Speculative Computations in Or-Parallel Tabled Logic Programs

Ricardo Rocha¹, Fernando Silva¹, and Vítor Santos Costa²

¹ DCC-FC & LIACC
University of Porto, Portugal
{ricroc,fds}@ncc.up.pt
² COPPE Systems & LIACC
Federal University of Rio de Janeiro, Brazil
vitor@cos.ufrj.br

Abstract. Pruning operators, such as cut, are important to develop efficient logic programs as they allow programmers to reduce the search space and thus discard unnecessary computations. For parallel systems, the presence of pruning operators introduces the problem of speculative computations. A computation is named speculative if it can be pruned during parallel evaluation, therefore resulting in wasted effort when compared to sequential execution. In this work we discuss the problems behind the management of speculative computations in or-parallel tabled logic programs. In parallel tabling, not only the answers found for the query goal may not be valid, but also answers found for tabled predicates may be invalidated. The problem here is even more serious because to achieve an efficient implementation it is required to have the set of valid tabled answers released as soon as possible. To deal with this, we propose a strategy to deliver tabled answers as soon as it is found that they are safe from being pruned, and present its implementation in the OPTYap parallel tabling system.

1 Introduction

Logic programming is a programming paradigm based on Horn Clause Logic, a subset of First Order Logic. Given a theory (or program) and a query, execution of logic programs uses a simple theorem prover that performs refutation in order to search for alternative ways to satisfy the query. Prolog implements a refutation strategy called SLD resolution. Further, subgoals in a query are always solved from left to right, and that clauses that match a subgoal are always applied in the textual order as they appear in the program.

In order to make Prolog an useful programming language, Prolog designers were forced to introduce features not found within First Order Logic. One such feature is the cut operator. The cut operator adds a limited form of control to the execution by pruning alternatives from the computation. Cut is an asymmetric pruning operator because it only prunes alternatives to the right. Some Prolog systems also implement symmetric pruning operators, with a generic name of
commit. In practice, pruning operators are almost always required when developing actual programs, because they allow programmers to reduce the search space and thus discard unnecessary computation.

Because their semantics are purely operational, pruning operators cause difficulties when considering alternative execution strategies for logic programs. The implementation of or-parallel systems is one example [1–4]. Namely, it has been observed that the presence of pruning operators during parallel execution introduces the problem of speculative computations. Ciepielewski defines speculative computations as work which would not be done in a system with one processor [5]. Alternatives picked for parallel execution, may later be pruned away by a cut. Earlier execution of such computations results in wasted effort when compared to sequential execution.

Pruning operators also raise questions in the context of tabling based execution models for Prolog. The basic idea behind tabling is straightforward: programs are evaluated by storing newly found answers of current subgoals in an appropriate data space, called the table space. New calls to a predicate check this table to verify whether they are repeated. If they are, answers are recalled from the table instead of the call being re-evaluated against the program clauses.

We can consider two types of cut operations in a tabling environment: cuts that do not prune alternatives in tabled predicates – inner cut operations, and cuts that prune alternatives in tabled predicates – outer cut operations. Inner cuts can be easily implemented in sequential systems. On the other hand, because tabling intrinsically changes the left-to-right semantics of Prolog, outer cuts present major difficulties, both in terms of semantics and of implementation.

In this work we address the problem of how to do inner pruning on systems that combine tabling with or-parallelism. Our interest stems from our work in the OPTYap system [6], to our knowledge the first available system that can exploit parallelism from tabled programs. Our experience has shown that many applications do require support for inner pruning. In contrast, outer pruning is not widely used in current tabling systems. Unfortunately, new problems arise even when performing inner pruning in parallel systems. Namely, speculative answers found for tabled predicates may later be invalidated. In the worst case, tabling such speculative answers may allow them to be consumed elsewhere in the tree, generating in turn more speculative computation and eventually cause wrong answers to occur. Answers for tabled predicates can only be tabled when they are safe from being pruned. On the other hand, finding and consuming answers is the natural way to get a tabled computation going forward. Delaying the consumption of valid answers too much may compromise such flow. Therefore, tabled answers should be released as soon as it is found that they are not speculative.

The main contribution of this paper is a design that allows the correct and efficient implementation of inner pruning in an or-parallel tabling system. To do so, we generalise Ali and Karlsson cut scheme [3], which prunes useless work as early as possible, to tabling systems. Our design allows speculative answers to