A Modular Structural Operational Semantics for Delimited Continuations

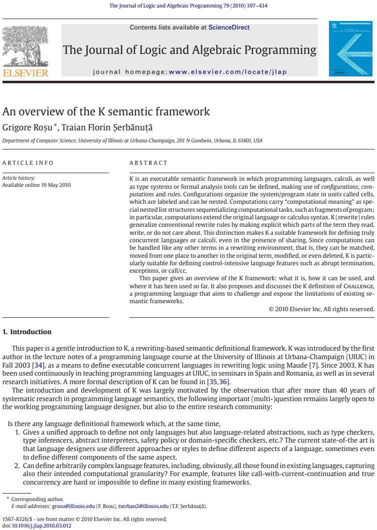
Neil Sculthorpe, Paolo Torrini and Peter D. Mosses

PLanCompS Project
Department of Computer Science
Swansea University
{N.A.Sculthorpe,P.Torrini,P.D.Mosses}@swansea.ac.uk

First Workshop on Continuations
London, England
12th April 2015

The Challenge

"While the use of labels gives MSOS the ability to modularly deal with some forms of control, such as abrupt termination, at our knowledge it still cannot support the definition of arbitrarily complex control-intensive features such call/cc."



The Journal of Logic and Algebraic Programming 79 (2010) 397–434

Contents lists available at ScienceDirect

The Journal of Logic and Algebraic Programming

journal homepage: www.elsevier.com/locate/jlap

An overview of the K semantic framework

Grigore Roşu *, Traian Florin Şerbănuţă

Department of Computer Science, University of Illinois at Urbana-Champaign, 201 N Goodwin, Urbana, IL 61801, USA

ARTICLE INFO

Article history:
Available online 19 May 2010

ABSTRACT

K is an executable semantic framework in which programming languages, calculi, as well as type systems or formal analysis tools can be defined, making use of *configurations*, *computations* and *rules*. Configurations organize the system/program state in units called cells, which are labeled and can be nested. Computations carry "computational meaning" as special nested list structures sequentializing computational tasks, such as fragments of program; in particular, computations extend the original language or calculus syntax. K (rewrite) rules generalize conventional rewrite rules by making explicit which parts of the term they read, write, or do not care about. This distinction makes K a suitable framework for defining truly concurrent languages or calculi, even in the presence of sharing. Since computations can be handled like any other terms in a rewriting environment, that is, they can be matched, moved from one place to another in the original term, modified, or even deleted, K is particularly suitable for defining control-intensive language features such as abrupt termination, exceptions, or call/cc.

This paper gives an overview of the K framework: what it is, how it can be used, and where it has been used so far. It also proposes and discusses the K definition of CHALLENGE, a programming language that aims to challenge and expose the limitations of existing semantic frameworks.

© 2010 Elsevier Inc. All rights reserved.

1. Introduction

This paper is a gentle introduction to K, a rewriting-based semantic definitional framework. K was introduced by the first author in the lecture notes of a programming language course at the University of Illinois at Urbana-Champaign (UIUC) in Fall 2003 [34], as a means to define executable concurrent languages in rewriting logic using Maude [7]. Since 2003, K has been used continuously in teaching programming languages at UIUC, in seminars in Spain and Romania, as well as in several research initiatives. A more formal description of K can be found in [35,36].

The introduction and development of K was largely motivated by the observation that after more than 40 years of systematic research in programming language semantics, the following important (multi-)question remains largely open to the working programming language designer, but also to the entire research community:

Is there any language definitional framework which, at the same time,

1. Gives a unified approach to define not only languages but also language-related abstractions, such as type checkers, type inferencers, abstract interpreters, safety policy or domain-specific checkers, etc.? The current state-of-the art is that language designers use different approaches or styles to define different aspects of a language, sometimes even to define different components of the same aspect.
2. Can define arbitrarily complex language features, including, obviously, all those found in existing languages, capturing also their intended computational granularity? For example, features like call-with-current-continuation and true concurrency are hard or impossible to define in many existing frameworks.

* Corresponding author.
E-mail addresses: grosu@illinois.edu (F. Roşu), tserban2@illinois.edu (T.F. Şerbănuţă).

1567-8326/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2010.03.012

The Challenge

"While the use of labels gives MSOS the ability to modularly deal with some forms of control, such as abrupt termination, at our knowledge it still cannot support the definition of arbitrarily complex control-intensive features such call/cc."

Yes it can!

The Journal of Logic and Algebraic Programming 79 (2010) 397–434

Contents lists available at ScienceDirect

The Journal of Logic and Algebraic Programming

journal homepage: www.elsevier.com/locate/jlap

An overview of the K semantic framework

Grigore Roşu *, Traian Florin Şerbănuţă

Department of Computer Science, University of Illinois at Urbana-Champaign, 201 N Goodwin, Urbana, IL 61801, USA

ARTICLE INFO

Article history:
Available online 19 May 2010

ABSTRACT

K is an executable semantic framework in which programming languages, calculi, as well as type systems or formal analysis tools can be defined, making use of *configurations*, *computations* and *rules*. Configurations organize the system/program state in units called cells, which are labeled and can be nested. Computations carry "computational meaning" as special nested list structures sequentializing computational tasks, such as fragments of program; in particular, computations extend the original language or calculus syntax. K (rewrite) rules generalize conventional rewrite rules by making explicit which parts of the term they read, write, or do not care about. This distinction makes K a suitable framework for defining truly concurrent languages or calculi, even in the presence of sharing. Since computations can be handled like any other terms in a rewriting environment, that is, they can be matched, moved from one place to another in the original term, modified, or even deleted, K is particularly suitable for defining control-intensive language features such as abrupt termination, exceptions, or call/cc.

This paper gives an overview of the K framework: what it is, how it can be used, and where it has been used so far. It also proposes and discusses the K definition of CHALLENGE, a programming language that aims to challenge and expose the limitations of existing semantic frameworks.

© 2010 Elsevier Inc. All rights reserved.

1. Introduction

This paper is a gentle introduction to K, a rewriting-based semantic definitional framework. K was introduced by the first author in the lecture notes of a programming language course at the University of Illinois at Urbana-Champaign (UIUC) in Fall 2003 [34], as a means to define executable concurrent languages in rewriting logic using Maude [7]. Since 2003, K has been used continuously in teaching programming languages at UIUC, in seminars in Spain and Romania, as well as in several research initiatives. A more formal description of K can be found in [35,36].

The introduction and development of K was largely motivated by the observation that after more than 40 years of systematic research in programming language semantics, the following important (multi-)question remains largely open to the working programming language designer, but also to the entire research community:

Is there any language definitional framework which, at the same time,
1. Gives a unified approach to define not only languages but also language-related abstractions, such as type checkers, type inferencers, abstract interpreters, safety policy or domain-specific checkers, etc.? The current state-of-the art is that language designers use different approaches or styles to define different aspects of a language, sometimes even to define different components of the same aspect.
2. Can define arbitrarily complex language features, including, obviously, all those found in existing languages, capturing also their intended computational granularity? For example, features like call-with-current-continuation and true concurrency are hard or impossible to define in many existing frameworks.

* Corresponding author.
E-mail addresses: grosu@illinois.edu (F. Roşu), tserban2@illinois.edu (T.F. Şerbănuţă).

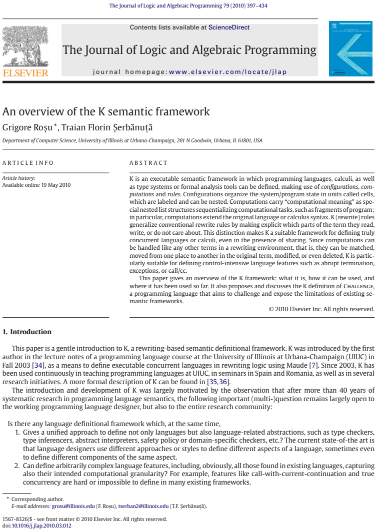
1567-8326/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2010.03.012

The Challenge

"While the use of labels gives MSOS the ability to modularly deal with some forms of control, such as abrupt termination, at our knowledge it still cannot support the definition of arbitrarily complex control-intensive features such call/cc."

Yes it can!

(And *control*/*prompt*, and *shift*/*reset*.)


The Journal of Logic and Algebraic Programming 79 (2010) 397–434

Contents lists available at ScienceDirect

The Journal of Logic and Algebraic Programming

journal homepage: www.elsevier.com/locate/jlap

An overview of the K semantic framework

Grigore Roșu *, Traian Florin Șerbănuță

Department of Computer Science, University of Illinois at Urbana-Champaign, 201 N Goodwin, Urbana, IL 61801, USA

ARTICLE INFO

Article history:
Available online 19 May 2010

ABSTRACT

K is an executable semantic framework in which programming languages, calculi, as well as type systems or formal analysis tools can be defined, making use of *configurations*, *computations* and *rules*. Configurations organize the system/program state in units called cells, which are labeled and can be nested. Computations carry "computational meaning" as special nested list structures sequentializing computational tasks, such as fragments of program; in particular, computations extend the original language or calculus syntax. K (rewrite) rules generalize conventional rewrite rules by making explicit which parts of the term they read, write, or do not care about. This distinction makes K a suitable framework for defining truly concurrent languages or calculi, even in the presence of sharing. Since computations can be handled like any other terms in a rewriting environment, that is, they can be matched, moved from one place to another in the original term, modified, or even deleted, K is particularly suitable for defining control-intensive language features such as abrupt termination, exceptions, or call/cc.

This paper gives an overview of the K framework: what it is, how it can be used, and where it has been used so far. It also proposes and discusses the K definition of CHALLENGE, a programming language that aims to challenge and expose the limitations of existing semantic frameworks.

© 2010 Elsevier Inc. All rights reserved.

1. Introduction

This paper is a gentle introduction to K, a rewriting-based semantic definitional framework. K was introduced by the first author in the lecture notes of a programming language course at the University of Illinois at Urbana-Champaign (UIUC) in Fall 2003 [34], as a means to define executable concurrent languages in rewriting logic using Maude [7]. Since 2003, K has been used continuously in teaching programming languages at UIUC, in seminars in Spain and Romania, as well as in several research initiatives. A more formal description of K can be found in [35,36].

The introduction and development of K was largely motivated by the observation that after more than 40 years of systematic research in programming language semantics, the following important (multi-)question remains largely open to the working programming language designer, but also to the entire research community:

Is there any language definitional framework which, at the same time,

1. Gives a unified approach to define not only languages but also language-related abstractions, such as type checkers, type inferencers, abstract interpreters, safety policy or domain-specific checkers, etc.? The current state-of-the art is that language designers use different approaches or styles to define different aspects of a language, sometimes even to define different components of the same aspect.
2. Can define arbitrarily complex language features, including, obviously, all those found in existing languages, capturing also their intended computational granularity? For example, features like call-with-current-continuation and true concurrency are hard or impossible to define in many existing frameworks.

* Corresponding author.
E-mail addresses: grosu@illinois.edu (F. Roșu), tserban2@illinois.edu (T.F. Șerbănuță).

1567-8326/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2010.03.012

This Talk

- Overview and motivate MSOS
- MSOS specifications of *control* and *prompt*

This Talk

- Overview and motivate MSOS
- MSOS specifications of *control* and *prompt*
- Specifications of *shift* and *call/cc* in terms of *control* and *prompt*

Modular Structural Operational Semantics (MSOS)

- A *modular* variant of Plotkin's SOS framework.

$$\frac{\rho_2 \vdash Y \xrightarrow{s_2} Y'}{\rho_1 \vdash X \xrightarrow{s_1} X'}$$

- Benefit: rules need not mention unused auxiliary entities.

- We use a flavour of MSOS called *Implicitly Modular SOS* (I-MSOS):
  - unmentioned entities are propagated between premise and conclusion;
  - when there is no premise, unmentioned signals have a default value.

- This talk will use *small-step* transition rules.

I-MSOS Specification of Lambda Calculus

$$
\begin{array}{rcl}
E & ::= & V \\
 & | & \textbf{bv}(I) \\
 & | & \textbf{lambda}(I, E) \\
 & | & \textbf{apply}(E, E) \\
 & | & \ldots \\
V & ::= & \textbf{closure}(\rho, I, E) \\
 & | & \ldots
\end{array}
$$

I-MSOS Specification of Lambda Calculus

$$
\begin{array}{rcl}
E & ::= & V \\
 & | & \mathbf{bv}(I) \\
 & | & \mathbf{lambda}(I, E) \\
 & | & \mathbf{apply}(E, E) \\
 & | & \ldots \\
V & ::= & \mathbf{closure}(\rho, I, E) \\
 & | & \ldots
\end{array}
$$

$$
\frac{\rho(I) = V}{\mathrm{env}\ \rho \vdash \mathbf{bv}(I) \rightarrow V}
$$

I-MSOS Specification of Lambda Calculus

$$
\begin{array}{rcl}
E & ::= & V \\
 & | & \mathbf{bv}(I) \\
 & | & \mathbf{lambda}(I, E) \\
 & | & \mathbf{apply}(E, E) \\
 & | & \ldots \\
V & ::= & \mathbf{closure}(\rho, I, E) \\
 & | & \ldots
\end{array}
$$

$$
\frac{\rho(I) = V}{\text{env } \rho \vdash \mathbf{bv}(I) \rightarrow V}
$$

$$
\text{env } \rho \vdash \mathbf{lambda}(I, E) \rightarrow \mathbf{closure}(\rho, I, E)
$$

I-MSOS Specification of Lambda Calculus

$$
\begin{array}{rcl}
E & ::= & V \\
 & | & \mathbf{bv}(I) \\
 & | & \mathbf{lambda}(I, E) \\
 & | & \mathbf{apply}(E, E) \\
 & | & \ldots \\
V & ::= & \mathbf{closure}(\rho, I, E) \\
 & | & \ldots
\end{array}
$$

$$\frac{\rho(I) = V}{\text{env } \rho \vdash \mathbf{bv}(I) \to V}$$

$$\text{env } \rho \vdash \mathbf{lambda}(I, E) \to \mathbf{closure}(\rho, I, E)$$

$$\frac{\text{env } \rho \vdash E_1 \to E_1'}{\text{env } \rho \vdash \mathbf{apply}(E_1, E_2) \to \mathbf{apply}(E_1', E_2)}$$

$$\frac{\text{val}(V) \qquad \text{env } \rho \vdash E \to E'}{\text{env } \rho \vdash \mathbf{apply}(V, E) \to \mathbf{apply}(V, E')}$$

$$\frac{\text{val}(V) \qquad \text{env } (\{I \mapsto V\}/\rho) \vdash E \to E'}{\text{env } \_ \vdash \mathbf{apply}(\mathbf{closure}(\rho, I, E), V) \to \mathbf{apply}(\mathbf{closure}(\rho, I, E'), V)}$$

$$\frac{\text{val}(V_1) \qquad \text{val}(V_2)}{\text{env } \rho \vdash \mathbf{apply}(\mathbf{closure}(\rho, I, V_1), V_2) \to V_1}$$

I-MSOS Specification of Lambda Calculus

$$
\begin{array}{rcl}
E & ::= & V \\
 & | & \mathbf{bv}(I) \\
 & | & \mathbf{lambda}(I, E) \\
 & | & \mathbf{apply}(E, E) \\
 & | & \ldots \\
V & ::= & \mathbf{closure}(\rho, I, E) \\
 & | & \ldots
\end{array}
$$

$$\frac{\rho(I) = V}{\mathsf{env}\ \rho \vdash \mathbf{bv}(I) \to V}$$

$$\mathsf{env}\ \rho \vdash \mathbf{lambda}(I, E) \to \mathbf{closure}(\rho, I, E)$$

$$\frac{E_1 \to E_1'}{\mathbf{apply}(E_1, E_2) \to \mathbf{apply}(E_1', E_2)}$$

$$\frac{\mathsf{val}(V) \qquad E \to E'}{\mathbf{apply}(V, E) \to \mathbf{apply}(V, E')}$$

$$\frac{\mathsf{val}(V) \qquad \mathsf{env}\ (\{I \mapsto V\}/\rho) \vdash E \to E'}{\mathsf{env}\ \_ \vdash \mathbf{apply}(\mathbf{closure}(\rho, I, E), V) \to \mathbf{apply}(\mathbf{closure}(\rho, I, E'), V)}$$

$$\frac{\mathsf{val}(V_1) \qquad \mathsf{val}(V_2)}{\mathbf{apply}(\mathbf{closure}(\rho, I, V_1), V_2) \to V_1}$$

I-MSOS Specification of Exception Handling

$$
\begin{array}{rcl}
E, H & ::= & \textbf{throw}(E) \\
& | & \textbf{catch}(E, H) \\
& | & \textbf{stuck} \\
& | & \ldots
\end{array}
$$

I-MSOS Specification of Exception Handling

$$E, H ::= \mathbf{throw}(E) \mid \mathbf{catch}(E, H) \mid \mathbf{stuck} \mid \ldots$$

$$\frac{E \xrightarrow{\text{exc } X} E'}{\mathbf{throw}(E) \xrightarrow{\text{exc } X} \mathbf{throw}(E')}$$

$$\frac{\mathsf{val}(V)}{\mathbf{throw}(V) \xrightarrow{\text{exc } \mathsf{some}(V)} \mathbf{stuck}}$$

I-MSOS Specification of Exception Handling

$$E, H ::= \mathbf{throw}(E) \mid \mathbf{catch}(E, H) \mid \mathbf{stuck} \mid \ldots$$

$$\frac{E \longrightarrow E'}{\mathbf{throw}(E) \longrightarrow \mathbf{throw}(E')}$$

$$\frac{\mathsf{val}(V)}{\mathbf{throw}(V) \xrightarrow{\mathrm{exc}\ \mathbf{some}(V)} \mathbf{stuck}}$$

I-MSOS Specification of Exception Handling

$$E, H ::= \mathbf{throw}(E) \mid \mathbf{catch}(E, H) \mid \mathbf{stuck} \mid \ldots$$

$$\frac{E \longrightarrow E'}{\mathbf{throw}(E) \longrightarrow \mathbf{throw}(E')}$$

$$\frac{\mathsf{val}(V)}{\mathbf{throw}(V) \xrightarrow{\text{exc } \mathbf{some}(V)} \mathbf{stuck}}$$

$$\frac{E \xrightarrow{\text{exc } \mathbf{none}} E'}{\mathbf{catch}(E, H) \xrightarrow{\text{exc } \mathbf{none}} \mathbf{catch}(E', H)}$$

$$\frac{E \xrightarrow{\text{exc } \mathbf{some}(V)} E'}{\mathbf{catch}(E, H) \xrightarrow{\text{exc } \mathbf{none}} \mathbf{apply}(H, V)}$$

$$\frac{\mathsf{val}(V)}{\mathbf{catch}(V, H) \xrightarrow{\text{exc } \mathbf{none}} V}$$

I-MSOS Specification of Exception Handling

$$E, H ::= \mathbf{throw}(E) \mid \mathbf{catch}(E, H) \mid \mathbf{stuck} \mid \ldots$$

$$\frac{E \longrightarrow E'}{\mathbf{throw}(E) \longrightarrow \mathbf{throw}(E')}$$

$$\frac{\mathsf{val}(V)}{\mathbf{throw}(V) \xrightarrow{\mathsf{exc}\ \mathbf{some}(V)} \mathbf{stuck}}$$

$$\frac{E \xrightarrow{\mathsf{exc}\ \mathbf{none}} E'}{\mathbf{catch}(E, H) \xrightarrow{\mathsf{exc}\ \mathbf{none}} \mathbf{catch}(E', H)}$$

$$\frac{E \xrightarrow{\mathsf{exc}\ \mathbf{some}(V)} E'}{\mathbf{catch}(E, H) \xrightarrow{\mathsf{exc}\ \mathbf{none}} \mathbf{apply}(H, V)}$$

$$\frac{\mathsf{val}(V)}{\mathbf{catch}(V, H) \longrightarrow V}$$

SOS Specification of Lambda Calculus with Exceptions

$$\frac{\rho(I) = V}{\text{env } \rho \vdash \textbf{bv}(I) \xrightarrow{\text{exc } \textbf{none}} V}$$

$$\text{env } \rho \vdash \textbf{lambda}(I, E) \xrightarrow{\text{exc } \textbf{none}} \textbf{closure}(\rho, I, E)$$

$$\frac{\text{env } \rho \vdash E_1 \xrightarrow{\text{exc } X} E_1'}{\text{env } \rho \vdash \textbf{apply}(E_1, E_2) \xrightarrow{\text{exc } X} \textbf{apply}(E_1', E_2)}$$

$$\frac{\text{val}(V) \qquad \text{env } \rho \vdash E \xrightarrow{\text{exc } X} E'}{\text{env } \rho \vdash \textbf{apply}(V, E) \xrightarrow{\text{exc } X} \textbf{apply}(V, E')}$$

$$\frac{\text{val}(V) \qquad \text{env } (\{I \mapsto V\}/\rho) \vdash E \xrightarrow{\text{exc } X} E'}{\text{env } \_ \vdash \textbf{apply}(\textbf{closure}(\rho, I, E), V) \xrightarrow{\text{exc } X} \textbf{apply}(\textbf{closure}(\rho, I, E'), V)}$$

$$\frac{\text{val}(V_1) \qquad \text{val}(V_2)}{\text{env } \rho \vdash \textbf{apply}(\textbf{closure}(\rho, I, V_1), V_2) \xrightarrow{\text{exc } \textbf{none}} V_1}$$

$$\frac{\text{env } \rho \vdash E \xrightarrow{\text{exc } X} E'}{\text{env } \rho \vdash \textbf{throw}(E) \xrightarrow{\text{exc } X} \textbf{throw}(E')}$$

$$\frac{\text{val}(V)}{\text{env } \rho \vdash \textbf{throw}(V) \xrightarrow{\text{exc } \textbf{some}(V)} \textbf{stuck}}$$

$$\frac{\text{env } \rho \vdash E \xrightarrow{\text{exc } \textbf{none}} E'}{\text{env } \rho \vdash \textbf{catch}(E, H) \xrightarrow{\text{exc } \textbf{none}} \textbf{catch}(E', H)}$$

$$\frac{\text{env } \rho \vdash E \xrightarrow{\text{exc } \textbf{some}(V)} E'}{\text{env } \rho \vdash \textbf{catch}(E, H) \xrightarrow{\text{exc } \textbf{none}} \textbf{apply}(H, V)}$$

$$\frac{\text{val}(V)}{\text{env } \rho \vdash \textbf{catch}(V, H) \xrightarrow{\text{exc } \textbf{none}} V}$$

I-MSOS Specification of Lambda Calculus with Exceptions

$$\frac{\rho(I) = V}{\text{env } \rho \vdash \textbf{bv}(I) \longrightarrow V}$$

$$\text{env } \rho \vdash \textbf{lambda}(I, E) \longrightarrow \textbf{closure}(\rho, I, E)$$

$$\frac{E_1 \longrightarrow E_1'}{\textbf{apply}(E_1, E_2) \longrightarrow \textbf{apply}(E_1', E_2)}$$

$$\frac{\text{val}(V) \qquad E \longrightarrow E'}{\textbf{apply}(V, E) \longrightarrow \textbf{apply}(V, E')}$$

$$\frac{\text{val}(V) \qquad \text{env } (\{I \mapsto V\}/\rho) \vdash E \longrightarrow E'}{\text{env } \_ \vdash \textbf{apply}(\textbf{closure}(\rho, I, E), V) \longrightarrow \textbf{apply}(\textbf{closure}(\rho, I, E'), V)}$$

$$\frac{\text{val}(V_1) \qquad \text{val}(V_2)}{\textbf{apply}(\textbf{closure}(\rho, I, V_1), V_2) \longrightarrow V_1}$$

$$\frac{E \longrightarrow E'}{\textbf{throw}(E) \longrightarrow \textbf{throw}(E')}$$

$$\frac{\text{val}(V)}{\textbf{throw}(V) \xrightarrow{\text{exc } \textbf{some}(V)} \textbf{stuck}}$$

$$\frac{E \xrightarrow{\text{exc } \textbf{none}} E'}{\textbf{catch}(E, H) \xrightarrow{\text{exc } \textbf{none}} \textbf{catch}(E', H)}$$

$$\frac{E \xrightarrow{\text{exc } \textbf{some}(V)} E'}{\textbf{catch}(E, H) \xrightarrow{\text{exc } \textbf{none}} \textbf{apply}(H, V)}$$

$$\frac{\text{val}(V)}{\textbf{catch}(V, H) \longrightarrow V}$$

Formulating *control* and *prompt*

- We formulate *control* as a unary operator that takes a higher-order function as its argument:

$$1 + \mathit{prompt}(2 * \mathit{control}(\lambda k.\ k(k\ 7))) \quad \rightsquigarrow \quad 29$$

- In our notation:

$$\textbf{plus}(1, \textbf{prompt}(\textbf{times}(2, \textbf{control}(\textbf{lambda}(K, \textbf{apply}(\textbf{bv}(K), \textbf{apply}(\textbf{bv}(K), 7)))))))$$

$$\boxed{\begin{array}{rcl} E & ::= & \textbf{control}(E) \\ & \mid & \textbf{prompt}(E) \\ & \mid & \textbf{times}(E, E) \\ & \mid & \textbf{plus}(E, E) \\ & \mid & \ldots \\ V & ::= & \mathit{integers} \\ & \mid & \ldots \end{array}}$$

Specifying *control* and *prompt*

Key ideas:

- We *don't* maintain an explicit representation of the current continuation.
- We *construct* the continuation from the program term when needed.
- Use *signals* to communicate between control operators and delimiters:
  - **control** emits a signal when executed;
  - **prompt** catches that signal and handles it.

I-MSOS Specification of *control* and *prompt*

$$E ::= \textbf{control}(E) \mid \textbf{prompt}(E) \mid \ldots$$

$$\frac{E \rightarrow E'}{\textbf{control}(E) \rightarrow \textbf{control}(E')}$$

$$\frac{\mathsf{val}(F) \qquad \mathsf{fresh\text{-}id}(I)}{\textbf{control}(F) \xrightarrow{\text{control } \textbf{some}(F,I)} \textbf{bv}(I)}$$

I-MSOS Specification of *control* and *prompt*

$$E ::= \mathbf{control}(E) \mid \mathbf{prompt}(E) \mid \ldots$$

$$\frac{E \rightarrow E'}{\mathbf{control}(E) \rightarrow \mathbf{control}(E')}$$

$$\frac{\mathsf{val}(F) \qquad \mathsf{fresh\text{-}id}(I)}{\mathbf{control}(F) \xrightarrow{\mathsf{control}\ \mathbf{some}(F,I)} \mathbf{bv}(I)}$$

$$\frac{E \xrightarrow{\mathsf{control}\ \mathbf{none}} E'}{\mathbf{prompt}(E) \xrightarrow{\mathsf{control}\ \mathbf{none}} \mathbf{prompt}(E')}$$

I-MSOS Specification of *control* and *prompt*

$$E ::= \mathbf{control}(E) \mid \mathbf{prompt}(E) \mid \ldots$$

$$\frac{E \rightarrow E'}{\mathbf{control}(E) \rightarrow \mathbf{control}(E')}$$

$$\frac{\mathrm{val}(F) \qquad \text{fresh-id}(I)}{\mathbf{control}(F) \xrightarrow{\text{control } \mathbf{some}(F,I)} \mathbf{bv}(I)}$$

$$\frac{E \xrightarrow{\text{control } \mathbf{none}} E'}{\mathbf{prompt}(E) \xrightarrow{\text{control } \mathbf{none}} \mathbf{prompt}(E')}$$

$$\frac{E \xrightarrow{\text{control } \mathbf{some}(F,I)} E' \qquad K = \mathbf{lambda}(I, E')}{\mathbf{prompt}(E) \xrightarrow{\text{control } \mathbf{none}} \mathbf{prompt}(\mathbf{apply}(F, K))}$$

I-MSOS Specification of *control* and *prompt*

$$E ::= \mathbf{control}(E) \mid \mathbf{prompt}(E) \mid \ldots$$

$$\frac{E \rightarrow E'}{\mathbf{control}(E) \rightarrow \mathbf{control}(E')}$$

$$\frac{\mathsf{val}(F) \qquad \mathsf{fresh\text{-}id}(I)}{\mathbf{control}(F) \xrightarrow{\text{control } \mathbf{some}(F,I)} \mathbf{bv}(I)}$$

$$\frac{E \xrightarrow{\text{control } \mathbf{none}} E'}{\mathbf{prompt}(E) \xrightarrow{\text{control } \mathbf{none}} \mathbf{prompt}(E')}$$

$$\frac{E \xrightarrow{\text{control } \mathbf{some}(F,I)} E' \qquad K = \mathbf{lambda}(I, E')}{\mathbf{prompt}(E) \xrightarrow{\text{control } \mathbf{none}} \mathbf{prompt}(\mathbf{apply}(F, K))}$$

$$\frac{\mathsf{val}(V)}{\mathbf{prompt}(V) \rightarrow V}$$

I-MSOS Specification of *control* and *prompt*

$$E ::= \textbf{control}(E) \mid \textbf{prompt}(E) \mid \ldots$$

$$\frac{E \rightarrow E'}{\textbf{control}(E) \rightarrow \textbf{control}(E')}$$

$$\frac{\mathsf{val}(F) \qquad \mathsf{fresh\text{-}id}(I)}{\textbf{control}(F) \xrightarrow{\mathsf{control}\ \textbf{some}(F,I)} \textbf{meta-bv}(I)}$$

$$\frac{E \xrightarrow{\mathsf{control}\ \textbf{none}} E'}{\textbf{prompt}(E) \xrightarrow{\mathsf{control}\ \textbf{none}} \textbf{prompt}(E')}$$

$$\frac{E \xrightarrow{\mathsf{control}\ \textbf{some}(F,I)} E' \qquad K = \textbf{lambda}(I, \textbf{meta-let-in}(I, \textbf{bv}(I), E'))}{\textbf{prompt}(E) \xrightarrow{\mathsf{control}\ \textbf{none}} \textbf{prompt}(\textbf{apply}(F, K))}$$

$$\frac{\mathsf{val}(V)}{\textbf{prompt}(V) \rightarrow V}$$

The Meta-environment

- An auxiliary environment that doesn't interact with closures.
- Used here to achieve the same effect as substitution.

$$\boxed{\begin{array}{lll} E & ::= & \textbf{meta-bv}(I) \\ & \mid & \textbf{meta-let-in}(I, E, E) \\ & \mid & \ldots \end{array}}$$

$$\frac{\rho(I) = V}{\text{meta-env } \rho \vdash \textbf{meta-bv}(I) \rightarrow V}$$

$$\frac{\text{meta-env } \rho \vdash E_1 \rightarrow E_1'}{\text{meta-env } \rho \vdash \textbf{meta-let-in}(I, E_1, E_2) \rightarrow \textbf{meta-let-in}(I, E_1', E_2)}$$

$$\frac{\text{val}(V) \qquad \text{meta-env } (\{I \mapsto V\}/\rho) \vdash E \rightarrow E'}{\text{meta-env } \rho \vdash \textbf{meta-let-in}(I, V, E) \rightarrow \textbf{meta-let-in}(I, V, E')}$$

$$\frac{\text{val}(V_1) \qquad \text{val}(V_2)}{\text{meta-env } \rho \vdash \textbf{meta-let-in}(I, V_1, V_2) \rightarrow V_2}$$

Conclusion

- MSOS allows programming constructs to be specified *independently*.
- Contrary to popular belief, specifying control operators in MSOS is fairly straightforward.
- In the paper we have also specified *call/cc*, *shift* and *reset*.
- Specifications tested on 70 test programs (including Mondo Bizarro!), using our I-MSOS interpreter.
- Test suite available online: `http://www.plancomps.org/woc2016`

I-MSOS Specification of *shift* and *reset*

$$E ::= \mathbf{shift}(E) \mid \mathbf{reset}(E) \mid \ldots$$

$$\mathbf{reset}(E) \to \mathbf{prompt}(E)$$

$$\frac{E \to E'}{\mathbf{shift}(E) \to \mathbf{shift}(E')}$$

$$\frac{\mathrm{val}(F) \qquad \text{fresh-id}(K) \qquad \text{fresh-id}(X)}{\mathbf{shift}(F) \to \mathbf{control}(\mathbf{lambda}(K, \mathbf{apply}(F, \mathbf{lambda}(X, \mathbf{reset}(\mathbf{apply}(\mathbf{bv}(K), \mathbf{bv}(X)))))))}$$

- i.e. $\mathit{shift}(f) \;=\; \mathit{control}(\lambda k.\ f(\lambda x.\ \mathit{reset}(k\ x)))$

I-MSOS Specification of *abort* and *call/cc*

$$E ::= \mathbf{abort}(E) \mid \mathbf{callcc}(E) \mid \ldots$$

$$\frac{E \rightarrow E'}{\mathbf{abort}(E) \rightarrow \mathbf{abort}(E')}$$

$$\frac{\mathsf{val}(V) \qquad \mathsf{fresh\text{-}id}(I)}{\mathbf{abort}(V) \rightarrow \mathbf{control}(\mathbf{lambda}(I, V))}$$

I-MSOS Specification of *abort* and *call/cc*

$$E ::= \mathbf{abort}(E) \mid \mathbf{callcc}(E) \mid \ldots$$

$$\frac{E \to E'}{\mathbf{abort}(E) \to \mathbf{abort}(E')}$$

$$\frac{\mathsf{val}(V) \qquad \mathsf{fresh\text{-}id}(I)}{\mathbf{abort}(V) \to \mathbf{control}(\mathbf{lambda}(I, V))}$$

$$\frac{E \to E'}{\mathbf{callcc}(E) \to \mathbf{callcc}(E')}$$

$$\frac{\mathsf{val}(F) \qquad \mathsf{fresh\text{-}id}(K) \qquad \mathsf{fresh\text{-}id}(X)}{\begin{array}{l}\mathbf{callcc}(F) \to \\ \quad \mathbf{control}(\mathbf{lambda}(K, \mathbf{apply}(\mathbf{bv}(K), \mathbf{apply}(F, \mathbf{lambda}(X, \mathbf{abort}(\mathbf{apply}(\mathbf{bv}(K), \mathbf{bv}(X))))))))\end{array}}$$

- i.e. $callcc(f) \;=\; control(\lambda k.\ k\ (f\ (\lambda x.\ abort(k\ x))))$
- N.B. To simulate an undelimited *call/cc*, the program should contain only a single top-level delimiter.

Eugene Kohlbecker's Mondo Bizarro

$$
\begin{array}{l}
\textbf{let}\ \mathit{mondo\_bizarro}\ () = \textbf{let}\ k = \textbf{callcc}(\textbf{function}\ c \to c) \\
\qquad\qquad\qquad\qquad\quad \textbf{in}\ \mathit{print}\ 1\ ; \\
\qquad\qquad\qquad\qquad\qquad \textbf{callcc}(k)\ ; \\
\qquad\qquad\qquad\qquad\qquad \mathit{print}\ 2\ ; \\
\qquad\qquad\qquad\qquad\qquad \textbf{callcc}(k)\ ; \\
\qquad\qquad\qquad\qquad\qquad \mathit{print}\ 3\ ;;
\end{array}
$$

$$\textbf{prompt}(\mathit{mondo\_bizarro}())\ ;;$$

Output: [1 ; 1 ; 2 ; 1 ; 3]