Towards Operations on Operational Semantics

Mauro Jaskelioff
mjj@cs.nott.ac.uk

School of Computer Science & IT
The University of Nottingham

22nd British Colloquium for Theoretical Computer Science

The Context

- We need semantics to *reason* about programs.
- Operational semantics is a popular way of giving semantics to languages.
- Languages evolve over time and need to be extended.
- We want to use what we alredy knew to reason about the extended language.
- However, operational semantics have poor modularity.

Modularity in SOS

- An arithmetics language

$$a ::= Con\ n \mid Add\ a\ a$$

where n ranges over $\mathbb{Z}$.

$$\frac{}{\text{Con}\ x \Downarrow x} \qquad \frac{t \Downarrow x \quad u \Downarrow y}{\text{Add}\ t\ u \Downarrow (x + y)}$$

Modularity in SOS

- An arithmetics language

$$a ::= Con\ n \mid Add\ a\ a$$

where n ranges over $\mathbb{Z}$.

$$\frac{}{\text{Con}\ x \Downarrow x} \qquad \frac{t \Downarrow x \quad u \Downarrow y}{\text{Add}\ t\ u \Downarrow (x + y)}$$

- An exceptions language

$$e ::= Throw \mid Catch\ e\ e$$

$$\frac{}{\text{Throw} \Downarrow \text{Nothing}} \qquad \frac{t \Downarrow \text{Just}\ x}{\text{Catch}\ t\ u \Downarrow \text{Just}\ x} \qquad \frac{t \Downarrow \text{Nothing} \quad u \Downarrow y}{\text{Catch}\ t\ u \Downarrow y}$$

A Combined Language

$$t ::= Con\ n \mid Add\ t\ t \mid Throw \mid Catch\ t\ t$$

A Combined Language

$$t ::= Con\ n \mid Add\ t\ t \mid Throw \mid Catch\ t\ t$$

$$\frac{}{\text{Con } x \Downarrow \text{Just } x} \qquad \frac{t \Downarrow x \quad u \Downarrow y}{\text{Add } t\ u \Downarrow \text{Just } (x + y)}$$

$$\frac{}{\text{Throw} \Downarrow \text{Nothing}} \qquad \frac{t \Downarrow \text{Just } x}{\text{Catch } t\ u \Downarrow \text{Just } x} \qquad \frac{t \Downarrow \text{Nothing} \quad u \Downarrow y}{\text{Catch } t\ u \Downarrow y}$$

$$\frac{t \Downarrow \text{Nothing}}{\text{Add } t\ u \Downarrow \text{Nothing}} \qquad \frac{u \Downarrow \text{Nothing}}{\text{Add } t\ u \Downarrow \text{Nothing}}$$

A Combined Language

$$t ::= Con\ n \mid Add\ t\ t \mid Throw \mid Catch\ t\ t$$

$$\frac{}{\text{Con } x \Downarrow \text{Just } x} \qquad \frac{t \Downarrow x \quad u \Downarrow y}{\text{Add } t\ u \Downarrow \text{Just } (x + y)}$$

$$\frac{}{\text{Throw} \Downarrow \text{Nothing}} \qquad \frac{t \Downarrow \text{Just } x}{\text{Catch } t\ u \Downarrow \text{Just } x} \qquad \frac{t \Downarrow \text{Nothing} \quad u \Downarrow y}{\text{Catch } t\ u \Downarrow y}$$

$$\frac{t \Downarrow \text{Nothing}}{\text{Add } t\ u \Downarrow \text{Nothing}} \qquad \frac{u \Downarrow \text{Nothing}}{\text{Add } t\ u \Downarrow \text{Nothing}}$$

- What is the relation between this semantics and the previous ones?

A Combined Language

$$t ::= Con\ n \mid Add\ t\ t \mid Throw \mid Catch\ t\ t$$

$$\frac{}{\text{Con } x \Downarrow \text{Just } x} \qquad \frac{t \Downarrow x \quad u \Downarrow y}{\text{Add } t\ u \Downarrow \text{Just } (x + y)}$$

$$\frac{}{\text{Throw} \Downarrow \text{Nothing}} \qquad \frac{t \Downarrow \text{Just } x}{\text{Catch } t\ u \Downarrow \text{Just } x} \qquad \frac{t \Downarrow \text{Nothing} \quad u \Downarrow y}{\text{Catch } t\ u \Downarrow y}$$

$$\frac{t \Downarrow \text{Nothing}}{\text{Add } t\ u \Downarrow \text{Nothing}} \qquad \frac{u \Downarrow \text{Nothing}}{\text{Add } t\ u \Downarrow \text{Nothing}}$$

- What is the relation between this semantics and the previous ones?
- Can we obtain rules that just propagate Nothing for free?

Functorial Operational Semantics

- Abstract formulation of operational semantics using category theory.
- Rules of SOS are expressed in terms of
  - The signature Σ (set of operations)
  - The observable behaviour B

  That is,

$$\mathcal{R}(\Sigma, B)$$

D. Turi. and G. Plotkin. Towards a mathematical operational semantics. *12th LICS Conf.*, 1997.

What if. . .

. . . we have some operations on rules $\mathcal{R}(\Sigma, B)$ such that:

What if...

...we have some operations on rules $\mathcal{R}(\Sigma, B)$ such that:

- We could join to languages with different signatures, but same behaviour.

$$join\colon (\mathcal{R}(\Sigma, B),\ (\mathcal{R}(\Sigma', B))) \to \mathcal{R}(\Sigma + \Sigma', B)$$

What if...

...we have some operations on rules $\mathcal{R}(\Sigma, B)$ such that:

- We could join to languages with different signatures, but same behaviour.

$$\mathit{join} : (\mathcal{R}(\Sigma, B),\ (\mathcal{R}(\Sigma', B))) \to \mathcal{R}(\Sigma + \Sigma', B)$$

- We could lift a rule to some effect F.

$$\mathit{lift} : \mathcal{R}(\Sigma, B) \to \mathcal{R}(\Sigma, F \cdot B)$$

What if. . .

. . . we have some operations on rules $\mathcal{R}(\Sigma, B)$ such that:

- We could join to languages with different signatures, but same behaviour.

$$\mathit{join} : (\mathcal{R}(\Sigma, B),\ (\mathcal{R}(\Sigma', B))) \rightarrow \mathcal{R}(\Sigma + \Sigma', B)$$

- We could lift a rule to some effect F.

$$\mathit{lift} : \mathcal{R}(\Sigma, B) \rightarrow \mathcal{R}(\Sigma, F \cdot B)$$

- We could construct rules with behaviour $F \cdot B$ that are well-defined for any B.

$$\rho_T : \forall B.\mathcal{R}(\Sigma, F \cdot B)$$

Then we could…

…answer the previous questions.
Semantics of arithmetics:

$$\rho_A : \mathcal{R}(\Sigma_A, K_{\mathbb{Z}})$$

Semantics of exceptions:

$$\rho_T : \forall B.\mathcal{R}(\Sigma_E, \text{Maybe} \cdot B)$$

with Maybe $X = 1 + X$

Then we could. . .

. . . answer the previous questions.
Semantics of arithmetics:

$$\rho_A : \mathcal{R}(\Sigma_A, K_{\mathbb{Z}})$$

Semantics of exceptions:

$$\rho_\tau : \forall B.\mathcal{R}(\Sigma_E, \text{Maybe} \cdot B)$$

with Maybe $X = 1 + X$

$$\dfrac{\dfrac{\rho_A : \mathcal{R}(\Sigma_A, K_{\mathbb{Z}})}{\mathit{lift}(\rho_A) : \mathcal{R}(\Sigma_A, \text{Maybe} \cdot K_{\mathbb{Z}})} \qquad \dfrac{\rho_\tau : \forall B.\mathcal{R}(\Sigma_E, \text{Maybe} \cdot B)}{\rho_{\tau_K} : \mathcal{R}(\Sigma_E, \text{Maybe} \cdot K_{\mathbb{Z}})}}{\mathit{join}(\rho_A, \rho_{\tau_K}) : \mathcal{R}(\Sigma_A + \Sigma_E, \text{Maybe} \cdot K_{\mathbb{Z}})}$$

Abstract Operational Rules

Our rules $\mathcal{R}(\Sigma, B)$, are actually *abstract operational rules*, natural transformations

$$\rho\colon \Sigma \cdot (Id \times B) \rightarrow B \cdot T_\Sigma$$

where

- T_Σ is the free monad on the signature Σ. ($T_\Sigma X$ is the set of terms with variables from X.)

$$\rho_X : \Sigma \cdot (X \times BX) \rightarrow (B \cdot T_\Sigma)X$$

Abstract Operational Rules

Our rules $\mathcal{R}(\Sigma, B)$, are actually *abstract operational rules*, natural transformations

$$\rho\colon \Sigma \cdot (Id \times B) \to B \cdot T_\Sigma$$

where

- T_Σ is the free monad on the signature Σ. ($T_\Sigma X$ is the set of terms with variables from X.)

$$\rho_X : \Sigma \cdot (X \times BX) \to (B \cdot T_\Sigma)X$$

Example: One rule for a binary sequence operator

$$(;)\colon X \times X \to X$$

$$\frac{t \xrightarrow{a} t'}{t;u \xrightarrow{a} t';u} \quad \Rightarrow \quad \begin{array}{c} ((X \times BX) \times (X \times BX)) \to (B \cdot T_\Sigma)X \\ (;)((t, \langle a, t' \rangle) \times (u, \_)) \to \langle a, t'; u \rangle \end{array}$$

Joining Rules

join puts together two languages with different signatures, but same behaviour.

$$\frac{\rho : \Sigma \cdot (Id \times B) \rightarrow B \cdot T_{\Sigma} \qquad \rho' : \Sigma' \cdot (Id \times B) \rightarrow B \cdot T_{\Sigma'}}{\begin{array}{lll} join\ (\rho, \rho') & : & (\Sigma + \Sigma') \cdot (Id \times B) \\ & = & \quad \{ \text{ Coproduct of Functors } \} \\ & & \Sigma \cdot (Id \times B) + \Sigma' \cdot (Id \times B) \\ & \rightarrow & \quad \{ \ \rho + \rho' \ \} \\ & & B \cdot T_{\Sigma} + B \cdot T_{\Sigma'} \\ & \rightarrowtail & \quad \{ \ [Binl + Binr] \ \} \\ & & B \cdot (T_{\Sigma} + T_{\Sigma'}) \\ & \rightarrowtail & \quad \{ \ \text{B[fold (inl, inr.inl),fold (inl,inr.inr)]} \ \} \\ & & B \cdot (T_{\Sigma + \Sigma'}) \end{array}}$$

Lifting Rules

lift lifts a rule with behaviour B to a behaviour $F \cdot B$.

- For F strong and a distributivity law $\Sigma \cdot F \to F \cdot \Sigma$

$$\frac{\rho \quad : \quad \Sigma \cdot (Id \times B) \to B \cdot T_\Sigma}{\begin{array}{lll} \mathit{lift}_F\ \rho & : & \Sigma \cdot (Id \times F \cdot B) \\ & \to & \quad \{\ \text{strength of } F\ \} \\ & & \Sigma \cdot F \cdot (Id \times B) \\ & \to & \quad \{\ \text{distributivity law}\ \} \\ & & F \cdot \Sigma \cdot (Id \times B) \\ & \to & \quad \{\ F\rho\ \} \\ & & F \cdot B \cdot T_\Sigma \end{array}}$$

Lifting Rules

lift lifts a rule with behaviour B to a behaviour $F \cdot B$.

- For F strong and a distributivity law $\Sigma \cdot F \rightarrow F \cdot \Sigma$

$$\frac{\rho \quad : \quad \Sigma \cdot (Id \times B) \rightarrow B \cdot T_\Sigma}{\begin{array}{lll} lift_F\ \rho & : & \Sigma \cdot (Id \times F \cdot B) \\ & \rightarrow & \quad \{\ \text{strength of } F\ \} \\ & & \Sigma \cdot F \cdot (Id \times B) \\ & \rightarrow & \quad \{\ \text{distributivity law}\ \} \\ & & F \cdot \Sigma \cdot (Id \times B) \\ & \rightarrow & \quad \{\ F \rho\ \} \\ & & F \cdot B \cdot T_\Sigma \end{array}}$$

- if F is *applicative* and Σ *traversable*, we obtain the strength and distributivity law for free.
- For simple signatures and all monadic effects, we get “propagation rules” for free.

Rule Transformers

- A *rule transformer* is a mapping from a behaviour B to a rule $\rho_\tau : \Sigma \cdot (Id \times F \cdot B) \to F \cdot B \cdot T_\Sigma$.

Rule Transformers

- A *rule transformer* is a mapping from a behaviour B to a rule $\rho_\tau : \Sigma \cdot (Id \times F \cdot B) \to F \cdot B \cdot T_\Sigma$.
- They can be generated from a *transformer germ*: a natural transformation $\tau : \Sigma \cdot F \to F$.

$$\frac{\tau : \Sigma \cdot F \to F}{\begin{array}{lll} \rho_\tau & : & \Sigma \cdot (Id \times F \cdot B) \\ & \to & \{\ \Sigma\pi_2\ \} \\ & & \Sigma \cdot F \cdot B \\ & \to & \{\ \tau_B\ \} \\ & & F \cdot B \\ & \to & \{\ (F \cdot B)\eta\ \} \\ & & F \cdot B \cdot T_\Sigma \end{array}}$$

Lifting T to D-coalgebras

Functorial operational semantics are a distributivity law

$$\lambda \colon T \cdot D \to D \cdot T$$

between

- a monad T (corresponding to syntax)
- a comonad D (corresponding to behaviours)

Equivalently, a lifting $\tilde{T}$ of T to the D-coalgebras:
For all $k \colon X \to DX$,

$$\tilde{T}(k) \colon TX \to D(TX)$$

To execute a program (a closed term $T\emptyset$) we unfold
$\tilde{T}(e) \colon T\emptyset \to D(T\emptyset)$, where $e \colon \emptyset \to D\emptyset$.

Summary

- We can easily reason about operational semantics by working in the abstract (category-theoretical) setting of functorial operational semantics.
- We can build complex semantics out of simpler building blocks, using operations on abstract operational rules (but with some limitations.)

- Future Work
  - Broaden the class of languages that we can represent (variable binding).
  - Construct more powerful operations to combine two languages (instead of *transforming* one.)

Thanks for listening

Haskell code will be available for downloading at
`http://www.cs.nott.ac.uk/~mjj/`