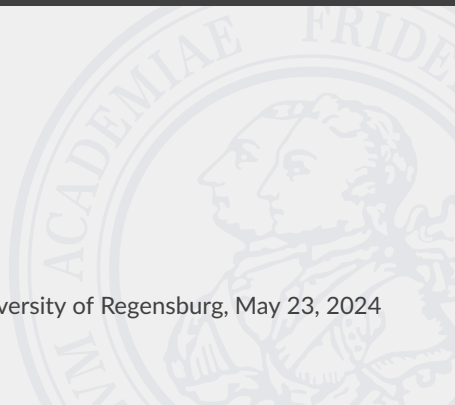


# Higher-Order Program Equivalence in the Abstract





Sergey Goncharov

FAU Erlangen-Nürnberg






Chair of Theoretical Computer Science, University of Regensburg, May 23, 2024



# HO GSOS Semantics Unravelling


-  Goncharov, Milius, Schröder, Tsampas, and Urbat, “Towards a Higher-Order Mathematical Operational Semantics”, POPL 2023
-  Urbat, Tsampas, Goncharov, Milius, and Schröder, “Weak Similarity in Higher-Order Mathematical Operational Semantics”, LICS 2023
-  Goncharov, Santamaria, Schröder, Tsampas, and Urbat, “Logical Predicates in Higher-Order Mathematical Operational Semantics”, FoSSaCS 2024
-  Goncharov, Milius, Tsampas, and Urbat, “Bialgebraic Reasoning on Higher-Order Program Equivalence”, LICS 2024

# HO GSOS Semantics Unravelling

-  Goncharov, Milius, Schröder, Tsampas, and Urbat, “Towards a Higher-Order Mathematical Operational Semantics”, POPL 2023
-  Urbat, Tsampas, Goncharov, Milius, and Schröder, “Weak Similarity in Higher-Order Mathematical Operational Semantics”, LICS 2023
-  Goncharov, Santamaria, Schröder, Tsampas, and Urbat, “Logical Predicates in Higher-Order Mathematical Operational Semantics”, FoSSaCS 2024
-  Goncharov, Milius, Tsampas, and Urbat, “Bialgebraic Reasoning on Higher-Order Program Equivalence”, LICS 2024 

# Higher-Order Operational Semantics

# Semantics First (!)

 Why care about semantics and about quality of semantics?



Implementation ( $\Leftarrow$  operational semantics)



Verification ( $\Leftarrow$  logical semantics)



Optimization ( $\Leftarrow$  denotational semantics)

But also

- Certified correctness
- Secure compilation
- Modelling and simulation
- Language design (Haskell, Scala, Coq, Agda,...)
- Transferring knowledge across domains

# Operational v.s. Denotational

- **Operational Semantics** (how programs behave?)


$$\begin{aligned} s(s(0)) + s(s(0)) &\rightarrow s(s(0) + s(s(0))) \\ &\rightarrow s(s(0 + s(s(0)))) \rightarrow s(s(s(s(0)))) \end{aligned}$$

- **Denotational Semantics** (what programs denote?)

$$\llbracket s(s(0)) + s(s(0)) \rrbracket = \llbracket s(s(0)) \rrbracket + \llbracket s(s(0)) \rrbracket = 2 + 2 = 4$$


# Semantics in Use


## Denotational:

 Compositional by design:

$$\llbracket p \rrbracket = \llbracket q \rrbracket \Rightarrow \llbracket C[p] \rrbracket = \llbracket C[q] \rrbracket$$


for any **program context**  $C$


 Mathematically rigorous and precise

 Ease to define: from hard to impossible

## Operational:

 Lightweight and easy to define even for complex languages

 Nonuniform and fragile

 Hard to reason about (because of lack of compositionality)

# Semantics in Use

## Denotational:

- 😊 Compositional by design:

$$\llbracket p \rrbracket = \llbracket q \rrbracket \Rightarrow \llbracket C[p] \rrbracket = \llbracket C[q] \rrbracket$$

for any **program context**  $C$

- 😊 Mathematically rigorous and precise
- ☹ Ease to define: from hard to impossible

## Operational:

- 😊 Lightweight and easy to define even for complex languages !
- ☹ Nonuniform and fragile
- ☹ Hard to reason about (because of lack of compositionality)



# Call-by-Name (Lazy) $\lambda$ -calculus

- Operational (small-step, call-by-name) semantics **rules**

$$\frac{}{(\lambda x. p)q \rightarrow p[q/x]} \qquad \frac{p \rightarrow p'}{pq \rightarrow p'q}$$

- ▶ involved terms (=programs) are closed (!)

# Call-by-Name (Lazy) $\lambda$ -calculus

- Operational (small-step, call-by-name) semantics **rules**

$$\frac{}{(\lambda x. p)q \rightarrow p[q/x]} \qquad \frac{p \rightarrow p'}{pq \rightarrow p'q}$$

- ▶ involved terms (=programs) are closed (!)
- **Values**  $v$  are irreducible closed terms:  $v \nrightarrow t$  for any  $t$ 
  - ▶ **Example:**  $I := \lambda x. x$  – value,  $\Omega := (\lambda x. xx)(\lambda x. xx)$  – non-value

# Call-by-Name (Lazy) $\lambda$ -calculus

- Operational (small-step, call-by-name) semantics **rules**

$$\frac{}{(\lambda x. p)q \rightarrow p[q/x]} \qquad \frac{p \rightarrow p'}{pq \rightarrow p'q}$$

- ▶ involved terms (=programs) are closed (!)
- Values**  $v$  are irreducible closed terms:  $v \nrightarrow t$  for any  $t$ 
  - ▶ **Example:**  $I := \lambda x. x$  – value,  $\Omega := (\lambda x. xx)(\lambda x. xx)$  – non-value
- Termination:**  $t \Downarrow = "t \rightarrow^* v$  for some value  $v"$

# Call-by-Name (Lazy) $\lambda$ -calculus

- Operational (small-step, call-by-name) semantics **rules**

$$\frac{}{(\lambda x. p)q \rightarrow p[q/x]} \qquad \frac{p \rightarrow p'}{pq \rightarrow p'q}$$

- ▶ involved terms (=programs) are closed (!)
- Values**  $v$  are irreducible closed terms:  $v \nrightarrow t$  for any  $t$ 
  - ▶ **Example:**  $I := \lambda x. x$  – value,  $\Omega := (\lambda x. xx)(\lambda x. xx)$  – non-value
- Termination:**  $t \downarrow =$  “ $t \rightarrow^* v$  for some value  $v$ ”
- Contextual preorder:**  $s \lesssim_{ctx} t$  if  $C[s] \downarrow$  implies  $C[t] \downarrow$  for **all** contexts  $C$ 
  - ▶ **Example:**  $f \lesssim_{ctx} \lambda x. fx$  (how to prove it?)

# Call-by-Name (Lazy) $\lambda$ -calculus

- Operational (small-step, call-by-name) semantics **rules**

$$\frac{}{(\lambda x. p)q \rightarrow p[q/x]} \qquad \frac{p \rightarrow p'}{pq \rightarrow p'q}$$

- ▶ involved terms (=programs) are closed (!)
- Values**  $v$  are irreducible closed terms:  $v \dashv\vdash t$  for any  $t$ 
  - ▶ **Example:**  $I := \lambda x. x$  – value,  $\Omega := (\lambda x. xx)(\lambda x. xx)$  – non-value
- Termination:**  $t \downarrow =$  “ $t \rightarrow^* v$  for some value  $v$ ”
- Contextual preorder:**  $s \lesssim_{ctx} t$  if  $C[s] \downarrow$  implies  $C[t] \downarrow$  for **all** contexts  $C$ 
  - ▶ **Example:**  $f \lesssim_{ctx} \lambda x. fx$  (how to prove it?)
- Contextual equivalence:**  $s \simeq_{ctx} t$  if  $s \lesssim_{ctx} t$  and  $t \lesssim_{ctx} s$ 
  - ▶ **Example:**  $f \not\approx_{ctx} \lambda x. fx$ , because  $\lambda x. \Omega x \not\lesssim_{ctx} \Omega$

# Scaling Up: Higher-Order Operational Semantics

Aspects that add/vary:

- Evaluation strategy (**call-by-name** v.s. **call-by-value**)
- **Types**  $\rightsquigarrow$  other notions of contextual equivalence
- Computational **effects** (non-determinism, exceptions, store, . . .)
- Other language features (recursive types, type constructors, polymorphism)

**Operational Methods** are complicated, fragile and boilerplate

❓ Can we build a mathematical theory of higher-order operational semantics, abstracting and unifying these methods?

# Higher-Order Abstract GSOS

# A Bit of Category Theory

From the programming perspective:

- **(Endo-)functor** is a type constructor, e.g.  $FX = X \times X$
- **Natural transformation**  $\alpha: F \rightarrow G$  is a polymorphic function  $\alpha_X: FX \rightarrow GX$ , e.g.  $swap: X \times X \rightarrow X \times X$
- **Algebra** is a map  $a: FX \rightarrow X$ , e.g. the free algebra of  $\Sigma$ -terms  $\Sigma(\Sigma^*X) \rightarrow \Sigma^*X$  over variables  $X$
- **Coalgebra** is a map  $c: X \rightarrow FX$ , e.g. a **labelled transition system**  $X \rightarrow \mathcal{P}(A \times X)$



# First-Order Abstract GSOS

Turi and Plotkin's abstraction of GSOS rule format<sup>1</sup>:

- **Signature endo-functor**  $\Sigma$
- **Behaviour endo-functor**  $B$
- **GSOS law** – natural transformation  $\rho_X: \Sigma(X \times BX) \rightarrow B(\Sigma^*X)$

**Example (Process Algebra):**

- $\Sigma = \{ | /2, \emptyset /0 \} \cup \{ a. (-) /1 \mid a \in A \}$
- $BX = \mathcal{P}(A \times X)$
- GSOS law encodes rules like:

$$\frac{p \xrightarrow{a} p'}{p \mid q \xrightarrow{a} p' \mid q}$$

---

<sup>1</sup>Turi and Plotkin, "Towards a Mathematical Operational Semantics".

# First-Order Abstract GSOS

Turi and Plotkin's abstraction of GSOS rule format<sup>1</sup>:

- Signature endo-functor  $\Sigma$
- Behaviour endo-functor  $B$
- GSOS law – natural transformation  $\rho_X: \Sigma(X \times BX) \rightarrow B(\Sigma^*X)$

Example (Process Algebra):

- $\Sigma = \{ | / 2, \emptyset / 0 \} \cup \{ a. (-) / 1 \mid a \in A \}$
- $BX = \mathcal{P}(A \times X)$
- GSOS law encodes rules like:

operation from  $\Sigma$

$$\frac{p \xrightarrow{a} p'}{p \mid q \xrightarrow{a} p' \mid q}$$

behaviour from  $B$

<sup>1</sup>Turi and Plotkin, "Towards a Mathematical Operational Semantics".

# First-Order Abstract GSOS

Turi and Plotkin's abstraction of GSOS rule format<sup>1</sup>:

- **Signature endo-functor**  $\Sigma$
- **Behaviour endo-functor**  $B$
- **GSOS law** – natural transformation  $\rho_X: \Sigma(X \times B X) \rightarrow B(\Sigma^* X)$

**Example (Process Algebra):**

- $\Sigma = \{ | / 2, \emptyset / 0 \} \cup \{ a. (-) / 1 \mid a \in A \}$
- $BX = \mathcal{P}(A \times X)$
- GSOS law encodes rules like:

$$\frac{p \xrightarrow{a} p'}{p \mid q \xrightarrow{a} p' \mid q}$$

---

<sup>1</sup>Turi and Plotkin, "Towards a Mathematical Operational Semantics".

# First-Order GSOS

Theory of first-order GSOS takes  $\Sigma$ ,  $B$ ,  $\rho$  as input parameters, and produces

- 😊 operational semantics  $\gamma: \Sigma^*\emptyset \rightarrow B(\Sigma^*\emptyset)$  (**operational model**)
- 😊 notion of program equivalence  $\sim \subseteq \Sigma^*\emptyset \times \Sigma^*\emptyset$  (**strong bisimilarity**)
- 😊 generic compositionality:  $p \sim q \Rightarrow C[p] \sim C[q]$  for any context

But


- 😞  $\sim$  is too fine-grained for programming languages
- 😞 first-order  $\not\subseteq$  higher-order  $\rightsquigarrow$  no  $\lambda$ -calculus

# First-Order GSOS

Theory of first-order GSOS takes  $\Sigma$ ,  $B$ ,  $\rho$  as input parameters, and produces

- 😊 operational semantics  $\gamma: \Sigma^*\emptyset \rightarrow B(\Sigma^*\emptyset)$  (**operational model**)
- 😊 notion of program equivalence  $\sim \subseteq \Sigma^*\emptyset \times \Sigma^*\emptyset$  (**strong bisimilarity**)
- 😊 generic compositionality:  $p \sim q \Rightarrow C[p] \sim C[q]$  for any context

But

- 😞  $\sim$  is too fine-grained for programming languages
- 😞 first-order  $\subsetneq$  higher-order  $\rightsquigarrow$  no  $\lambda$ -calculus 

# (Call-by-Name Extended) Combinatory Logic

- $K (= \lambda p. \lambda q. p)$
- $S (= \lambda p. \lambda q. \lambda r. (p \cdot r) \cdot (q \cdot r))$
- plus  $S'$ ,  $S''$  and  $K'$  for partially reduced terms

$$\begin{array}{cccc} K \xrightarrow{p} K'(p) & K'(p) \xrightarrow{q} p & S \xrightarrow{p} S'(p) & S'(p) \xrightarrow{q} S''(p, q) \\ S''(p, q) \xrightarrow{r} (p \cdot r) \cdot (q \cdot r) & \frac{p \rightarrow p'}{p \cdot q \rightarrow p' \cdot q} & \frac{p \xrightarrow{q} p'}{p \cdot q \rightarrow p'} & \end{array}$$

► **Example:**  $Spqr \rightarrow S'(p)qr \rightarrow S''(p, q)r \rightarrow (pr)(qr)$

- This is very similar to original GSOS, but it is not

# (Call-by-Name Extended) Combinatory Logic

- $K (= \lambda p. \lambda q. p)$
- $S (= \lambda p. \lambda q. \lambda r. (p \cdot r) \cdot (q \cdot r))$
- plus  $S', S''$  and  $K'$  for partially reduced terms

$$K \xrightarrow{p} K'(p) \quad K'(p) \xrightarrow{q} p \quad S \xrightarrow{p} S'(p) \quad S'(p) \xrightarrow{q} S''(p, q)$$
$$S''(p, q) \xrightarrow{r} (p \cdot r) \cdot (q \cdot r) \quad \frac{p \rightarrow p'}{p \cdot q \rightarrow p' \cdot q} \quad \frac{p \xrightarrow{q} p'}{p \cdot q \rightarrow p'} \quad !$$

► **Example:**  $Spqr \rightarrow S'(p)qr \rightarrow S''(p, q)r \rightarrow (pr)(qr)$

- This is very similar to original GSOS, but it is not

# Higher-Order Abstract GSOS

A higher-order GSOS law consists of

- Signature  $\Sigma$
- **Mixed variance** (!) behaviour functor  $B$
- Family of maps  $\rho_{X,Y}: \Sigma(X \times B(X, Y)) \rightarrow B(X, \Sigma^*(X + Y))$  **natural** in  $Y$  and **dinatural** in  $X$

**Example:** For **combinatory logic**:  $B(X, Y) = Y^X + Y$ ,  $\rho$  is induced by rules



Most of Turi and Plotkin's theory carries over



Program equivalence is **strong applicative bisimilarity** – still too fine-grained



# Applicative Bisimilarity

So, how can we prove contextual equivalence/inequality anyway?

One approach: **weak (applicative) bisimilarity** as a sound proof method:

1. Let  $\Rightarrow := (\rightarrow^*)$ ,  $\overset{t}{\Rightarrow} := (\Rightarrow \cdot \overset{t}{\rightarrow})$  and define weak bisimilarity as strong bisimilarity for  $\Rightarrow$
2. Prove that ensuing similarity relation  $\lesssim$  is a congruence (hard)
3. Derive that  $\lesssim \subseteq \lesssim_{ctx}$  (easy)

**Example:** for combinatory logic:  $t \lesssim s$  if

- $t \rightarrow t'$  implies  $s \rightarrow^* s'$  and  $t' \lesssim s'$  for some  $s'$
- $t \overset{r}{\rightarrow} t'$  implies  $s \overset{r}{\Rightarrow} s'$  and  $t' \lesssim s'$  for some  $s'$

Showing  $f \lesssim_{ctx} S \cdot (K \cdot I) \cdot f$  (analogue of  $f \lesssim_{ctx} \lambda x. fx$ ) reduces to showing  $f \lesssim S \cdot (K \cdot I) \cdot f$ , which is easy

# Ground Contexts

- Recently<sup>2</sup> we identified conditions on  $\Sigma$ ,  $B$ ,  $\rho$ , and category, enabling weak applicative (bi-)similarity as an abstract sound method
  - ▶ Hard part: proving congruence  $\sim$  categorical **Howe's method**
- But this would not work for the following flavour of contextual preorder in typed setting:

$$s \lesssim_{ctx}^{bool} t \quad \text{if} \quad C[s] \downarrow \Rightarrow C[t] \downarrow \quad \text{for all} \quad C: bool$$

Now:  $f \simeq_{ctx}^{bool} \lambda x. fx$  —  $f := \Omega$  does not break it!

- This can be resolved with (step-indexed) logical relations!

---

<sup>2</sup>Urbat, Tsampas, Goncharov, Milius, and Schröder, “Weak Similarity in Higher-Order Mathematical

# Step-Indexing in the Abstract

# Step-Indexing for Combinatory Logic

The **step-indexed logical relation**  $\mathcal{L}$  for combinatory logic is the inductively defined family  $(\mathcal{L}^\alpha \subseteq \Sigma^*\emptyset \times \Sigma^*\emptyset)_{\alpha \leq \omega}$ :

$$\mathcal{L}^0 = \top, \quad \mathcal{L}^{n+1} = \mathcal{L}^n \cap \mathcal{E}(\mathcal{L}^n) \cap \mathcal{V}(\mathcal{L}^n, \mathcal{L}^n), \quad \mathcal{L}^\omega = \bigcap_{n < \omega} \mathcal{L}^n$$

where  $\mathcal{E}$  and  $\mathcal{V}$  are relation transformers:

$$\begin{aligned} \mathcal{E}(R) &= \{(t, s) \mid \text{if } t \rightarrow t' \text{ then } \exists s'. s \Rightarrow s' \wedge R(t', s')\} \\ \mathcal{V}(Q, R) &= \{(t, s) \mid \text{for all } r_1, r_2, Q(r_1, r_2), \\ &\quad \text{if } t \xrightarrow{r_1} t' \text{ then } \exists s'. s \xrightarrow{r_2} s' \wedge R(t', s')\} \end{aligned}$$

As a slogan: "related programs applied to related arguments produce related results"

# Step-Indexing for Combinatory Logic: Use

- $\mathcal{L}^\omega$  is a fixpoint  $\mathcal{L}^\omega = \mathcal{L}^\omega \cap \mathcal{E}(\mathcal{L}^\omega) \cap \mathcal{V}(\mathcal{L}^\omega, \mathcal{L}^\omega)$
- In first-order case we would reduce to the familiar fixpoint theory and **Kleene/Knaster-Tarski theorems**, but because of higher-order, we generally do not (!)
- Every  $\mathcal{L}^\alpha$  is a congruence
- $\mathcal{L}^\omega$  is sound for contextual preorder:  $\mathcal{L}^\alpha(s, t)$  implies  $s \lesssim_{ctx} t$

**Example:** We can reprove  $f \lesssim_{ctx} S \cdot (K \cdot I) \cdot f$ , by showing by induction on  $n$  that  $\mathcal{L}^n(t, S \cdot (K \cdot I) \cdot f)$  whenever  $f \Rightarrow t$

# Contextual Preorders in the Abstract

- Given a preorder  $O \subseteq \Sigma^*\emptyset \times \Sigma^*\emptyset$ , a relation  $R \subseteq \Sigma^*\emptyset \times \Sigma^*\emptyset$  is  **$O$ -adequate** if  $R \subseteq O$
- The greatest  $O$ -adequate congruence is the **contextual preorder w.r.t.  $O$**  and denoted  $\lesssim^O$ 
  - ▶ If  $O = \{(t, s) \mid t \downarrow \Rightarrow s \downarrow\}$  then  $\lesssim^O = \lesssim_{ctx}$

**Theorem:** under general conditions  $\lesssim^O$  exists and is a preorder

# Step-Indexing in the Abstract

Additional parameters:

1. Coalgebra  $\tilde{\gamma}: \Sigma^*\emptyset \rightarrow B(\Sigma^*\emptyset, \Sigma^*\emptyset)$  abstracting weak transitions ' $\Rightarrow$ ' (additionally to the automatic coalgebra  $\gamma$  of strong transitions ' $\rightarrow$ ')
2. **Relation lifting** of  $B$ , i.e. its action on relations

**Results:** under general assumptions,

1. There is an abstract (ordinal-indexed) logical relation  $(\Box^\alpha \top)_\alpha$
2. The limit  $\Box^\vee \top = \bigcap_\alpha \Box^\alpha \top$  exists
3. Every  $\Box^\alpha \top$  is a congruence
4. If  $\Box^\vee \top$  is  $O$ -adequate then  $\Box^\vee \top \subseteq \lesssim^O$

# Ground Contextual Equivalence Revisited

- By redefining weak transitions  $t \Rightarrow^r s$  via

$$(\exists t'. t \Rightarrow t' \wedge t' \xrightarrow{r} s) \vee (\exists t'. t \Rightarrow t' \wedge s = t' r)$$

we obtain a different logical relation  $\square\top$

- By taking

$$O_{bool} = \{(t, s) \mid t \downarrow \Rightarrow s \downarrow\} \quad \text{and} \quad O_\tau = \top \quad \text{for } \tau \neq bool$$

we obtain the ground contextual preorder  $\lesssim_{ctx}^{bool} = \lesssim^0$

- $\square^v\top$  is  $O$ -adequate, hence  $\square^v\top \subseteq \lesssim_{ctx}^{bool}$



# Proving the “ $\eta$ -Law”

- Recall:  $t \xrightarrow{r} s = (\exists t'. t \Rightarrow t' \wedge t' \xrightarrow{r} s) \vee (\exists t'. t \Rightarrow t' \wedge s = t' r)$

$$\mathcal{L}^{n+1} = \mathcal{L}^n \cap \mathcal{E}(\mathcal{L}^n) \cap \mathcal{V}(\mathcal{L}^n, \mathcal{L}^n)$$

$$\mathcal{E}(\mathcal{L}^n) = \{(t, s) \mid \text{if } t \rightarrow t' \text{ then } \exists s'. s \Rightarrow s' \wedge \mathcal{L}^n(t', s')\}$$

$$\mathcal{V}(\mathcal{L}^n, \mathcal{L}^n) = \{(t, s) \mid \text{for all } r_1, r_2, \mathcal{L}^n(r_1, r_2),$$

$$\text{if } t \xrightarrow{r_1} t' \text{ then } \exists s'. s \xrightarrow{r_2} s' \wedge \mathcal{L}^n(t', s')\}$$

- Proof of  $\mathcal{L}^n(S \cdot (K \cdot I) \cdot f, f)$  by induction on  $n$ , in particular:

$$\begin{array}{ccccccccc}
 S \cdot (K \cdot I) \cdot f & \rightarrow & S'(K \cdot I) \cdot f & \rightarrow & S''(K \cdot I, f) & \xrightarrow{t} & (K \cdot I \cdot t) \cdot (f \cdot t) & \xrightarrow{*} & f \cdot t \\
 \downarrow \mathcal{L}^n & & \downarrow \mathcal{L}^{n-1} & & \downarrow \mathcal{L}^{n-2} & & \downarrow \mathcal{L}^{n-3} & & \downarrow \mathcal{L}^{n-6} \\
 f & \Longrightarrow & f & \Longrightarrow & f & \xrightarrow{t'} & f \cdot t' & \Longrightarrow & f \cdot t'
 \end{array}$$

# Conclusions

Our present construction of  $\square\top$  and results are highly flexible and cover

- various choices of the underlying category (e.g. category of presheaves for  $\lambda$ -calculus)
- type constructors and recursive types
- nondeterminism

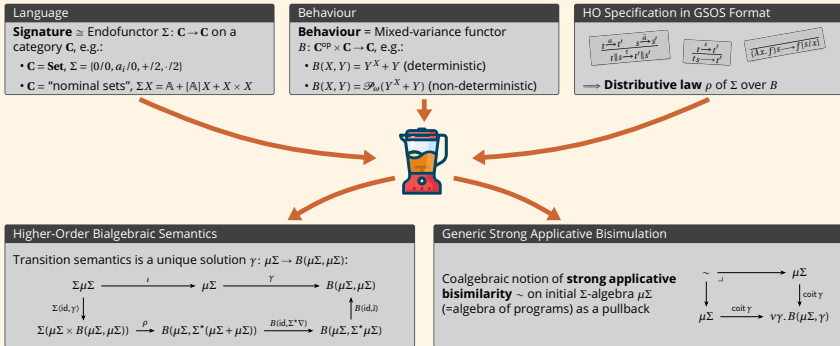
## Further Work:

- Call-by-value
- Metric, probabilistic, quantal, fibrational generalizations
- Modelling polymorphic languages
- Modelling effectful languages
- $\square\top$  is included applicative bisimilarity. When they are equal?

# Thank You for Your Attention!



## Higher-Order Abstract GSOS


### Categorical Framework for Higher-Order Operational Semantics





Central Result: Compositionality for Free

Under certain general assumptions,  $\sim$  is a congruence

-  Goncharov, Sergey, Stefan Milius, Lutz Schröder, Stelios Tsampas, and Henning Urbat. “Towards a Higher-Order Mathematical Operational Semantics”. In: Proc. ACM Program. Lang. 7 (2023), pp. 632–658. doi: 10.1145/3571215.
-  Goncharov, Sergey, Stefan Milius, Stelios Tsampas, and Henning Urbat. “Bialgebraic Reasoning on Higher-Order Program Equivalence”. In: *LICS*. 2024, pp. 1–13.

 Goncharov, Sergey, Alessio Santamaria, Lutz Schröder, Stelios Tsampas, and Henning Urbat. “Logical Predicates in Higher-Order Mathematical Operational Semantics”. In: *Foundations of Software Science and Computation Structures - 27th International Conference, FoSSaCS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part II*. Ed. by Naoki Kobayashi and James Worrell. Vol. 14575. Lecture Notes in Computer Science. Springer, 2024, pp. 47–69. doi: 10.1007/978-3-031-57231-9\\_3. url: [https://doi.org/10.1007/978-3-031-57231-9\\\_3](https://doi.org/10.1007/978-3-031-57231-9\_3).

-  Turi, D. and G. Plotkin. “Towards a Mathematical Operational Semantics”. In: *Logic in Computer Science*. IEEE. 1997, pp. 280–291.
-  Urbat, Henning, Stelios Tsampas, Sergey Goncharov, Stefan Milius, and Lutz Schröder. “Weak Similarity in Higher-Order Mathematical Operational Semantics”. In: *LICS*. 2023, pp. 1–13. doi: [10.1109/LICS56636.2023.10175706](https://doi.org/10.1109/LICS56636.2023.10175706). url: <https://doi.org/10.1109/LICS56636.2023.10175706>.