Science of Computer Programming 95 (2014) 176–199

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

On the syntactic and functional correspondence between hybrid (or layered) normalisers and abstract machines ☆

A. García-Pérez [a,b,*], P. Nogueira [b,*]

[a] IMDEA Software Institute, Campus de Montegancedo, 28223 Pozuelo de Alarcón, Madrid, Spain
[b] Babel Research Group, Universidad Politécnica de Madrid, Campus de Montegancedo, 28660 Boadilla del Monte, Madrid, Spain

ARTICLE INFO

Article history:
Received 20 May 2013
Received in revised form 16 May 2014
Accepted 19 May 2014
Available online 29 May 2014

Keywords:
Operational semantics
Program transformation
Reduction strategies
Abstract machines
Full reduction

ABSTRACT

We show how to connect the syntactic and the functional correspondence for normalisers and abstract machines implementing hybrid (or layered) reduction strategies, that is, strategies that depend on subsidiary sub-strategies. Many fundamental strategies in the literature are hybrid, in particular, many full-reducing strategies, and many full-reducing and complete strategies that deliver a fully reduced result when it exists. If we follow the standard program-transformation steps the abstract machines obtained for hybrids after the syntactic correspondence cannot be refunctionalised, and the junction with the functional correspondence is severed. However, a solution is possible based on establishing the shape invariant of well-formed continuation stacks. We illustrate the problem and the solution with the derivation of substitution-based normalisers for normal order, a hybrid, full-reducing, and complete strategy of the pure lambda calculus. The machine we obtain is a substitution-based, eval/apply, open-terms version of Pierre Crégut's full-reducing Krivine machine KN.

1. Introduction

Background An operational semantics is founded on a *reduction strategy* that specifies the order in which reducible terms (or reducible expressions, 'redices' for short, singular 'redex') must be reduced. An operational semantics can be *small-step* (concerned with single reduction steps) or *big-step* (concerned with final results). In both cases reduction may terminate delivering an *irreducible* term, or may diverge entering an infinite loop. Traditional approaches to small-step operational semantics are structural [1], context-based (or reduction semantics) [2], and abstract machines [3]. The latter are state-transition functions that, unlike virtual machines, operate directly on terms, have no instruction set, and no need for a compiler. Two approaches to big-step operational semantics are natural semantics [4] and big-step abstract machines. The latter are first-order tail-recursive presentations of state-transition functions.

All these different semantic styles can be implemented as programs that can be *inter-derived by means of program transformation*. We call such programs 'semantic artefacts', a terminology perhaps coined for mathematical descriptions of semantics but often used by extension for their implementations [5]. 'Inter-derivation' of semantic artefacts is used in the literature in

☆ Research partially funded by the Spanish 'Ministerio de Economía y Competitividad' through projects DESAFIOS10 TIN2009-14599-C03-00 and STRONGSOFT TIN2012-39391-C04-02, and by 'Comunidad de Madrid' through programme PROMETIDOS P2009/TIC-1465. The first author has been supported by Comunidad de Madrid grant CPI/0622/2008.

* Corresponding authors.
E-mail addresses: agarcia@babel.ls.fi.upm.es (A. García-Pérez), pablo.nogueira@upm.es (P. Nogueira).

http://dx.doi.org/10.1016/j.scico.2014.05.011
0167-6423/

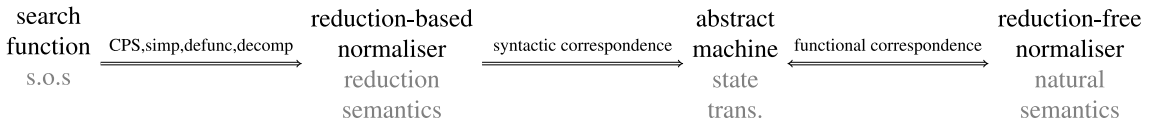

Fig. 1. Semantic artefacts and derivation paths.

the specific sense that the artefacts derive by program transformation to the same (implementation of an) abstract machine. The literature on inter-derivation we assume throughout the paper is [5–11], the last one an excellent tutorial introduction on which we have based part of our presentation. Although we introduce all the terminology and transformation steps we use, we expect the reader to be familiar with that literature.

Fig. 1 illustrates the derivation paths among semantic artefacts. From left to right, a *search function* is a simple artefact that mirrors the compatibility rules of a structural operational semantics (the rules that express how to navigate a term to locate a redex). A search function delivers for an input term the redex subterm to be contracted or the input term back if the input term is irreducible. A search function derives to a *reduction-based normaliser* by applying the following program-transformation steps: CPS-transformation, simplification, defunctionalisation, and turning the search into a decomposition function which, additionally to the next redex, delivers the context where the redex appears. A reduction-based normaliser is a program that implements a reduction semantics by iterating (i) the unique decomposition of a term into a context and a redex within the context hole, (ii) the contraction of the redex and, (iii) the recomposition of the resulting term. The additional recomposition function consists of a left fold over the contexts. A reduction-based normaliser derives to (a big-step implementation of) an *abstract machine* by applying the following steps: refocusing (which optimises the iteration loop), lightweight fusion by fixed-point promotion, and inlining-of-iterate-function steps. This latter derivation is called a *syntactic correspondence* and its steps are in general not reversible. The abstract machine derives to a *reduction-free normaliser* by applying refunctionalisation and direct-style transformation. This derivation is reversible by CPS-transformation and defunctionalisation and is called a *functional correspondence*. A reduction-free normaliser is a program implementing a natural semantics, typically a recursive evaluator for deeply-embedded terms.

Reduction-based and reduction-free normalisers (and intermediate abstract machines) are equivalent because the transformation steps are equivalence-preserving. Consequently, the artefacts implement the same reduction strategy. A search function is a simpler artefact which, although not strictly equivalent, is sufficient to characterise the structural operational semantics [11]. It connects the structural and context-based semantics, recomposition is straightforward to add and, more importantly for us, the simplification and defunctionalisation of the search function reveals the continuation stack, which is not the case if the starting point is a whole implementation of the structural operational semantics that searches the input term, contracts the redex, and delivers the next reduct.

The problem If we follow the standard program-transformation steps [5–11] it is not possible to connect the syntactic and the functional correspondence for normalisers implementing 'hybrid' strategies. The correspondences and the connection have been successfully established for 'uniform' strategies such as call-by-name and call-by-value, and a functional correspondence between a big-step virtual machine and a reduction-free normaliser has been established for a hybrid strategy, namely, normal order [7].

We have borrowed the uniform/hybrid terminology from [12] where it is used informally. A strategy is *uniform* when it is defined as a *single function* that only depends on itself, e.g., no other function occurs in the premisses of the inference rules of its natural semantics. In contrast, a strategy is *hybrid* (or layered) when it is defined as a *single function* that depends on (at least) another *subsidiary* strategy.[1] For example, the natural semantics of a hybrid $\Downarrow_h$ will have inference rules where a subsidiary $\Downarrow_s$ occurs in one or more premisses. Here is a possible example rule:

$$\frac{M \Downarrow_s M' \quad M' \Downarrow_h N}{M \Downarrow_h N}\ (\textsc{Rule})$$

In words, and reading relational notation functionally, Rule says that $\Downarrow_h$ reduces M to N by first reducing M to M' using $\Downarrow_s$ and then reducing M' to N recursively. The term M' is the point at which $\Downarrow_s$ stops and $\Downarrow_h$ resumes.

In many practical strategies (see Section 9), the subsidiary is employed by the hybrid to reduce some subterms *less* in order to uphold some properties. Which strategy, subsidiary or hybrid, is to start, continue, or resume the next reduction is clear in the semantics. In the syntactic correspondence, several semantic artefacts (subsidiaries and hybrid) are written in parallel. However, the refocusing and inlining-of-iterate-function steps become context dependent, and the dispatcher of the abstract machine has to inspect the continuation stack (the arguments of value constructors that represent defunctionalised continuations) deeply to find out which strategy is to continue. This prevents the refunctionalisation of the machine.

[1] We insist on 'single-function' to avoid confusion with definitions in *eval-readback style* (Section 4), a degenerate case of normalisation-by-evaluation where a strategy is defined as the composition of *two* single functions, e.g., two natural semantics.