A Programmer's Text Editor for a Logical Theory: The SUMOjEdit Editor (System Description)

Adam Pease(✉)

Articulate Software, San Jose, USA
apease@articulatesoftware.com

Abstract. SUMOjEdit is a programmer's text editor for the SUO-KIF language and SUMO http://www.ontologyportal.org theory. Modern procedural programming is done in a text editor with tool support. Development of ontologies and taxonomies has often been done in graphical editors, leading many developers to employ only logics of very limited expressiveness that can be manipulated visually. Developers in the theorem proving community typically work in text editors but often without the same degree of tool support that most programmers rely on. Beginners working with SUMO make some very predictable errors in syntax, logical formulation, and use of the library of theories. Many of these errors can be flagged during editing, resulting in reduced time to become a productive developer. An editor designed for working with SUMO has the potential to aid beginners and experienced SUMO developers.

1 Introduction

The Suggested Upper Merged Ontology (SUMO) [6,9] is a logical theory stated in a higher-order logic, in SUO-KIF syntax[1]. It has roughly 20,000 constant symbols and 80,000 statements that have been written by hand since the start of the effort in the year 2000. It has been mapped by hand [7] to all 100,000 word senses in the WordNet lexicon [4] and to the Open Multilingual WordNet [3] that includes some two dozen languages. SUMO has been used as a source theory in several CASC competitions [13]. The Sigma knowledge engineering environment (SigmaKEE) [11] is the tool set employed in development of SUMO. It includes translators that translate SUMO from SUO-KIF to TPTP FOF [12], TFF0 [10] and THF [2] and interfaces to provers including E [14] and Vampire [5].

2 Motivation: Support for New Ontologists

We recently employed SUMO on a project to improve a taxonomy of customer service requests for a consumer electronics company. They had thousands of tags

[1] https://github.com/ontologyportal/sigmakee/blob/master/suo-kif.pdf.

N. Peltier and V. Sofronie-Stokkermans (Eds.): IJCAR 2020, LNAI 12167, pp. 472–479, 2020.
https://doi.org/10.1007/978-3-030-51054-1_31

that were used to classify problems, products and solutions to problems. The tags were created by customer service people, essentially forming a"folksonomy" that had considerable overlap and duplication of concepts. Because of a lack of definition of the tags, they were often reused in an inconsistent manner, since users of the tags would often not have the same understanding of their intended meaning as their author.

The project started quickly, with very little time to recruit and train developers. Three recent college graduates with Bachelor's degrees in Computer Science or Physics were employed and given one week of training based on the SUMO textbook [9]. They were then expected to begin work and ramp up their rate of constant symbol and definition authorship over the course of a month to a target of 50 concepts with definitions per week. None had any prior experience writing logic expressions. The training consisted of exercises in the SUO-KIF language and the most commonly used terms in SUMO, and use of the Sigma environment to view and test new SUMO content. The training followed the textbook [9] on SUMO. It became clear over the course of the project that although developers could load their new theories into Sigma and get feedback on errors, and were required to do so each week, that getting more immediate feedback would be valuable. Getting feedback from Sigma is essentially a batch process. If a syntax error is found and the system can't recover and interpret the rest of the statements in a file, then that error has to be corrected and then the process must start over. If instead errors could be caught at editing time, it would improve productivity.

A retrospective analysis of coding errors was performed. Given the sample size of only three developers and that this was not a controlled study, it must be considered anecdotal, but may still be informative. Many corrections were provided in person to the developers by the author of SUMO, so only email comments could be analyzed. The result was a set of 104 corrections that have been grouped into 7 categories. Only data for the first month of work was sampled as after that time basic coding errors became less frequent. Figure 1 shows the results of the analysis. The seven categories of errors are:

- *syntax error* - This refers to any number of violations of the BNF grammar of SUO-KIF [8].
- *unused var & var name* - A variable may appear in a quantifier list and never get used or a variable might be used only once. In the first case that is logically harmless but may indicate a mistake. In the second case, it's allowed if the intent of the formula is to specify some restriction about an argument to a relation, without respect to the other arguments. However, it is also often indicative of a mistake.
- *term name* - The developer uses a name for a constant that is likely misspelled and not defined elsewhere.
- *arg order or num* - A common error is to reverse the arguments to case roles or other binary relations, or to forget an argument to a higher-arity relation.
- *class vs. instance* - It is a common error for beginners to confuse an instance with a type. An example of this error would be stating that the result of

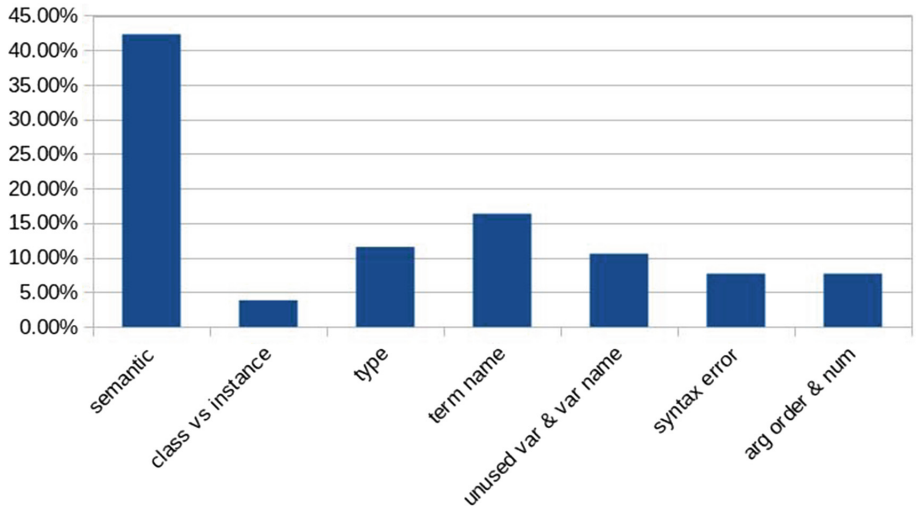

Fig. 1. Knowledge coding errors

a `Cooking` process is the class `PreparedFood` rather than an instance of `PreparedFood`.

- *type* - This refers to any error with types. All relations in SUMO have required types. Errors can include using an argument of the wrong type to a relation as well as using variables as arguments to relations that would require the variable to have two disjoint types.
- *semantic* - This refers to comments and corrections about whether the developer stated a fact that accurately reflects the domain. This sort of comment would be expected to be the most common correction. It's also one that automation cannot address. Semantic corrections become a much larger percentage as users gain facility in using the mechanics of SUMO and the SUO-KIF logical language.

3 SUMOjEdit Features and Functions

While theorem proving should in theory be able to find most contradictions, on large theories such as SUMO it may fail to find them in a reasonable amount of time. Theorem proving will also not be able to find a problem that doesn't result in a contradiction, like use of a constant symbol that only appears once. Much simpler inference procedures, coded in Java, can find many straightforward issues that would result in a logical contradiction, such as argument type violations, very quickly. In addition, such dedicated procedures can also find all such issues at once, and without the need for the user to diagnose the root cause by reading a potentially long or complicated proof. Apart from errors that result in logical contradictions, there are also many issues that can be classified as "warnings", that are indicative of possible errors, but are still logically consistent. Lastly,

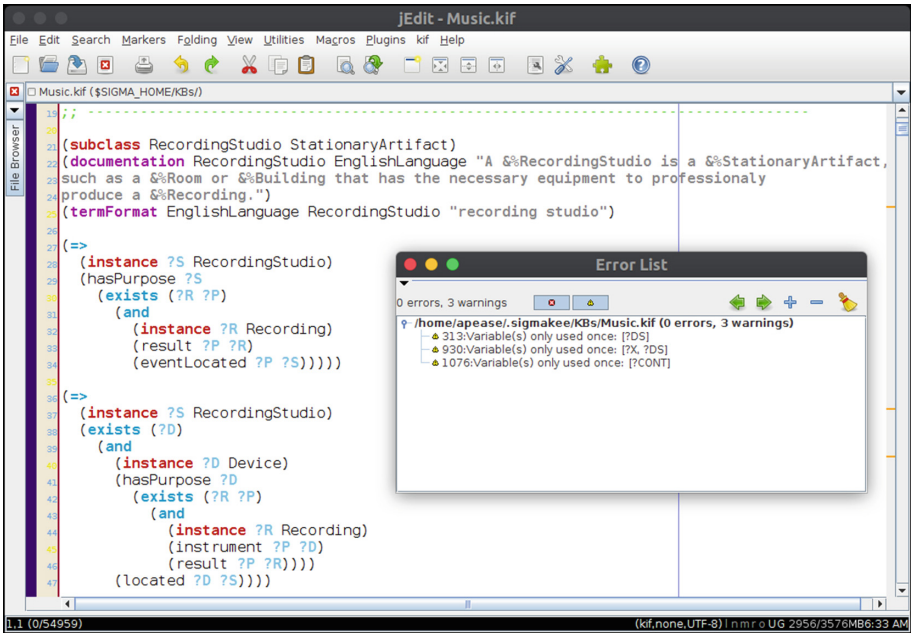

Fig. 2. An editor screen

some features may be viewed as helpful productivity enhancements, such as finding the likely definition of a constant symbol.

The features currently implemented in the editor are:

- *color coding* - there are only eight "reserved words" in the SUO-KIF language - `and`, `or`, `not`, `=>`, `<=>`, `equal`, `forall`, `exists` and they are given their own color. Comments, numbers and strings are uniquely color coded as shown in Fig. 2. The most fundamental defining relations in SUMO are given their own color. They are `instance`, `subclass`, `domain`, `domainSubclass`, `range`, `rangeSubclass`. Documentation predicates are also color-coded. These predicates are: `documentation`, `termFormat` and `format`. Lastly `True` and `False` are highlighted. Color coding often makes some errors obvious, such as a case of comment text not being preceded by a semicolon comment symbol.
- *check for errors and warnings* - There are many errors that can be checked for by the Sigma system and this feature applies all the checks to a file. A common error is for a developer to make an instance of an `Object` as the `Agent` of a `Process`. `Agent`(s) however must be sentient beings and not inanimate objects. Another common error is for a particular variable not to get used because one occurrence of it has a typo. Beginners often do copy-and-modify as an approach to writing rules and wind up with a conjunction with just one element, or a quantifier enclosing a set of literals, rather than a conjunction of literals. This function should catch all of the errors (other than "semantic") listed in Fig. 1.

- *formatting* - This is analogous to a Lisp "pretty-print" function and formats a statement to conform to the convention for SUO-KIF.
- *open browser* - This opens the browser-based Sigma system to the page showing all statements for a given term that is highlighted in the editor.
- *go to definition* - Sigma users define which files of SUO-KIF content comprise the current knowledge base either interactively or via editing a configuration file. Although formulas in different files can refer to a constant symbol, a good developer will put the basic information about a constant symbol - its class membership and documentation string at least - in the same place in one file. This function looks in order for statements involving the constant symbol to make a good guess at where to move the cursor. In order, it looks for `instance`, `subclass`, `subAttribute`, `subrelation`, `domain` and `documentation`, stopping at the first such statement.
- *theorem proving* - The user highlights a formula in the editor and this is sent as a query to a theorem prover. In other work [12] we have described the transformations needed to convert a SUO-KIF formula to TPTP FOF formula. The resulting proof is then converted back into SUO-KIF and a new editing buffer is opened that contains the proof (if found) (see Fig. 3). With this feature, it becomes easier to use automated theorem provers as proof assistants, trying to assemble lemmas that can be proven into a larger proof, adding new formulas that are required as a desired proof is built up from smaller proofs.

4 Related Work

Most prior work in this area is on editing tools for proofs using proof assistants. The focus of SUMO, and the SUMO editor, is on authoring logical theories, where the primary applications of the theories are as metadata for software engineering, including development of taxonomies and schemas.

The Logic text editor - [2] is designed for human authoring of statements and proofs. It is a set of extensions for the Markdown language. No automation support is provided for checking the statements or proofs that a user creates.

ProofGeneral [1] - [3] is an editor based on Emacs (and more recently, there is a proposed port to the Eclipse IDE) for proof assistants and supports many provers including Coq, EasyCrypt and PhoX. It includes color coding of proofs and interactive application of tactics.

Isabelle/jEdit [17][4] is an interface to the Isabelle interactive prover with many sophisticated functions. It includes color coding and the ability to display non-ASCII symbols in formulas. It does theory formatting. Many kinds of autocompletion of formulas and symbols can be performed. Proof checking can be done continuously as a background task.

[2] http://davidagler.com/teaching/logic/handouts/supplemental_material/MarkdownForSymbolicLogic.html.
[3] https://proofgeneral.github.io/.
[4] https://isabelle.in.tum.de/dist/doc/jedit.pdf.

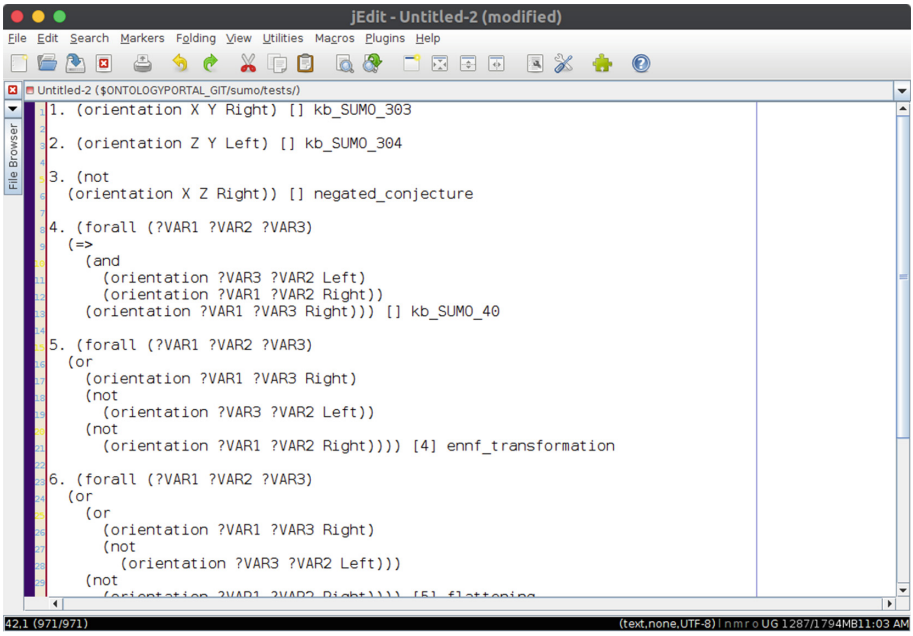

Fig. 3. A proof window

MizarMode [15,16] is focused on interactive theorem proving with the Mizar library. Like ProofGeneral, it runs on top of Emacs. It supports browsing semantically disambiguated queries, placing the cursor at the definition of a constant and sending a formula to the Mizar Proof Advisor.

Other systems include CoqIDE[5] and the Lean prover's UI[6].

5 Code and Future Work

SUMOjEdit is implemented in Java as a plugin for jEdit. It relies on the Sigma framework that is also written in Java, as well as the ErrorList jEdit plugin. It is open source and available at https://github.com/ontologyportal/SUMOjEdit.

Auto-complete would be a useful feature, especially given that there are 20,000 constant symbols in SUMO, and spelling unusual ones correctly can sometime be a challenge. A more thorough review of the proof assistant tools listed in the Related Work section will be helpful to see if the approach of proof assistants can be more thoroughly applied to work with automated provers. Finally, in this first version, much could be done to improve speed and efficiency. Load time is rather slow, and implementing jEdit's *listener* functionality would help to get the editor running while more complex operations proceed in the background and

[5] https://coq.inria.fr/refman/practical-tools/coqide.html.
[6] https://leanprover.github.io/.

then trigger the availability of the more sophisticated forms of analysis. We are currently using the system interactively to develop spatial reasoning problems and SUMO-based proofs for those problems. More experience with the system will likely yield more practical improvements.

References

1. Aspinall, D.: Proof general: a generic tool for proof development. In: Graf, S., Schwartzbach, M. (eds.) TACAS 2000. LNCS, vol. 1785, pp. 38–43. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-46419-0_3
2. Benzmüller, C., Pease, A.: Progress in automating higher-order ontology reasoning. In: Konev, B., Schmidt, R., Schulz, S. (eds.) Workshop on Practical Aspects of Automated Reasoning (PAAR-2010). CEUR Workshop Proceedings, Edinburgh (2010)
3. Bond, F., Fellbaum, C., Hsieh, S.-K., Huang, C.-R., Pease, A., Vossen, P.: A multilingual lexico-semantic database and ontology. In: Buitelaar, P., Cimiano, P. (eds.) Towards the Multilingual Semantic Web, pp. 243–258. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43585-4_15
4. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press, Cambridge (1998)
5. 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). https://doi.org/10.1007/978-3-642-39799-8_1
6. 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), pp. 2–9 (2001)
7. Niles, I., Pease, A.: Linking lexicons and ontologies: mapping WordNet to the suggested upper merged ontology. In: Proceedings of the IEEE International Conference on Information and Knowledge Engineering, pp. 412–416 (2003)
8. Pease, A.: (2009). https://github.com/ontologyportal/sigmakee/blob/master/suo-kif.pdf
9. Pease, A.: Ontology: A Practical Guide. Articulate Software Press, Angwin (2011)
10. Pease, A.: Arithmetic and inference in a large theory. In: AI in Theorem Proving (2019)
11. Pease, A., Benzmüller, C.: Sigma: an integrated development environment for logical theories. In: Proceedings of the ECAI 2010 Workshop on Intelligent Engineering Techniques for Knowledge Bases (I-KBET-2010), Lisbon, Portugal (2010)
12. Pease, A., Schulz, S.: Knowledge engineering for large ontologies with Sigma KEE 3.0. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS (LNAI), vol. 8562, pp. 519–525. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08587-6_40
13. Pease, A., Sutcliffe, G., Siegel, N., Trac, S.: Large theory reasoning with SUMO at CASC. AI Commun. Spec. Issue Pract. Aspects Autom. Reasoning **23**(2–3), 137–144 (2010)
14. Schulz, S., Cruanes, S., Vukmirović, P.: Faster, higher, stronger: E 2.3. In: Fontaine, P. (ed.) CADE 2019. LNCS (LNAI), vol. 11716, pp. 495–507. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29436-6_29
15. Urban, J.: Mizarmode - an integrated proof assistance tool for the mizar way of formalizing mathematics. J. Appl. Logic **4**(4), 414–427 (2006). Towards Comput. Aided Math

16. Urban, J., Rudnicki, P., Sutcliffe, G.: ATP and presentation service for Mizar formalizations. J. Autom. Reasoning **50**(2), 229–241 (2013). https://doi.org/10.1007/s10817-012-9269-y
17. Wenzel, M.: Isabelle/jEDIT as IDE for domain-specific formal languages and informal text documents. In: Electronic Proceedings in Theoretical Computer Science, vol. 284, pp. 71–84, November 2018. https://doi.org/10.4204/EPTCS.284.6