Logic Programming Debugger
Using Control Flow Specification

Hideo Tamura
Hideo Aiso
Keio University
Japan

In this paper, a control specification for logic programming and the Deterministic Trimming Method for bug detection are Proposed. Using information from the control specification, the query complexity of our Deterministic Trimming Method is less than that of Shapiro's debugging system.

§1 Introduction
Prolog is a popular language for programming prototypes of parallel language interpreters and natural language processors, and for knowledge representation. But it is said that it is difficult to show Prolog's execution flow explicitly, so one cannot understand another's programs. To reduce the programmer's load, many Prolog debugging systems have been proposed and used. They are classified into two groups.

1. Programmer oriented
   Using debugger's facilities to detect bugs by programmer.
   [Byrd78][Numao85]

2. System oriented
   Debugging system asks programmer about a test program, analyses that information and detects bugs.
   [Shapiro83][Pereira,L.M 86][Lloyd87][Maechi87]
   [Takahasi86]

If there were a program-specification that did not put additional load on programmer and did not let down the Prolog's abilities, it would be very useful and make Prolog more powerful. In this paper, a control specification is proposed, that is based on the Control Flow Model [Byrd78], very familiar to Prolog users as the Box Model. This control specification can represent the programmer's intention of how the program is to perform.

We propose the Deterministic Trimming Method, which uses control specification information based on Shapiro's Algorithmic Program Debugging [Shapiro83].

§2 Control Specification
If there are specifications of each goal's mode declarations, argument types and determinacy, it is easy to debug and maintain a program. Even a beginner will understand the program's meaning easily.

There is not a general Prolog program specification notation, even if there were a new one proposed, it probably would be difficult to get acquainted with, and
programmer's load would increase linearly with the number of program steps.

Therefore a control specification based on the popular control flow model is adopted and applied to this debugging system.

§2.1 Control flow specification

Whereas the control flow model represents the dynamic runtime behavior of a program, control specification represents it statically. Although control specification cannot represent a program's entire behavior, it can represent the programmer's practical intention.

In the following, control specification is described, and we indicate how the programmer's intention is represented and its relationship with the debugger.

The control specification mainly represents relations between goals of program text. In figure 2.1, its difference from the control flow model is an additional 'FAIL' line. This additional 'FAIL' line is used for 'cut fail', which means backtracking invoked after a 'cut' operation does not affect goals before the 'cut' in the same clause.

![Fig 2.1](image)

There are 5 patterns of control specification, which the programmer can use to express control flow.

![Fig 2.2](image)

- Goal ① succeeds deterministically does not allow backtracking (deep backtrack).
- Goal ② finitely fails.
- Goal ③ always succeeds. If there is backtracking after the goal's success, there must be alternative solutions of the goal. There is never the case of no solution of the goal.
- Goal ④ permits both success of failure. If the goal succeeds the goal does not allow backtracking (deep backtracking).

Examples of control specifications for some built-in predicates is shown in figure 2.3.