Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 338 (2005) 315–349

Theoretical Computer Science

www.elsevier.com/locate/tcs

Functions with local state: Regularity and undecidability

Andrzej S. Murawski

Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, UK

Received 2 March 2004; accepted 9 December 2004

Communicated by D. Sannella

Abstract

We study programs of a finitary ML-like language RML_f with ground-type references. RML_f permits the use of functions with locally declared variables that remain private and persist from one use of the function to the next. Using game semantics we show that this leads to undecidability of program equivalence already at second order. We also examine the extent to which this feature can be captured by regular languages. This gives a decidability result for a second-order fragment $\mathsf{RML}_\mathrm{f}^-$ of RML_f, which comprises many examples studied in the literature.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Game semantics; Automata; Software verification; Model checking

1. Introduction

Game semantics has contributed fully abstract models for a variety of programming languages. Each of these models gives a semantic account of program equivalence: the interpretations of programs coincide if and only if the programs are equivalent in the respective languages. This makes it possible, at least in principle, to reason about program equivalence with the help of game models. However, their structure does not facilitate such reasoning, since the game categories are obtained via non-trivial quotienting. Fortunately, for languages with ground-type references, the quotient can be characterized explicitly via sets of special plays, making the model more accessible and usable.

E-mail address: andrzej.murawski@comlab.ox.ac.uk.

Plays in game semantics are sequences of moves equipped with pointers. In some cases, however, the pointer structure can be shown to be uniquely reconstructible and consequently can be ignored. Then sets of plays can be regarded simply as languages over the alphabet of moves and program equivalence can be analyzed as language equivalence. If the associated language equivalence is decidable, so must be program equivalence. In order to obtain decidability, finitary language fragments are considered: with finite datatypes and with iteration instead of unrestricted recursion.

The first result establishing that pattern of reasoning has been obtained by Ghica and McCusker [10,11] and concerned second-order Idealized Algol and regular expressions. This discovery initiated research into algorithmic aspects of game semantics and its potential to become a foundation for software model-checking [2]. The initial decidability result has also been extended in various directions: to third-order Idealized Algol [22,21], concurrency [12] and a call-by-value language with arrays [9]. The last of these papers, by Ghica, investigated a language with first-order procedures and block-allocated variables, such as those used in imperative programs.

In this paper we consider the call-by-value case as well but we shall focus on dynamically allocated (integer-valued) variables used in languages such as Standard ML. Access to such variables can be passed outside their original allocation block, which opens up new ways of manipulating the program state. One can also define functions which have “private” local variables that persist over invocations and accumulate information throughout their lifetime, like in the simple example given below:

$$(\lambda l^{\mathsf{int\,ref}}.\,(\lambda x^{\mathsf{int}}.\mathsf{if}\ !l < \mathtt{max}\ \mathsf{then}\ (l := !l+1; x)\ \mathsf{else}\ \Omega))(\mathsf{ref}(0)) : \mathsf{int} \to \mathsf{int}$$

where the local variable restricts the number of function calls and causes divergence after $\mathtt{max}$ uses (see also Examples 3, 21, 27 and 28). This encapsulates the state within the function much like in object-oriented programming. Indeed, that mechanism can be employed to define objects and implement basic object-oriented features [25]. The combination of imperative and functional features present in ML turns out quite difficult to reason about [23]. In fact we are going to show that already at second order finitary program equivalence is undecidable. On the other hand, we will identify a language fragment with second-order procedures which can be captured via regular languages and which still contains many examples considered in the literature [23,26]. The language will include some terms whose game semantics is not strictly regular. Then, instead of the full semantics, we will use a suitable regular representative.

Our language of study is finitary RML for which a fully abstract game model was given by Abramsky and McCusker [5]. RML bears close resemblance to Reduced ML as studied by Pitts and Stark [23,26] with one important distinction. RML is equipped with a variable constructor mkvar, which can be used to design user-defined variable objects that do not have to behave like standard memory cells. In general this feature makes RML contexts more discriminating as far as program equivalence is concerned, but this happens only when the types of the terms involved have negative[1] occurrences of $\mathsf{int\,ref}$. In other cases RML can simply be considered a conservative extension of Reduced ML and then our results are immediately applicable to Reduced ML, including the undecidability result which does not

[1] i.e. in the left-hand scope of a odd number of arrows.