ABSTRACT

To resolve the search-incompleteness of depth-first logic program interpreters, a new interpretation method based on the tabulation technique is developed and modeled as a refinement to SLD resolution. Its search space completeness is proved, and a complete search strategy consisting of iterated stages of depth-first search is presented. It is also proved that for programs defining finite relations only, the method under an arbitrary search strategy is terminating and complete.

1. Introduction

The most fundamental principle of logic programming is the equivalence of the declarative and the procedural semantics[1], which has led many researchers to believe logic programming to be a suitable framework for various program manipulation tasks such as verification, synthesis, transformation, and formal debugging. The unfortunate fact is that this equivalence is always sacrificed in real implementations like Prolog, for the sake of execution efficiency. As a result, logically correct logic programs do not necessarily run correctly on Prolog.

One may argue that Prolog is just a programming language with its own procedural semantics, of which programmers should have sufficient knowledge. But the abandonment of the equivalence is so deeply concerned with the philosophy and the potential of logic programming that it could not be approved easily.

There are several causes of this dis-equivalence: the absence of occur check, the depth-first search strategy, and inclusion of many extra-logical features. We attack the second problem in this paper: we develop an interpretation method which is complete even under essentially depth-first search strategies.

The completeness of SLD refutation [2,3,4] ensures that given a conjunction of atomic formulas as a query, every instance of it implied by the program can be obtained as a result of some computation path. Though a typical Prolog interpreters are essentially SLD refutation procedures, they are not complete in the sense that they can, with their depth-first search strategy, be trapped by an infinite computation path in the search tree, and fail to find an actually existing successful computation path.
The breadth-first strategy might seem a sufficient theoretical answer to this problem, since it will eventually find any successful computation path in the search tree. But apart from the practical problem of its storage requirement, it also suffers from infinite paths of the search tree. In this case, though it is not trapped from successful computation paths, it can be trapped from termination if all solutions are requested, even when the set of solutions is finite.

Several authors, including Brough and Walker [5], proposed techniques to prune the infinite paths in the search tree by detecting identical or matching goals on a path. However, all of such techniques are incomplete in two ways (as was studied in [5]) : some infinite paths can escape the pruning, and some pruned infinite paths can have side branches which constitute successful computation paths.

The interpretation method we develop here can be considered as a remedy to such pruning techniques, as well as a generalization of the tabulation techniques [6] for functional programs, where the result of evaluating a function call is stored in a table to eliminate repeated evaluation of the function calls for the same arguments. Though the same effect of avoiding the redundant evaluation of a goal is achieved as a side benefit, the principal purpose of applying the technique here is to prevent the interpreter from repeatedly entering the evaluation of the same goal in a single computation path and thus from being trapped by an infinite path. The suspended computation node in the path is later fed, through the table, with the solutions of the other computation paths for the goal in question. In this way, the completeness of the SLD refutation is preserved.

The next Section presents an example to illustrate the above discussion and to informally explain our interpretation method. Section 3 and Section 4 contain more detailed description and some completeness results. Finally in Section 5, we conclude by summarizing the advantage of our method.

2. Examples

2.1 Infinite paths in search trees

Consider the following program to define the reachability relation in a directed graph. (We follow the DEC10 Prolog convention of designating variables by upper case letters.)

PROGRAM 2.1 (graph reachability)

(C1) reach(X,Y) ← reach(X,Z), edge(Z,Y).
(C2) reach(X,X).
(C3) edge(a,b).
(C4) edge(a,c).
(C5) edge(b,a).
(C6) edge(b,d).