An External Module for Implementing Linear Tabling in Prolog

Cláudio Silva, Ricardo Rocha, and Ricardo Lopes*

DCC-FC & LIACC
University of Porto, Portugal
ccaldas@dcc.online.pt {ricroc,rslopes}@ncc.up.pt

Extended Abstract

In previous work [1], we have presented a proposal to combine the power of tabling with the Extended Andorra Model (EAM) in order to produce an execution model with advanced control strategies that guarantees termination, avoids looping, reduces the search space, and is less sensitive to goal ordering.

To address the integration between tabling and the EAM, through the BEAM system [2], we have identified several tasks [1]. In particular, to study how tabling interacts with the BEAM, we proposed the ability to use an external module for implementing tabling primitives that provide direct control over the search strategy. This approach may compromise efficiency, if compared to systems that implement tabling support at the low-level engine, but allows tabling to be easily incorporated into any Prolog system. For our work, it will serve as the basis to study and detect in advance the potential integration problems before extending the BEAM system to support tabling running within the EAM environment.

In the past years several alternative mechanisms for tabling have been proposed and implemented in systems like XSB, Yap, B-Prolog, ALS-Prolog and Mercury. In these implementations, we can distinguish two main categories of tabling mechanisms: *delaying-based tabling mechanisms* in the sense that the computation state of suspended tabled subgoal calls has to be preserved, either by freezing the whole stacks or by copying the execution stacks to separate storage; and *linear tabling mechanisms* where a new call always extends the latest one, therefore maintaining only a single SLD tree in the execution stacks. Delaying-based mechanisms can be considered more complicated to implement but obtain better results. The weakness of the linear mechanisms is the necessity of re-computation for computing fix-points.

Implementing tabling through an external module restrict us to linear tabling mechanisms, because external modules cannot directly interact with the execution stacks. Therefore, we have decided to design a module that implements the two available mechanisms that, to the best of our knowledge, implement linear tabling: the SLDT strategy of Zhou *et al.* [3]; and the DRA technique of Guo and Gupta [4]. The key idea of the SLDT strategy is to let a tabled subgoal call

* 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.

execute from the backtracking point of a former variant call if such a call exists. When there are available answers in the table space, the variant call consumes them; otherwise, it uses the predicate clauses to produce answers. Meanwhile, if a call that is a variant of some former call occurs, it takes the remaining clauses from the former call and tries to produce new answers by using them. The variant call is then repeatedly re-executed, until all the available answers and clauses have been exhausted, that is, until a fix-point is reached. The DRA technique is based on dynamic reordering of alternatives with variant calls. This technique tables not only the answers to tabled subgoals, but also the alternatives leading to variant calls, the *looping alternatives*. It then uses the looping alternatives to repeatedly recompute them until a fix-point is reached.

Currently, we have already a preliminary implementation of both approaches in our external module. The module uses the C language interface of the Yap Prolog system to implement external tabling primitives that provide direct control over the search strategies for a transformed program. According to the tabling mechanism to be used, a tabled logic program is first transformed to include the tabling primitives through source level transformations and only then, the resulting program is compiled. Our module is independent from the Yap Prolog's engine which makes it easily portable to other Prolog systems with a C language interface. To implement the table space data structures we use *tries* as originally implemented in the XSB Prolog system [5].

Preliminaries results, on a set of common benchmarks for tabled execution, allows us to make a first and fair comparison between the SLDT and the DRA mechanisms and, therefore, better understand the advantages and weaknesses of each. Starting from these results, we are now working on a new proposal that tries to combine the best features of both in order to produce a more robust and efficient linear tabling mechanism to experiment with the BEAM.

References

1. Rocha, R., Lopes, R., Silva, F., Costa, V.S.: IMPACT: Innovative Models for Prolog with Advanced Control and Tabling. In: International Conference on Logic Programming. Number 3668 in LNCS, Springer-Verlag (2005) 416–417
2. Lopes, R., Santos Costa, V., Silva, F.: A Novel Implementation of the Extended Andorra Model. In: International Symposium on Pratical Aspects of Declarative Languages. Number 1990 in LNCS, Springer-Verlag (2001) 199–213
3. Zhou, N.F., Shen, Y.D., Yuan, L.Y., You, J.H.: Implementation of a Linear Tabling Mechanism. In: Practical Aspects of Declarative Languages. Number 1753 in LNCS, Springer-Verlag (2000) 109–123
4. Guo, H.F., Gupta, G.: A Simple Scheme for Implementing Tabling based on Dynamic Reordering of Alternatives. In: Conference on Tabulation in Parsing and Deduction. (2000) 141–154
5. Ramakrishnan, I.V., Rao, P., Sagonas, K., Swift, T., Warren, D.S.: Efficient Access Mechanisms for Tabled Logic Programs. Journal of Logic Programming **38** (1999) 31–54