The Barcelona Prover

ROBERT NIEUWENHUIS, JOSÉ MIGUEL RIVERO and
MIGUEL ÁNGEL VALLEJO
Technical University of Catalonia
Barcelona, Spain
roberto@lsi.upc.es

Abstract. Here we describe the equational theorem prover Barcelona, in its version that participated in the CADE'96 theorem proving competition. The system was built on top of our toolkit of data structures and algorithms for automated deduction in first-order logic with equality, and was devised mainly to test the performance of this toolkit.

Key words: Automated theorem proving, competition, Barcelona, data structures and algorithms, implementation

1. Introduction

During the last decade, research on automated deduction in our group has mainly focussed on theoretical results for first-order logic with equality. New techniques for e.g., clausal rewriting and deduction with constrained clauses have been developed and completeness results established. Many necessary underlying results on term orderings, constraint solving and answer computation have been given, with their decidability and complexity characteristics (cf. `http://www-lsi.upc.es/dept/sectp.html`). In order to better understand these new techniques and learn more about their practical behaviour, during their development we have always worked with prototype implementations written in Prolog. These experimental systems converged in 1992 into our laboratory implementation *Saturate* [9, 3]. Although it has been applied successfully in many contexts and has led to interesting new insight and theoretical results (see also [1, 7]), the *Saturate* system is not a high-performance theorem prover.

Therefore, in order to focus more on practice, some years ago we decided to work on efficiency-oriented systems as well. When studying the existing provers, we found a wide range of different *ad-hoc* data structures for implementing very similar calculi and control mechanisms. E.g., for term representation there are the linear flatterms [2] and diverse other types of nonlinear terms, and for indexing, discrimination trees [2, 6], path indexing [10, 6] and substitution trees [4], among others.

In spite of the evident qualities of all these data structures, we somewhat missed a more uniform framework, like the Warren Abstract Machine (WAM) [11] in logic programming. In such a framework structure sharing, indexing for all the main operations and perhaps even compilation of static parts of the clause sets should be possible in a standard, elegant, well-structured and reusable way. After a large number of experiments with many techniques (most of them with rather negative results), we finally came to a kernel of data structures satisfying these requirements.

We chose a WAM-like heap structure for storing terms as DAG's, where structure sharing is possible but not forced. Regarding indexing, we developed several techniques based on substitution trees, which turn out to be surprisingly well combinable with WAM terms due to conceptual similarities. Many known techniques from the WAM, like variable binding, backtracking and memory management, which are the result of many years of work in logic programming implementation, can be smoothly incorporated here. Another requirement is satisfied as well: static clause (sub)sets can be compiled in this framework into efficient abstract machine code for inference computation and redundancy proving. All this led to a toolkit called *Dedam* (Deduction Abstract Machine) [8] in which all basic operations (indexing, variable management, I/O, etc.) are provided, and on top of which one can build theorem provers in a simple way.

Here we describe the equational theorem prover Barcelona, in its version that participated in the CADE-13 theorem proving competition. The system was built on top of (a first version of) our kernel of data structures, and was devised mainly to test the performance of this toolkit. During the competition the relatively high throughput of the data structures seems to have compensated for the fact that the prover itself is just an unfailing Knuth-Bendix completion procedure without many heuristics or tuning for the specific class of problems.

2. Architecture

2.1. Calculus

As said, the Barcelona prover is essentially an implementation of unfailing Knuth-Bendix completion. Hence a term ordering $\succ$ is central. For completeness reasons, it must be a reduction ordering that is (extendable to) a total ordering on all ground terms, so as a simple choice we have taken the lexicographic path ordering (LPO) [5]. The LPO extends a *precedence* ordering on the function symbols.

In the Barcelona prover the only inference rule is the well-known rule of equational superposition:

$$\frac{s' = t' \qquad s = t}{(s[t']_p = t)\sigma} \qquad \text{where:} \quad \begin{array}{l} \sigma \text{ is the } mgu \text{ of } s' \text{ and } s|_p \\ s|_p \notin \mathit{Vars}(s) \\ t\sigma \not\succeq s\sigma \text{ and } t'\sigma \not\succeq s'\sigma \end{array}$$

(here $s|_p$ denotes the subterm of s at position p, and $s[t']_p$ is the result of replacing it in s by t'). The conclusion $(s[t']_p = t)\sigma$ of the inference is called a *critical pair.*

Furthermore there are two main mechanisms for detecting redundant equations: forward and backward demodulation and forward subsumption. For efficiency reasons, in order to avoid checking with LPO at each rewrite step, we only considered demodulation with rules that can be oriented once and for all: an equation $l = r$ where $l \succ r$ (i.e., $l = r$ is an oriented rewrite rule) *demodulates* an equation $s[l\sigma]_p = t$ into $s[r\sigma]_p = t$. An equation $s = t$ *subsumes* another equation $s\sigma = t\sigma$ if σ is some matching substitution.

2.2. CONTROL AND TUNING FOR THE COMPETITION

The following is a typical main loop of completion:

```
 1. $new$ := the set of initial equations
 2. $old$ := $\emptyset$
 3. **While** $new \neq \emptyset$ **And** no inconsistency detected **Do**
 4.     select one equation $eq$ in $new$
 5.     remove $eq$ from $new$ and add it to $old$
 6.     **For** all critical pairs $cp$ between $eq$ and $old$ **Do**
 7.         **If** $cp$ is not redundant **Then**
 8.             orient $cp$ (if possible) and add it to $new$
 9.             use $cp$ to detect redundancies in $new$ and $old$
10.         **EndIf**
11.     **EndFor**
12. **EndWhile**
```

The previous scheme has many degrees of freedom, which were solved as follows for the competition:

Line 3: What notion of inconsistency is used? The system only deals with (universally quantified) positive equations. At the competition, the input axioms were of this form, but the theorem could be an arbitrarily quantified equation. After negation and Skolemization, it can however be expressed as $thm(s,t) = false$ and, if an additional $thm(x,x) = true$ is input, an inconsistency exists iff the equation $true = false$ appears at some step.

Line 4: Which equation *eq* is selected? In our system for the competition we used the following measure of term and equation size: $size(x) = 1$ for a variable x; furthermore, $size(f(t_1,\ldots,t_n)) = 3 + size(t_1) + \ldots + size(t_n)$ and $size(s = t) = size(s) + size(t)$. The selection takes the smallest new equation according to this size, except that once in each five iterations it takes the smallest descendant of $thm(s,t) = false$ (if there is such a descendant in *new*).

Line 7: What is done for checking forward redundancy? As said, we use demodulation with orientable equations and subsumbtion with non-orientable equations, in both methods with equations from both *new* and *old*. The rewriting strategy applied for normalization is a (leftmost) innermost strategy with marking of irreducible subterms to avoid unnecessary reduction attempts.

Line 8: How is orientation done? The precedence of the LPO ordering we use is as follows: symbols with greater arity are bigger in the precedence and between symbols with the same arity we compare the natural numbers that internally represent each symbol.

Line 9: How to detect backward redundancies? Only orientable equations *cp* are used. First, it is checked whether *cp* demodulates any of the equations in *new* or *old*. If this is the case, then the reduced equation is further treated as a critical pair (with some optimizations taking advantage of its previous orientation).

3. Implementation

The system is implemented in C. We have been especially careful regarding the readability and the structuring of the data structures, since our aim is to provide a clean, reusable and standard framework for the implementation of first-order provers with equality. The prover has no further language or hardware requirements. Probably due to the fact that we always keep the data base of equations fully interreduced and simplified, and that there is a considerable amount of sharing in the indexing data structures, memory has not been a bottleneck in any of the competition problems.

The user interface of the system has been kept flexible in the sense that a large number of output settings can be enabled or disabled. A proof in tree format is output by the system after each successful run.

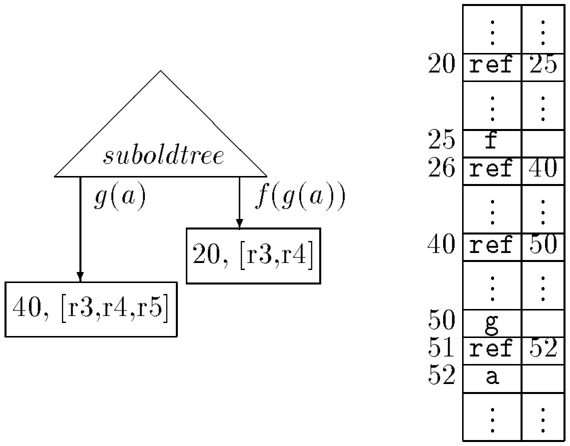
3.1. Data Structures

There is an overall heap data structure for all terms, with their basic operations: input/ouput, LPO, etc. Furthermore, there are four indexing data structures:

- *demodtree* is specialized in matching, and contains all left hand sides of rewrite rules applicable in demodulation;
- *backdemodtree* is specialized in finding instances (i.e., reverse matching) for backward demodulation and contains the (shared) subterms of old and new rules;
- *oldtree* and *suboldtree* are specialized in unification for inferences and contain the maximal side(s) of old rules and their (shared) subterms respectively.

Each critical pair is demodulated *as soon as it is generated* by (backtrackable) rewriting, and *only if it is not convergent* it is copied, oriented and stored as a new rule; then the demodulation steps are undone and the critical pair search goes on by backtracking in the unification indexing tree (*oldtree* or *suboldtree*).

As a simple example, in the figure below we show how sharing in our data structures leads to efficiency for inference computation (for e.g., backward demodulation there are similar mechanisms).



At the right hand side in this figure a small part of our WAM-like heap is shown. Roughly (we are omitting a number of details here), terms are represented on the heap as usual in the WAM: each function symbol of arity n is followed by n contiguous `ref` positions pointing to its arguments.

The figure at the left represents the tree *suboldtree* and two of its leaves, corrsponding to the terms $g(a)$ and $f(g(a))$ respectively. At each leaf a heap address and a list of rule numbers is stored.

Suppose we are looking for inferences with the rule $g(x) \rightarrow h(x)$ (not shown here on the heap) using *suboldtree*. When we arrive at the leaf corresponding to the term $g(a)$, the variable x will have been instantiated accordingly, and we find the heap address (40) of a `ref` that points to $g(a)$ at position 50. By temporarily changing this `ref` at position 40, making it point instead to the term $h(x)$, we can simply read off all the critical pairs from the given list of rules [r3,r4,r5] which share the subterm $g(a)$ through position 40. Note that $g(a)$ is also shared by other terms in the tree like $f(g(a))$, and hence the rule set for $g(a)$ contains the one at $f(g(a))$ as a subset.

4. Performance

During the competition (and especially the first 15-20 problems) the relatively high throughput of the data structures seems to have compensated the simplicity of the prover itself and its lack of many heuristics or tuning to the specific class of problems. As it was developed only very recently, there are no further experimental results outside the competition.

5. Conclusion

The Barcelona prover —in its CADE-13 competition version— was built on top of a first version of our Dedam kernel of data structures, and was devised mainly to test the performance of this kernel. The strength of the prover came from the efficiency of Dedam, which we believe is now starting to be useful for enhancing the efficiency of state-of-the-art provers.

The prover's main weakness was the lack of heuristics or tuning to the rather specific class of problems, aspects which we are currently working on.

Regarding further work on the Dedam kernel of data structures, it seems that there is still a lot of room for progress. For example, the performance of matching for demodulation has recently been enhanced importantly thanks to some new ideas on indexing data structures, and there are also some other recent improvements regarding memory management: by applying WAM-based techniques we can in fact almost completely avoid it. Both matching and memory management are well-known main bottlenecks in equational theorem proving. In the near future we will continue investigating some further ideas related to the underlying data structures. We will also build a prover for full first-order clauses with equality on top of Dedam.

References

1. David Basin and Harald Ganzinger. Complexity Analysis Based on Ordered Resolution. In *11th IEEE LICS*, pages 456–465, New Brunswick, NJ, July, 1996.
2. Jim Christian. Flatterms, Discrimination Nets, and Fast Term rewriting. *Journal of Automated Reasoning*, 10:95–113, 1993.
3. Harald Ganzinger, Robert Nieuwenhuis, and Pilar Nivela. The Saturate System, 1995. See `http://www.mpi-sb.mpg.de/SATURATE/Saturate.html` for software and documentation.
4. Peter Graf. Substitution Tree Indexing. In J. Hsiang, editor, *6th RTA*, LNCS 914, pages 117–131, Kaiserslautern, Germany, April 4–7, 1995. Springer-Verlag.
5. S. Kamin and J.-J. Levy. Two generalizations of the recursive path ordering. Unpublished note, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, 1980.
6. William McCune. Experiments with discrimination tree indexing and path indexing for term retrieval. *Journal of Automated Reasoning*, 9(2):147–167, October 1992.
7. Robert Nieuwenhuis. Basic paramodulation and decidable theories. In *11th IEEE LICS*, pages 473–482, New Brunswick, NJ, USA, July 27–30, 1996.
8. Robert Nieuwenhuis, Josè Miguel Rivero, and Miguel Angel Vallejo. An implementation kernel for theorem proving with equality clauses. Technical report, Dept. LSI, Technical University of Catalonia, Barcelona, May 1996.
9. Pilar Nivela and Robert Nieuwenhuis. Practical results on the saturation of full first-order clauses: Experiments with the saturate system. (system description). In Proc. *5th RTA*, LNCS 690, Montreal, June 16–18, 1993. Springer-Verlag.
10. Mark Stickel. The path-indexing method for indexing terms. Technical Report 473, Artificial Intelligence Center, SRI International, October 1989.
11. David H.D. Warren. An Abstract Prolog Instruction Set. Technical Report Technical Note 309, SRI International, Menlo Park, CA, October 1983.