A.: Toward a standard upper ontology. In: Welty, C., Smith, B. (eds.) Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001) (2001)

27. Paulsson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliff, G., Ternovska, E., Schulz, S. (eds.) Proceedings of the 8th International Workshop on the Implementation of Logics (IWIL-2010), Yogyakarta, Indonesia. EPiC, vol. 2 (2012)
28. Pease, A., Sutcliffe, G.: First order reasoning on a large ontology. In: Urban, J., Sutcliffe, G., Schulz, S. (eds.) Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories, pp. 61–70. No. 257 in CEUR Workshop Proceedings (2007)
29. Pease, A.: Ontology: A Practical Guide. Articulate Software Press, Angwin (2011)
30. Pease, A., Benzmüller, C.: Sigma: an integrated development environment for logical theories. AI Commun. **26**, 9–97 (2013)
31. Pease, A., Niles, I., Li, J.: The suggested upper merged ontology: a large ontology for the semantic web and its applications. In: Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web (2002)
32. Priest, G.: Paraconsistent logic. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic, vol. 6, pp. 287–393. Kluwer Academic Publishers (2002)
33. Ramachandran, D., Reagan, P., Goolsbey, K.: First-orderized ResearchCyc: expressiveness and efficiency in a common sense knowledge base. In: Shvaiko, P. (ed.) Proceedings of the AAAI Workshop on Contexts and Ontologies: Theory, Practice and Applications (C&O-2005) (2005)
34. Reynolds, A., Blanchette, J.C., Cruanes, S., Tinelli, C.: Model finding for recursive functions in SMT. In: Olivetti, N., Tiwari, A. (eds.) IJCAR 2016. LNCS (LNAI), vol. 9706, pp. 133–151. Springer, Cham (2016). doi:10.1007/978-3-319-40229-1_10
35. Schulz, S.: E - A brainiac theorem prover. J. AI Commun. **15**(2/3), 111–126 (2002)
36. Schulz, S.: System description: E 1.8. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR 2013. LNCS, vol. 8312, pp. 735–743. Springer, Heidelberg (2013). doi:10.1007/978-3-642-45221-5_49
37. Seligman, E., Schubert, T., Achutha Kiran Kumar, M.: Formal Verification: An Essential Toolkit for Modern VLSI Design. Morgan Kaufmann, San Francisco (2015)
38. Sutcliffe, G.: The CADE-23 automated theorem proving system competition - CASC-23. AI Commun. **25**(1), 49–63 (2012)
39. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. J. Autom. Reasoning. (2017, to appear)
40. Sutcliffe, G., Puzis, Y.: SRASS - a semantic relevance axiom selection system. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 295–310. Springer, Heidelberg (2007). doi:10.1007/978-3-540-73595-3_20
41. Sutcliffe, G., Urban, J., Schulz, S. (eds.): Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories, CEUR Workshop Proceedings, vol. 257 (2007)
42. Urban, J.: MPTP 0.2: design, implementation, and initial experiments. J. Autom. Reasoning **37**(1), 21–43 (2006)
43. Urban, J.: BliStr: The blind strategymaker. In: Gottlob, G., Sutcliffe, G., Voronkov, A. (eds.) Proceedings of the Global Conference on Artificial Intelligence, Tibilisi, Georgia. EPiC, vol. 36, pp. 312–319. EasyChair (2015)
44. Urban, J., Rudnicki, P., Sutcliffe, G.: ATP and presentation service for Mizar formalizations. J. Autom. Reasoning **50**(2), 229–241 (2013)

45. Urban, J., Sutcliffe, G., Pudlák, P., Vyskočil, J.: MaLARea SG1 - machine learner for automated reasoning with semantic guidance. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 441–456. Springer, Heidelberg (2008). doi:10.1007/978-3-540-71070-7_37
46. Zalta, E., Fitelson, B.: Steps toward a computational metaphysics. Australas. J. Philos. **36**(2), 227–247 (2007)