Efficient solving and exploration of scope ambiguities

Alexander Koller and **Stefan Thater**
Department of Computational Linguistics
Saarland University, Saarbrücken, Germany
{koller,stth}@coli.uni-sb.de

Abstract

We present the currently most efficient solver for scope underspecification; it also converts between different underspecification formalisms and counts readings. Our tool makes the practical use of large-scale grammars with (underspecified) semantic output more feasible, and can be used in grammar debugging.

1 Introduction

One of the most exciting recent developments in computational linguistics is that large-scale grammars which compute semantic representations are becoming available. Examples for such grammars are the HPSG English Resource Grammar (ERG) (Copestake and Flickinger, 2000) and the LFG ParGram grammars (Butt et al., 2002); a similar resource is being developed for the XTAG grammar (Kallmeyer and Romero, 2004).

But with the advent of such grammars, a phenomenon that is sometimes considered a somewhat artificial toy problem of theoretical semanticists becomes a very practical challenge: the presence of scope ambiguities. Because grammars often uniformly treat noun phrases as quantifiers, even harmless-looking sentences can have surprisingly many readings. The median number of scope readings for the sentences in the Rondane Treebank (distributed with the ERG) is 55, but the treebank also contains extreme cases such as (1) below, which according to the ERG has about 2.4 trillion (10^{12}) readings:

(1) Myrdal is the mountain terminus of the Flåm rail line (or Flåmsbana) which makes its way down the lovely Flåm Valley (Flåmsdalen) to its sea-level terminus at Flåm. (Rondane 650)

In order to control such an explosion of readings (and also to simplify the grammar design process), the developers of large-scale grammars typically use methods of packing or *underspecification* to specify the syntax-semantics interface. The general idea is that the parser doesn't compute all the individual scope readings, but only a compact *underspecified description*, from which the individual readings can then be extracted at a later stage of processing – but the underspecified description could also be used as a platform for the integration of lexical and context information, so as to restrict the set of possible readings without enumerating the wrong ones.

Such an approach is only feasible if we have access to efficient tools that support the most important operations on underspecified descriptions. We present `utool`, the Swiss Army Knife of Underspecification, which sets out to do exactly this. It supports the following operations:

1. enumerate all scope readings represented by an underspecified description;
2. check whether a description has any readings, and compute how many readings it has without explicitly enumerating them;
3. convert underspecified descriptions between different underspecification formalisms (at this point, Minimal Recursion Semantics (Copestake et al., 2003), Hole Semantics (Bos, 1996), and dominance constraints/graphs (Egg et al., 2001; Althaus et al., 2003)).

Proceedings of the ACL Interactive Poster and Demonstration Sessions,
pages 9–12, Ann Arbor, June 2005.

Our system is the fastest solver for underspecificied description available today; that is, it is fastest at solving Task 1 above (about 100.000 readings per second on a modern PC). It achieves this by implementing an efficient algorithm for solving dominance graphs (Bodirsky et al., 2004) and caching intermediate results in a chart data structure. To our knowledge, it is the *only* system that can do Tasks 2 and 3. It is only because `utool` can compute the number of readings without enumerating them that we even know that (1) has trillions of readings; even `utool` would take about a year to enumerate and count the readings individually.

`utool` is implemented in C++, efficient and portable, open source, and freely downloadable from `http://utool.sourceforge.net`.

2 Technical Description

2.1 Solving dominance graphs

At the core of `utool` is a solver for *dominance graphs* (Bodirsky et al., 2004) – graph representations of *weakly normal dominance constraints*, which constitute one of the main formalisms used in scope underspecification (Egg et al., 2001; Althaus et al., 2003). Dominance graphs are directed graphs with two kinds of edges, *tree edges* and *dominance edges*. They can be used to describe the set of all trees into which their tree edges can be embedded, in such a way that every dominance edge in the graph is realised as reachability in the tree. Dominance graphs are used as underspecified descriptions by describing sets of trees that are encodings of the formulas of some language of semantic representations, such as predicate logic.

Fig. 1 shows an example of a constraint graph for the sentence "every student reads a book." It consists of five *tree fragments* – sets of nodes that are connected by (solid) tree edges – which are connected by dominance edges (dotted lines). Two of the fragments have two *holes* each, into which other fragments can be "plugged". The graph can be embedded into the two trees shown in the middle of Fig. 1, which correspond to the two readings of the sentence. By contrast, the graph cannot be embedded into the tree shown on the right: a dominance edge stipulates that "read$_{x,y}$" must be reachable from "some$_y$", but it is not reachable from "some$_y$" in the tree. We call the two trees into which the graph can be embedded its *solutions*.

The Bodirsky et al. algorithm enumerates the solutions of a dominance graph (technically, its *solved forms*) by computing the set of its *free fragments*, which are the fragments that can occur at the root of some solution. Then it picks one of these fragments as the root and removes it from the graph. This splits the graph into several connected subgraphs, which are then solved recursively.

This algorithm can call itself for the same subgraph several times, which can waste a lot of time because the set of all solutions was already computed for the subgraph on the first recursive call. For this reason, our implementation caches intermediate results in a chart-like data structure. This data structure maps each subgraph G to a set of *splits*, each of which records which fragment of G should be placed at the root of the solution, what the subgraphs after removal of this fragment are, and how their solutions should be plugged into the holes of the fragment. In the worst case, the chart can have exponential size; but in practice, it is much smaller than the set of all solutions. For example, the chart for (1) contains 74.960 splits, which is a tiny number compared to the 2.4 trillion readings, and can be computed in a few seconds.

Now solving becomes a two-phase process. In the first phase, the chart data structure is filled by a run of the algorithm. In the second phase, the complete solutions are extracted from the chart. Although the first phase is conceptually much more complex than the second one because it involves interesting graph algorithms whose correctness isn't trivial to prove, it takes only a small fraction of the entire runtime in practice.

Instead of enumerating all readings from the chart, we can also compute the number of solutions represented by the chart. For each split, we compute the numbers of solutions of the fragment sets in the split. Then we multiply these numbers (choices for the children can be combined freely). Finally, we obtain the number of solutions for a subgraph by adding the numbers of solutions of all its splits. This computation takes linear time in the size of the chart.

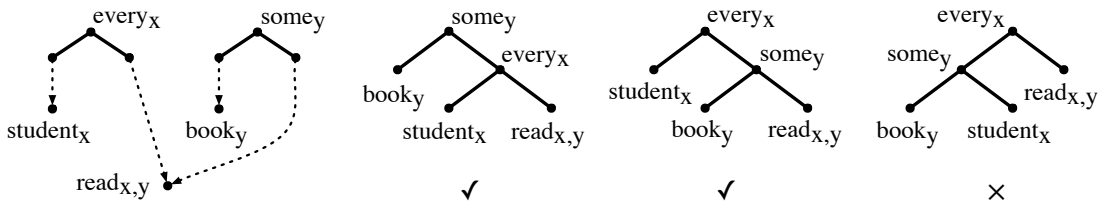

Figure 1: A dominance graph (left), two solutions (middle) and a non-solution (right).

2.2 Translating between formalisms

One of the most significant obstacles in the development of tools and resources for scope underspecification is that different resources (such as grammars and solvers) are built for different underspecification formalisms. To help alleviate this problem, `utool` can read and write underspecified descriptions and write out solutions in a variety of different formats:

- *dominance graphs*;
- descriptions of *Minimal Recursion Semantics*;
- descriptions of *Hole Semantics*.

The input and output functionality is provided by *codecs*, which translate between descriptions in one of these formalisms and the internal dominance graph format. The codecs for MRS and Hole Semantics are based on the (non-trivial) translations in (Koller et al., 2003; Niehren and Thater, 2003) and are only defined on *nets*, i.e. constraints whose graphs satisfy certain structural restrictions. This is not a very limiting restriction in practice (Flickinger et al., 2005). `utool` also allows the user to test efficiently whether a description is a net.

In practice, `utool` can be used to convert descriptions between the three underspecification formalisms. Because the codecs work with concrete syntaxes that are used in existing systems, `utool` can be used as a drop-in replacement e.g. in the LKB grammar development system (Copestake and Flickinger, 2000).

2.3 Runtime comparison

To illustrate `utool`'s performance, we compare its runtimes for the enumeration task with the (already quite efficient) MRS constraint solver of the LKB system (Copestake and Flickinger, 2000). Our data set consists of the 850 MRS-nets extracted from the Rondane treebank which have less than one million solutions (see Fig. 2). Fig. 3 displays the runtimes for enumerating all solutions, divided by the number of solutions, for both solvers. The horizontal axis shows the description sizes (number of tree fragments), and the (logarithmic!) vertical axis shows the average runtime per solution for descriptions of this size.

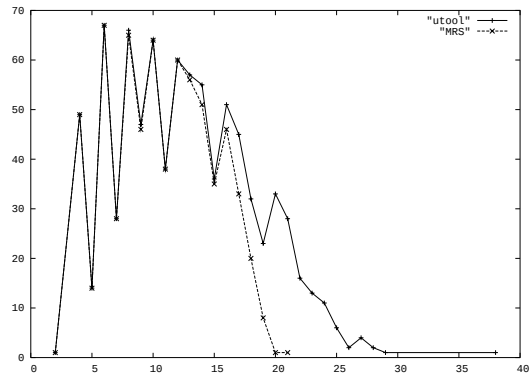

Figure 2: Distribution of constraints in Rondane over different sizes. The solid line shows the constraints in the data set, and the dashed line shows the constraints that the LKB solver could solve.

Due to memory limitations, the LKB solver could only solve descriptions with up to 21 tree fragments, which account for 80% of the test data. `utool` solved all descriptions in the test set. The evaluation was done using a 1.2 GHz PC with 2 GB of memory.

The figure shows that `utool` is generally faster than the LKB solver, up to a factor of approx. 1000. We should note that the LKB solver displays a dramatically higher variation in runtimes for constraints of the same size. Note that for small constraints, the runtimes tend to be too small to measure them accurately.

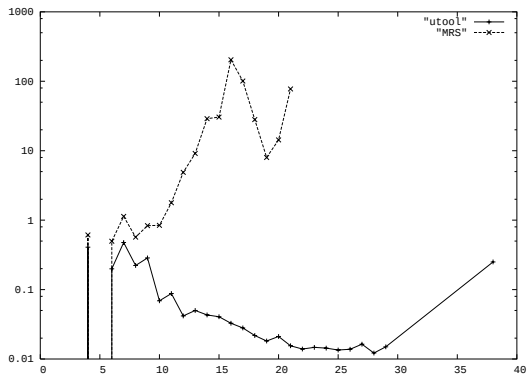

Figure 3: Runtimes per solution (in ms) for the MRS nets in the Rondane treebank for LKB and `utool`.

3 Conclusion

We have presented `utool`, a tool that supports a variety of operations related to scope underspecification. It is the most efficient solver for underspecification available today, and provides functionality for counting readings, testing whether a description is a net, and converting between different underspecification formalisms. It collects the results of several years of formal and computational research on dominance graphs into one convenient system.

The most obvious use of `utool` is the enumeration of readings of underspecified descriptions produced by large-scale grammars. This means that a user can realistically map the semantic output of these grammars into actual semantic representations. However, the tool is also useful for *developers* of such grammars. It can be used to count and explore the readings of the underspecified descriptions the grammar computes, and has already been used in the debugging of the syntax-semantics interface of the ERG (Flickinger et al., 2005).

From a more general perspective, the real appeal of underspecification is that it could allow us to eliminate readings that contradict the context or our world knowledge, without having to enumerate these readings first. Such inferences could already take place on the level of the underspecified description (Koller and Niehren, 2000). But the new chart data structure that `utool` computes is a more explicit packed representation of the possible readings, and still relatively small in practice. Thus it could open up avenues for more theoretical future research as well.

References

Ernst Althaus, Denys Duchier, Alexander Koller, Kurt Mehlhorn, Joachim Niehren, and Sven Thiel. 2003. An efficient graph algorithm for dominance constraints. *Journal of Algorithms*, 48:194–219.

Manuel Bodirsky, Denys Duchier, Joachim Niehren, and Sebastian Miele. 2004. An efficient algorithm for weakly normal dominance constraints. In *ACM-SIAM Symposium on Discrete Algorithms*. The ACM Press.

Johan Bos. 1996. Predicate logic unplugged. In *Proceedings of the Tenth Amsterdam Colloquium*, pages 133–143.

Miriam Butt, Helge Dyvik, Tracey Holloway King, Hiroshi Masuichi, and Christian Rohrer. 2002. The parallel grammar project. In *Proceedings of the COLING 2002 Workshop on Grammar engeneering and evaluation*.

Ann Copestake and Dan Flickinger. 2000. An open-source grammar development environment and broad-coverage English grammar using HPSG. In *Conference on Language Resources and Evaluation*.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan Sag. 2003. Minimal recursion semantics: An introduction. Available at `http://lingo.stanford.edu/sag/papers/copestake.pdf`.

Markus Egg, Alexander Koller, and Joachim Niehren. 2001. The Constraint Language for Lambda Structures. *Logic, Language, and Information*, 10:457–485.

Dan Flickinger, Alexander Koller, and Stefan Thater. 2005. A new well-formedness criterion for semantics debugging. In *Proceedings of the 12th HPSG Conference*, Lisbon.

Laura Kallmeyer and Maribel Romero. 2004. LTAG semantics with semantic unification. In *Proceedings of the TAG+7 Workshop*, Vancouver.

Alexander Koller and Joachim Niehren. 2000. On underspecified processing of dynamic semantics. In *Proceedings of COLING-2000*, Saarbrücken.

Alexander Koller, Joachim Niehren, and Stefan Thater. 2003. Bridging the gap between underspecification formalisms: Hole semantics as dominance constraints. In *Proceedings of the 10th EACL*, Budapest.

Joachim Niehren and Stefan Thater. 2003. Bridging the gap between underspecification formalisms: Minimal recursion semantics as dominance constraints. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*.