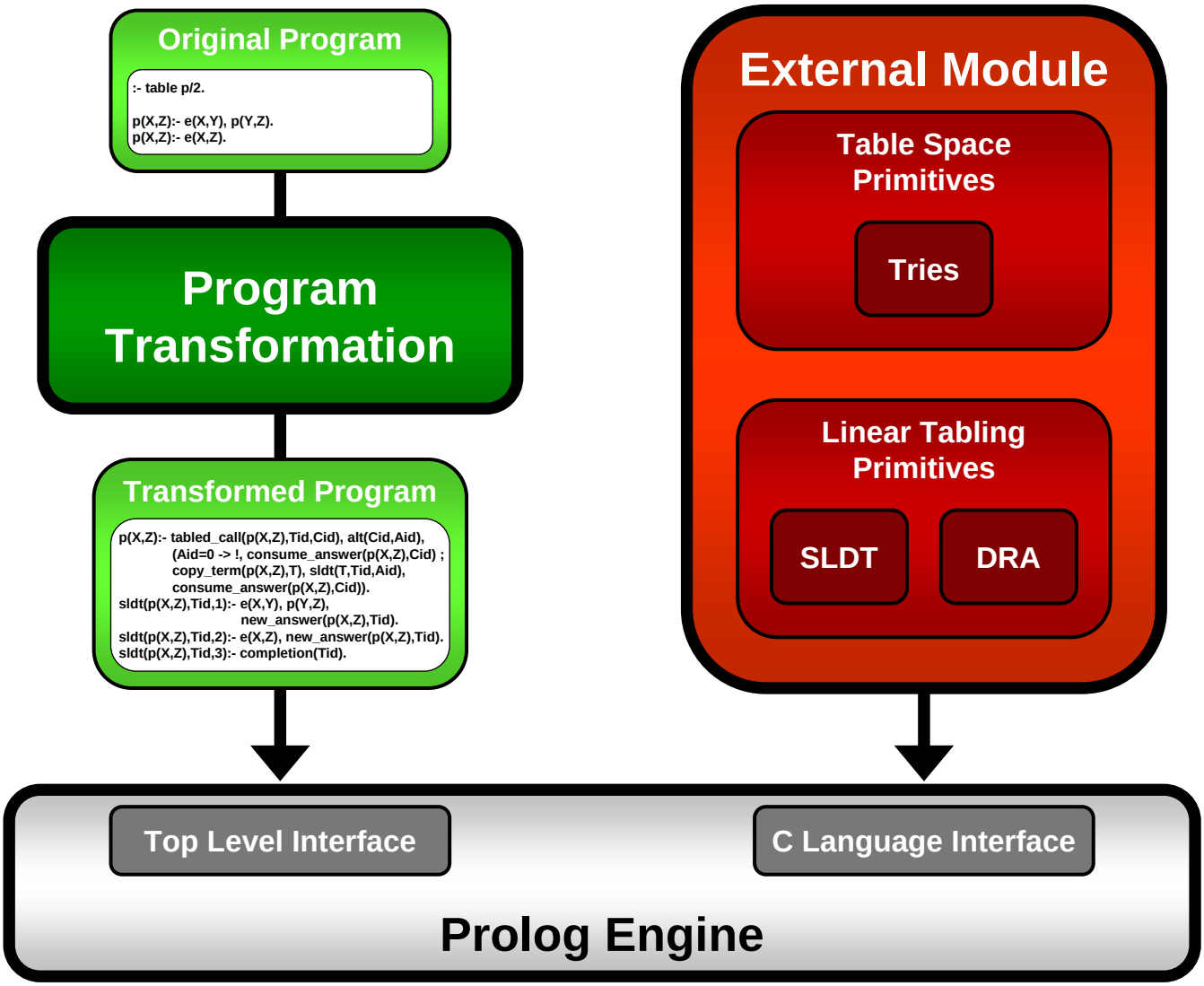
An External Module for Implementing Linear Tabling in Prolog

Cláudio Silva, Ricardo Rocha and Ricardo Lopes
DCC-FC & LIACC, University of Porto, Portugal



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.

In this work, we have designed a module to implement the two available mechanisms that, to the best of our knowledge, implement linear tabling: the **SLDT strategy** of Zhou *et al.*; and the **DRA technique** of Guo and Gupta. The key idea of the SLDT strategy is to let a tabled subgoal call 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.

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.