Error Diagnosis in Prolog Programs, A Critical View

Gabriella Kókai
Institute of Informatics, József Attila University
Árpád tér 2, H-6720 Szeged, Hungary
e-mail: kokai@sol.cc.u-szeged.hu
now visiting Department of Computer Science, Programming Languages
Friedrich-Alexander University of Erlangen-Nürnberg
Martensstr. 3. D-91058 Erlangen, Germany
e-mail: kokai@informatik.uni-erlangen.de

Abstract

Programming environments which include program development, program analysis and program debugging tools ([2]) are essential for the acceptance of programming languages as Prolog or other logic programming languages as a "real" one. Especially testing which is a process of executing a program with the intend of finding error (see [10]), and debugging which goal is to analyse the program to locate and fix the detected error (see [6]) are central issues of programming environments.

Research on this topic started with the development of Shapiro's APD-System ([12]). The algorithmic program debugging method introduced by Shapiro can isolate an erroneous procedure if a program and an input on which it behaves incorrectly is given. Shapiro's model was originally applied to Prolog programs to diagnose the following three types of errors: termination with incorrect output, termination with missing output and nontermination. A major drawback of this debugging method is the great number of queries put to the user about the correctness of intermediate results of procedure calls.

Many other debugging techniques and tools of Prolog programs were developed. A few of them base on Shapiro's algorithm (APROPOS [9], EDS [5], DED [8] and ADAss [1]) or used other methods (PRESET [14], RD [11], OPIUM [2] and PTP [4]) but the main disadvantage of these systems is that they can handle only *pure* Prolog programs. It means that these systems cannot be applied for *realistic* programs. For example let us take the following short Prolog program which parses simple arithmetic expressions and computes the result:

```
expression(Value) --> term(A), "+", expression(B), {Value is A + B}.
expression(Value) --> term(A), "-", expression(B), {Value is A - B}.
expression(Value) --> term(Value).
term(Value) --> atomicexpression(Value).
term(Value) --> atomicexpression(A), "*", term(B), {Value is A * B}.
term(Value) --> atomicexpression(A), "/", term(B), {Value is A / B}.
atomicexpression(Value) --> number(Value, _).
atomicexpression(Value) --> "(", expression(Value), ")".
digit --> "1"; "2"; "3"; "4"; "5"; "6"; "7"; "8"; "9"; "0".
number(Value, Expo, C, D) :- 'C'(C,Code,E), digit(C,E),
                              (Expo = 10 -> Value is Code - "0", D=E;
                                Value is Code * B + A, Expo is B * 10, number(A, B, E, D)).
compute(Value, A, B) :- expression(Value, A, B).
```

The systems listed above cannot handle the extensions of Prolog used in this program. This paper describes a new system developed for algorithmic debugging and functional testing of Prolog programs based on Shapiro's work . This system is able to find the error in a program containing the following

- Declarations
- Control facilities: if (->), or (;), repeat
- Definit Clause Grammars
- Handling Built-In Predicates

The task of the system is the following:

- Develop a grammar for Prolog. With the help of this grammar parse the program and generate a tree for the syntax. From the tree produce a prooftree in the programming language *C* which produces the graphical representation for the graphical user interface (see Figure 1)
- The modifications of Shapiro's debugging methods that they can handle the whole special features of ISO Prolog

References

[1] Drabent, W., Nadjm-Tehrani, S., Małuszyński, J.: *Algorithmic Debugging with Assertions*, In Proc Meta-Programming in Logic Programming, 1988, 375-378,

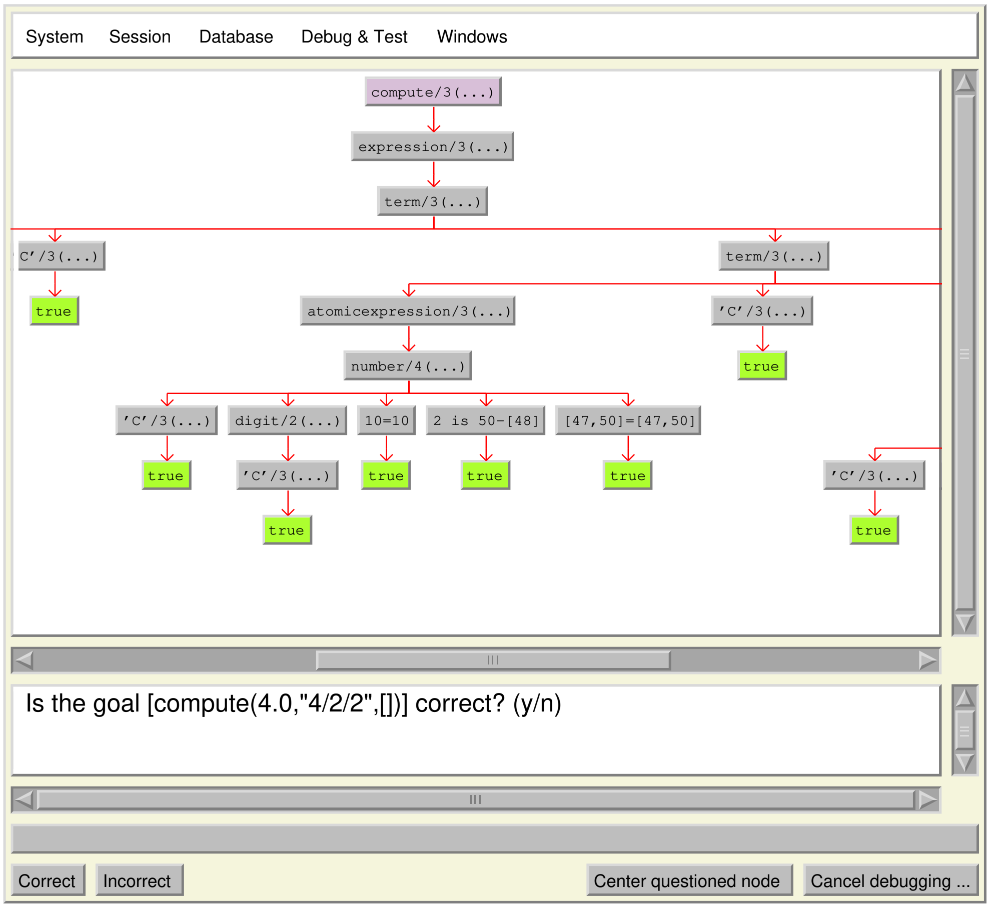

Figure 1: The generated prooftree in main window of the system

[2] Ducasse, M.: *Opium - An Extensible Tracer for Prolog*, Technical Report Lp14, January 1987, 203-210

[3] Ducasse, M. Noye J.: *Logic Programming Environments Dynamic Program Analysis and Debugging*, Journal of Logic Programming, 19/20, 1994, 351-384

[4] Eisenstadt, Marc: *Retrospective Zooming: A Knowledge Based Tracing and Debugging Methodology for Logic Programming*, In Proc of the 9.th IJCAI, 1985

[5] Ferrand, Gérard:, *Error Diagnosis in Logic Programming an Adaptation of E.Y. Shapiros's Method*, Journal of Logic Programming, 1987, 177-198

[6] Kamkar M., Shahmehri N., Fritzson P.: *Bug localization by algorithmic debugging and program slicing*, In Proc. of the Programming Language Implementation and Logic Programming Symposium, Linköping, Sweden, 1993, Deransart and Małuszyński, LNCS 456 Springer

[7] Kókai, G., Harmath, L., Gyimóthy, T.: *Algorithmic Debugging and Testing of Prolog Programs* ICLP '97 The Fourteenth International Conference on Logic Programming, Eighth Workshop on Logic Programming Environments Leuven, Belgium, 8-12 July 1997, 14-21

[8] Lloyd J. W.: *Declarative Error Diagnosis*, New Generation Computing, 1987, 133-154, OHMSHA LTD and Springer Verlag

[9] Chee-Kit Looi: *Analysing Novices' Programs in Prolog Intelligent Teaching System*, In Proc of the European Conference on Artificial Intelligence, Munich 1988, 314-319,

[10] Myers. G. J.: *The art of software testing Business data processing* John Wiley and Sons, 1979

[11] Pereira, L. M. : *Rational Debugging in Logic Programming* In Proc of the Third Logic Programming Conference, London, England, July 1986, 203-210

[12] Shapiro, E. Y.: *Algorithmic Program Debugging* MIT Press (1983)

[13] Sterling, L., Shapiro, E., Y.: *The Art of Prolog* The MIT Press Cambridge Mass. 1986

[14] Takahashi, H., Shibayama, E.: *PRESET - A Debugging Environment for Prolog* , Proc of the International Conference on Logic Programming, Tokyo, 1985