Available online at www.sciencedirect.com
SCIENCE DIRECT
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 132 (2005) 73–93
www.elsevier.com/locate/entcs

A Proof Calculus for Natural Semantics Based on Greatest Fixed Point Semantics

Sabine Glesner

Institute for Program Structures and Data Organization
University of Karlsruhe, 76128 Karlsruhe, Germany
http://www.info.uni-karlsruhe.de/~glesner

Abstract

Formal semantics of programming languages needs to model the potentially infinite state transition behavior of programs as well as the computation of their final results simultaneously. This requirement is essential in correctness proofs for compilers. We show that a greatest fixed point interpretation of natural semantics is able to model both aspects equally well. Technically, we infer this interpretation of natural semantics based on an easily omprehensible introduction to the dual definition and proof principles of induction and coinduction. Furthermore, we develop a proof calculus based on it and demonstrate its application for two typical problems.

Keywords: Formal semantics, formal compiler correctness, natural semantics, coinductive/greatest fixed point interpretation, proof calculus.

1 The Need for Greatest Fixed Point Semantics

Programming language semantics incorporates two dual aspects: The execution of a program triggers a potentially infinite state transition sequence. If this transition sequence terminates, then it defines the final result of program execution. A formalism for the semantics of programming languages should model both aspects simultaneously. If the execution of a program terminates, then its final result should be defined based on the finite state transition sequence. Moreover, a semantics formalism should specify a more meaningful semantics than just "undefined" for non-terminating programs. This requirement is essential in practical applications. Many programs (e.g. operating systems, data bases, control software in embedded systems or reactive systems) are not intended to terminate while still having a very special semantics.

1571-0661/$ – see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.entcs.2005.02.011

We show that a greatest fixed point interpretation of natural semantics is able to model both aspects simultaneously. This greatest fixed point interpretation gives rise to a proof calculus consisting of inductive and coinductive proof rules. It can be used in the formal reasoning about programming languages. As examples, we consider two applications. The first concerns the correctness proofs of translations, e.g. in compilers. Thereby one needs to prove that the observable behavior of the translated programs is preserved. This is a stronger requirement than just preserving their final results. The second example regards proofs for properties of programming languages, e.g. type safety. They need to consider terminating and non-terminating programs.

Our proof calculus is based on the well-established trend that a combination of algebraic and coalgebraic methods can be used successfully in the specification of and reasoning about programming languages, especially for potentially non-terminating processes. We restate the corresponding proof principles of induction and coinduction in a simple form which is yet powerful enough to model deterministic, possibly infinite computations. We describe the two dual definition and proof principles, in contrast to the common literature utilizing category theory, in a purely set-theoretic and easily comprehensible manner. We show that the state transition behavior of programs must be defined coinductively and that the final result is defined inductively on top of it. While automated theorem provers, e.g. Isabelle [13], have the potential to reason coinductively, the standard practice does not use it. All automated as well as "paper and pencil" proofs based on natural semantics exploit induction and, hence, do only hold for terminating computations. The results of this paper demonstrate that this is not sufficient and can easily be replaced by coinductive reasoning.

The Insufficiency of Induction Proofs

Let us start with a motivation why induction is not the appropriate proof principle for infinite computations.

```
{P}
proc p
    ...
        {P}
        p
        {Q}
    ...
endproc
{Q}
```

Consider one of the well-known proof rules of the Hoare calculus [6]. If one wants to prove that a recursive procedure p is correct wrt. a precondition P and a postcondition Q, then one assumes that for all recursive calls of p within the body of p, precondition P and postcondition Q hold. If p always terminates, then this is an induction proof. The recursion depth of the inner calls is always smaller than the recursion depth of p itself. If the procedure p does not terminate, it is no longer a valid induction proof. The state transition sequence in the inner procedure's body is infinitely long as well as the state transition sequence of the outer procedure. Hence, we do not have an induction premise about a strictly smaller