Deriving Mixed Evaluation from Standard Evaluation for a Simple Functional Language

JOHN HANNAN and DALE MILLER

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389 USA

We demonstrate how a specification for the standard evaluation of a simple functional programming language can be systematically extended to a specification for mixed evaluation. Using techniques inspired by natural semantics we specify a standard evaluator by a set of inference rules. The evaluation of programs is then performed by a restricted kind of theorem proving in this logic. We then describe a systematic method for extending the proof system for standard evaluation to a new proof system that provides greater flexibility in treating bound variables in the object-level functional programs. We demonstrate how this extended proof system provides the capabilities of a mixed evaluator and how correctness with respect to standard evaluation can be proved in a simple and direct manner. The current work focuses only on a primitive notion of mixed evaluation for a simple functional programming language, but we believe that our methods will extend to more sophisticated kinds of evaluations and richer languages.

1 Introduction

The formal derivation and correctness of program analysis tools play a central role in many programming language research efforts. In this paper we focus on evaluators for programming languages. We shall use natural deduction techniques to specify and derive evaluators for a simple functional language. With a natural deduction theorem prover, one constructs formal proofs of propositions using a particular set of inference rules. If we encode programs as terms then we can build propositions expressing relationships between programs. A proof system can then axiomatize such relationships. This approach to program analysis shares much with the work on structural operational semantics [19] and natural semantics [12].

Since what we shall call “mixed evaluation” is related to the terms “partial evalua-
tion" and "mixed computation," we provide a brief description of our use of these terms. *Standard evaluation* refers to a conventional notion of evaluation or interpretation of functional programs. *Partial evaluation* is a systematic method of constructing an efficient program based on a given program and a part of its input [3]. In general terms, it can be described as follows. Let \( f \) be some functional program of two arguments \( x \) and \( y \) and consider the application \( f(c, y) \) for some constant (known) value \( c \) and variable (unknown) value \( y \). We wish to construct a new functional program \( f_c \) such that \( f_c(y) = f(c, y) \) for all values of \( y \), such that for any value of \( y \), computing \( f_c(y) \) should be easier (or faster) than computing \( f(c, y) \). Such improvement is possible by "compiling" the information that \( x = c \) in \( f \) into the definition of \( f_c \).

*Mixed evaluation*, also called *symbolic evaluation* in [5], is the process of evaluating expressions, which may contain free variables (i.e., not bound to any values), to some canonical form. This process must deal with the proper treatment of the interaction of known and unknown (symbolic) values and hence the adjective "mixed." The key task of mixed evaluation is making maximal use of the partial information or performing as much computation in advance as possible. Unfortunately, there are languages for which no evaluation strategy can perform such maximal compiling [10]. We would, however, like to produce mixed evaluators that are capable of a reasonable level of performance.

The importance of mixed evaluation was elucidated by Futamura [3] when he described the construction of compiled programs, compilers, and compiler generators via mixed evaluation. Thus mixed evaluation is a means for understanding and constructing a wide range of translation tools. But where do mixed evaluators come from? In particular, can mixed evaluators be formally derived from standard evaluators? Few research efforts have addressed the formal construction of mixed evaluators using principled techniques. We address this question by demonstrating how, for a simple functional programming language, a specification for mixed evaluation can be derived from a specification for standard evaluation.

The remainder of this paper is organized as follows. In Section 2 we introduce a simple functional programming language, giving both a concrete and an abstract syntax. Following this we specify a standard evaluator for this language in Section 3. In Section 4 we use the signature of the functional program's abstract syntax to construct a mixed evaluator and in Section 5 we prove a form of its correctness. Issues of implementation are discussed in Section 6. Summary comments and a description of related work are provided in Section 7.

## 2 Abstract Syntax as Lambda Terms

We introduce a simple functional programming language for which we shall specify two evaluators in subsequent sections. There is strong connection between the choice of *abstract syntax* for the formal representation of functional programs and the complexity of the presentation of these evaluators. An appropriate choice of abstract syntax will