Mixed-Strategies for Linear Tabling in Prolog

Miguel Areias and Ricardo Rocha[⋆]

CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto
Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal
{miguel-areias,ricroc}@dcc.fc.up.pt

Abstract. Tabling is an implementation technique that solves some limitations of Prolog's operational semantics in dealing with recursion and redundant sub-computations. Arguably, the SLDT and DRA strategies are the two most successful extensions to standard linear tabled evaluation. In this work, we propose a new strategy for linear tabling, named DRS, and we present a framework, on top of the Yap system, that supports the combination of variants of these three strategies.

1 Introduction

Tabled evaluation is a recognized and powerful technique that can considerably reduce the search space, avoid looping and have better termination properties than SLD resolution [1]. Tabling consists of storing intermediate solutions for subgoals so that they can be reused when a repeated subgoal appears during the resolution process. We can distinguish two main categories of tabling mechanisms: *suspension-based tabling* and *linear tabling*.

Suspension-based tabling mechanisms need to preserve the computation state of suspended tabled subgoals in order to ensure that all solutions are correctly computed. Linear tabling mechanisms use iterative computations of tabled subgoals to compute fix-points. While suspension-based mechanisms are considered to obtain better results in general, they have more memory space requirements and are more complex and hard to implement than linear tabling mechanisms.

Arguably, the SLDT [2] and DRA [3] strategies are the two most successful extensions to standard linear tabling evaluation. As these strategies optimize different aspects of the evaluation, they are, in principle, orthogonal to each other and thus it should be possible to combine both in the same system. In this work, we propose a new strategy, named *Dynamic Reordering of Solutions (DRS)*, and we present a framework, on top of the Yap Prolog system, that integrates and supports the combination of these three strategies. Our implementation shares the underlying execution environment and most of the data structures used to implement tabling in Yap. We thus argue that all these common support features allow us to make a first and fair comparison between these different linear tabling strategies and, therefore, better understand the advantages and weaknesses of each, when used solely or combined with the others.

[⋆] This work has been partially supported by the FCT research projects STAMPA (PTDC/EIA/67738/2006) and HORUS (PTDC/EIA-EIA/100897/2008).

2 Standard Linear Tabled Evaluation

Tabling works by storing intermediate solutions for tabled subgoals so that they can be reused when a repeated call appears. In a nutshell, first calls to tabled subgoals are considered *generators* and are evaluated as usual, using program resolution, but their solutions are stored in a global data space, called the *table space*. Repeated calls to tabled subgoals are considered *consumers* and are not re-evaluated against the program clauses because they can potentially lead to infinite loops, instead they are resolved by consuming the solutions already stored for the corresponding generator. During this process, as further new solutions are found, we need to ensure that they will be consumed by all the consumers, as otherwise we may miss parts of the computation and not fully explore the search space. To do that, linear tabling mechanisms maintain a single execution tree where tabled subgoals are iteratively computed until reaching a fix-point.

A generator call C thus keeps trying its matching clauses until reaching a fix-point. If no new solutions are found during one cycle of trying the matching clauses, then we have reached a fix-point and we can say that C is completely evaluated. However, if a number of subgoal calls is mutually dependent, thus forming a *Strongly Connected Component (SCC)*, then completion is more complex and we can only complete the calls in a SCC together. SCCs are usually represented by the *leader call*, i.e., the generator call which does not depend on older generators. A leader call defines the next completion point, i.e., if no new solutions are found during one cycle of trying the matching clauses for the leader call, then we have reached a fix-point and we can say that all subgoal calls in the SCC are completely evaluated.

3 Linear Tabling Strategies

The standard linear tabling mechanism uses a naive approach to evaluate tabled logic programs. Every time a new solution is found during the last round of evaluation, the complete search space for the current SCC is scheduled for re-evaluation. However, some branches of the SCC can be avoided, since it is possible to know beforehand that they will only lead to repeated computations, hence not finding any new solutions. Next, we present three different approaches for optimizing standard linear tabled evaluation.

3.1 Dynamic Reordering of Execution

The first optimization, that we call *Dynamic Reordering of Execution (DRE)*, is based on the original SLDT strategy, as proposed by Zhou et al. [2]. The key idea of the DRE strategy is to let repeated calls to tabled subgoals execute from the *backtracking clause of the former call*. A first call to a tabled subgoal is called a *pioneer* and repeated calls are called *followers* of the pioneer. When backtracking to a pioneer or a follower, we use the same strategy, first we explore the remaining clauses and only then we try to consume solutions. The fix-point check operation is still only performed by pioneer calls.

3.2 Dynamic Reordering of Alternatives

The key idea of the *Dynamic Reordering of Alternatives (DRA)* strategy, as originally proposed by Guo and Gupta [3], is to memorize the clauses (or alternatives) leading to consumer calls, the *looping alternatives*, in such a way that when scheduling an SCC for re-evaluation, instead of trying the full set of matching clauses, we only try the looping alternatives. Initially, a generator call C explores the matching clauses as in standard evaluation and, if a consumer call is found, the current clause for C is memorized as a looping alternative. After exploring all the matching clauses, C enters the *looping state* and from this point on, it only tries the looping alternatives until reaching a fix-point.

3.3 Dynamic Reordering of Solutions

The last optimization, that we named *Dynamic Reordering of Solutions (DRS)*, is a new proposal that can be seen as a variant of the DRA strategy, but applied to the consumption of solutions. The key idea of the DRS strategy is to memorize the solutions leading to consumer calls, the *looping solutions*. When a non-leader generator call C consumes solutions to propagate them to the context of the previous call, if a consumer call is found, the current solution for C is memorized as a looping solution. Later, if C is scheduled for re-evaluation, instead of trying the full set of solutions, it only tries the looping solutions plus the new solutions found during the current round. In each round, the new solutions leading to consumer calls are added to the previous set of looping solutions.

4 Experimental Results

To the best of our knowledge, Yap is now the first tabling engine that integrates and supports the combination of different linear tabling strategies. We have thus the conditions to better understand the advantages and weaknesses of each strategy when used solely or combined with the others. In what follows, we present initial experiments comparing linear tabled evaluation with and without support for the DRE, DRA and DRS strategies. To put the performance results in perspective, we used the well-known $path/2$ predicate, that computes the transitive closure in a graph, combined with several different graph configurations.

Next, we show in Table 1 the execution time ratios of standard linear tabled evaluation to DRE, DRA and DRS solely and combined strategies. Ratios higher than 1.00 mean that the respective strategies have a positive impact on the execution time. The results obtained are the average of 5 runs for each configuration.

Globally, the results in Table 1 show that, for most of these experiments, DRE evaluation has no significant impact in the execution time. On the other hand, the results indicate that the DRA and DRS strategies are able to effectively reduce the execution time for most of the experiments, when compared with standard evaluation, and that by combining both strategies it is possible to reduce even further the execution time of the evaluation. In most cases, this

Table 1. Execution time ratios of standard to DRE, DRA and DRS strategies

Strategy	**Pyramid**			**Cycle**			**Grid**		
	1000	**2000**	**3000**	**1000**	**2000**	**3000**	**20**	**30**	**40**
DRE	0.98	1.00	0.88	0.94	0.95	**1.04**	0.83	0.99	0.99
DRA	**1.60**	**1.59**	**1.58**	**1.18**	**1.20**	**1.22**	**1.08**	**1.09**	**1.07**
DRS	0.99	0.98	0.99	**1.14**	**1.18**	**1.25**	**1.20**	**1.20**	**1.21**
DRE+DRA	1.58	**1.66**	**1.63**	**1.22**	**1.24**	1.22	**1.12**	**1.10**	1.07
DRE+DRS	1.00	**1.01**	**1.01**	**1.22**	**1.23**	1.23	0.95	1.14	1.14
DRA+DRS	**1.63**	**1.64**	**1.62**	**1.56**	**1.59**	**1.69**	**1.40**	**1.32**	**1.32**
DRE+DRA+DRS	1.59	1.57	1.56	**1.61**	1.55	1.60	1.36	**1.33**	1.30

reduction is higher than the sum of the reductions obtained with each strategy individually. This shows the potential of our framework and suggests that the overhead associated with this combination is negligible. When DRE is present, the results are, in general, worst than the results obtained with the DRA/DRS strategies solely. A possible explanation for this behavior is the fact that, as DRE has more space requirements, this leads to more expansions of the execution stacks, which in turn can lead to higher ratios of cache and page misses. Still, these results require further study and analysis.

5 Conclusions

We have presented a new strategy for linear tabled evaluation of logic programs, named DRS, and a framework, on top of the Yap system, that integrates and supports the combination of different linear tabling strategies. Our experiments for DRS evaluation showed that, the strategy of avoiding the consumption of non-looping solutions in re-evaluation rounds, can be quite effective for programs that can benefit from it, with insignificant costs for the other programs. Preliminary results for the combined framework were also very promising. In particular, the combination of the DRA and DRS strategies showed the potential of our framework to reduce even further the execution time of a linear tabled evaluation.

References

1. Chen, W., Warren, D.S.: Tabled Evaluation with Delaying for General Logic Programs. Journal of the ACM **43**(1) (1996) 20–74
2. 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
3. Guo, H.F., Gupta, G.: A Simple Scheme for Implementing Tabled Logic Programming Systems Based on Dynamic Reordering of Alternatives. In: International Conference on Logic Programming. Number 2237 in LNCS, Springer-Verlag (2001) 181–196