Implementing Bottom-up Procedures with Code Trees (abstract of the invited talk)

Andrei Voronkov

Computing Science Department
Uppsala University
(email voronkov@csd.uu.se)*

Research on automated reasoning is one of the topics developed inside the framework of the Fifth Generation Computer Systems project. Development of automated reasoning systems is important because such systems can be embedded in various applied intelligent systems. Research on automated theorem proving always supplied important ideas for other parts of computer science, maybe because automated reasoning has always been aimed at solving extremely hard (and even unsolved) problems. In order to solve such problems, one needs both research in computational logic and research on the implementation of logic. This talk discusses *code trees:* an *implementation* technique for bottom-up procedures. Code trees are implemented in our theorem prover `Vampire`.

Most automatic theorem proving systems can generally be divided in two parts by their mode of evaluation (or clause generation). *Top-down* systems start from a goal, reducing it to subgoals, until all subgoals become axioms. *Bottom-up* systems usually deal with a database of clauses (or facts, literals, tuples, sequents etc.) and generate new clauses by applying inference rules to clauses in the database. There are also systems which can combine top-down with bottom-up.

Surprisingly or not, bottom-up and combined systems show a superior performance when dealing with problems involving large search spaces. This has been widely recognized in the deductive database community. There

*Supported by a TFR grant

are two main reasons bottom-up systems become more efficient on hard problems:

1. Top-down algorithms retain no information and must often do the same job on different branches of the search tree.
2. In bottom-up algorithms one can implement major operations on the "set at a time" basis, opposite to "tuple a time" or "clause at a time" procedures used by e.g. Prolog.

Bottom-up procedures have been traditionally used in automated deduction, since its very beginning. Recently, the need for such procedures has been recognized in logic programming and deductive databases. In logic programming, it has been noted that tabulation can both speed up evaluation and give more declarative treatment of some subgoals. In deductive databases, where the databases can be very large, the old "tuple at a time" methods are not adequate any more. In logic programming and deductive databases the systems that make use of previously generated clauses (or goals) are usually considered as bottom-up procedures.

There are certain specific features of bottom-up systems which pose a challenge to people who implement them:

1. There is usually a small number of operations which must repeatedly be applied to clauses in the database (for example, resolution).
2. The database may be very large and dynamically changing.
3. The number of calls of these operations is so big that it is practically impossible to implement them on the "clause at a time" basis.

The main problems faced by bottom-up procedures are based on the necessity to efficiently handle large dynamically changing databases of clauses.

In our talk we deal with an implementation technique for bottom-up systems, which can be used to speed up many important algorithms. The technique is demonstrated on the forward subsumption problem for arbitrary clauses. The idea of code trees is very general and may as well be applied to various concepts of resolution. It can also be used for the implementation of equality reasoning procedures, like paramodulation and rewriting. In areas outside of automated reasoning the most natural candidates are deductive databases, logic programming, expert systems and parsing.

Our technique is based on two main ideas which we shall illustrate on forward subsumption. The first idea is to compile specialized subsumption procedures for kept clauses. It can be considered as a run time specialization of a general subsumption algorithm. This technique has much in common with the technique of WAM-based Prolog implementations. We call it *the abstract subsumption machine.* It gives a very efficient subsumption algorithm for the problem of subsuming a clause by a clause. A specific feature of the use of this approach is that the compilation must be performed at run time.

When the set D of kept clauses is large, the implementation of subsumption on the "clause at a time" basis becomes inefficient. The second idea of this paper is to perform subsumption on the "set at a time" basis via code trees which merge several specialized algorithms into one.

The structure of the talk is as follows.

1. Implementation of theorem proving systems and bottom-up algorithms in general.

2. Subsumption.

3. Survey of traditional implementation techniques. Indexing.

4. Code trees.

5. Experimental results.

References

[Gra94] P. Graf. Extended path-indexing. In A. Bundy, editor, *Automated Deduction — CADE-12. 12th International Conference on Automated Deduction.*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 514–528, Nancy, France, June/July 1994.

[Lus92] E.L. Lusk. Controlling redundancy in large search spaces: Argonne-style theorem proving through the years. In A. Voronkov, editor, *Logic Programming and Automated Reasoning. International Conference LPAR'92.*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 96–106, St.Petersburg, Russia, July 1992.

[McC92] William W. McCune. Experiments with discrimination-tree in indexing and path indexing for term retrieval. *Journal of Automated Reasoning*, 9(2):147–167, 1992.

[Vor 94] A. Voronkov. Implementing bottom-up procedures with code trees: a case study of forward subsumption. UPMAIL Technical Report 88, Uppsala University, Computing Science Department, October 1994 (also to be published in Journal of Automated Reasoning).

[WOL91] Larry Wos, Ross Overbeek, and Ewing Lusk. Subsumption, a sometimes undervalued procedure. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic. Essays in Honor of Alan Robinson.*, pages 3–40. The MIT Press, Cambridge, Massachusetts, 1991.