A Modular Structural Operational Semantics for Delimited Continuations

Neil Sculthorpe

(Joint work with Paolo Torrini and Peter D. Mosses)

PLanCompS Project
Department of Computer Science
Royal Holloway, University of London
neil.sculthorpe@rhul.ac.uk

Nottingham, England
4th March 2016

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 [sic] 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 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.

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

- Introduction to Control Operators
- Introduction to Modular Structural Operational Semantics (MSOS)
- Formal specifications of control operators using MSOS

Delimited Control Operators

- *control*:
  - capture the current continuation as a lambda abstraction
  - apply the argument of *control* to the captured continuation
- *prompt*:
  - a delimiter for *control*

Delimited Control Operators

- *control*:
  - capture the current continuation as a lambda abstraction
  - apply the argument of *control* to the captured continuation
- *prompt*:
  - a delimiter for *control*

$$\textbf{prompt}(\ldots\textbf{control}(f)\ldots)$$

Delimited Control Operators

- *control*:
  - capture the current continuation as a lambda abstraction
  - apply the argument of *control* to the captured continuation
- *prompt*:
  - a delimiter for *control*

continuation: $(\lambda x.\ \ldots x \ldots)$

$$\mathbf{prompt}(\ldots \mathbf{control}(f) \ldots)$$

Delimited Control Operators

- *control*:
  - capture the current continuation as a lambda abstraction
  - apply the argument of *control* to the captured continuation
- *prompt*:
  - a delimiter for *control*

$$\text{continuation: } (\lambda x.\ \ldots x \ldots)$$

$$\textbf{prompt}(\ldots \textbf{control}(f) \ldots) \longrightarrow \textbf{prompt}(f(\lambda x.\ \ldots x \ldots))$$

Delimited Control Operators

- *control*:
  - capture the current continuation as a lambda abstraction
  - apply the argument of *control* to the captured continuation
- *prompt*:
  - a delimiter for *control*

$$\text{continuation: } (\lambda x.\ \ldots x \ldots)$$

$$\mathbf{prompt}(\ldots \mathbf{control}(f) \ldots) \longrightarrow \mathbf{prompt}(f(\lambda x.\ \ldots x \ldots))$$

Approximate Types

$\mathbf{prompt} : A \rightarrow A$

$\mathbf{control} : ((B \rightarrow A) \rightarrow A) \rightarrow B$

Delimited Control Operators

- *control*:
  - capture the current continuation as a lambda abstraction
  - apply the argument of *control* to the captured continuation
- *prompt*:
  - a delimiter for *control*

$$1 + \mathbf{prompt}(2 * \mathbf{control}(\lambda k.\ k\ 7))$$

Approximate Types

$\mathbf{prompt} : A \to A$

$\mathbf{control} : ((B \to A) \to A) \to B$

Delimited Control Operators

- *control*:
  - capture the current continuation as a lambda abstraction
  - apply the argument of *control* to the captured continuation
- *prompt*:
  - a delimiter for *control*

$$\text{continuation: } (\lambda x.\ 2 * x)$$

$$1 + \textbf{prompt}(2 * \textbf{control}(\lambda k.\ k\ 7))$$

Approximate Types

$\textbf{prompt} : A \to A$

$\textbf{control} : ((B \to A) \to A) \to B$

Delimited Control Operators

- *control*:
  - capture the current continuation as a lambda abstraction
  - apply the argument of *control* to the captured continuation
- *prompt*:
  - a delimiter for *control*

$$\text{continuation: } (\lambda x.\ 2 * x)$$

$$1 + \textbf{prompt}(2 * \textbf{control}(\lambda k.\ k\ 7)) \quad \longrightarrow$$
$$1 + \textbf{prompt}((\lambda k.\ k\ 7)\ (\lambda x.\ 2 * x))$$

Approximate Types

$\textbf{prompt} : A \to A$

$\textbf{control} : ((B \to A) \to A) \to B$

Delimited Control Operators

- *control*:
  - capture the current continuation as a lambda abstraction
  - apply the argument of *control* to the captured continuation
- *prompt*:
  - a delimiter for *control*

continuation: $(\lambda x.\ 2 * x)$

$$1 + \mathbf{prompt}(2 * \mathbf{control}(\lambda k.\ k\ 7)) \quad \longrightarrow$$
$$1 + \mathbf{prompt}((\lambda k.\ k\ 7)\ (\lambda x.\ 2 * x)) \quad \longrightarrow^* \quad 15$$

Approximate Types

$$\mathbf{prompt} : A \to A$$
$$\mathbf{control} : ((B \to A) \to A) \to B$$

Delimited Control Operators

- *control*:
  - capture the current continuation as a lambda abstraction
  - apply the argument of *control* to the captured continuation
- *prompt*:
  - a delimiter for *control*

$$\text{continuation: } (\lambda x.\ 2 * x)$$

$$1 + \textbf{prompt}(2 * \textbf{control}(\lambda k.\ k\ 7)) \quad \longrightarrow^* \quad 15$$

Approximate Types

$\textbf{prompt} : A \rightarrow A$

$\textbf{control} : ((B \rightarrow A) \rightarrow A) \rightarrow B$

Delimited Control Operators

- *control*:
  - capture the current continuation as a lambda abstraction
  - apply the argument of *control* to the captured continuation
- *prompt*:
  - a delimiter for *control*

$$\text{continuation: } (\lambda x.\ 2 * x)$$

$$1 + \mathbf{prompt}(2 * \mathbf{control}(\lambda k.\ k\ 7)) \quad \longrightarrow^* \quad 15$$

$$1 + \mathbf{prompt}(2 * \mathbf{control}(\lambda k.\ k(k\ 7))) \quad \longrightarrow^* \quad 29$$

Approximate Types

$$\mathbf{prompt} : A \to A$$

$$\mathbf{control} : ((B \to A) \to A) \to B$$

Delimited Control Operators

- *control*:
  - capture the current continuation as a lambda abstraction
  - apply the argument of *control* to the captured continuation
- *prompt*:
  - a delimiter for *control*

$$\text{continuation: } (\lambda x.\ 2 * x)$$

$$
\begin{aligned}
1 + \textbf{prompt}(2 * \textbf{control}(\lambda k.\ k\ 7)) &\quad \longrightarrow^* \quad 15 \\
1 + \textbf{prompt}(2 * \textbf{control}(\lambda k.\ k(k\ 7))) &\quad \longrightarrow^* \quad 29 \\
1 + \textbf{prompt}(2 * \textbf{control}(\lambda k.\ 7)) &\quad \longrightarrow^* \quad 8
\end{aligned}
$$

Approximate Types

$$\textbf{prompt} : A \rightarrow A$$

$$\textbf{control} : ((B \rightarrow A) \rightarrow A) \rightarrow B$$

Delimited Control Operators

- *control*:
  - capture the current continuation as a lambda abstraction
  - apply the argument of *control* to the captured continuation
- *prompt*:
  - a delimiter for *control*

$$\textbf{prompt}(\mathit{print}\ \text{'A'}\ ;\ \textbf{control}(\lambda k.\ k\ ()\ ;\ k\ ())\ ;\ \mathit{print}\ \text{'B'})$$

Approximate Types

$\textbf{prompt} : A \to A$

$\textbf{control} : ((B \to A) \to A) \to B$

Delimited Control Operators

- *control*:
  - capture the current continuation as a lambda abstraction
  - apply the argument of *control* to the captured continuation
- *prompt*:
  - a delimiter for *control*

$$k = \lambda x.\ x\ ;\ \ \mathit{print}\ \text{'B'}$$

$$\mathbf{prompt}(\mathit{print}\ \text{'A'}\ ;\ \mathbf{control}(\lambda k.\ k\ ()\ ;\ k\ ())\ ;\ \mathit{print}\ \text{'B'}) \quad \xrightarrow{\text{ABB}}{}^{*} \quad ()$$

Approximate Types

$\mathbf{prompt} : A \rightarrow A$

$\mathbf{control} : ((B \rightarrow A) \rightarrow A) \rightarrow B$

Encoding Exceptions

- *control* and *prompt* can express throwing and catching exceptions

Encoding Exceptions

- *control* and *prompt* can express throwing and catching exceptions

$$\textbf{data}\ \mathit{Result}\ a = \mathsf{Exc} \mid \mathsf{Val}\ a$$

$$\mathit{throw} : A$$
$$\mathit{throw} = \textbf{control}(\lambda\_.\ \mathsf{Exc})$$

$$\mathit{catch} : A \to A \to A$$
$$\begin{aligned}\mathit{catch}\ e\ h = \textbf{case prompt}(\mathsf{Val}\ e)\ \textbf{of}\ & \\ \mathsf{Val}\ v &\to v \\ \mathsf{Exc} &\to h\end{aligned}$$

Encoding other Control Operators

$$abort(v) \;=\; \mathbf{control}(\lambda\_.\ v)$$

$$callcc(f) \;=\; \mathbf{control}(\lambda k.\ k\ (f\ (\lambda x.\ abort(k\ x))))$$

$$shift(f) \;=\; \mathbf{control}(\lambda k.\ f(\lambda x.\ \mathbf{prompt}(k\ x)))$$

Modular Structural Operational Semantics (MSOS)

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: unused semantic entities can be abstracted away.
- We use *Implicitly Modular SOS* (I-MSOS) [MN09]:
  - syntactic sugar for 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

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

I-MSOS Specification of Lambda Calculus

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

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

I-MSOS Specification of Lambda Calculus

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

$$\frac{\rho(I) = V}{\mathsf{env}\ \rho \vdash \textbf{bound}(I) \rightarrow V}$$

$$\mathsf{env}\ \rho \vdash \textbf{lambda}(I, E) \rightarrow \textbf{closure}(\rho, I, E)$$

I-MSOS Specification of Lambda Calculus

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

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

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

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

I-MSOS Specification of Lambda Calculus

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

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

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

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

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

I-MSOS Specification of Lambda Calculus

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

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

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

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

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

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

I-MSOS Specification of Lambda Calculus

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

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

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

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

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

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

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

I-MSOS Specification of Lambda Calculus

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

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

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

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

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

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

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

I-MSOS Specification of Exception Handling

$$
\begin{array}{rcl}
E, H & ::= & \textbf{throw}(E) \\
& | & \textbf{catch}(E, H) \\
& | & \textbf{stuck} \\
& | & \ldots \\
V & ::= & \textbf{none} \\
& | & \textbf{some}(V) \\
& | & \ldots
\end{array}
$$

I-MSOS Specification of Exception Handling

$$
\begin{array}{lcl}
E, H & ::= & \mathbf{throw}(E) \\
 & | & \mathbf{catch}(E, H) \\
 & | & \mathbf{stuck} \\
 & | & \ldots \\
V & ::= & \mathbf{none} \\
 & | & \mathbf{some}(V) \\
 & | & \ldots
\end{array}
$$

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

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

I-MSOS Specification of Exception Handling

$$
\begin{array}{rcl}
E, H & ::= & \mathbf{throw}(E) \\
& | & \mathbf{catch}(E, H) \\
& | & \mathbf{stuck} \\
& | & \ldots \\
V & ::= & \mathbf{none} \\
& | & \mathbf{some}(V) \\
& | & \ldots
\end{array}
$$

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

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

I-MSOS Specification of Exception Handling

$$
\begin{array}{rcl}
E, H & ::= & \mathbf{throw}(E) \\
 & \mid & \mathbf{catch}(E, H) \\
 & \mid & \mathbf{stuck} \\
 & \mid & \ldots \\
V & ::= & \mathbf{none} \\
 & \mid & \mathbf{some}(V) \\
 & \mid & \ldots
\end{array}
$$

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

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

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

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

I-MSOS Specification of Exception Handling

$$
\begin{array}{lcl}
E, H & ::= & \mathbf{throw}(E) \\
 & \mid & \mathbf{catch}(E, H) \\
 & \mid & \mathbf{stuck} \\
 & \mid & \ldots \\
V & ::= & \mathbf{none} \\
 & \mid & \mathbf{some}(V) \\
 & \mid & \ldots
\end{array}
$$

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

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

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

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

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

I-MSOS Specification of Exception Handling

$$
\begin{array}{rcl}
E, H & ::= & \mathbf{throw}(E) \\
 & | & \mathbf{catch}(E, H) \\
 & | & \mathbf{stuck} \\
 & | & \ldots \\
V & ::= & \mathbf{none} \\
 & | & \mathbf{some}(V) \\
 & | & \ldots
\end{array}
$$

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

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

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

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

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

SOS Specification of Lambda Calculus with Exceptions

SOS Specification of Lambda Calculus with Exceptions

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

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

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

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

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

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

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

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

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

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

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

I-MSOS Specification of Lambda Calculus with Exceptions

$$\frac{\rho(I) = V}{\mathsf{env}\ \rho \vdash \mathbf{bound}(I) \longrightarrow V}$$

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

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

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

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

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

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

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

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

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

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

Specifying *control* and *prompt*

Specifying *control* and *prompt*

- Conventional specifications of control operators tend to be given in frameworks that use evaluation contexts.
- MSOS doesn't provide evaluation contexts, and we don't want to extend the framework to include them if we can avoid it.
- Instead we *construct* the continuation from the program term when needed.
- We use *signals* to communicate between control operators and delimiters:
  - **control** emits a signal when executed;
  - **prompt** catches that signal and handles it.

Auxiliary constructs: plug and hole

$$\boxed{\begin{array}{rcl} E & ::= & \mathbf{plug}(E, E) \\ & | & \mathbf{hole} \\ & | & \ldots \end{array}}$$

$$\frac{E_1 \rightarrow E_1'}{\mathbf{plug}(E_1, E_2) \rightarrow \mathbf{plug}(E_1', E_2)}$$

$$\frac{\mathsf{val}(V) \qquad E \xrightarrow{\text{plg } \mathbf{some}(V)} E'}{\mathbf{plug}(V, E) \xrightarrow{\text{plg } \mathbf{none}} E'}$$

$$\mathbf{hole} \xrightarrow{\text{plg } \mathbf{some}(V)} V$$

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

$$
\begin{array}{rcl}
E & ::= & \mathbf{control}(E) \\
 & \mid & \mathbf{prompt}(E) \\
 & \mid & \ldots
\end{array}
$$

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

$$\frac{\mathsf{val}(F)}{\mathbf{control}(F) \xrightarrow{\mathsf{ctrl}\ \mathsf{some}(F)} \mathbf{hole}}$$

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

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

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

$$\frac{\mathsf{val}(F)}{\mathbf{control}(F) \xrightarrow{\text{ctrl } \mathbf{some}(F)} \mathbf{hole}}$$

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

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

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

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

$$\frac{\mathsf{val}(F)}{\mathbf{control}(F) \xrightarrow{\mathsf{ctrl}\ \mathbf{some}(F)} \mathbf{hole}}$$

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

$$\frac{E \xrightarrow{\mathsf{ctrl}\ \mathbf{some}(F)} E' \qquad \mathsf{fresh\text{-}id}(I) \qquad K = \mathbf{lambda}(I, \mathbf{plug}(\mathbf{bound}(I), E'))}{\mathbf{prompt}(E) \xrightarrow{\mathsf{ctrl}\ \mathbf{none}} \mathbf{prompt}(\mathbf{apply}(F, K))}$$

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

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

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

$$\frac{\mathsf{val}(F)}{\mathbf{control}(F) \xrightarrow{\text{ctrl } \mathbf{some}(F)} \mathbf{hole}}$$

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

$$\frac{E \xrightarrow{\text{ctrl } \mathbf{some}(F)} E' \qquad \mathsf{fresh\text{-}id}(I) \qquad K = \mathbf{lambda}(I, \mathbf{plug}(\mathbf{bound}(I), E'))}{\mathbf{prompt}(E) \xrightarrow{\text{ctrl } \mathbf{none}} \mathbf{prompt}(\mathbf{apply}(F, K))}$$

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

Testing

- We have a pre-existing executable specification of Caml Light, given by translation to fundamental constructs (funcons) [CMST15].
- Added **control** and **prompt** as new keywords, and specified them by direct translation to the **control** and **prompt** funcons.
- Modular: no other modification to the specification of Caml Light was required!
- Specification and test suite available at: `http://www.plancomps.org/woc2016`

Conclusion

- MSOS allows programming-language constructs to be specified *independently* of unrelated semantic entities.
- Control operators can be specified in MSOS without needing evaluation contexts.
- More details, and a specification of *call/cc*, in our paper [STM16].

References

Martin Churchill, Peter D. Mosses, Neil Sculthorpe, and Paolo Torrini.
Reusable components of semantic specifications.
In *Transactions on Aspect-Oriented Software Development XII*, volume 8989 of *Lecture Notes in Computer Science*, pages 132–179. Springer, 2015.
https://doi.org/10.1007/978-3-662-46734-3_4.

Peter D. Mosses and Mark J. New.
Implicit propagation in structural operational semantics.
In *Workshop on Structural Operational Semantics*, volume 229(4) of *Electronic Notes in Theoretical Computer Science*, pages 49–66. Elsevier, 2009.
https://doi.org/10.1016/j.entcs.2009.07.073.

Neil Sculthorpe, Paolo Torrini, and Peter D. Mosses.
A modular structural operational semantics for delimited continuations.
In *Post-proceedings of the 2015 Workshop on Continuations*, volume 212 of *Electronic Proceedings in Theoretical Computer Science*, pages 63–80. Open Publishing Association, 2016.
https://doi.org/10.4204/EPTCS.212.5.