Pruning in the Extended Andorra Model

Ricardo Lopes¹, Vítor Santos Costa², and Fernando Silva¹

¹ DCC-FC & LIACC, University of Porto
Rua do Campo Alegre, 823, 4150-180 Porto, Portugal
Tel. +351 226078830, Fax. +351 226003654
{rslopes,fds}@ncc.up.pt

² COPPE/Sistemas, Universidade Federal do Rio de Janeiro, Brasil
vitortcos.ufrj.br

Abstract. One of the major problems that actual logic programming systems have to address is whether and how to prune undesirable parts of the search space. A region of the search space would definitely be undesirable if it can only repeat previously found solutions, or if it is well-known that the whole computation will fail. Or it may be the case that we are interested in a subset of solutions. In this work we discuss how the BEAM addresses pruning issues. The BEAM is an implementation of David Warren’s Extended Andorra Model. Because the BEAM relies on a very flexible execution mechanism, all cases of pruning discussed above should be considered. We show that all these different forms of pruning can be supported, and study their impact in applications.

Keywords: Logic Programming, Extended Andorra Model, Pruning, Language Implementation.

1 Introduction

Logic programs are sets of statements defining an intended model for a problem. Logic programming offers programmers a simple, yet powerful, first-order logic based language, for which efficient inference mechanisms exist and which has been used to program significant applications. Most work on logic programming relies the Prolog language, which uses an efficient selection function and search rule for inference. Prolog has well-known limitations, though, which have been recognised since the early days of logic programming [11]. Approaches that propose more flexible execution strategies, such as tabling [1] and co-routining [3] [2], have thus been proposed.

One of the major problems that actual logic programming systems have to address is whether and how to prune undesirable parts of the search space. A region of the search space would definitely be undesirable if it can only repeat previously found solutions, or if it is well-known that the whole computation will fail. Or it may be the case that we are interested in a single solution, and thus there is no point in performing extra computation. We may want any solution, but in many cases programmers have an implicit ordering over the quality of
solutions. Last, programmers often go a step further and explicitly want to do incomplete search. Programs that rely on ordering or that demand incomplete search, are only meaningful given a specified selection function and search rule, and can be extremely difficult to run with other execution mechanisms.

In this work we discuss how pruning issues are addressed in the BEAM

1

an implementation of the Extended Andorra Model. Because the BEAM relies on a very flexible execution mechanism, all cases of pruning discussed above should be considered. Our work presents several contributions. The BEAM supports early completion of successful or failed computations naturally in its execution rules. The BEAM supports explicit pruning through the pruning operators cut and commit, by using the notions of quiet and noisy pruning, as originally proposed by Warren [21] and Haridi [5]. Quiet pruning allows for full co-routining, and is the desirable solution. Noisy pruning allows for Prolog compatibility, but reduces the amount of co-routining.

The paper is organised as follows. First, we present the BEAM design. Next, we give an overview of the issues related with cut on an EAM implementation and how we address them in BEAM.

2 The BEAM

We briefly present the BEAM, an implementation of the main concepts in Warren’s Extended Andorra Model with Implicit Control [15], with several refinements [13]. Our implementation owes to the experience in the design and implementation of Andorra-I, a predecessor of the BEAM, to Gupta’s first EAM interpreter [4], and to the work in the AKL language [9]. The BEAM model has been implemented for the Herbrand domain [12], although the EAM does support other constraint domains [19,8].

2.1 BEAM Concepts

A BEAM computation is a series of rewriting operations, performed on And-Or Trees. And-Or Trees contain two kinds of nodes: and-boxes represent a conjunction of positive literals, and store goals $G_1, \ldots, G_n$, new local variables $X_1, \ldots, X_m$, and a set of constraints $\sigma$; or-boxes represent alternative clauses.

A configuration is an And-Or Tree, describing a state of the computation. A computation is a sequence of configurations obtained by successive applications of rewrite rules that define valid state transitions. The initial configuration is an and-box representing the query. The constraints over the top-and box on a final configuration are called an answer. We define an And-Or Tree as compact when all children of and-boxes are or-boxes and when all children of or-boxes are and-boxes.

A goal is said to be deterministic when there is at most one candidate that succeeds for the goal. Otherwise it is said to be non-deterministic.

1 Not to be confused with the Erlang BEAM virtual machine [6].