Abstract

ABSTRACT. A Prolog or a Concurrent Prolog program can be seen as a specialization or refinement of a program in Horn Clause Logic: In addition to the logic component a Prolog or Concurrent Prolog program contains information about the flow of control. In Prolog we have the cut statement and a leftmost depth first search strategy, in Concurrent Prolog we have read-only variables and commits. In this paper we study the flow of control of these languages by giving transition systems for abstract versions of Prolog, Horn Clause Logic and Concurrent Prolog. On the basis of these transition systems we define operational semantics for all three languages. Three basic sets (success set, finite failure set and the infinite failure set or divergence set) can be derived from the operational semantics. A comparison is made between the different sets: for Prolog we show that the success set and the finite failure sets of a Prolog program are smaller than the corresponding sets of a Horn Clause Logic program. The infinite failure sets are incomparable. A similar comparison is made between the success set and the finite failure sets for Horn Clause Logic and Concurrent Prolog. These comparisons give some feeling what happens if we put extra logical information in Horn Clause Logic programs.

Remark: part of the work was carried out in the ESPRIT Basic Research Action Integration.

Key words: Logic programming, operational semantics, Horn Clause Logic, Prolog, Concurrent Prolog, cut operator, backtracking, committed-choice, synchronization mechanisms.
1 Introduction

The ideal way of programming is to start from a specification and to derive in several refinement steps a program that can be executed efficiently. At first sight, logic programming seems to be very suited for this way of programming: the specification equals the program and there are no refinement steps needed. It turns out, however, that the programs we get in this way are not very efficient. Therefore in almost all realistic logic programming languages we have control components like the cut operator of Prolog, the commit operator and the synchronization mechanisms of concurrent logic languages and the different kinds of search strategies. Hence it requires at least one refinement step to go from a specification (a pure logic program) to a program say in Prolog in which we add the control component.

In this paper we look how the addition of control components influences the semantics of logic programs. Also we are interested how certain basis sets are related. The basis sets we consider are

1. the success set: This is the set of instantiations of atoms that (can) successfully terminate (those atoms which associated goal has a finite refutation). We instantiate the atoms with the computed answer substitutions.
2. the finite failure set: the set of atoms for which the execution can end in deadlock (atoms that can fail).
3. the infinite failure set: the set of atoms that can produce infinite computations. (Using a different terminology one might call this set the divergence set.)

These sets are defined in a formal way with the aid of transition systems.

These transition systems are defined for more abstract languages than Prolog, HCL and CP, but they can be seen as a kind of subsets of the abstract languages. In this paper we do not give introductions to the three languages.

The transition systems and the operational semantics that are derived from them focus on the flow of control of the three languages. There are different kinds of semantics that look at other aspects of the languages (for example denotational and declarative semantics). We do not consider them