Classifying Scope Ambiguities

MALTE GABSDIL, *Department of Computational Linguistics, University of the Saarland, Germany. E-mail: gabsdil@coli.uni-sb.de*

KRISTINA STRIEGNITZ, *Department of Computational Linguistics, University of the Saarland, Germany. E-mail: kris@coli.uni-sb.de*

Abstract

We describe the architecture and implementation of a system which compares and sorts semantic representations of natural language input with respect to equivalence of logical content and context change potential. By using automated theorem proving we compute a graph-like structure which represents the relationships that hold between different readings of a given sentence.

The information encoded in the graph-structure can be useful for discourse processing systems (such as Johan Bos' DORIS system [4]), where knowledge about the relative logical strength of readings might be used to reduce the number of readings that have to be considered during processing.

The system relies heavily on existing implementations and code available via the internet. These are integrated and put to the desired use by a Prolog interface. By illustrating the architecture of this system, we want to argue that it is possible to build rather complex systems involving multiple levels of linguistic processing without having to spend an unreasonably large amount of time on the implementation of basic functionalities.

Keywords: scope ambiguity, automated theorem proving

1 Introduction

Scopal ambiguity arising from the interaction of several quantifiers in one sentence has been a major topic of investigation in formal semantics. But although it has been shown that all possible permutations of the quantifiers need not give rise to well-formed readings, in principle the number of readings is exponential in the number of quantifiers in a sentence. Since each of the readings has to be considered separately, further semantic processing, which usually involves inference, can be quite inefficient.

One possible solution to this problem is to use underspecified semantic representations. However, inference on underspecified structures is still in its infancy. In contrast, "normal" deduction on formulas of first order logic is well understood. There is a long tradition of research on automated theorem proving in computer science and many efficient implementations are available. So, it seems a good idea to make use of this well developed technology for semantic processing (see also [2]).

But this means that the problem of an exponentially large number of inference tasks has to be tackled differently. And in fact, it can be mitigated by exploiting equivalence and entailment relationships that may hold between different readings. Logically equivalent readings can be collapsed into one group. Everything that can be proved to be true for one arbitrary member of such a group should also hold for all others (though, as we shall see below, members of a group can differ in their *dynamic* potential). Furthermore these groups can be ordered according to the entailment relations that hold between members of different groups. This can be useful for further reducing the number of readings that have to be considered during processing, since certain results for a specific reading can be transferred to all stronger readings or to all weaker readings.

We describe a system which takes natural language discourse as input, enumerates all possible readings, and orders them in a DAG-like structure. This graph gives a clear representation of equivalence of readings and the entailment relationships that hold between different readings.

The linguistic modules, grammar, semantic construction, and enumeration of readings, are based on code presented in [1], which is available via the World Wide Web. The comparison of readings as to their logical equivalence is formulated as entailment problems which are then send to theorem provers.

By using existing systems and assembling them in a "plug and play" fashion, we were able to build a system which makes use of state of the art techniques and covers several levels of linguistic processing. And we could do so without having to devote unreasonably much time to the development of the basic modules, before being able to turn to the subjects we were originally interested in.

The result is a tool which, due to its modularity, is well suited for further extension. Integration with other systems (for example the DORIS system [4]) is planned. Also, we think that it should be useful for educational purposes, for example to illustrate the effect of scopal ambiguity and demonstrate how logically equivalent sentences may differ in their dynamic potential.[1]

2 Architecture

The System consists of three different components that are linked together by a Prolog interface (see Figure 1).

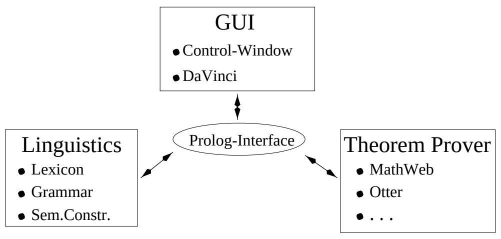

FIG. 1. System architecture.

The linguistics module is based on code presented in [1]. It comprises a 150 word lexicon and a grammar for a small fragment of English. As for semantic construction, there are several different formalisms included: Keller-Storage [7] and Hole Semantics [3] applied to predicate logic as well as DRT. All these formalisms come with their own interface to the lexicon and can therefore use the same grammar.

The second module consists of interfaces to various different theorem provers. First, a simple tableau-prover again taken from [1]. Second, local installations of the reso-

[1] An anonymous reviewer suggested that the tool could also be of use in the field of formal approaches to social and organizational theories. Although a first glance at the subject revealed interesting possibilities for application, we did not further pursue this idea.

lution based provers Otter [8], Bliksem [9] and Spass [10]. Finally, we can make use of the MathWeb society of distributed theorem proving agents [5].

The third module of the system is the user interface, which again is subdivided into two parts. First, a control panel which accepts user input and offers menus for selecting different semantic formalisms, theorem provers, and sorting strategies. Second, an interface to the DaVinci system developed at the University of Bremen [6]. DaVinci is a tool for representing trees and lattice-structures. It is well suited for our purposes because we represent implication relations between readings in terms of graphs. By clicking on the nodes of such a graph, one can see the readings and (according to the chosen semantic formalism) DRSs that are associated with them.

The Prolog interface has to read in the user input, interpret the chosen options, and coordinate the work of the different modules accordingly. Furthermore it comprises the core of our system, i.e. the sorting algorithm which orders a set of readings according to the entailment relationships that hold between them.

The first processing step is the linguistic analysis of the natural language input. The result is a list of semantic representations, which may be logical formulas or DRSs, depending on which semantic formalism was specified.

Then the readings are sorted. For this purpose DRSs are translated to formulas of first order logic. Our algorithm arranges them in a graph structure expressing the entailment relationships that hold between the readings.

In the following section we describe the sorting algorithm and the resulting representation in more detail.

3 Sorting of Readings

The main idea is to find out whether two readings r1 and r2 are related through an entailment relationship by asking an automated theorem prover to prove “r1 implies r2” and “r2 implies r1”. If proofs are found, we can conclude that the corresponding entailment relationship holds.

In principle, this strategy will lead to $n(n-1)$ proofs (where n is the number of readings), which have to be done to determine the entailment relationship between each pair of readings. The worst case scenario will always be this, but by exploiting equivalence of readings and the transitivity of the entailment relationship it is possible to often get along with fewer proofs.

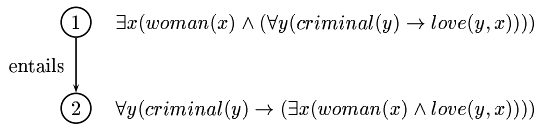
1
$\exists x(woman(x) \wedge (\forall y(criminal(y) \rightarrow love(y,x))))$
entails
2
$\forall y(criminal(y) \rightarrow (\exists x(woman(x) \wedge love(y,x))))$

FIG. 2. Entailment relationships represented as a graph.

We chose a graph structure to represent an ordered set of readings, because it gives us easy access to all readings which are equivalent, stronger, weaker or unrelated compared to a specific reading. Nodes in this graph represent groups of equivalent readings. Entailment between members of different groups is encoded via the dominance relation between nodes. This works like this: Let r1 and r2 be two readings which are associated with nodes n1 and n2 respectively. Then r1 implies r2 if n1 dominates n2. For illustration Figure 2 shows a simple entailment graph for the two readings of *Every criminal loves a woman.*

The sorting algorithm starts out with one node for each reading and the assumption that they are not connected by any edges. Its task is then to determine the appropriate way of connecting these nodes by edges, i.e. find the entailment relationships between the readings associated with them.

As said above we do not want to do a pairwise comparison of all readings. By the following methods it is possible to save proofs: Whenever the readings of two nodes are found to be equivalent, they are merged into one. In this way it is possible to reduce the total numbers of proofs to be done because the number of nodes is reduced steadily. Comparing two groups of equivalent readings boils down to comparing one representative of the first group with one representative of the second (hereby avoiding unnecessary comparisons to distinct readings which are already known to be equivalent). Further proofs are saved by exploiting transitivity during the sorting process. Imagine we found that the readings in node n1 are stronger than the readings of node n2. If we already know that the readings of node n2 in turn are stronger than the readings of node n3, there is no need to compare nodes n1 and n3, since it follows that the readings of n1 must be stronger than the readings of n3.

Our system provides two sorting algorithms which differ in the way they choose the proofs which have to be done next (i.e. which nodes have to be compared next). The first algorithm follows an arbitrary order in processing the nodes. This results in a kind of bubble-sort behavior, where the sequence in which items are sorted into the graph is arbitrary. The second algorithm uses a cleverer approach. It tries to detect equivalent nodes as soon as possible, which reduces the number of proofs to be done. Dominance information is used to "guide" the search for possibly equivalent nodes. If, for example, the readings of node n1 are stronger than the readings of node n2, the algorithm first checks all other readings stronger than n2, since one of them might be equivalent to n1. In practice, the second algorithm often needs less proofs than the first. However, it is less efficient, so that it is usually only slightly faster.

We now want to illustrate by means of an example how collapsing logically equivalent readings into one group reduces the number of proofs necessary to compute an entailment graph.

18 different readings are derived for the example given in 3.1.

(3.1) Every owner of a hash bar gives every criminal a big kahuna burger.

These readings are clustered into 11 groups. In the worst case, a pairwise comparison of 18 different readings would involve $n(n-1) = 18*17 = 306$ proofs. By grouping equivalent readings together and exploiting transitivity, the system can reduce this number by more than half to 136.

The graph (see Figure 3) representing the entailment relationships between these groups has a rather interesting form as it falls into two parts, which means that the readings represented in the left lattice stand in no entailment relation to the readings on the right side, and vice versa. In particular, there is no single weakest reading and no single strongest reading either. Note that while this phenomena is known for sentences containing generalized quantifiers, Example 3.1 contains only the standard first-order quantifiers. In our example the structure is due to the fact that *a hash bar* can either be in the restriction of *every owner* or can outscope it.

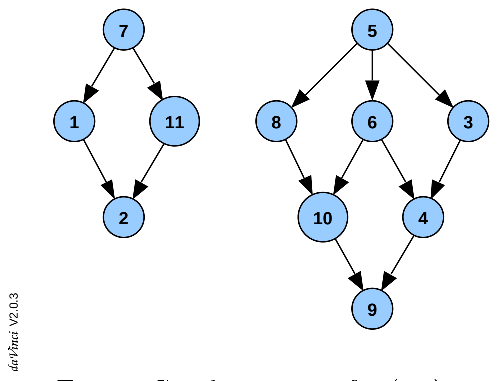

FIG. 3. Graph structure for (3.1).

4 Logical Equivalence and Dynamic Potential

We represent the dynamic potential of a semantic description as the set of discourse referents that are accessible for a continuation of the discourse, i.e. the discourse referents introduced in the top level DRS.

It is worth emphasizing that readings may differ in their dynamic potential, although they are logically equivalent. Consider the following example.

(4.1) No criminal does not love a woman.

Due to scopal ambiguity between the negation and the two quantifiers, this sentence has (among others) the two readings displayed in Figure 4.

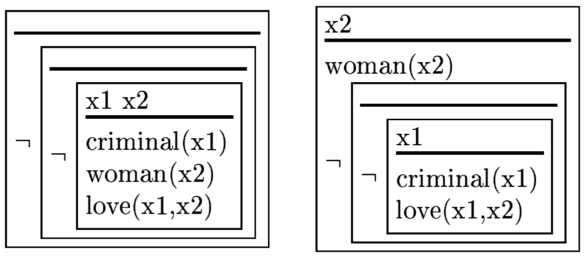

FIG. 4. Two readings of (4.1).

These readings are logically equivalent, but differ in terms of their dynamic potential. Assuming that negation restricts accessibility, only the second reading can be continued by *Her name is Mia* meaning that every criminal loves one specific woman, namely Mia. In case of the first reading this is not possible, because the discourse referent introduced by *a woman* is not accessible.

5 Conclusion

We built a system that orders readings of scopally ambiguous sentences with respect to logical entailment and dynamic potential, using a graph-like representation to improve efficiency.

By freely using available implementations we could base our system on advanced techniques in natural language processing and theorem proving.

We are confident that the described system can be useful for educational purposes. Another interesting experiment would be to incorporate it into other discourse processing systems in order to cut down the total number of readings. Also it should be interesting to investigate, how it could be used for selecting preferred readings of sentences. In the case of presupposition, for instance, it has been argued that always the strongest reading is preferred.

Acknowledgements

Thanks to all anonymous reviewers who helped to improve and clarify the presentation of this paper. Furthermore, we would like to thank Patrick Blackburn, Johan Bos, and Alexander Koller for many helpful comments.

References

[1] P. Blackburn and J. Bos. *Representation and Inference for Natural Language. A First Course in Computational Semantics.* `http://www.coli.uni-sb.de/~bos/comsem/`, 1998.

[2] P. Blackburn, J. Bos, M. Kohlhase, and H. de Nivelle. Inference and Computational Semantics. In Bunt and Thijsse, editors, *IWCS-3*, Tilburg, NL, 1999.

[3] J. Bos. Predicate Logic Unplugged. In *10th Amsterdam Colloquium*, 1995.

[4] J. Bos. The DORIS-System. `http://www.coli.uni-sb.de/~bos/atp/doris.html`, 1998.

[5] A. Franke and M. Kohlhase. System Description: MathWeb, an Agent-Based Communication Layer for Distributed Automated Theorem Proving. In *Cade '99*, 1999.

[6] Fröhlich, M. and Werner, M. The daVinci System. `http://www.informatik.uni-bremen.de/~davinci/`, 1998.

[7] W. Keller. Nested Cooper Storage: The proper treatment of quantification in ordinary noun phrases. In Reyle and Rohrer, editors, *Natural Language Parsing and Linguistic Theory.* D. Reidel Publishing Company, 1988.

[8] W. McCune. *Otter Reference Manual and Guide.* `http://www-unix.mcs.anl.gov/AR/otter/`, 1994.

[9] Hans de Nivelle. Bliksem User Manual. `http://www.mpi-sb.mpg.de/~nivelle/programs/bliksem/doc.html`.

[10] The SPASS Development Team. SPASS - a first-order theorem prover and a CNF-translator. `http://spass.mbi-sb.mpg.de`.

Received 30 September 1999.