MECHANIZATION OF AN ORACLE IN A DEBUGGING SYSTEM

Anneli Edman and Sten-Ake Tirnlund
UPMAIL
Uppsala Programming Methodology and Artificial Intelligence Laboratory
Department of Computing Science, Uppsala University
P.O. Box 2050, S - 750 02 Uppsala, Sweden

Abstract

We shall make use of a *programming calculus* to derive the correct result a program is expected to compute. In this way, the decision by an *oracle* whether or not a result is correct can be derived formally from a specification (model) of the program, and thus the oracle can be mechanized. The *debugging system* consists mainly of a *derivation editor* and a *fault finding program* that collects an erroneous procedure.

Keywords: Debugger, Derivation Editor, Logic Programming, Specification, Sub-Specification, Verified Test.

1. Introduction

The problem of correcting programs has troubled computer scientists since the very first computer programs. Studies of the debugging process itself have, for example, led to a theory of skill acquisition (Sussman, 1075) and to inferring theories (Shapiro, 1081). We shall take up the debugging problem from what seems to be a new point of view, namely *a verified test* that can lead to a mechanized oracle.

Let us first briefly take up the relations between *formal derivations, verification* and *formal tests* of programs before we go into details of the latter idea. The situation, especially for logic programs (Kowalski, 1074) is currently changing its character entirely. The advent of derivation editors (Eriksson, Johansson and Tarnlund, 1082) makes it possible to semi-automatic ally and formally derive correct non-trivial programs from their specifications, for example, derivations of the programs in (Hogger, 1070) and (Hansson and Tarnlund, 1070) and in (Clark and Darlington, 1080) are reported in (Blomberg, Carlsson and Eriksson, 1082) and (Johansson, 1083). They also report on semi-automatical formal proofs of correctness of non-trivial programs, for example, the programs in (Clark and Tarnlund, 1077).

The idea of deriving a program from a specification is to construct a correct program. In contrast, the idea of verifying that a program is correct requires that the program is constructed by an oracle. When it is correct we intend to confirm this fact, but when it is not we have, eventually, to ask an oracle for another program to verify. However, when debugging a program we know that we can not confirm that a program is correct even after arbitrarily many tests, but still worse even a single test is not verified correct, since it has to rely on a judgment by an oracle i.e., an oracle decides whether or not a computed result is correct. It is this judgment that we shall formalize so the answer can be proved correct and furthermore lead to a mechanization of the oracle.

In order to carry out this idea we shall make use of a specification (model T*)* of a program, IP, in particular, a sub-specification (sub-model *OCT)* that exactly specifies a particular program. We shall use a *derivation editor* to derive an object w corresponding to an input object 7 from the sub-specification & that a program IP is supposed to compute i.e.,

$$\Theta(\gamma,\omega) \vdash \Psi(\gamma,\omega) \tag{1}$$

This work was sponsored by the National Swedish Board For Technical Development (STU).

2. Specifications of a class of programs and sub-specifications for a particular program

Some of the main advantages of a specification, Γ, are its abstractness, clearness and shortness, as a consequence we often not only specify *one program*, Ψ, but a class Ω of programs Ψ i.e.,

$$\Omega = \{\Psi : \Gamma \vdash \Psi\} \tag{2}$$

This is fine both when we develop programs formally and verify correctness of programs, but not so good when we want to compute, from the specification, objects that a particular program should compute. For example, it could happen that we derive the input-output pair γ,ω of a predicate Φ i.e., $\Gamma(\gamma,\omega) \vdash \Phi(\gamma,\omega)$, but that we could not derive this pair from the program Ψ i.e., $\Psi(\gamma,\omega) \not\vdash \Phi(\gamma,\omega)$. What we want to find is a sub-specification $\Theta \subseteq \Gamma$ such that a derivation of the objects γ', ω' from the sub-specification Θ i.e., $\Theta(\gamma',\omega') \vdash \Phi(\gamma',\omega')$ necessarily implies that there exists a derivation of these objects from the program Ψ i.e., $\Psi(\gamma',\omega') \vdash \Phi(\gamma',\omega')$.

Let us now explain the situation more generally. Suppose that we have a specification Γ and a set Ω of programs as in (2) and a program $\Psi \in \Omega$ that we shall test, then we want to find a specification Θ, from which we can derive exactly the objects that Ψ should compute. A sufficent condition is to find a relation, Δ, that gives a sub-specification Θ i.e.,

$$\Theta \rightarrowtail \Gamma \wedge \Delta \tag{3}$$

such that the program, Ψ, is the only element in the set

$$A = \{\Psi : \Theta \vdash \Psi\}. \tag{4}$$

The relation, Δ , must satisfy the condition,

$$\Delta \vdash \Sigma \tag{5}$$

where,

$$\Gamma \rightarrowtail \Sigma \wedge \Pi \tag{6}$$

hence,

$$\Theta \rightarrowtail \Pi \wedge \Delta \tag{7}$$

The relation Δ eliminates Σ of Γ from Θ and can in this way, as we shall see below, shorten the computation of objects from $\Theta \subset \Gamma$ in comparison to a derivation from Γ.

3. Derlving objects from a specification of a particular program

Suppose that we have a specification of an insert relation Γ , then according to the relations (3) and (4) we want to find a condition Δ leading to a sub-specification Θ of our logic program Ψ for inserting a label into an ordered binary tree (ob-tree). Let us now write down these expressions, in more detail in English followed by a formalization.

We have first the specification, Γ, of an insert relationship between two ordered binary trees (ob-tree) and a label. The ob-tree w' is the result of inserting the label u to the ob-tree w if and only if all elements that belong to w' also belong to w or is the label u, and moreover w and w' are both ob-trees and u a label. We write this formally as:

$$\forall w \forall u \forall w' (\text{insert}(w, u, w') \leftrightarrow \text{obtree}(w) \wedge \text{obtree}(w') \wedge \text{label}(u) \wedge p(w, u, w')) \quad (8)$$

$$\forall w \forall u \forall w' (p(w, u, w') \leftrightarrow \forall v (v \in w \vee v = u \leftrightarrow v \in w')) \quad (9)$$

A label u belongs to an ob-tree w if and only if w is a label and w an ob-tree constructed as $t(x, y, z)$ and u is the label y or u belongs to the ob-tree x or u belongs to the ob-tree z. Formally,

$$\forall u \forall w (u \in w \leftrightarrow \text{label}(u) \wedge \text{tree}(w) \wedge \exists x \exists y \exists z (w = t(x, y, z) \wedge (u = y \vee u \in x \vee u \in z))) \quad (10)$$

An ob-tree w is either empty ,ϕ, or constructed $t(x, y, z)$ where both x and z are ob-trees and y a label, moreover, all elements that belong to x are less than or equal to y and all elements that belong to z are greater than y. Formally,

$$\forall w (\text{obtree}(w) \leftrightarrow w = \phi \vee \exists x \exists y \exists z (w = t(x, y, z) \wedge \text{obtree}(x) \wedge \text{obtree}(z) \wedge \text{label}(y) \wedge \forall u (u \in x \rightarrow u \leq y) \wedge \forall u (u \in z \rightarrow u > y))) \quad (11)$$

A tree is either empty or constructed of subtrees and a label.

$$\forall w (\text{tree}(w) \leftrightarrow w = \phi \vee \exists x \exists y \exists z (w = t(x, y, z) \wedge \text{tree}(x) \wedge \text{tree}(z) \wedge \text{label}(y))) \quad (12)$$

Now all ob-trees are trees (but not conversely).

$$\forall w (\text{obtree}(w) \rightarrow \text{tree}(w)) \quad (13)$$

All constructed trees are different from the empty tree.

$$\forall x \forall y \forall z (\neg t(x, y, z) = \phi) \quad (14)$$

However, the set Γ of programs is not unique, so we need a sub-specification Θ (see (7)) that characterizes our particular program Ψ. We introduce the following condition as our Δ (see (5))

$$\forall w \forall u \forall w' (s(w, u, w') \leftrightarrow \text{tree}(w) \wedge \text{tree}(w') \wedge \text{label}(u) \wedge (w = \phi \wedge w' = t(\phi, u, \phi) \vee \exists x \exists y \exists z \exists x' \exists y' \exists z' (w = t(x, y, z) \wedge w' = t(x', y', z') \wedge y = y' \wedge (x = x' \wedge s(x, u, z') \vee z = z' \wedge s(z, u, x')))) \quad (15)$$

which satisfies $\Delta \vdash \Sigma$, where Σ is (9), and leads to the following sub-specification Θ of our program Ψ

$$\forall w \forall u \forall w' (\text{insert}(w, u, w') \leftrightarrow \text{obtree}(w) \wedge \text{obtree}(w') \wedge \text{label}(u) \wedge s(w, u, w')) \quad (16)$$

Let us write down three statements that comprise our program, Ψ.

The result of inserting a label u into an empty tree ϕ is $t(\phi, u, \phi)$.

$$\text{insert}(\phi, u, t(\phi, u, \phi)) \quad (17)$$

The result of inserting a label u into an ob-tree $t(x, y, z)$ is $t(x', y, z)$ if u is less than or equal to y and x' is the resulting ob-tree from inserting u into the ob-tree x.

$$\text{insert}(t(x, y, z), u, t(x', y, z)) \leftarrow u \leq y \wedge \text{insert}(x, u, x') \quad (18)$$

There is a symmetrical case when the label u is greater than y.

$$\text{insert}(t(x, y, z), u, t(x, y, z')) \leftarrow u > y \wedge \text{insert}(z, u, z') \quad (19)$$

So, we can now derive from our sub-specification Θ objects that our program Ψ shall compute. In contrast, we notice that from the specification Γ we can derive objects that Ψ should not compute.

4. Example

The result of deriving a formula , $\exists w \text{insert}(t(t(\phi, 5, \phi), 9, \phi), 7, w)$ from the specification, Θ gives the result that w is the ordered binary tree, $t(t(\phi, 5, t(\phi, 7, \phi)), 9, \phi)$. For reasons of space, we show in the appendix the main structure of the derivation carried out in *NATDED* (Eriksson, Johansson and Tärnlund, 1982).

5. Conclusions

We have described the essential part of a debugging system which is based on the the programming calculus of (Hansson and Tarnlund, 1079). We use the derivation editor to compute elements of the input output pair and then ask the program to compute the same result, or conversely verify by the editor the pairs computed by the program. This leads to a formal treatment of testing, and moreover the decision by an oracle whether or not a result is correct is substituted by a (semi)mechanized oracle. When a program does not compute a correct result a fault finding system selects the procedure that is responsible for the wrong instantiation. An oracle is then supposed to produce a better procedure and we can restart the testing cycle. Mechanization of the latter oracle is a goal of formal program development.

The programs we are debugging are logic programs. The system is programmed in Prolog (Colmerauer, Kanoui, Pascro and Roussel, 1073) and running on DEC TOPS-20 Prolog (Warren, Pereira and Pereira, 1970).

6. Acknowledgement

The hospitality, when writing a first version of this paper, from Nils Nilsson and SRI is much appreciated. We also thank our colleagues at UPMAIL, in particular, Gunnar Blomberg and Martin Nilsson for their help with typesetting this paper using TEX.

7. References

[I] Blomberg G., Carlsson M., and Eriksson L-H., *Deriving Logic Program;* SAIS-82, Uppsala 1082.

[2] Clark K., and Tarnlund S-A., *A Firtt Order Theory of Data, and Program;* IFIP-77, Toronto, North-Holland, 1077.

[3] Clark K., and Darlington J., *Algorithm Classification Through Synthesis,* The Computer Journal 23 no 1, 1080.

[4] Colmerauer A., Kanoui H., Pasero R., and Roussel P., *Un Systeme de Communication Homme-machine en Francais,* Research Report, Groupe Intelligence Artiflcielle, University Aix-Marseille II, 1073.

[5] Eriksson A., Johansson A-L., and Tarnlund S-A., *Towards a Derivation Editor,* First International Logic Programming Conference, Marseille, 1082.

|6| Hansson A., and Tarnlund S-A., *A Natural Programming Calculus,* IJCAI-70, Tokyo, 1070.

[7] Hansson A., and Tarnlund S-A., *Derivationt in a Natural Programming Calculus,* Electrotechnical Laboratoy Tokyo, 1070.

|8) *Hogger* C, *Derivation of Logic Programme,* Ph.D. Thesis Department of Computing, Imperial College, London, 1070.

(0) Johansson A-L., *Results of Program Derivations in a Derivation Editor - NATDED,* UPMAIL Report, Computing Science Departrment, Uppsala University, 1083.

[10] Kowalski R., *Predicate Logic at a Programming Language,* IFIP-74, Stockholm, North-Holland, 1074.

(II) Shapiro E., An *Algorithm that Infers Theories from Facts,* IJCAI-81, Vancouver, 1081.

(12) Sussman G., *A Computer Model of Skill Acquisition,* American Elsevier, 1075.

(13] Warren D., Pereira F., and Pereira L., *User's Guide to Decsystem-10 Prolog,* Department of Artificial Intelligence, University of Edinburgh, 1070.

Appendix

SPECIFICATIONS

$$\forall w \forall u \forall w'(\mathrm{insert}(w,u,w') \leftrightarrow \mathrm{obtree}(w) \wedge \mathrm{obtree}(w') \wedge \mathrm{label}(u) \wedge s(w,u,w'))$$

$$\forall w(\mathrm{obtree}(w) \leftrightarrow w = \phi \vee \exists x \exists y \exists z(w = t(x,y,z) \wedge \mathrm{obtree}(x) \wedge \mathrm{obtree}(z) \wedge \mathrm{label}(y) \wedge \forall u(u \in x \rightarrow u \leq y) \wedge \forall u(u \in z \rightarrow u > y)))$$

$$\forall x\, \mathrm{label}(x)$$

$$\forall w \forall u \forall w'(s(w,u,w') \leftrightarrow \mathrm{tree}(w) \wedge \mathrm{tree}(w') \wedge \mathrm{label}(u) \wedge (w = \phi \wedge w' = t(\phi,u,\phi) \vee \exists x \exists y \exists z \exists x' \exists y' \exists z'(w = t(x,y,z) \wedge w' = t(x',y',z') \wedge y = y' \wedge (x = x' \wedge s(x,u,x') \vee z = z' \wedge s(z,u,z')))))$$

$$\forall w(\mathrm{tree}(w) \leftrightarrow w = \phi \vee \exists x \exists y \exists z(w = t(x,y,z) \wedge \mathrm{tree}(x) \wedge \mathrm{tree}(z) \wedge \mathrm{label}(y)))$$

$$\forall w(\mathrm{obtree}(w) \rightarrow \mathrm{tree}(w))$$

$$\forall x \forall y \forall z(\neg t(x,y,z) = \phi)$$

$$\forall u \forall w(u \in w \leftrightarrow \mathrm{label}(u) \wedge \mathrm{tree}(w) \wedge \exists x \exists y \exists z(w = t(x,y,z) \wedge (u = y \vee u \in x \vee u \in z)))$$

$$\forall u(\neg u \in \phi)$$

$$\forall x \forall y \forall z \forall x' \forall y' \forall z'(f(x,y,z) = f(x',y',z') \rightarrow x' = x \wedge y' = y \wedge z' = z)$$

DERIVATION

$$\vdots$$

$$\mathrm{tree}(\phi)$$

$$\vdots$$

$$\mathrm{tree}(t(\phi,7,\phi))$$

$$\vdots$$

$$t(\phi,5,t(\phi,7,\phi)) = t(\phi,5,t(\phi,7,\phi)) \wedge \mathrm{tree}(\phi) \wedge \mathrm{tree}(t(\phi,7,\phi)) \wedge \mathrm{label}(5)$$

$$t(\phi,5,t(\phi,7,\phi)) = \phi \vee \exists x \exists y \exists z(t(\phi,5,t(\phi,7,\phi)) = t(x,y,z) \wedge \mathrm{tree}(x) \wedge \mathrm{tree}(z) \wedge \mathrm{label}(y))$$

$$\mathrm{tree}(t(\phi,5,t(\phi,7,\phi))) \leftrightarrow t(\phi,5,t(\phi,7,\phi)) = \phi \vee \exists x \exists y \exists z(t(\phi,5,t(\phi,7,\phi)) = t(x,y,z) \wedge \mathrm{tree}(x) \wedge \mathrm{tree}(z) \wedge \mathrm{label}(y))$$

$$\vdots$$

$$\mathrm{tree}(t(\phi,5,t(\phi,7,\phi)))$$

$$\vdots$$

$$\mathrm{tree}(t(t(\phi,5,t(\phi,7,\phi)),9,\phi))$$

$$\vdots$$

$$\mathrm{obtree}(\phi)$$

$$\vdots$$

$$\mathrm{obtree}(t(\phi,5,\phi))$$

$$\vdots$$

$$\mathrm{obtree}(t(t(\phi,5,\phi),9,\phi))$$

$$\vdots$$

$$\mathrm{obtree}(t(\phi,7,\phi))$$

$$\vdots$$

$$\forall u(u \in \phi \rightarrow u \leq 5)$$

$$\begin{array}{|l} u \in t(\phi,7,\phi) \\ \quad \vdots \\ \quad \exists x \exists y \exists z(t(\phi,7,\phi) = t(x,y,z) \wedge (u = y \vee u \in x \vee u \in z)) \\ \hline t(\phi,7,\phi) = t(x,y,z) \wedge (u = y \vee u \in x \vee u \in z) \\ \quad \vdots \\ \quad u = 7 \vee u \in \phi \vee u \in \phi \end{array}$$

$$\begin{array}{|l} u = 7 \\ \hline \neg u > 5 \\ \quad \neg 7 > 5 \\ \quad \bot \\ \hline u > 5 \\ \hline u = 7 \rightarrow u > 5 \\ \quad \vdots \\ u \in \phi \rightarrow u > 5 \\ \hline u > 5 \\ \hline u > 5 \end{array}$$

$$u \in t(\phi,7,\phi) \rightarrow u > 5$$

$$\forall u(u \in t(\phi,7,\phi) \rightarrow u > 5)$$

$$\vdots$$

$$t(\phi,5,t(\phi,7,\phi)) = t(\phi,5,t(\phi,7,\phi)) \wedge \mathrm{obtree}(\phi) \wedge \mathrm{obtree}(t(\phi,7,\phi)) \wedge \mathrm{label}(5) \wedge \forall u(u \in \phi \rightarrow u \leq 5) \wedge \forall u(u \in t(\phi,7,\phi)) \rightarrow u > 5)$$

$$\vdots$$

$$t(\phi,5,t(\phi,7,\phi)) = \phi \vee \exists x \exists y \exists z(t(\phi,5,t(\phi,7,\phi)) = t(x,y,z) \wedge \mathrm{obtree}(x) \wedge \mathrm{obtree}(z) \wedge \mathrm{label}(y) \wedge \forall u(u \in x \rightarrow u \leq y) \wedge \forall u(u \in z) \rightarrow u > y))$$

$$\vdots$$

$$\mathrm{obtree}(t(\phi,5,t(\phi,7,\phi)))$$

$$\vdots$$

$$\mathrm{obtree}(t(t(\phi,5,t(\phi,7,\phi)),9,\phi))$$

$$\vdots$$

$$\phi = \phi \wedge t(\phi,7,\phi) = t(\phi,7,\phi)$$

$$\phi = \phi \wedge t(\phi,7,\phi) = t(\phi,7,\phi) \vee \exists x \exists y \exists z \exists x' \exists y' \exists z'(\phi = t(x,y,z) \wedge t(\phi,7,\phi) = t(x',y',z') \wedge y = y' \wedge (x = x' \wedge s(x,7,x') \vee z = z' \wedge s(z,7,z')))$$

$$\vdots$$

$$\mathrm{tree}(\phi) \wedge \mathrm{tree}(t(\phi,7,\phi)) \wedge \mathrm{label}(7) \wedge (\phi = \phi \wedge t(\phi,7,\phi) = t(\phi,7,\phi) \vee \exists x \exists y \exists z \exists x' \exists y' \exists z'(\phi = t(x,y,z) \wedge t(\phi,7,\phi) = t(x',y',z') \wedge y = y' \wedge (x = x' \wedge s(x,7,x') \vee z = z' \wedge s(z,7,z'))))$$

$$s(\phi,7,t(\phi,7,\phi)) \leftrightarrow \mathrm{tree}(\phi) \wedge \mathrm{tree}(t(\phi,7,\phi)) \wedge \mathrm{label}(7) \wedge (\phi = \phi \wedge t(\phi,7,\phi) = t(\phi,7,\phi) \vee \exists x \exists y \exists z \exists x' \exists y' \exists z'(\phi = t(x,y,z) \wedge t(\phi,7,\phi) = t(x',y',z') \wedge y = y' \wedge (x = x' \wedge s(x,7,x') \vee z = z' \wedge s(z,7,z'))))$$

$$\vdots$$

$$s(\phi,7,t(\phi,7,\phi))$$

$$t(\phi,5,\phi) = t(\phi,5,\phi) \wedge t(\phi,5,t(\phi,7,\phi)) = t(\phi,5,t(\phi,7,\phi)) \wedge 5 = 5 \wedge (\phi = \phi \wedge s(\phi,7,t(\phi,7,\phi)) \vee \phi = t(\phi,7,\phi) \wedge s(\phi,7,\phi))$$

$$\exists x \exists y \exists z \exists x' \exists y' \exists z'(t(\phi,5,\phi) = t(x,y,z) \wedge t(\phi,5,t(\phi,7,\phi)) = t(x',y',z') \wedge y = y' \wedge (x = x' \wedge s(x,7,x') \vee z = z' \wedge s(z,7,z')))$$

$$\mathrm{obtree}(t(\phi,5,\phi)) \rightarrow \mathrm{tree}(t(\phi,5,\phi))$$

$$\mathrm{tree}(t(\phi,5,\phi))$$

$$\mathrm{tree}(t(\phi,5,\phi)) \wedge \mathrm{tree}(t(\phi,5,t(\phi,7,\phi))) \wedge \mathrm{label}(7) \wedge (t(\phi,5,\phi) = \phi \wedge t(\phi,5,t(\phi,7,\phi)) = t(\phi,7,\phi) \vee \exists x \exists y \exists z \exists x' \exists y' \exists z'(t(\phi,5,\phi) = t(x,y,z) \wedge t(\phi,5,t(\phi,7,\phi)) = t(x',y',z') \wedge y = y' \wedge (x = x' \wedge s(x,7,x') \vee z = z' \wedge s(z,7,z'))))$$

$$\vdots$$

$$s(t(\phi,5,\phi),7,t(\phi,5,t(\phi,7,\phi)))$$

$$\vdots$$

$$s(t(t(\phi,5,\phi),9,\phi),7,t(t(\phi,5,t(\phi,7,\phi)),9,\phi))$$

$$\mathrm{obtree}(t(t(\phi,5,\phi),9,\phi)) \wedge \mathrm{obtree}(t(t(\phi,5,t(\phi,7,\phi)),9,\phi)) \wedge \mathrm{label}(7) \wedge s(t(t(\phi,5,\phi),9,\phi),7,t(t(\phi,5,t(\phi,7,\phi)),9,\phi))$$

$$\mathrm{insert}(t(t(\phi,5,\phi),9,\phi),7,t(t(\phi,5,t(\phi,7,\phi)),9,\phi)) \leftrightarrow \mathrm{obtree}(t(t(\phi,5,\phi),9,\phi)) \wedge \mathrm{obtree}(t(t(\phi,5,t(\phi,7,\phi)),9,\phi)) \wedge \mathrm{label}(7) \wedge s(t(t(\phi,5,\phi),9,\phi),7,t(t(\phi,5,t(\phi,7,\phi)),9,\phi))$$

$$\vdots$$

$$\mathrm{insert}(t(t(\phi,5,\phi),9,\phi),7,t(t(\phi,5,t(\phi,7,\phi)),9,\phi))$$