Available online at www.sciencedirect.com
SCIENCE DIRECT

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 128 (2005) 39–59
www.elsevier.com/locate/entcs

Toward the Concept of Backtracking Computation

Marija Kulaš[1]

FB Informatik, FernUniversität Hagen, Hagen, Germany

Abstract

This article proposes a new mathematical definition of the execution of pure Prolog, in the form of axioms in a structural operational semantics. The main advantage of the model is its ease in representing backtracking, due to the functionality of the transition relation and its converse. Thus, *forward and backward* derivation steps are possible. A novel concept of *stages* is introduced, as a refinement of final states, which captures the evolution of a backtracking computation. An advantage over the traditional stack-of-stacks approaches is a *modularity* property. Finally, the model combines the intuition of the traditional 'Byrd box' metaphor with a compact representation of execution state, making it feasible to formulate and prove theorems about the model. In this paper we introduce the model and state some useful properties.

Keywords: backtracking, Prolog, operational semantics

1 Motivation, aims and results

In this paper, we introduce S_1:PP, a new operational semantics for pure Prolog, and establish some useful properties, aiming toward an algebraic definition of the concept of Prolog computation. On the way, we obtain some new concepts useful for characterizing backtracking, but possibly also useful for objects which evolve over time. Such an object is in logic programming *the goal*, in its dynamic sense ('this unique, run-time invocation of a Prolog procedure'), as opposed to its static or syntactic sense ('this goal formula').

The goal is a basic concept of logic programming, but nevertheless one which proved hard to grasp in a formal way, even in the case of pure Prolog.

[1] Email: marija.kulas@fernuni-hagen.de

doi:10.1016/j.entcs.2004.10.026

The problem is the possible evolution of 'the' goal through slightly different identities, in case of a non-deterministic procedure.

For example, assume we pose the following query to a Prolog program: $\mathsf{p(X, Y), q(Y), fail}$. Assume two answers for $\mathsf{p(X, Y)}$, say $\mathsf{p(a, Z)}$ and $\mathsf{p(b, 3)}$. In the static sense, we have only one goal $\mathsf{q(Y)}$, but dynamically we can have two different goals: first $\mathsf{q(Z)}$, and if $\mathsf{q(Z), fail}$ terminates, then $\mathsf{q(3)}$. Both goals have their own creation, lifetime of forward and backward execution, and possible expiration. It is vital for an operational semantics of backtracking to differentiate between these objects. Moreover, assume $\mathsf{q}/1$ is recursive. During the computation of e. g. the goal $\mathsf{q(Z)}$, we need to pay attention to all the recursive invocations of $\mathsf{q}/1$ as well, in order to know exactly where to go after a failure. Are they all to be considered distinct objects as well, of the same kind as the above two, $\mathsf{q(Z)}$ and $\mathsf{q(3)}$, and how to manage them anyway?

In the rest of this paper we proceed as follows. First a canonical form of predicates is defined, into which the original pure Prolog program shall be transformed. Then, in Section 3, a novel operational semantics of pure Prolog is defined, in a structural operational manner. Throughout the Section 4 – Section 7 we develop formal tools (concepts and theorems) suitable for characterizing Prolog computation. This obviously includes defining in some way or other the (dynamic) concept of the goal as well. We solved this problem by means of *stages*, through which an initial event (representing the creation of a goal) passes in the course of computation. Stages can be seen as a generalization of the normal form idea, in the sense that stages are independent on the context of computation, as shown in Section 7, but organized by macro transitions. Starting from individual transitions as given in the model, simple derivations (forward and backward) are built, which are the basis of simple passes, and simple passes aggregate into composed passes. Finally, we show in Section 8 how composed passes model Prolog computations.

Our approach can be seen as a formalization of the original Byrd model. But there is an important detail: we extend the notion of a port, initially conceived by Byrd for selected atoms, to *general goal formulas*. The shifting of attention from atoms to general goal formulas proved to be a key idea and made a very simple model possible. The model in its first version, called S:PP, was proposed in [18]. But the handling of variables turned out to be difficult. The new model S_1:PP improves on that.

2 Preliminaries

Before it can be interpreted in our model, the original Prolog program has to be transformed into a canonical form, the common single-clause representation.