Handling Incomplete and Complete Tables in Tabled Logic Programs

Ricardo Rocha*

DCC-FC & LIACC
University of Porto, Portugal
`ricroc@ncc.up.pt`

Extended Abstract

Most of the recent proposals in tabling technology were designed as a means to improve the performance of particular applications in key aspects of tabled evaluation like re-computation and scheduling. The discussion we address in this work was also motivated by our recent attempt [1] of applying tabling to Inductive Logic Programming (ILP) [2]. ILP applications are very interesting for tabling because they have huge search spaces and do a lot of re-computation. Moreover, we found that they are an excellent case study to improve some practical limitations of current tabling execution models. In particular, we next focus on the table space and how to efficiently handle incomplete and complete tables.

Tabling is about storing answers for subgoals so that they can be reused when a repeated call appears. On the other hand, most ILP algorithms are interested in example satisfiability, not in the answers: query evaluation stops as soon as an answer is found. This is usually implemented by *pruning* at the Prolog level. Unfortunately, pruning over tabled computations results in *incomplete tables*: we may have found several answers but not the complete set. Thus, usually, when a repeated call appears we cannot simply trust the answers from an incomplete table because we may loose part of the computation. The simplest approach, and the one that has been implemented in most tabling systems, is to throw away incomplete tables, and restart the evaluation from scratch.

In this work, we propose a more aggressive approach where, by default, we keep incomplete tables around. Whenever a call for an incomplete table appears, we first consume the answers from the table. If the table is exhausted, then we will restart the evaluation from the beginning. Later, if the subgoal is pruned again, then the same process is repeated until eventually the subgoal is completely evaluated. The main goal of this proposal is to avoid re-computation when the already stored answers are enough to evaluate a repeated call. This idea is closer to the spirit of the *just enough tabling (JET)* proposal of Sagonas and Stuckey [3]. Our approach works well in the ILP setting, where queries are often very similar, and thus already stored answers are enough to evaluate a

* This work has been partially supported by Myddas (POSC/EIA/59154/2004) and by funds granted to LIACC through the Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia and Programa POSC. We are very thankful to Nuno Fonseca for his support with the April ILP System.

repeated call. When this is not the case, we may not benefit from having kept an incomplete table, but we do not pay any cost either.

On the other hand, complete tables can also be a problem. When we use tabling for applications that build very many queries or that store a huge number of answers, we can build arbitrarily very many or very large tables, quickly running out of memory space. In general, we will have no choice but to throw away some of the tables (ideally, the least likely to be used next). A common control implemented in most tabling systems is to have a set of tabling primitives that the programmer can use to dynamically abolish some of the tables. However, this can be hard to implement and difficult to decide what are the potentially useless tables that should be deleted.

In order to allow useful deletion without compromising efficiency, we propose a more suitable approach for large dynamic searches, a memory management strategy based on a *least recently used* algorithm, that dynamically recovers space from the least recently used tables when the system runs out of memory. With our approach, the programmer can still force the deletion of particular tables, but can also rely on the effectiveness of the memory management algorithm to completely avoid the problem of deciding what potentially useless tables should be deleted.

Both proposals have been implemented in the YapTab tabling system [4] with minor changes to the original design. To the best of our knowledge, YapTab is the first tabling system that implements support to handle incomplete and complete tables as discussed above. Preliminaries results using the April ILP system [5] showed very substantial performance gains and a substantial increase of the size of the problems that can be solved by combining ILP with tabling. Despite the fact that we used ILP as the motivation for this work, our proposals are not restricted to ILP applications and can be generalised and applied to any other applications.

References

1. Rocha, R., Fonseca, N., Costa, V.S.: On Applying Tabling to Inductive Logic Programming. In: European Conference on Machine Learning. Number 3720 in LNAI, Springer-Verlag (2005) 707–714
2. Muggleton, S.: Inductive Logic Programming. In: Conference on Algorithmic Learning Theory, Ohmsma (1990) 43–62
3. Sagonas, K., Stuckey, P.: Just Enough Tabling. In: ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming, ACM (2004) 78–89
4. Rocha, R., Silva, F., Santos Costa, V.: YapTab: A Tabling Engine Designed to Support Parallelism. In: Conference on Tabulation in Parsing and Deduction. (2000) 77–87
5. Fonseca, N., Camacho, R., Silva, F., Santos Costa, V.: Induction with April: A Preliminary Report. Technical Report DCC-2003-02, Department of Computer Science, University of Porto (2003)