Algorithmic Debugging for Imperative Languages with Side-effects*

Nahid Shahmehri (E-mail: nsh@ida.liu.se) Peter Fritzson (E-mail: paf@ida.liu.se)
Dept. of Comp. & Info. Science, Linköping University, S-581 83 Linköping, Sweden.

Abstract

Algorithmic debugging is a technique for semi-automatic localization of program errors. So far, this technique has been limited to programs without side-effects, and has only been applied to Prolog programs. In this paper, we generalize the algorithmic debugging method to programs written in imperative languages, e.g. Pascal, which may contain side-effects.

Our method combines program transformation techniques with data flow analysis techniques to achieve this goal. Programs which contain side-effects are transformed or mapped to programs without side-effects. These transformations are guided by data flow analysis results. The conventional procedure level algorithmic debugging technique is used on the transformed or mapped program, but the debugging process is presented to the user in terms of the original program. The model can be extended to statement level debugging. A prototype for a subset of Pascal has been implemented. A larger prototype including transformations is being implemented within the DICE system - a programming environment based on incremental compilation. Currently we restrict ourselves to bug localization for terminating programs. Also, side-effects related to pointers are currently not considered.

What is Algorithmic Debugging?

A program is erroneous if there is a difference between its actual result, i.e. *actual program behavior*, and its expected result, i.e. *intended program behavior*. This difference is usually the symptom of a bug in the program. The process of program debugging can be divided into two stages: bug localization and bug correction [Shapiro-83]. This paper only considers the bug localization problem.

The user activates the algorithmic debugger when he/she notices a bug. The algorithmic debugger then builds an *execution tree* for the program, i.e. a tree structure containing information about the actual program execution, such as a trace of actual parameters. The execution tree is subsequently inspected by the debugger in order to localize the program bug. Procedure level algorithmic debugging is based on the assumption that an error in an execution of a procedure p is caused by one or both of the following conditions:

- There is at least one procedure call inside procedure p which contains a bug and returns a wrong result.
- There is an error in the computations inside the body of procedure p.

The assumption above guarantees that an error is finally localized in the program.

The debugger gathers knowledge about the intended program behavior, while inspecting the execution tree. This knowledge is obtained through a number of queries to the user. The de-

* *This work was supported by STU, The Swedish Board for Technical Development.*

bugger asks the user questions of the form: is procedure *q* supposed to return the output value *m* on its input value *n*? The user answers "yes" or "no", or he/she can give an *assertion* about the intended behavior of procedure *q* [Drabent, et al-88]. An *assertion* is a predicate which expresses an input-output relation for a procedure. A given assertion may eliminate some future questions. Finally, a bug will be localized at the procedure level.

Language Requirements and Program Transformations

Shapiro has given a general model which can be applied to purified applicative versions of many programming languages. Examples of such languages are pure Lisp, Pure Prolog and Loop-free Algol-like languages with no side-effects.

Shapiro has imposed two restrictions on programs that are subject to his method of algorithmic debugging: *loop-freeness* and *side-effect freeness*. However, loop-freeness is not an essential property [Shahmehri, Fritzson-89]. Side-effect freeness is however essential to the precision of the bug localization process [Shahmehri, Fritzson-89]. There is a discussion in [Kamkar, et al-90] on relaxing those transformations which are not absolutely necessary for the purpose of procedure level algorithmic debugging.

Shapiro's model covers a very restricted class of programs in Algol-like languages. However most practical systems do not conform to such restrictions. Our extension of algorithmic debugging covers a more general class of programs which can be written in imperative languages where side-effects and loops are allowed.

Our model for algorithmic debugging consists of two phases, a *program transformation phase* and an *algorithmic program debugging phase*. The first phase transforms a program in an imperative language to an intermediate functional representation of the same program which is accepted by Shapiro's model. The transformation phase is guided by results from data flow analysis techniques. The second phase applies algorithmic debugging to this intermediate representation. The programmer never sees the program in this intermediate representation, as all dialogues refer to the original source program. The debugging system maintains a mapping between the original and transformed program constructs.

The program transformations deals with variable assignments, global variables, loops and goto-statements (local or non-local). A detailed description of these transformations can be found in [Shahmehri, Fritzson-89].

Current State of Implementation

An algorithmic debugger has been implemented for a subset of Pascal. The system is implemented within the DICE system [Fritzson-83]. The *algorithmic debugging phase* is fully implemented. The *program transformation phase* is based on the transformations described in [Kamkar, et al-90]. However, loop transformations are not yet implemented.

References

[Drabent, et al-88] Drabent W., Nadjm-Tehrani S. and Maluszynski J., The Use of Assertions in Algorithmic Debugging. Proceeding of the FGCS Conference, Tokyo. 1988, pp.573-581.

[Fritzson-83] Fritzson P., Symbolic Debugging Through Incremental Compilation in an Integrated Environment. The Journal of Systems and Software 3, 1983, pp.285-294.

[Kamkar, et al-90] Kamkar M., Shahmehri N. and Fritzson P., Bug Localization by Algorithmic Debugging and Program Slicing. Proceeding of the PLILP'90, Linköping, Sweden. LNCS, Vol 456, pp.60-74.

[Shahmehri, Fritzson-89] Shahmehri N., Fritzson P., Algorithmic debugging for Imperative Languages with Side-effects, Research Report LiTH-IDA-R-89-49, Linköping University, Sweden, 1989.

[Shapiro-83] Shapiro E. Y., Algorithmic Program Debugging. MIT press, 1983.