The Sigma Ontology Development Environment

Adam Pease
Teknowledge
1800 Embarcadero Rd
Palo Alto CA 94303
650 424-0500
650 493-2645
apease@teknowledge.com
keywords: natural language, ontology

Abstract

Abstract: We discuss the development of an environment for formal knowledge engineering. The Sigma system is an advance over previously developed systems in that it integrates a number of modern ontology development tools, which has motivated a number of research issues. Primary components include an ontology browsing and editing environment, a first order logic inference system and a natural language to logic translator. Although largely independent of any particular ontology, it supports a number of publicly available formal ontologies.

Introduction

Human knowledge is complex, and human language is very expressive. Computer systems that process natural language have been limited to a shallow level of understanding by the very complexity of the information that they are expected to handle. Conversely, systems that manage information in databases have been left to reason either in very narrow and pre-determined ways, or to perform processing tasks that must be interpreted by humans.

It is desirable to employ computer languages for representing knowledge that are as expressive as possible, and as close to the expressiveness of human language. Expressiveness does have a computational burden. In the opinion of the author, the best compromise available for many knowledge representation efforts is first order logic. While there are other choices, such as description logic and higher order logic, they represent departures from a middle ground. First order logic is undecidable, but can be made tractable for certain kinds of reasoning, which will be detailed later. Description logic (Baader, et al, 2003) has some very attractive formal properties, but is significantly less expressive than first order logic, and therefore limits the knowledge engineer's ability to express the semantic content of statements found in normal human discourse. Higher order logic (Carreno et al, 2002) on the other hand is much harder to reason with.

The Sigma system was designed to integrate several different kinds of tools for working with knowledge in first order logic.

Browsing and Editing

The Sigma system contains several different sorts of browsers for viewing formal knowledge bases. The most basic component is a term browser, which presents all the statements in which a particular term appears. The statements are sorted by argument position and then the appearance of the term in rules and non-rule statements are then shown. All the statements are hyperlinked to the terms that appear in them.

Two types of tree browser are provided. One provides an automatic graph layout. Another shows a textual hierarchy. The user can chose the term to start with, the number of "levels" that should be presented from the term, and the binary relation to chose as the predicate that links the different nodes in the graph. For example, if the user asks for a graph of the term **`IntentionalProcess`** and for the **`subclass`** relation, the system will go "up" the graph to display the term **`Process`**, and down the graph to display the terms **`Keeping`**, **`Guiding`**, **`Maintaining`** etc. The user can ask for more levels up or down the graph. Note that any relation can be chosen so for example a presentation of partonomies, or attribute hierarchies is also supported by choosing the relations **`part`** and **`subAttribute`** respectively.

The editing system is currently rather rudimentary, consisting of the ability to type formulae and have some simple syntax and other error checking performed. We plan to offer a frame-based editing system similar to, or incorporating open-source components from, the Protégé (Gennari et al, 2002) system.

We have developed some more sophisticated tools for collaborative knowledge engineering. The System for Collaborative Open Ontology Production (SCOOP) (Pease & Li, 2003) supports an automated workflow process for development of ontologies and resolution of conflicts among ontologies. The SCOOP system employs a theorem prover to detect inconsistencies among ontologies.

SCOOP addresses several types of inconsistencies. The first we term *vertical inconsistency*. This refers to when a set of ontologies are loaded and one ontology has a contradiction with another ontology on which it depends. Inconsistency of this type is never allowed, because from a

contradictory knowledge base, any proposition can be concluded to be true. SCOOP enforces a workflow process that requires a resolution to such a situation.

The second kind of inconsistency we term *horizontal inconsistency*. This refers to a situation in which knowledge products created by different knowledge engineers, and which do not have a mutual dependency, are mutually contradictory. SCOOP alerts developers to such a situation, but does not require that it be eliminated. It is possible for two knowledge bases to represent different perspectives, which are contradictory, but locally valid.

We employ the KIF (Genesereth, 1991) language and have a translator to and from DAML (Hendler & McGuinness, 2000). A translator to and from Protégé (Gennari et al, 2002) is partially completed at this time.

Language Generation

Each statement can also be presented as a natural language paraphrase similar to (Sevcenko, 2003). Note that we also use Sevcenko's language templates for English and Czech. Each term in an ontology can be given a natural language presentation form, indexed by the human language. For example, the SUMO term DiseaseOrSyndrome can be presented as "disease or syndrome" and an Italian presentation would be shown as "malattia o sindrome". Combined with the English understanding mechanisms described below, this gives us a very rudimentary language translation capability.

Although presentation of terms is straightforward, presentation of statements is more complicated, and roughly patterned after C-language printf statements. For example the relation "part", which states that one object is a part of another, has the corresponding language generation template "%1 is %n a part of %2". The first argument to the logical relation is substituted for %1 etc. Note that this substitution is recursive, so that complex statements with nested formulae can be translated effectively. The %n signifies that if the statement is negated, that the negation operator for the appropriate human language should be inserted in that position.

We currently have language templates for English, Czech, German and Italian. Some work has been done on Hindi, Telugu, Tagalog and Russian.

Natural Language Understanding

We take a restricted language approach to natural language understanding. Deep understanding of unrestricted human languages is too hard with present technology. Our approach is to create a restricted version of English, which is grammatical but more limited than true natural language. Statements must be present tense singular. Although different tense and number can be accepted, the system paraphrases such sentences into a present tense singular form. Ambiguous word choices default to the most popular sense of the word, with an escape mechanism for the user to select a different sense. Anaphoric references are handled with a simple mechanical algorithm. Some kinds of quantification (every, some, all) can be handled.

The current system is capable of taking simple English sentences and translating them into KIF, using terms from our SUMO upper ontology (described below). We have mapped all 100,000 WordNet (Miller, et al, 1993) word senses (synsets) to SUMO, one at a time, by hand over the period of a year (Niles & Pease, 2003). This endows our natural language translation system with a very large vocabulary.

Our approach is presented more fully in (Murray, Pease & Sams, 2003).

Inference

One basic issue is that of allowing variables in the predicate position. While the general case of this results in expressions being beyond first order logic, there is a slightly more restricted case, which is first order. A statement with a variable in the predicate position is only higher order if a quantifer ranges of all possible relations, rather than the finite set of relations in a particular knowledge base. Since prepending another relation in front of every statement makes everything first order for such a restricted practical case, this therefore must mean that the statement wasn't really higher-order in the first place.

There is also an issue with prohibiting statements as arguments to relations. When arguments to relations are statements, then such a statement is higher order. We could either have a quote operator or define a Statement class and require all predicates have their argument types defined (which would allow us to know in advance whether an argument is a Statement, and therefore needs to be implicitly quoted). A statement that has this issue is

```
(believes John (likes Bob Mary))
```

Currently, such statements are quoted, and therefore become opaque first-order objects rather than statements.

Another issue is row variables, for use in variable arity relations. One proposed version of KIF (Hayes & Menzel, 2001) has what are alternately called row, or sequence variables. They are denoted by the '@' symbol in KIF statements. They are analogous to the lisp @REST variable. This is not first order if the number of arguments it can "swallow up" is infinite. However, if row variables have a definite number of arguments, it can be treated like a macro, and becomes first order. For example,

```
(=>
   (and
      (subrelation ?REL1 ?REL2)
      (holds ?REL1 @ROW))
   (holds ?REL2 @ROW))
```

would become

```
(=>
```

```
   (and
      (subrelation ?REL1 ?REL2)
      (holds ?REL1 ?ARG1))
   (holds ?REL2 ?ARG1))

(=>
   (and
      (subrelation ?REL1 ?REL2)
      (holds ?REL1 ?ARG1 ?ARG2))
   (holds ?REL2 ?ARG1 ?ARG2))
```

etc.

Also note that this "macro" style expansion has the problem that unlike the true semantics of row variables, that it is not infinite. If the macro processor only expands to five variables, there is a problem if the knowledge engineer uses a relation with six. Because of that, Sigma's syntax checker must prohibit relations with more arguments than the row variable preprocessor expands to.

Default reasoning

Given the example KB:

```
(=>
  (instance ?X Bird)
  (canFly ?X))

(=>
  (instance ?X Penguin)
  (not
    (canFly ?X)))

(subclass Penguin Bird)
```

We would like the second rule to override the first. The general case of default reasoning is very difficult (Schlechta, 1997). However, we believe that there is an easy way to address the most common case and get the utility we need. Given a relation

```
(exception <formula-specific> <formula2-general>)
```

we can implement a macro that modifies the first formula above to exclude the exception specified and generate the KB

```
(=>
  (and
    (instance ?X Bird)
    (not (instance ?X Penguin))
  (canFly ?X))

(=>
  (instance ?X Penguin)
  (not
    (canFly ?X)))

(subclass Penguin Bird)
```

Of course, this is only easy in the specific case where two rules have opposite conclusions and the antecedent of one subsumes the other. It's possible that in easy cases we might be able to have Sigma even generate the **(exception**... clause that the macro will use.

The algorithm for the macro simply would be, given the **(exception**... statement, to negate the antecedent of the <formula-specific> and add it as a conjunction to the <formula2-general>. Like other macros, this has implications for proof presentation that are also unresolved.

Proof Presentation

Sigma includes several options for proof presentation. Despite the fact that most textbooks present proofs as linear structures, proofs are trees. The same sub-proof may be used several times in reaching different intermediate conclusions. While this is a byproduct of automated reasoning, it is rarely useful to the human user. Therefore, Sigma eliminates redundant paths of reasoning in order to create a linear proof.

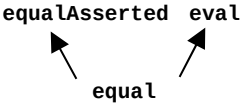
Sigma also has options for how it presents multiple proofs. Often, the same proof, with the same proof steps, can be reached via a different search order. This means that although two tree-structured proofs may be different, they may work out to be identical when redundant paths are removed and a linear proof structure generated. Sigma supports an option for hiding such duplicate proofs.

On additional option is to suppress proofs that may have different steps but which lead to the same answer. Sometime it is useful to a knowledge engineer to see these alternate proofs, and sometime not.

Reasoning with Equality

One problem in a first order theorem prover with procedural attachments for arithmetic is that the equality operator masks normal unification and inference. Currently Sigma converts the SUMO term 'equal' to '=' when sending assertions and queries to the theorem prover and then converts back for proof formatting. We are experimenting with changing the name to 'eval' and mapping that string to the theorem prover's '='.

Our current idea is to have three predicates, one, which handles evaluation, one, which handles inference, and another, which combines the two.



The following examples of assertions and queries should make clear how this will work.

assert:

```
(equal (CardinalityFn Continent) 7)
```

macro expands to:

```
(eval (CardinalityFn Continent) 7)
(equalAssigned (CardinalityFn Continent) 7)
```

query:

```
(eval (CardinalityFn Continent) ?X)
```
result:
```
(CardinalityFn Continent)
```

query:
```
(equalAssigned (CardinalityFn Continent) ?X)
```
result:
```
7
```

query:
```
(equal (CardinalityFn Continent) ?X)
```
macro expands to:
```
(eval (CardinalityFn Continent) ?X)
(equalAssigned (CardinalityFn Continent) ?X)
```
result:
```
(CardinalityFn Continent)
7
```

query:
```
(eval (AdditionFn 2 3) ?X)
```
result:
```
5
```

query:
```
(equalAssigned (AdditionFn 2 3) ?X)
```
result:
```
[no]
```

query:
```
(equal (AdditionFn 2 3) ?X)
```
macro expands to:
```
(eval (AdditionFn 2 3) ?X)
(equalAssigned (AdditionFn 2 3) ?X)
```
result:
```
5
```

The formal semantics of the macro is

```
(=>
  (equal ?X ?Y)
  (eval ?X ?Y))

(=>
  (equal ?X ?Y)
  (equalAssigned ?X ?Y))
```

Ontology

The Sigma system has been designed to be as independent of a particular ontology as possible, but there are many features that are available when a standard ontology is used. Current features are primarily to do with error checking. Knowing a standard method for defining argument type restrictions, class-subclass relations and documentation allows us to alert the user when such statements are conflicting or not present.

We employ the Suggested Upper Merged Ontology (SUMO) (Niles & Pease, 2001) as the system's standard ontology. SUMO has been released to the public for free since its first version in December of 2000. Now in its 47th version, the SUMO has been reviewed by hundreds of people and subject to formal validation with a theorem prover, to ensure that it is free of contradictions. The SUMO contains roughly 1000 terms and 4000 statements, of which 750 are rules. As mentioned above, it has been mapped by hand to the WordNet lexicon, which has acted as an additional check on completeness and coverage.

A number of domain ontologies have been created that extend SUMO and can be used in the Sigma system. They include

- A Quality of Service ontology, covering computer systems and networks
- An ecommerce services ontology
- An ontology of biological viruses
- A financial ontology
- An ontology of terrain features
- Ontologies of weapons of mass destruction and terrorism
- An ontology of government and governmental organizations
- Taxonomies of the periodic table of the elements and industry types

Related Work

There have been a number of ontology editing environments created including Ontolingua (Farquhar et al, 1996), Ontosaurus (Swartout et al, 1996), Protégé (Gennari et al, 2002), and Cyc (Lenat, 1995). Ontosaurus and Cyc both have inference components. Cyc is the only system to contain a standard ontology. Cyc also contains a natural language component although little or no public information about that work is available.

The GKB editor (Paley et al, 1997) is an example of a graphical ontology editor. SNARK (Stickel et al, 1994), and Otter (Kalman, 2001) are examples of theorem provers. Many theorem provers have been tested on the CADE/TPTP (Sutcliffe et al, 2002) competitions. However, limited efforts have been made to apply formal first order theorem proving to expert system and common sense reasoning on large knowledge bases. The Sigma system brings all these types of components together, and in the process, addresses a number of research issues that have not been evident in use of these separate components.

Acknowledgements

The author would like to thank Randy Schulz for his efforts on early versions of this system.

References

Baader, F., Calvanese, D., McGuinness, Nardi, D., and Patel-Schneider, P., (eds). (2003) The Description Logic

Handbook Theory, Implementation and Applications, 574 pp. ISBN: 0521781760

Carreno, V., Munoz, C., and Tahar, S., (eds.) (2002). Theorem Proving in Higher Order Logics. Proceedings of the 15th International Conference, TPHOLs 2002, Hampton, VA, USA, August 20-23, 2002. Springer-Verlag.

Farquhar, A., Fikes, R., and Rice, J., (1996). The Ontolingua Server: a Tool for Collaborative Ontology Construction. Proceedings of Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop. Available at http://ksi.cpsc.ucalgary.ca/KAW/KAW96/farquhar/farquhar.html#RTFToC15

Gennari, J., M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, S. W. Tu (2002). The Evolution of Protégé: An Environment for Knowledge-Based Systems Development. Stanford SMI technical report SMI-2002-0943 .http://smi-web.stanford.edu/pubs/SMI_Abstracts/SMI-2002-0943.html

Genesereth, M., (1991). ``Knowledge Interchange Format", In Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning, Allen, J., Fikes, R., Sandewall, E. (eds), Morgan Kaufman Publishers, pp 238-249.

Hayes, P., and Menzel, C., (2001). A Semantics for Knowledge Interchange Format, in Working Notes of the IJCAI-2001 Workshop on the IEEE Standard Upper Ontology.

Hendler, J., and McGuinness, D., (2000), The DARPA Agent Markup Language, IEEE Intelligent Systems, 15 (6) (November), 67-73. Available: http://www.ksl.stanford.edu/people/dlm/papers/ieee-daml01-abstract.html

Kalman, J. K. (2001) Automated Reasoning with OTTER. Rinton Press.

Lenat, D. (1995). "Cyc: A Large-Scale Investment in Knowledge Infrastructure." Communications of the ACM 38, no. 11 (November).

MacGregor, R., (1991). The Evolving Technology of Classification-Based Knowledge Representation Systems. In Principles of Semantic Networks: Explorations in the Representation of Knowledge, ed J. Sowa, 385-400. San Francisco, Calif., Morgan Kaufmann

Miller, G., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. "Introduction to WordNet: An On-line Lexical Database." 1993.

Murray, W., Pease, A., and Sams, M. (2003). Applying Formal Methods and Representations in a Natural Language Tutor to Teach Tactical Reasoning. To appear.

Niles, I & Pease A., (2001) "Towards A Standard Upper Ontology." In Proceedings of Formal Ontology in Information Systems (FOIS 2001), October 17-19, Ogunquit, Maine, USA, pp 2-9. See also http://ontology.teknowledge.com

Niles, I., & Pease, A., (2003). Mapping WordNet to the SUMO Ontology. To appear. See also http://ontology.teknowledge.com:8080/rsigma/nilesWordNet.pdf

Paley, S.M., Lowrance, J.D., and Karp, P.D. (1997) "A Generic Knowledge-Base Browser and Editor," In Proceedings of the 1997 National Conference on Artificial Intelligence, Providence, RI.

Pease, A., and Li, J., (2003). Agent-Mediated Knowledge Engineering Collaboration, in Proceedings of the AAAI Spring Symposium on Agent Mediated Knowledge Management. Stanford, CA March 24-26. To appear.

Pease, A., Niles, I., Li, J., (2002), 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. http://projects.teknowledge.com/AAAI-2002/Pease.ps

Schlechta, K., (1997). Nonmonotonic Logics: Basic Concepts, Results, and Techniques (Lecture Notes in Artificial Intelligence) by Karl 243 pp. Springer Verlag pub. ISBN: 3540624821

Stickel, M., R. Waldinger, M. Lowry, T. Pressburger, and I. Underwood. (1994) Deductive composition of astronomical software from subroutine libraries. Proceedings of the Twelfth International Conference on Automated Deduction (CADE-12), Nancy, France, June 1994, 341-355. See also http://www.ai.sri.com/~stickel/snark.html.

Sutcliffe G., Suttner C.B., Pelletier F.J. (2002), The IJCAR ATP System Competition, Journal of Automated Reasoning 28(3), pp.307-320.

Swartout, B., Patil, R., Knight, K. and Russ, T. (1996). Ontosaurus: a tool for browsing and editing ontologies, Gaines, B.R. and Musen, M.A., Ed. Proceedings of Tenth Knowledge Acquisition Workshop. pp.69-1-69-12 (http://ksi.cpsc.ucalgary.ca/KAW/KAW96/swartout/ontosaurus_demo.html)