Logical Specification of Operational Semantics *

Peter D. Mosses
BRICS and Department of Computer Science, University of Aarhus
Ny Munkegade, bldg. 540, DK-8000 Aarhus C, Denmark
Home page: http://www.brics.dk/~pdm/

Abstract. Various logic-based frameworks have been proposed for specifying the operational semantics of programming languages and concurrent systems, including inference systems in the styles advocated by Plotkin and by Kahn, Horn logic, equational specifications, reduction systems for evaluation contexts, rewriting logic, and tile logic.

We consider the relationship between these frameworks, and assess their respective merits and drawbacks—especially with regard to the modularity of specifications, which is a crucial feature for scaling up to practical applications. We also report on recent work towards the use of the Maude system (which provides an efficient implementation of rewriting logic) as a meta-tool for operational semantics.

1 Introduction

The designers, implementors, and users of a programming language all need to acquire an intrinsically operational understanding of its semantics. Programming language reference manuals attempt to provide such an understanding using informal, natural language; but they are prone to ambiguity, inconsistency, and incompleteness, and totally unsuitable as a basis for sound reasoning about the effects of executing programs—especially when concurrency is involved.

Various mathematical frameworks have been proposed for giving formal descriptions of programming language semantics. Denotational semantics generally tries to avoid direct reference to operational notions, and its abstract domain-theoretic basis remains somewhat inaccessible to most programmers (although modelling programs as higher-order functions has certainly given useful insight to language designers and to theoreticians). Operational semantics, which directly aims to model the program execution process, is generally based on familiar first-order notions; it has become quite popular, and has been preferred to denotational semantics for defining programming languages [28] and process algebras [26].

Despite the relative popularity of operational semantics, there have been some “semantic engineering” problems with scaling up to descriptions of full practical programming languages. A significant feature that facilitates scaling up is good modularity; the formulation of the description of one construct should

* Written while visiting SRI International and Stanford University.
not depend on the presence (or absence) of other constructs in the language. Recently, the author has proposed a solution to the modularity problem for the structural approach to operational semantics [31,32].

There are different ways of specifying operational semantics for a programming language: an interpreter for programs—written in some (other) programming language, or defined mathematically as an abstract machine—is an algorithmic specification, determining how to execute programs; a logic for inferring judgements about program executions is a declarative specification, determining what program executions are allowed, but leaving how to find them to logical inference. Following Plotkin’s seminal work [37], much interest has focussed on logical specification of operational semantics.

In fact various kinds of logic have been found useful for specifying operational semantics: arbitrary inference systems, natural deduction systems, Horn logic, equational logic, rewriting logic, and tile logic, among others. Sections 2 and 3 review and consider the relationship between these applied logics, pointing out some of their merits and drawbacks—especially with regard to the modularity of specifications. The brief descriptions of the various logics are supplemented by illustrative examples of their use. It is hoped that the survey thus provided will be useful as an introduction to the main techniques available for logical specification of operational semantics.

The inference of a program execution in some logic is clearly not the same thing as the inferred execution itself. Nevertheless, a system implementing logical inference may be used to execute programs according to their operational semantics. Section 4 reports on recent work towards the use of the Maude system (which provides an efficient implementation of rewriting logic) as a meta-tool for operational semantics.

2 Varieties of Structural Operational Semantics

The structural style of operational semantics (SOS) is to specify inference rules for steps (or transitions) that may be made not only by whole programs but also by their constituent phrases: expressions, statements, declarations, etc. The steps allowed for a compound phrase are generally determined by the steps allowed for its component phrases, i.e., the steps are defined inductively according the (abstract) syntax of the described programming language. An atomic assertion of the specified logic (such as $\gamma \rightarrow \gamma'$) asserts the possibility of a step from one configuration $\gamma$ to another $\gamma'$. Some configurations are usually distinguished as terminal, and have no further steps, whereas initial and intermediate configurations have phrases that remain to be executed as components.

Small-step SOS: In so-called small-step SOS [37], a single step for an atomic phrase often gives rise to a single step for its enclosing phrase (and thus ultimately for the whole program). A complete program execution is modelled as a succession—possibly infinite—of these small steps. During such a program execution, phrases whose execution has terminated get replaced by the values that they have computed.