Relational Interpretations of Recursive Types in an Operational Setting  
(Summary)

Lars Birkedal and Robert Harper

birkedal@cs.cmu.edu and rwh@cs.cmu.edu
School of Computer Science, Carnegie Mellon University

Abstract. Relational interpretations of type systems are a useful tool for establishing properties of programming languages. For languages with recursive types the existence of a relational interpretation is often difficult to establish. The most well-known approach is to pass to a domain-theoretic model of the language, using the structure of the domain to define a suitable system of relations. Here we study the construction of relational interpretations for an ML-like language with recursive functions and recursive types in a purely operational setting. The construction is an adaptation of results of Pitts on relational properties of domains to an operational setting, making use of techniques introduced by Mason, Smith, and Talcott for proving operational equivalence of expressions. To illustrate the method we give a relational proof of correctness of the continuation-passing transformation used in some compilers for functional languages.

1 Introduction

The interpretation of types as relations is a fundamental technique in the study of type systems (see, for example, Mitchell’s survey [18] and monograph [19] for examples and references to the literature). The general idea is to associate to each type a relation over a suitable value space in such a way that well-typed terms are related appropriately by the interpretation. The construction of relational interpretations of type systems often raises interesting technical problems. For example, Girard’s proof of strong normalization for the second-order λ-calculus [10] may be understood as a relational interpretation for a type system with impredicative type quantification.

In this paper we are concerned with the construction of relational interpretations for an ML-like language $L$ with recursive functions and one recursive type. The operational semantics of the language specifies an “eager” or “call-by-value” evaluation strategy, as in Standard ML [17]. We make no restrictions on the occurrence of the recursively-defined type in its definition — both positive and negative occurrences are permitted. This complicates the construction of a relational interpretation of the language.

The usual approach is to pass to a specific model (for example, a domain-theoretic model such as Scott’s $D_\infty$) and to exploit the structure of the model to
construct the required system of relations. One disadvantage of this approach is that one must then give a denotational semantics for the language in the model under consideration, and this must be proved adequate with respect to the operational semantics. Another disadvantage is that the interpretation is defined for a specific model, and it is not clear to what extent the result applies to other models of the language. The question of generality was recently addressed by Pitts [22], who exploited Freyd’s analysis of solutions of recursive domain equations [8,7,9] to construct relational interpretations over domain models of recursive types. In particular, Pitts showed that the existence of a relational interpretation can, under very general conditions, be reduced to the minimal invariant property of solutions to recursive domain equations given by Freyd. Very roughly, the minimal invariant property characterizes the “minimal” solution to a recursive domain equation by a universal property.

The starting point for our work is the observation that the minimal invariant property for a model of $\mathcal{L}$ can be stated entirely in terms of the constructs of the language itself. This opens the way to carrying out the construction of a relational interpretation of the type system in a purely operational setting — that is, without consideration of a domain-theoretic denotational semantics of the language. The key is to establish a “syntactic minimal invariant” property for terms of the language taken modulo a suitable notion of operational equivalence. With this in hand we may adapt Pitts’s results to construct relational interpretations of types over operational equivalence classes of closed terms. The choice of operational equivalence is guided by the requirements of the proof. Candidates for operational equivalence include contextual equivalence [20,23] (coincidence of evaluation in all program contexts), bisimilarity [13,21] (existence of a correspondence between evaluation steps), and experimental equivalence [15] (coincidence of closed instances in all evaluation contexts). It turns out that all three notions coincide for the language $\mathcal{L}$, so our decision to work with experimental equivalence is entirely pragmatic — it supported a relatively straightforward proof of the critical minimal invariant property for $\mathcal{L}$.

Relational interpretations of types have a number of applications. Pitts [22] uses relations to characterize the approximation relation in minimal domain models of FPC and to give a proof of adequacy of the denotational semantics for FPC in a minimal domain model relative to an operational semantics for it. Here we focus on the application to the correctness of program transformations used in compilers of functional languages. In particular, we consider the correctness of the translation into continuation-passing style [6,23], called the $\text{cps}$ transform. The proof relies on the construction of a relational interpretation of $\mathcal{L}$ that establishes a correspondence between the evaluation of a program and its continuation-passing transform. The result generalizes Reynolds’s proof [25] of the relation between direct and continuation semantics for an untyped language to the case of a typed language with a recursive type (which could be taken to be the recursive type corresponding to the untyped $\lambda$-calculus). In constrast to Reynolds’ proof we do not rely on a specific domain-theoretic interpretation of $\mathcal{L}$, but instead work directly over the operational semantics.