Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 304 (2014) 97–110

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Abstract Semantics for Alias Analysis in $\mathbb{K}$

Irina Măriuca Asăvoae[3]

Computer Science Faculty
University Alexandru Ioan Cuza
Iaşi, Romania

Abstract

This paper presents an approach to integrating analysis and verification methods in the $\mathbb{K}$ framework. We adopt the abstract interpretation perspective where the concrete system to be analyzed/verified is mapped into a suitable abstract system, and collecting semantics is applied over the abstract system to obtain the analysis/verification method itself. As such, we present the $\mathbb{K}$ perspective of collecting semantics over $\mathbb{K}$ operational semantics for abstract systems. For a good degree of generality we consider that abstract systems are $\mathbb{K}$ specifications of (finite) pushdown systems. We give the collecting semantics as a generic set of $\mathbb{K}$ rules parametrized by the $\mathbb{K}$ specification of a finite pushdown system. Further, we describe a case study which instances collecting semantics with alias analysis. For this, the abstract system is defined as an imperative language which maintains enough pointer and flow information for alias analysis to be decidable. The $\mathbb{K}$ specification of this imperative language fits the frame of a finite pushdown system specification.

Keywords: abstraction, collecting semantics, pushdown systems, alias analysis

1 Introduction

The spark of the $\mathbb{K}$ framework [14] is the observation that computation is expressed naturally with rewriting. The source of inspiration for $\mathbb{K}$ is the Rewriting Logic Semantics project [9,19,10] which has the declared purpose of unifying algebraic denotational semantics and operational semantics. This unification is achieved by considering the two semantics as different views over the same object. Namely, denotational semantics views the rewriting logic specification of a language as a designated model, while operational semantics focuses on the execution of the same specification.

$\mathbb{K}$ is built upon a continuation-based technique and a series of notational conventions to allow for more compact and modular programming language definitions. $\mathbb{K}$ definitions can be mechanically translated into rewriting logic, and in particular

[1] Contract ANCS POS CCE, O2.1.2, ID nr. 602/12516, ctr.nr 161/15.06.2010, DAK.

[2] European Social Fund in Romania, under the responsibility of the Managing Authority for the Sectoral Operational Programme for Human Resources Development 2007-2013 [grant POSDRU/88/1.5/S/47646]

[3] Email: `mariuca.asavoae@info.uaic.ro`

http://dx.doi.org/10.1016/j.entcs.2014.05.005
1571-0661/© 2014 Published by Elsevier B.V.

into Maude, to obtain program analysis tools or interpreters based on term rewriting. This capability makes $\mathbb{K}$ an executable framework, with $\mathbb{K}$-Maude its prototype implementation [15,18].

A $\mathbb{K}$ definition is an executable specification of a transition system whose computations are obtained by the execution of the $\mathbb{K}$ definition. Moreover, one can also reuse a $\mathbb{K}$ definition to enable richer executions as, for example, sets of computations. When producing the plain computations, $\mathbb{K}$ can be seen as an interpreter while, when producing sets of computations, $\mathbb{K}$ can also be used as an analyzer/verifier for the specified transition system. This is the idea of the current paper in a nutshell and we frame it under the methodology proposed by abstract interpretation.

In abstract interpretation, a particular analysis/verification method is achieved by defining collecting semantics over the examined transition system [4,5]. Namely, the transition system is first transformed into a simpler "abstract" one such that the operational semantics of the two systems "agree" on the analyzed/verified set of properties. Then, collecting semantics relies on the operational semantics of the abstract transition system and collects its computations via a forward or backward fixpoint iteration. Hence, the analysis/verification methods are a semantic reflection of the operational semantics.

1.1 *Contributions summary*

In this paper we present an infrastructure for expressing in $\mathbb{K}$ the reflection of operational semantics into collecting semantics, and the alias analysis instantiation of this reflection. The cornerstone of this infrastructure is the choice of pushdown systems as suitable $\mathbb{K}$ definitions. The semantics reflection is nicely captured in $\mathbb{K}$ by the configuration abstraction mechanism and definitional modularity. The choice of pushdown systems as focal point for this study is justified by the generality of the notion, the already available theoretical results, and the close resemblance with $\mathbb{K}$ definitions. By the latter similitude we mean that the continuation-based technique used in $\mathbb{K}$ gives the stack aspect to the k-cell, while $\mathbb{K}$ rules usually rely on a pushed down stack mechanism.

In Section 2 we present an infrastructure for the $\mathbb{K}$ specification of analysis/verification methods for pushdown systems. In more detail, in Section 2.1 we present a discussion on the $\mathbb{K}$ representation of pushdown systems. We use the $\mathbb{K}$ specification of a pushdown system as support for deriving the analysis/verification infrastructure in Section 2.2. We also argue the opportunity to consider pushdown systems and their $\mathbb{K}$ specification in Section 2.3.

In Section 3 we present a case study of an abstract imperative programming language with procedures and objects, *SILK*, via its $\mathbb{K}$ specification. *SILK* is of interest in the context of the $\mathbb{K}$ framework for the following reasons:

- This is a research language introduced in [17,16] with several bisimilar semantics. In Section 3.1 we present the $\mathbb{K}$ specification of one of these semantics. In particular, this semantics exhibits algorithmic details which emphasize the versatility of $\mathbb{K}$ in the area of algorithm formulation.