### Towards Operations on Operational Semantics

Mauro Jaskelioff mjj@cs.nott.ac.uk

School of Computer Science & IT

The University of Nottingham

22<sup>nd</sup> British Colloquium for Theoretical Computer Science

\* ロ ト \* 母 ト \* 臣 ト \* 臣

### The Context

◆□▶ ◆□▶ ◆臣▶ ◆臣▶ ○臣

- 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  $\mathbb{Z}$ .

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



# Modularity in SOS

An arithmetics language

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

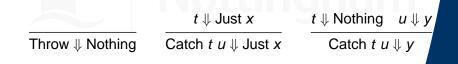
where *n* ranges over  $\mathbb{Z}$ .

Con  $x \Downarrow x$ 

 $\frac{t \Downarrow x \quad u \Downarrow y}{\operatorname{Add} t u \Downarrow (x+y)}$ 

An exceptions language

e ::= Throw | Catch e e



t ::= Con n | Add t t | Throw | Catch t t



t ::= Con n | Add t t | Throw | Catch t t $t \Downarrow x \quad u \Downarrow y$ Add  $t \ u \Downarrow \text{Just} (x + y)$ Con  $x \Downarrow Just x$  $t \Downarrow \text{Nothing} \quad u \Downarrow y$  $t \Downarrow Just x$ Throw U Nothing Catch *t*  $u \Downarrow y$ Catch *t*  $u \Downarrow$  Just *x*  $t \Downarrow Nothing$  $u \Downarrow \text{Nothing}$ Add *t*  $u \Downarrow$  Nothing Add t  $u \Downarrow$  Nothing

 $t ::= Con n \mid Add t t \mid Throw \mid Catch t t$  $t \Downarrow x \quad u \Downarrow y$ Add  $t u \Downarrow \text{Just} (x + y)$ Con  $x \Downarrow Just x$  $t \Downarrow \text{Nothing } u \Downarrow y$  $t \Downarrow Just x$ Throw ↓ Nothing Catch *t*  $u \Downarrow$  Just *x* Catch t  $u \Downarrow y$  $t \Downarrow Nothing$  $u \Downarrow \text{Nothing}$ Add t  $u \Downarrow$  Nothing Add t  $u \Downarrow$  Nothing What is the relation between this semantics and the previous ones?

 $t ::= Con n \mid Add t t \mid Throw \mid Catch t t$  $t \Downarrow x \quad u \Downarrow y$ Add  $t u \Downarrow \text{Just} (x + y)$ Con  $x \Downarrow Just x$  $t \Downarrow \text{Nothing } u \Downarrow y$  $t \Downarrow \text{Just } x$ Throw U Nothing Catch *t*  $u \Downarrow$  Just *x* Catch t  $u \Downarrow y$  $t \Downarrow Nothing$  $u \Downarrow Nothing$ Add t  $u \Downarrow$  Nothing Add t  $u \Downarrow$  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  $\Sigma$  (set of operations)
  - The observable behaviour B

That is,

イロト イポト イヨト イヨト

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

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

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



- ... we have some operations on rules  $\mathcal{R}(\Sigma, B)$  such that:
  - We could join to languages with different signatures, but same behaviour.

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

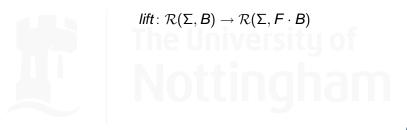


イロト イヨト イヨト イ

- ... we have some operations on rules  $\mathcal{R}(\Sigma, B)$  such that:
  - We could join to languages with different signatures, but same behaviour.

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

▶ We could lift a rule to some effect *F*.



- ... we have some operations on rules  $\mathcal{R}(\Sigma, B)$  such that:
  - We could join to languages with different signatures, but same behaviour.

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

▶ We could lift a rule to some effect *F*.

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

► We could construct rules with behaviour *F* · *B* that are well-defined for any *B*.

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

### Then we could...

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

$$\rho_{\mathcal{A}} \colon \mathcal{R}(\Sigma_{\mathcal{A}}, \mathcal{K}_{\mathbb{Z}})$$

Semantics of exceptions:

큰

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

# with Maybe X = 1 + X

#### Then we could...

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

$$\rho_{\mathcal{A}} \colon \mathcal{R}(\Sigma_{\mathcal{A}}, \mathcal{K}_{\mathbb{Z}})$$

Semantics of exceptions:

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

with Maybe X = 1 + X

 $\frac{\frac{\rho_{A} \colon \mathcal{R}(\Sigma_{A}, \mathcal{K}_{\mathbb{Z}})}{\text{lift}(\rho_{A}) \colon \mathcal{R}(\Sigma_{A}, \text{Maybe} \cdot \mathcal{K}_{\mathbb{Z}})} \qquad \frac{\rho_{\tau} \colon \forall B.\mathcal{R}(\Sigma_{E}, \text{Maybe} \cdot B)}{\rho_{\tau_{K}} \colon \mathcal{R}(\Sigma_{E}, \text{Maybe} \cdot \mathcal{K}_{\mathbb{Z}})}}{\text{join}(\rho_{A}, \rho_{\tau_{K}}) \colon \mathcal{R}(\Sigma_{A} + \Sigma_{E}, \text{Maybe} \cdot \mathcal{K}_{\mathbb{Z}})}$ 

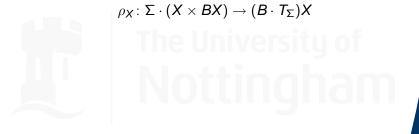
### Abstract Operational Rules

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

$$\rho \colon \Sigma \cdot (\mathit{Id} \times \mathit{B}) \to \mathit{B} \cdot \mathit{T}_{\Sigma}$$

where

*T*<sub>Σ</sub> is the free monad on the signature Σ. (*T*<sub>Σ</sub>*X* is the set of terms with variables from *X*.)



### Abstract Operational Rules

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

$$\rho \colon \Sigma \cdot (\mathit{Id} \times \mathit{B}) \to \mathit{B} \cdot \mathit{T}_{\Sigma}$$

where

 $\frac{t \xrightarrow{a} t'}{t: u \xrightarrow{a} t': u}$ 

*T*<sub>Σ</sub> is the free monad on the signature Σ. (*T*<sub>Σ</sub>*X* is the set of terms with variables from *X*.)

 $\rho_{\mathbf{X}}: \Sigma \cdot (\mathbf{X} \times \mathbf{B}\mathbf{X}) \to (\mathbf{B} \cdot \mathbf{T}_{\Sigma})\mathbf{X}$ 

Example: One rule for a binary sequence operator

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

 $((X \times BX) \times (X \times BX)) \to (B \cdot T_{\Sigma})X$ (;)((t, (a, t')) × (u, \_))  $\to$  (a, t'; u)

### **Joining Rules**

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

$$\begin{array}{rcl} \rho \colon \Sigma \cdot (\mathit{Id} \times \mathit{B}) \to \mathit{B} \cdot \mathit{T}_{\Sigma} & \rho' \colon \Sigma' \cdot (\mathit{Id} \times \mathit{B}) \to \mathit{B} \cdot \mathit{T}_{\Sigma'} \\ \hline \textit{join} (\rho, \rho') & \vdots & (\Sigma + \Sigma') \cdot (\mathit{Id} \times \mathit{B}) \\ & = & \{ \text{ Coproduct of Functors } \} \\ & \Sigma \cdot (\mathit{Id} \times \mathit{B}) + \Sigma' \cdot (\mathit{Id} \times \mathit{B}) \\ & \to & \{ \rho + \rho' \} \\ & & B \cdot \mathit{T}_{\Sigma} + \mathit{B} \cdot \mathit{T}_{\Sigma'} \\ & \to & \{ [\mathit{Binl} + \mathit{Binr}] \} \\ & & B \cdot (\mathit{T}_{\Sigma} + \mathit{T}_{\Sigma'}) \\ & \to & \{ B[\textit{fold (inl, inr.inl),\textit{fold (inl, inr.inr)}] } \} \\ & & B \cdot (\mathit{T}_{\Sigma + \Sigma'}) \end{array}$$

# Lifting Rules

큰

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

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

$$\rho : \Sigma \cdot (Id \times B) \rightarrow B \cdot I_{\Sigma}$$

$$\lim_{H \to \infty} Iift_{F} \rho : \Sigma \cdot (Id \times F \cdot B)$$

$$\rightarrow \qquad \{ \text{ strength of } F \}$$

$$\Sigma \cdot F \cdot (Id \times B)$$

$$\rightarrow \qquad \{ \text{ distributivity law } \}$$

$$F \cdot \Sigma \cdot (Id \times B)$$

$$\rightarrow \qquad \{ F\rho \}$$

$$F \cdot B \cdot T_{\Sigma}$$

$$\rho$$
 :  $\Sigma \cdot (Id \times B) \rightarrow B \cdot T_{\Sigma}$ 

# Lifting Rules

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

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

$$\rho$$
 :  $\Sigma \cdot (Id \times B) \rightarrow B \cdot T_{\Sigma}$ 

| $lift_F \rho$ | :             | $\Sigma \cdot (\mathit{Id} 	imes \mathit{F} \cdot \mathit{B})$ |
|---------------|---------------|--|
|               | $\rightarrow$ | { strength of F }  |
|               |               | $\Sigma \cdot F \cdot (\mathit{Id} \times \mathit{B})$         |
|               | $\rightarrow$ | { distributivity law }   |
|               |               | $F \cdot \Sigma \cdot (Id 	imes B)$                            |
|               | $\rightarrow$ | $\{F\rho\}$  |
|               |               | $F \cdot B \cdot T_{\Sigma}$                                   |

- If F is applicative and ∑ 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 ρ<sub>τ</sub>: Σ · (Id × F · B) → F · B · T<sub>Σ</sub>.



### **Rule Transformers**

- A rule transformer is a mapping from a behaviour B to a rule ρ<sub>τ</sub>: Σ · (Id × F · B) → F · B · T<sub>Σ</sub>.
- They can be generated from a *transformer germ*: a natural transformation *τ* : Σ · *F* → *F*.

$$\begin{array}{cccc} & \tau \colon \Sigma \cdot F \to F \\ \hline \rho_{\tau} & \vdots & \Sigma \cdot (Id \times F \cdot B) \\ & \to & \{ \Sigma \pi_2 \} \\ & \Sigma \cdot F \cdot B \\ & \to & \{ \tau_B \} \\ & F \cdot B \\ & \to & \{ (F \cdot B)\eta \} \\ & F \cdot B \cdot T_{\Sigma} \end{array}$$

# Lifting T to D-coalgebras

Functorial operational semantics are a distributivity law

 $\lambda : \mathbf{T} \cdot \mathbf{D} \rightarrow \mathbf{D} \cdot \mathbf{T}$ 

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  $k: X \to DX$ ,

$$\tilde{T}(k)$$
:  $TX \rightarrow D(TX)$ 

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

## 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

(日)

- 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/

