Abstract Transformation of Small-step Higher-order SOS to Big-step

Pouya Partow

A work with Sergey Goncharov and Stelios Tsampas

Chair of Theoretical Computer Science

Friedrich-Alexander-Universität Erlangen-Nürnberg

Oberseminar Theoretische Informatik, December 2024

Overview

1. Introduction: Operational semantics, small-step vs big-step, and abstract HO-GSOS.
2. Constraints: Strongly Separated HO-GSOS
3. Big-step SOS
4. A sketch of the proof
5. Examples

And an encoding in Haskell!

Operational Semantics

Operational Semantics: A set of inference rules, describing reduction.

Also called specification here.

Example: *xCL*

$$t, s ::= S \mid K \mid I \mid S'(t) \mid K'(t) \mid S''(t, s) \mid t \cdot s.$$

$$\frac{}{S \xrightarrow{t} S'(t)} \qquad \frac{}{S'(t) \xrightarrow{s} S''(t, s)} \qquad \frac{}{S''(t, s) \xrightarrow{r} (t \cdot r) \cdot (s \cdot r)}$$

$$\frac{}{K \xrightarrow{t} K'(t)} \qquad \frac{}{K'(t) \xrightarrow{s} t} \qquad \frac{}{I \xrightarrow{t} t} \qquad \frac{t \to t'}{t \cdot s \to t' \cdot s} \qquad \frac{t \xrightarrow{s} t'}{t \cdot s \to t'}$$

Why *xCL*? Syntactic simplicity (we'll see...), while being highly expressive.

Small-step vs Big-step

Operational semantics are in different flavors: small-step vs big-step, call-by-name vs call-by-value, ...

Small-step is fine-grained (like the given example).

Big-step does all at once.

Like the following *xCL* specification (all $\cdot$ are omitted):

$$\frac{}{v \Downarrow v} \qquad \frac{s \Downarrow I \quad t \Downarrow v}{st \Downarrow v} \qquad \frac{s \Downarrow K \quad K'(t) \Downarrow v}{st \Downarrow v} \qquad \frac{s \Downarrow S \quad S'(t) \Downarrow v}{st \Downarrow v}$$

$$\frac{s \Downarrow K'(r) \quad r \Downarrow v}{st \Downarrow v} \qquad \frac{s \Downarrow S'(r) \quad S''(r,t) \Downarrow v}{st \Downarrow v} \qquad \frac{s \Downarrow S''(r,q) \quad (rt)(qt) \Downarrow v}{st \Downarrow v}$$

Stating the Problem

Definition (Equivalence)

Two specifications (small-step and big-step) for one language are equivalent iff for all the terms t and v:

$$t \to^\star v \wedge v \downarrow \iff t \Downarrow v$$

Usually, small-step is designed first, and big-step is derived.

We want to generalize this as much as possible.

One gets equivalent big-step for free from a designed small-step.

First Step For Generalization (Rule Formats)

- To generalize, we need rule formats (reasoning on them).

  Like this one (HO-GSOS[1]):

$$\frac{(x_j \to y_j)_{j \in W} \quad (x_i \xrightarrow{z} y_i^z)_{i \in \{1,\ldots,m\} \setminus W, z \in \{x, x_1, \ldots, x_m\}}}{f(x_1, \ldots, x_m) \xrightarrow{x} t}$$

- We create the transformation for a specific rule format.
- Working with rule formats may be messy, though!
- Category theory helps!

[1] Goncharov, et al., "Towards Higher-Order Mathematical Operational Semantics", 2023

Next Step For Generalization (Abstract HO-GSOS)

HO-GSOS is represented by natural transformations of the following form:

$$\rho_{X,Y} : \Sigma(X \times B(X,Y)) \to B(X, \Sigma^*(X+Y))$$

Example

xCL

$$\Sigma X = \amalg_{f \in \mathcal{O}_p} X^{\mathrm{ar}(f)(\sigma)} \qquad B(X,Y) = Y^X + Y$$

$$\rho(S'(t)) = \; r \mapsto S''(t,r)$$

$$\rho(S''(t,s)) = \; r \mapsto (t \cdot r) \cdot (s \cdot r)$$

$$\rho((t,f) \cdot (s,\_)) = \; f(s) \quad \& \quad \rho((t,t') \cdot (s,\_)) = \; t'$$

And the uniquely derived operational model: $\gamma : \mu\Sigma \to B(\mu\Sigma, \mu\Sigma)$

xCL in Haskell: Syntax and Behaviour

```haskell
data XCL' x y = S | K | I | S' x | K' x
| S'' x x | Comp x y  --XCL, as a bifunctor.

newtype Mrg s x = Mrg (s x x) --Merge functor.

type XCL = Mrg XCL' --XCL as a functor.

data Free s x = Res x
| Cont (s (Free s x)) --{runCont :: s (Free s x)}

type Initial s = Free s Void --LFP of Sigma*

data Beh x y = Eval (x -> y) | Red y --Behaviour
```

xCL in Haskell: HO-GSOS

```haskell
class (Bifunctor s,MixFunctor b)=>HOGSOS s b where--HoGSOS law.
rho :: s (x , b x y) x -> b x (Free (Mrg s) (Either x y))
gamma :: Initial (Mrg s) ->
b (Initial (Mrg s)) (Initial (Mrg s)) --Operational model.
gamma (Cont (Mrg t)) = mx_second (>>= nabla)
$ rho $ first (id &&& gamma) t  --Derived from the diagram.
where nabla = either id id

instance HOGSOS XCL' Beh where
rho :: XCL' (x, Beh x y) x -> Beh x (Free XCL (Either x y))
rho K = Eval $ sigOp . K' . Res . Left
rho (K' (s , _ )) = Eval $ const $ Res $ Left s
rho (Comp (_ , Red s ) u ) = Red $ sigOp $ Comp (Res $ Right s)
rho (Comp (_ , Eval f) u )  = Red $ Res (Right $ f u)
```

About Abstract HO-GSOS

Not directly related (yet), but worth mentioning that:

- Bisimilarity is a congruence. relation[2]
- Howe's method. [3]
- Logical Relations [4][5]
- Operational methods for Call-by-push-value [6]

[2] Goncharov, et al., "Towards Higher-Order Mathematical Operational Semantics", 2023
[3] Urbat, et al., "Weak similarity in higher-order mathematical semantics", 2023
[4] Goncharov, et al., "Logical Predicates in Higher-Order Operational Semantics", 2024
[5] Goncharov, et al., "Bialgebraic Reasoning on Higher-Order Program Equivalence", 2024
[6] Goncharov, et al., "Abstract Operational Methods for Call-by-Push-Value", 2025

Separated HO-GSOS

A modified version of abstract HO-GSOS.

An effect monad $\boldsymbol{T}$ (ω-continuous[7], like $\mathcal{P}$ and $\mathcal{S}$) is installed on the setting.

Along with some constraints (on Σ,B, and ρ). Don't mind the future blues!

$$\Sigma' = \Sigma^{\mathsf{v}}\Pi_2 + \Sigma^{\mathsf{c}} \qquad \textit{Value formers and computation formers}$$

$$t \to^{\star} v \wedge v\downarrow \iff t \Downarrow v$$

Example

xCL

$$\mathcal{O}p = \{I, K, K', S, S', S''\} + \{\cdot\}$$

$$\Sigma^{\mathsf{v}}(X) = \amalg_{\mathsf{f}\in\mathcal{O}p\setminus\{\cdot\}} X^{\mathsf{ar}(\mathsf{f})} \qquad \Sigma^{\mathsf{c}}(X,Y) = X \times Y$$

[7] Goncharov, et al., "Unguarded recursion on coinductive resumptions", 2018

Separated HO-GSOS (2)

$$B(X,Y) = TD(X,Y) + TY$$

$$\rho = TD(\mathrm{id}, \Sigma^\star \mathrm{inl}) \cdot \rho^{\mathsf{v}} + \rho^{\mathsf{c}}, \quad \textit{Where}:$$

$$\rho^{\mathsf{v}}_X \colon \Sigma_{\mathsf{v}} X \to TD(X, \Sigma^\star X)$$

$$\rho^{\mathsf{c}}_{X,Y} \colon \Sigma_{\mathsf{c}}(X \times B(X,Y), X) \to T\Sigma^\star(X+Y)$$

Example

xCL

$$\mathbf{T} = \mathbf{Id} \quad \& \quad D(X,Y) = Y^X$$

$$\rho^{\mathsf{v}}(S'(t)) = \; r \mapsto S''(t,r)$$

$$\rho^{\mathsf{v}}(S''(t,s)) = \; r \mapsto (t \cdot r) \cdot (s \cdot r)$$

$$\rho^{\mathsf{c}}((t,f) \cdot s) = \; f(s) \quad \& \quad \rho^{\mathsf{v}}((t,t') \cdot s) = \; t'$$

xCL in Haskell: Separated HO-GSOS

```haskell
data SepSig sv sc x y = SigV (sv y) | SigC (sc x y)
data SepBeh d x y = BehV (d x y) | BehC y

class (MixFunctor d, Functor sv, Bifunctor sc) =>
SepHOGSOS sv sc d where
rhoV :: sv x -> d x (Free (SepSig sv sc) x)
rhoC :: sc (x , SepBeh d x y) x ->
Free (SepSig sv sc) (Either x y)

instance (SepHOGSOS sv sc d) =>
HOGSOS (SepSig' sv sc) (SepBeh d) where
rho :: SepSig' sv sc (x, SepBeh d x y) x ->
SepBeh d x (Free (SepSig sv sc) (Either x y))
rho (SigV v) = BehV $ mx_second (fmap Left) (rhoV v)
rho (SigC c) = BehC $ rhoC c
```

Strong Separation

This is our last last condition. Abstract form is a bit complex!

Definition (Strong Separation)

Every separable HO-GSOS ρ is strongly separated iff rules defining ρ^{c} are of the following form if $I \neq \emptyset$:

$$\frac{\{p_i \to p_i'\}_{i \in I} \qquad \{p_j \xrightarrow{s} p_j'\}_{s \in \{p_1,\ldots,p_n,q_1,\ldots,q_m\}, j \in J}}{f(p_1,\ldots,p_n,q_1,\ldots,q_m) \to f(p_1',\ldots,p_n',q_1,\ldots,q_m)}$$

$$J = W \setminus I \quad \& \quad \forall j \in J : p_j = p_j'$$

p_is are called the strict variables.

$$\rho^{c}_{X,Y} : \Sigma_c(X \times B(X,Y), X) \to T\Sigma^{\star}(X + Y)$$

Examples

For example, *xCL* satisfies this.

$$\frac{}{S \xrightarrow{t} S'(t)} \qquad \frac{}{S'(t) \xrightarrow{s} S''(t,s)} \qquad \frac{}{S''(t,s) \xrightarrow{r} (t \cdot r) \cdot (s \cdot r)}$$

$$\frac{}{K \xrightarrow{t} K'(t)} \qquad \frac{}{K'(t) \xrightarrow{s} t} \qquad \frac{}{I \xrightarrow{t} t} \qquad \frac{t \to t'}{t \cdot s \to t' \cdot s} \qquad \frac{t \xrightarrow{s} t'}{t \cdot s \to t'}$$

But the language

$$t ::= \Omega \mid f(t) \mid g(t),$$

with the following specification doesn't satisfy strong separation:

$$\frac{}{g(x) \xrightarrow{y} f(y)} \qquad \frac{}{\Omega \to \Omega} \qquad \frac{x \to y}{f(x) \to g(y)} \qquad \frac{x \xrightarrow{x} y}{f(x) \to x}$$

Big-step SOS

We need a rule format as the host for big-step specifications.

$$\frac{x_1 \Downarrow g_1(x_1^1, \ldots, x_{n_1}^1) \quad \ldots \quad x_k \Downarrow g_k(x_1^k, \ldots, x_{n_k}^k) \qquad t \Downarrow v}{f(x_1, \ldots, x_n) \Downarrow v}$$

Specifications in this format are encoded by the following natural-transformations:

$$\xi : \Sigma_c(\Sigma_v X, X) \to T(\Sigma^\star X).$$

Example

xCL $\quad \xi(I \cdot q) = q \qquad \xi(K \cdot q) = K'(q) \qquad \xi(S \cdot q) = S'(q)$

$\xi(K'(t) \cdot q) = t \qquad \xi(S'(t) \cdot q) = S''(t, q) \qquad \xi(S''(s, t) \cdot q) = (sq)(tq)$

A Sketch of the Proof of Equivalence

We have a strongly separable ρ, and we define the following two:

$$t \to^{\star} v \wedge v \downarrow \iff t \Downarrow v$$

Big-step operational model (derived from ξ derived from ρ):

$$\zeta : \Sigma_c(\mu\Sigma, \mu\Sigma) \to T(\Sigma^v(\mu\Sigma))$$

$$t \to^{\star} v \wedge v \downarrow \iff t \Downarrow v$$

Multi-step transitions (derived from the γ derived from ρ):

$$\beta : \Sigma_c(\mu\Sigma, \mu\Sigma) \to T(\Sigma^v(\mu\Sigma))$$

It has been proved that ζ and β coincide.

Examples: Ωfg

$$\frac{}{g(x) \overset{y}{\to} f(y)} \qquad \frac{}{\Omega \to \Omega} \qquad \frac{x \to y}{f(x) \to g(y)} \qquad \frac{x \overset{x}{\to} y}{f(x) \to x}$$

$$\frac{}{v \Downarrow v} \qquad \frac{x \Downarrow g(y) \qquad g(y) \Downarrow v}{f(x) \Downarrow v}$$

And it does not work (as expected)!

$$f(f(g(\Omega))) \to g(g(\Omega)),$$

but

$$f(f(g(\Omega))) \Downarrow g(\Omega)$$

Examples: Non-deterministic *xCL* in Haskell

```
instance SepHOGSOST VndxCL CndxCL (->) [] where
rhoVT :: VndxCL x -> [ x -> Free (SepSig VndxCL CndxCL) x ]
rhoVT NK = [sigOp . SigV . NK' . Res]
rhoVT (NK' t) = [const (Res t)]

rhoCT :: CndxCL (x , SepBehT [] (->) x y) x ->
[(Free (SepSig VndxCL CndxCL)) (Either x y)]
rhoCT (NComp (s , SepBehT (BehC s' : l)) t ) =
sigOp (SigC $ NComp (Res $ Right s') (Res $ Left t)) :
rhoCT (NComp (s , SepBehT l ) t)
rhoCT (NComp (s , SepBehT (BehV f  : l)) t ) =
Res (Right $ f t) : rhoCT (NComp (s , SepBehT l ) t)
rhoCT (NComp (s , SepBehT [] ) t ) = []

rhoCT (DSum s t) = [Res $ Left s, Res $ Left t]
```

Example: Call-by-value

$$\frac{t \to t'}{ts \to t's} \qquad \frac{t \xrightarrow{r} t' \quad s \to s'}{ts \to ts'} \qquad \frac{t \xrightarrow{s} t' \quad s \xrightarrow{r} s'}{ts \to t'}$$

The first rule is both

$$\frac{t \to t' \quad s \to s'}{t \cdot s \to t' \cdot s} \quad \textit{and} \quad \frac{t \to t' \quad s \xrightarrow{r} s'}{t \cdot s \to t' \cdot s} \;.$$

We can replace that with

$$\frac{t \to t' \quad s \to s'}{t \cdot s \to t' \cdot s'}$$

Example: Call-by-value (2)

And get the following equivalent big-step specification:

$$\frac{}{v \Downarrow v} \qquad \frac{s \Downarrow I \quad t \Downarrow v}{st \Downarrow v} \qquad \frac{s \Downarrow K \quad t \Downarrow w \quad K'(w) \Downarrow v}{st \Downarrow v}$$

$$\frac{s \Downarrow S \quad t \Downarrow w \quad S'(w) \Downarrow v}{st \Downarrow v} \qquad \frac{s \Downarrow K'(r) \quad t \Downarrow w \quad r \Downarrow v}{st \Downarrow v}$$

$$\frac{s \Downarrow S'(r) \quad t \Downarrow w \quad S''(r,w) \Downarrow v}{st \Downarrow v} \qquad \frac{s \Downarrow S''(r,q) \quad t \Downarrow w \quad (rw)(qw) \Downarrow v}{st \Downarrow v}$$

But it is not "the" of call-by value. For exmaple, we must have

$(I \cdot I) \cdot (I \cdot I) \to^* I \cdot (I \cdot I)$, but we have $(I \cdot I) \cdot (I \cdot I) \to I \cdot I$

Example: Call-by-Value (3)

Instead we can define the following

$$\frac{t \to t'}{t \cdot s \to t' \cdot s} \qquad \frac{s \xrightarrow{r} s'}{s \cdot t \to s \mathbin{\blacklozenge} t} \qquad \frac{s \to s'}{t \mathbin{\blacklozenge} s \to t \mathbin{\blacklozenge} s'}$$

$$\frac{t \xrightarrow{r} t'}{s \mathbin{\blacklozenge} t \to s \bullet t} \qquad \frac{t \to t'}{t \bullet s \to t' \bullet s} \qquad \frac{t \xrightarrow{r} t'}{t \bullet s \to t'}$$

And derive the following from it:

$$\frac{}{v \Downarrow v} \qquad \frac{s \Downarrow w \quad w \mathbin{\blacklozenge} t \Downarrow v}{st \Downarrow v} \qquad \frac{t \Downarrow w \quad s \bullet w \Downarrow v}{s \mathbin{\blacklozenge} t \Downarrow v}$$

$$\frac{s \Downarrow I \quad t \Downarrow v}{s \bullet t \Downarrow v} \qquad \frac{s \Downarrow K \quad K'(t) \Downarrow v}{s \bullet t \Downarrow v} \qquad \frac{s \Downarrow S \quad S'(t) \Downarrow v}{s \bullet t \Downarrow v}$$

$$\frac{s \Downarrow K'(r) \quad r \Downarrow v}{s \bullet t \Downarrow v} \qquad \frac{s \Downarrow S'(r) \quad S''(r,t) \Downarrow v}{s \bullet t \Downarrow v} \qquad \frac{s \Downarrow S''(r,q) \quad (rt)(qt) \Downarrow v}{s \bullet t \Downarrow v}$$

Typed *xCL* with Recursion and Control

We add the following operators (typed!):

$$\mathrm{fix}_\tau : (\tau \mapsto \tau) \mapsto \tau \qquad \mathrm{if}_\tau : \mathrm{bool}, \tau, \tau \to \tau \qquad \mathrm{true}, \mathrm{false} : \mathrm{bool}$$

And modify D (and B and Σ for types!):

$$D(X,Y)_{\tau_1 \mapsto \tau_2} = Y_{\tau_2}^{X_{\tau_1}} \qquad D(X,Y)_\tau = \begin{cases} \{\mathrm{true}, \mathrm{false}\} & \text{if } \tau = \mathrm{bool} \\ \{\} & \text{if } \tau \neq \mathrm{bool} \end{cases}$$

Typed *xCL* with Recursion and Control (2)

And we define the following small-step semantics:

$$\frac{}{\mathsf{fix}_\tau \xrightarrow{t} t \cdot (\mathsf{fix}_\tau \cdot t)} \qquad \frac{}{\mathsf{true} \downarrow_{\mathsf{true}}} \qquad \frac{}{\mathsf{false} \downarrow_{\mathsf{false}}}$$

$$\frac{\mathsf{if}_\tau(b, s, t) \to \mathsf{if}_\tau(b', s, t)}{b \to b'} \qquad \frac{\mathsf{if}_\tau(b, s, t) \to s}{b \downarrow_{\mathsf{true}}} \qquad \frac{\mathsf{if}_\tau(b, s, t) \to t}{b \downarrow_{\mathsf{false}}}$$

And the following big-step specification is derive from it:

$$\frac{p \Downarrow \mathsf{fix}_\tau \quad q \cdot (\mathsf{fix}_\tau \cdot q) \Downarrow v}{\mathsf{comp}_{\tau \mapsto \tau, \tau}(p, q) \Downarrow v} \qquad \frac{b \Downarrow \mathsf{true} \quad s \Downarrow v}{\mathsf{if}_\tau(b, s, t) \Downarrow v} \qquad \frac{b \Downarrow \mathsf{false} \quad t \Downarrow v}{\mathsf{if}_\tau(b, s, t) \Downarrow v}$$

Conclusion and Future Works

What we have new:

- An automatic way to derive big-step semantics from a large family of HO-GSOS specifications.
- A new big-step rule format.
- Multi-step transitions and operational model for big-step SOS.

How to improve it:

- λ-calculus!
- Formalization in Agda
- Different effects (statefulness, probability, non-determinis, ...)
- Considering other constraints.
- Finding necessary condition(s).

Thank you for your attention! :-)

Appendix

HO-GSOS for *xCL* in more detail:

Example

$$\Sigma X = \amalg_{\mathsf{f} \in \mathcal{O}_p} X^{\mathsf{ar}(\mathsf{f})(\sigma)} \qquad B(X,Y) = Y^X + Y$$

$$\rho(S'(t,\_)) = \ \mathsf{inl}(r \mapsto S''(t,r))$$

$$\rho(S''((t,\_),(s,\_))) = \ \mathsf{inl}(r \mapsto (t \cdot r) \cdot (s \cdot r))$$

$$\rho((t,\mathsf{inl}(f)) \cdot (s,\_)) = \ \mathsf{inr}(f(s)) \quad \& \quad \rho((t,\mathsf{inr}(t')) \cdot (s,\_)) = \ \mathsf{inr}(t')$$

ω-continuous monads:

ω-continuous monad

A strong monad $\mathbf{T}$ is ω-continuous iff the Kleisli category $\mathcal{C}_\mathbf{T}$ is enriched with ω-cpos and satisfies the following conditions:

- Strength (τ) is ω-continuous: $\tau(\mathrm{id} \times \amalg_i f_i) = \amalg_i \tau(\mathrm{id} \times f_i)$
- Copairing in $\mathcal{C}_\mathbf{T}$ is ω-continuous: $\left[\amalg_i f_i, \amalg_i g_i\right] = \amalg_i [f_i, g_i]$
- Bottom elements are preserved by strength and by postcomposition in $\mathcal{C}_\mathbf{T}$: $\tau(\mathrm{id} \times \bot) = \bot, f^\sharp \cdot \bot = \bot$.

"An ω-continuous monad is an Elgot monad, i.e. it supports an (Elgot) iteration operator that sends every $f : X \to T(Y + X)$ to $f^\dagger : X \to TY$, subject to several standard laws of iteration. Specifically, $f^\dagger = \mu g.\, [\eta, g]^\dagger \cdot f$."