Algorithmic Games for Full Ground References

Andrzej S. Murawski\textsuperscript{1} and Nikos Tzevelekos\textsuperscript{2,*}

\textsuperscript{1} University of Leicester
\textsuperscript{2} Queen Mary, University of London

Abstract. We present a full classification of decidable and undecidable cases for contextual equivalence in a finitary ML-like language equipped with full ground storage (both integers and reference names can be stored). The simplest undecidable type is unit $\rightarrow$ unit $\rightarrow$ unit. At the technical level, our results marry game semantics with automata-theoretic techniques developed to handle infinite alphabets. On the automata-theoretic front, we show decidability of the emptiness problem for register pushdown automata extended with fresh-symbol generation.

1 Introduction

Mutable variables in which numerical values can be stored for future access and update are the pillar of imperative programming. The memory in which the values are deposited can be allocated statically, typically to coincide with the lifetime of the defining block, or dynamically, on demand, with the potential to persist forever. In order to support memory management, modern programming languages feature mechanisms such as pointers or references, which allow programmers to access memory via addresses. Languages like C (through int*) or ML (via int ref ref) make it possible to store the addresses themselves, which creates the need for storing references to references etc. We refer to this scenario as full ground storage. In this paper we study an ML-like language GRef with full ground storage, which permits the creation of references to integers as well as references to integer references, and so on.

We concentrate on contextual equivalence\textsuperscript{1} in that setting. Reasoning about program equivalence has been a central topic in programming language semantics since its inception. This is in no small part due to important applications, such as verification problems (equivalence between a given implementation and a model implementation) and compiler optimization (equivalence between the original program and its transform). Specifically, we attack the problem of automated reasoning about our language in a finitary setting, with finite datatypes and with looping instead of recursion, where decidability questions become interesting and the decidability/undecidability frontier can be identified. In particular, it is possible to quantify the impact of higher-order types on decidability, which goes unnoticed in Turing-complete frameworks.

The paper presents a complete classification of cases in which GRef program equivalence is decidable. The result is phrased in terms of the syntactic shape of types.

\textsuperscript{*} Supported by a Royal Academy of Engineering Research Fellowship.

\textsuperscript{1} Two program phrases are regarded as contextually equivalent, or simply equivalent, if they can be used interchangeably in any context without affecting the observable outcome.
We write $\theta_1, \ldots, \theta_k \vdash \theta$ to refer to the problem of deciding contextual equivalence between two terms $M_1, M_2$ such that $x_1 : \theta_1, \ldots, x_m : \theta_m \vdash M_i : \theta$ ($i = 1, 2$). We investigate the problem using a fully abstract game model of $\text{GRef}$ \footnote{A model is \textit{fully abstract} if it captures contextual equivalence denotationally, i.e. equivalence can be confirmed/disproved by reference to the interpretations of terms.}. Such a model can be easily obtained by modifying existing models of more general languages, e.g. by either adding type information to Laird’s model of untyped references \footnote{14} or trimming down our own model for general references \footnote{18}. The models are nomina\footnote{1,14} l in that moves may involve elements from an infinite set of names to account for reference names. Additionally, each move is equipped with a store whose domain consists of all names that have been revealed (played) thus far and the corresponding values. Note that values of reference types also become part of the domain of the store. This representation grows as the play unfolds and new names are encountered. We shall rely on the model both for decidability and undecidability results. Our work identifies the following undecidable cases as minimal.

\[ \vdash \text{unit} \rightarrow \text{unit} \rightarrow \text{unit} \quad \text{(unit} \rightarrow \text{unit} \rightarrow \text{unit}) \rightarrow \text{unit} \vdash \text{unit} \]
\[ \vdash ((\text{unit} \rightarrow \text{unit}) \rightarrow \text{unit}) \rightarrow \text{unit} \quad (((\text{unit} \rightarrow \text{unit}) \rightarrow \text{unit}) \rightarrow \text{unit}) \rightarrow \text{unit} \vdash \text{unit} \]

Obviously, undecidability extends to typing judgments featuring syntactic supertypes of those listed above (for instance, when fourth-order types appear on the left-hand side of the turnstile or types of the shape $\theta_1 \rightarrow \theta_2 \rightarrow \theta_3$ occur on the right). The remaining cases are summarized by typing judgements in which each of $\theta_1, \ldots, \theta_m$ is generated by the grammar given on the left below, and $\theta$ by the grammar on the right,

\[
\Theta_L ::= \beta | \Theta_R \rightarrow \Theta_L \\
\Theta_R ::= \beta | \Theta_1 \rightarrow \beta
\]

where $\beta$ stands any ground type and $\Theta_1$ is a first-order type, i.e. $\beta ::= \text{unit} | \text{int} | \text{ref}^{\dagger} \text{int}$ and $\Theta_1 ::= \beta | \beta \rightarrow \Theta_1$. We shall show that all these cases are in fact decidable. In order to arrive at a decision procedure we rely on effective reducibility to a canonical ($\beta$-normal) form. These forms are then inductively translated into a class of automata over infinite alphabets that represent the associated game semantics. Finally, we show that the representations can be effectively compared for equivalence.

The automata we use are especially designed to read moves-with-stores in a single computational step. They are equipped with a finite set of registers for storing elements from the infinite alphabet (names). Moreover, in a single transition step, the content of a subset of registers can be pushed onto the stack (along with a symbol from the stack alphabet), to be popped back at a later stage. We use visibly pushdown stacks \footnote{4}, i.e. the alphabet can be partitioned into letters that consistently trigger the same stack actions (push, pop or no-op). Conceptually, the automata extend register pushdown automata \footnote{6} with the ability to generate fresh names, as opposed to their existing capability to generate names not currently present in registers. Crucially, we can show that the emptiness problem for the extended machine model remains decidable.

Because the stores used in game-semantic plays can grow unboundedly, one cannot hope to construct the automata in such a way that they will accept the full game semantics of terms. Instead we construct automata that, without loss of generality, will accept plays in which the domains of stores are bounded in size. Each such restricted