

On the Expressiveness and Decidability of Higher-Order Process Calculi

Ivan Lanese (Univ. of Bologna)
Jorge A. Pérez (Univ. of Bologna)
Davide Sangiorgi (Univ. of Bologna)
Alan Schmitt (INRIA Rhône-Alpes)

The Copenhagen Programming Language Seminar (COPLAS)
August 15 2008

Higher-order (HO) process calculi: features

- Languages which allow communication of **processes**, i.e. pieces of code a recipient can run.
- Usual operators: parallel composition, input and output prefixes, restriction. Infinite behavior can be encoded.
- As in the λ -calculus, computation involves **term instantiation**.
- Examples: CHOCS and Plain CHOCS (Thomsen); the Higher-Order π -calculus (Sangiorgi); Homer (Hildebrandt et al); Kell (Schmitt and Stefani).

HO calculi: facts

- They have been used primarily to investigate the expressiveness of **first-order calculi** such as the π -calculus.
- In π , the first- and higher-order paradigms have the same expressive power.
- Process communication has strong consequences on semantics. In particular, behavioral equivalences are problematic.
- First-order techniques usually do not carry over to the HO case.

HO languages are increasingly relevant nowadays

We find HO features in languages for emerging applications in concurrency:

- HO languages with localities have proven useful in **component-based programming** and **mobile computing**.
- Languages for **service-oriented computing** and **systems biology** usually include HO features (or elements reminiscent of them).

This work

We aim at a better understanding of the HO communication paradigm.

We propose to do so by:

- identifying a **core language** for HO concurrency;
- studying sensible **behavioral equivalences** for it;
- addressing issues little studied or not studied at all:
absolute expressiveness and **decidability**.

HOCORE: a core calculus for higher-order concurrency

Syntax

P, Q	$::=$	$\bar{a}\langle P \rangle$	output
		$a(x).P$	input prefix
		x	process variable
		$P \parallel Q$	parallel composition
		$\mathbf{0}$	nil

HOCORE: a core calculus for higher-order concurrency

Semantics

Labeled Transition System

$$\text{INP} \quad a(x). P \xrightarrow{a(x)} P \qquad \text{OUT} \quad \bar{a}\langle P \rangle \xrightarrow{\bar{a}\langle P \rangle} \mathbf{0}$$

$$\text{ACT1} \quad \frac{P_1 \xrightarrow{\alpha} P'_1 \quad \text{bv}(\alpha) \cap \text{fv}(P_2) = \emptyset}{P_1 \parallel P_2 \xrightarrow{\alpha} P'_1 \parallel P_2}$$

$$\text{TAU1} \quad \frac{P_1 \xrightarrow{\bar{a}\langle P \rangle} P'_1 \quad P_2 \xrightarrow{a(x)} P'_2}{P_1 \parallel P_2 \xrightarrow{\tau} P'_1 \parallel P'_2\{P/x\}}$$

Structural congruence, \equiv :

$$P \parallel \mathbf{0} \equiv P, \quad P_1 \parallel P_2 \equiv P_2 \parallel P_1, \quad P_1 \parallel (P_2 \parallel P_3) \equiv (P_1 \parallel P_2) \parallel P_3.$$

Reductions $P \longrightarrow P'$ are defined as $P \equiv \xrightarrow{\tau} \equiv P'$.

HOCORE: a core calculus for higher-order concurrency

Main features

- HO communication is **strict**: no name passing is allowed.
- No **output prefix**: asynchronous calculus.
- No **restriction operator**: every communication is public.
Behavior is **exposed**.

HOCORE: a core calculus for higher-order concurrency

Main features

- HO communication is **strict**: no name passing is allowed.
- No **output prefix**: asynchronous calculus.
- No **restriction operator**: every communication is public.
Behavior is **exposed**.

A **located, concurrent** λ -calculus:

- $a(x).P$: a function with parameter x and body P , located at a ;
- $\bar{a}\langle Q \rangle$: an argument Q for a function located at a .

Main Results

- 1 HOCORE is Turing complete.

Main Results

- 1 HOCORE is Turing complete.
- 2 Strong bisimilarity is decidable.

Main Results

- 1 HOCORE is Turing complete.
- 2 Strong bisimilarity is decidable.
- 3 A number of sensible equivalences coincide with strong bisimilarity.

Main Results

- 1 HOCORE is Turing complete.
- 2 Strong bisimilarity is decidable.
- 3 A number of sensible equivalences coincide with strong bisimilarity.
- 4 Strong bisimilarity has a sound and complete axiomatization.

Main Results

- 1 HOCORE is Turing complete.
- 2 Strong bisimilarity is decidable.
- 3 A number of sensible equivalences coincide with strong bisimilarity.
- 4 Strong bisimilarity has a sound and complete axiomatization.
- 5 Using (4), bisimulation checking has a polynomial complexity.

Main Results

- 1 HOCORE is Turing complete.
- 2 Strong bisimilarity is decidable.
- 3 A number of sensible equivalences coincide with strong bisimilarity.
- 4 Strong bisimilarity has a sound and complete axiomatization.
- 5 Using (4), bisimulation checking has a polynomial complexity.
- 6 Decidability breaks with four *static* restrictions.

Main Results

- 1 HOCORE is Turing complete.
- 2 Strong bisimilarity is decidable.
- 3 A number of sensible equivalences coincide with strong bisimilarity.
- 4 Strong bisimilarity has a sound and complete axiomatization.
- 5 Using (4), bisimulation checking has a polynomial complexity.
- 6 Decidability breaks with four *static* restrictions.

In this talk

I will focus on (1)-(3). Some hints on (4) and (6).

- 1 Motivation
- 2 A core calculus for higher-order concurrency
- 3 Focus on some of the results
 - Absolute Expressiveness
 - Behavioral Equivalences in HOCORE
- 4 A bird-eye view on other results
 - Axiomatization
 - Limits of decidability
- 5 Final Remarks

HOCORE is Turing complete

We show HOCORE is Turing complete by [encoding Minsky machines](#).

The encoding:

- Uses basic forms of replication and guarded choice.
- The cornerstone: counters that may be tested for zero.
- Counters and registers based on HO communication.
- Requires a finite number of fresh names (linear on the number of instructions).

The encoding is termination-preserving. Hence, termination in HOCORE is undecidable.

Expressivity of HOCore

Guarded Choice

Assume that, for $i \in \{1, 2\}$, a process P_i can be only triggered by a behavior selector \hat{a}_i :

$$(a_1.P_1 + a_2.P_2) \parallel \hat{a}_i \longrightarrow P_i$$

This is encoded as

$$\begin{aligned} \llbracket a_1.P_1 + a_2.P_2 \rrbracket_+ &= \bar{a}_1 \langle \llbracket P_1 \rrbracket_+ \rangle \parallel \bar{a}_2 \langle \llbracket P_2 \rrbracket_+ \rangle \\ \llbracket \hat{a}_i \rrbracket_+ &= a_1(x_1).a_2(x_2).x_i \end{aligned}$$

and an extra communication is introduced:

$$\llbracket (a_1.P_1 + a_2.P_2) \parallel \hat{a}_i \rrbracket_+ \longrightarrow \longrightarrow \llbracket P_i \rrbracket_+$$

Expressivity of HO CORE

Input-guarded replication

Divergence-free adaptation of the usual encoding of replication:

$$\llbracket !a(z). P \rrbracket_{i!} = a(z). (Q_c \parallel P) \parallel \bar{c}\langle a(z). (Q_c \parallel P) \rangle$$

where

- $Q_c = c(x). (x \parallel \bar{c}\langle x \rangle)$
- P contains no replications (nested replications are forbidden)
- $\llbracket \cdot \rrbracket_{i!}$ is an homomorphism for the other operators.

Encoding Minsky machines into HOCORE

Two-counter Minsky machines

Turing complete model with n labeled instructions and two registers.

- **Registers** r_j ($j \in \{0, 1\}$) can hold arbitrarily large natural numbers.
- **Instructions** can be of two kinds:

Instruction	$r_j == 0$	$r_j > 0$
INC(r_j)	$r_j = r_j + 1$	$r_j = r_j + 1$
DECJ(r_j, k)	jump to k	$r_j = r_j - 1$

- A **program counter** indicates the label of the instruction being executed.

Encoding Minsky machines into HOCORE

Numbers as nested higher-order processes

A number $k > 0$ is encoded as the *wrapping* of its predecessor in a “successor” channel, and a “non-zero” flag:

$$\langle \!| \ k + 1 \ | \rangle_j = \overline{r_j^S} \langle \!| \ k \ | \rangle_j \parallel \widehat{n}_j$$

Similarly, $\langle \!| \ 0 \ | \rangle_j = \overline{r_j^0} \parallel \widehat{z}_j$ (the “zero” channel and the “zero” flag).

Encoding Minsky machines into HO_{CORE}

Numbers as nested higher-order processes

A number $k > 0$ is encoded as the *wrapping* of its predecessor in a “successor” channel, and a “non-zero” flag:

$$\langle \langle k + 1 \rangle \rangle_j = \overline{r_j^S} \langle \langle k \rangle \rangle_j \parallel \widehat{n}_j$$

Similarly, $\langle \langle 0 \rangle \rangle_j = \overline{r_j^0} \parallel \widehat{z}_j$ (the “zero” channel and the “zero” flag).

Example: Encoding 2

$$\langle \langle 0 \rangle \rangle_j = \overline{r_j^0} \parallel \widehat{z}_j$$

To **increment** it:

put it as the argument of a message on r_j^S along with the \widehat{n}_j flag.

To **decrement** it:

consume the message on r_j^S and use \widehat{n}_j to trigger some behavior.

Encoding Minsky machines into HO_CORE

Numbers as nested higher-order processes

A number $k > 0$ is encoded as the *wrapping* of its predecessor in a “successor” channel, and a “non-zero” flag:

$$\langle \langle k + 1 \rangle \rangle_j = \overline{r_j^S} \langle \langle k \rangle \rangle_j \parallel \hat{n}_j$$

Similarly, $\langle \langle 0 \rangle \rangle_j = \overline{r_j^0} \parallel \hat{z}_j$ (the “zero” channel and the “zero” flag).

Example: Encoding 2

$$\langle \langle 1 \rangle \rangle_j = \overline{r_j^S} \langle \langle 0 \rangle \rangle_j \parallel \hat{n}_j$$

To **increment** it:

put it as the argument of a message on r_j^S along with the \hat{n}_j flag.

To **decrement** it:

consume the message on r_j^S and use \hat{n}_j to trigger some behavior.

Encoding Minsky machines into HOCore

Numbers as nested higher-order processes

A number $k > 0$ is encoded as the *wrapping* of its predecessor in a “successor” channel, and a “non-zero” flag:

$$\langle \langle k + 1 \rangle \rangle_j = \overline{r_j^S} \langle \langle k \rangle \rangle_j \parallel \widehat{n}_j$$

Similarly, $\langle \langle 0 \rangle \rangle_j = \overline{r_j^0} \parallel \widehat{z}_j$ (the “zero” channel and the “zero” flag).

Example: Encoding 2

$$\langle \langle 1 \rangle \rangle_j = \overline{r_j^S} \langle \overline{r_j^0} \parallel \widehat{z}_j \rangle \parallel \widehat{n}_j$$

To **increment** it:

put it as the argument of a message on r_j^S along with the \widehat{n}_j flag.

To **decrement** it:

consume the message on r_j^S and use \widehat{n}_j to trigger some behavior.

Encoding Minsky machines into HOCORE

Numbers as nested higher-order processes

A number $k > 0$ is encoded as the *wrapping* of its predecessor in a “successor” channel, and a “non-zero” flag:

$$\langle \langle k + 1 \rangle \rangle_j = \overline{r_j^S} \langle \langle k \rangle \rangle_j \parallel \hat{n}_j$$

Similarly, $\langle \langle 0 \rangle \rangle_j = \overline{r_j^0} \parallel \hat{z}_j$ (the “zero” channel and the “zero” flag).

Example: Encoding 2

$$\langle \langle 2 \rangle \rangle_j = \overline{r_j^S} \langle \langle 1 \rangle \rangle_j \parallel \hat{n}_j$$

To **increment** it:

put it as the argument of a message on r_j^S along with the \hat{n}_j flag.

To **decrement** it:

consume the message on r_j^S and use \hat{n}_j to trigger some behavior.

Encoding Minsky machines into HOCORE

Numbers as nested higher-order processes

A number $k > 0$ is encoded as the *wrapping* of its predecessor in a “successor” channel, and a “non-zero” flag:

$$\langle \langle k + 1 \rangle \rangle_j = \overline{r_j^S} \langle \langle k \rangle \rangle_j \parallel \widehat{n}_j$$

Similarly, $\langle \langle 0 \rangle \rangle_j = \overline{r_j^0} \parallel \widehat{z}_j$ (the “zero” channel and the “zero” flag).

Example: Encoding 2

$$\langle \langle 2 \rangle \rangle_j = \overline{r_j^S} \langle \overline{r_j^S} \langle \overline{r_j^0} \parallel \widehat{z}_j \rangle \parallel \widehat{n}_j \rangle \parallel \widehat{n}_j$$

To **increment** it:

put it as the argument of a message on r_j^S along with the \widehat{n}_j flag.

To **decrement** it:

consume the message on r_j^S and use \widehat{n}_j to trigger some behavior.

Encoding Minsky machines into HOCORE

Registers: counters that can be incremented and decremented

Operations as two mutually recursive behaviors on r_j^0 and r_j^S .

- **Increment:** the process sends a message on r_j^S containing the successor of the current register value.
- **Decrement:** the register is recreated with the decremented value (or zero) and the corresponding flag (\hat{z}_j or \hat{n}_j) is spawned.

Encoding Minsky machines into HOCORE

Instructions: encoded hand-in-hand with registers.

An instruction $(i : l_i)$ is a replicated process guarded by p_i .

- Once p_i is consumed, the instruction is active and an interaction with a register occurs.
- A choice representing the kind of instruction is sent to the register, which returns an acknowledgment.
- Upon reception of the acknowledgment, either
 - case INC: the next instruction is spawned
 - case DECJ: a jump to the specified instruction (if the register was zero) OR the next instruction is spawned (otherwise).

Encoding Minsky machines into HOCORE

INSTRUCTIONS $(i : l_i)$

$$\llbracket (i : \text{INC}(r_j)) \rrbracket_M = !p_i. (\widehat{\text{inc}}_j \parallel \text{ack}. \overline{p_{i+1}})$$

$$\llbracket (i : \text{DECJ}(r_j, k)) \rrbracket_M = !p_i. (\widehat{\text{dec}}_j \parallel \text{ack}. (z_j. \overline{p_k} + n_j. \overline{p_{i+1}}))$$

Encoding Minsky machines into HOCORE

INSTRUCTIONS ($i : l_i$)

$$\llbracket (i : \text{INC}(r_j)) \rrbracket_M = !p_i. (\widehat{\text{inc}}_j \parallel \text{ack}. \overline{p_{i+1}})$$

$$\llbracket (i : \text{DECJ}(r_j, k)) \rrbracket_M = !p_i. (\widehat{\text{dec}}_j \parallel \text{ack}. (z_j. \overline{p_k} + n_j. \overline{p_{i+1}}))$$

REGISTERS r_j

$$\llbracket r_j = 0 \rrbracket_M = (\text{inc}_j. (\overline{r_j^S} \langle \langle 1 \rangle_j \rangle \parallel \overline{\text{ack}}) + \text{dec}_j. (\langle \langle 0 \rangle_j \rangle \parallel \overline{\text{ack}})) \parallel \text{REG}_j$$

$$\llbracket r_j = m \rrbracket_M = (\text{inc}_j. (\overline{r_j^S} \langle \langle m \rangle_j \rangle \parallel \overline{\text{ack}}) + \text{dec}_j. (\langle \langle m - 1 \rangle_j \rangle \parallel \overline{\text{ack}})) \parallel \text{REG}_j$$

Encoding Minsky machines into HOCore

INSTRUCTIONS ($i : l_i$)

$$\llbracket (i : \text{INC}(r_j)) \rrbracket_M = !p_i. (\widehat{\text{inc}}_j \parallel \text{ack}. \overline{p_{i+1}})$$

$$\llbracket (i : \text{DECJ}(r_j, k)) \rrbracket_M = !p_i. (\widehat{\text{dec}}_j \parallel \text{ack}. (z_j. \overline{p_k} + n_j. \overline{p_{i+1}}))$$

REGISTERS r_j

$$\llbracket r_j = 0 \rrbracket_M = (\text{inc}_j. (\overline{r_j^S} \langle \langle 1 \rangle_j \rangle \parallel \overline{\text{ack}}) + \text{dec}_j. (\langle \langle 0 \rangle_j \rangle \parallel \overline{\text{ack}})) \parallel \text{REG}_j$$

$$\llbracket r_j = m \rrbracket_M = (\text{inc}_j. (\overline{r_j^S} \langle \langle m \rangle_j \rangle \parallel \overline{\text{ack}}) + \text{dec}_j. (\langle \langle m - 1 \rangle_j \rangle \parallel \overline{\text{ack}})) \parallel \text{REG}_j$$

where:

$$\text{REG}_j = !r_j^0. (\text{inc}_j. (\overline{r_j^S} \langle \langle 1 \rangle_j \rangle \parallel \overline{\text{ack}}) + \text{dec}_j. (\langle \langle 0 \rangle_j \rangle \parallel \overline{\text{ack}})) \parallel$$

$$!r_j^S(Y). (\text{inc}_j. (\overline{r_j^S} \langle \overline{r_j^S} \langle Y \rangle \rangle \parallel \widehat{n}_j \rangle \parallel \overline{\text{ack}}) + \text{dec}_j. (Y \parallel \overline{\text{ack}}))$$

$$\langle \langle k \rangle_j \rangle = \begin{cases} \overline{r_j^0} \parallel \widehat{z}_j & \text{if } k = 0 \\ \overline{r_j^S} \langle \langle k - 1 \rangle_j \rangle \parallel \widehat{n}_j & \text{if } k > 0. \end{cases}$$

Encoding Minsky machines into HOCORE

Lemma

Let $[[\cdot]]_M$ represent the encoding of Minsky machines into HOCORE.
Given a Minsky machine N , we have:

- $N \longrightarrow N'$ if and only iff $[[N]]_M \longrightarrow^* [[N']]_M$;
- $N \not\rightarrow$ if and only iff $[[N]]_M \not\rightarrow$;
- N diverges if and only iff $[[N]]_M$ diverges.

Encoding Minsky machines into HOCORE

Lemma

Let $[[\cdot]]_M$ represent the encoding of Minsky machines into HOCORE.
Given a Minsky machine N , we have:

- $N \longrightarrow N'$ if and only iff $[[N]]_M \longrightarrow^* [[N']]_M$;
- $N \not\rightarrow$ if and only iff $[[N]]_M \not\rightarrow$;
- N diverges if and only iff $[[N]]_M$ diverges.

Since the encoding preserves termination we have:

Corollary

Termination in HOCORE is undecidable.

Roadmap

- 1 Motivation
- 2 A core calculus for higher-order concurrency
- 3 Focus on some of the results
 - Absolute Expressiveness
 - Behavioral Equivalences in HOCORE
- 4 A bird-eye view on other results
 - Axiomatization
 - Limits of decidability
- 5 Final Remarks

Behavioral Equivalences in HO_CORE

HO_CORE has a unique reasonable relation of **strong bisimilarity** (\sim) that enjoys a number of nice properties:

- 1 it is **decidable**;
- 2 it is a **congruence** (and with a simple proof);
- 3 it coincides with a number of equivalences, including **barbed congruence**.

Bisimilarities for HO calculi in a nutshell

Consider two processes, P and Q :

- “Ordinary” (i.e. CCS-like) bisimilarity: P and Q are bisimilar if any action by one can be matched by an **identical action** from the other.

Drawback: This breaks basic laws, e.g., commutativity of \parallel :

$$\bar{a}\langle P \parallel Q \rangle \not\approx \bar{a}\langle Q \parallel P \rangle$$

Bisimilarities for HO calculi in a nutshell

Consider two processes, P and Q :

- “Ordinary” (i.e. CCS-like) bisimilarity: P and Q are bisimilar if any action by one can be matched by an **identical action** from the other.

Drawback: This breaks basic laws, e.g., commutativity of \parallel :

$$\bar{a}\langle P \parallel Q \rangle \not\approx \bar{a}\langle Q \parallel P \rangle$$

- In **Higher-order bisimilarity** (\sim_{HO}) one requires **bisimilarity**, rather than **identity**, of the processes emitted in an output action.

Drawback: It is over-discriminating and properties (e.g. congruence) may be very hard to establish.

Bisimilarities for HO calculi in a nutshell

- **Context bisimilarity** explicitly takes into account every possible context the emitted process could go to.

It yields more satisfactory process equalities, and it coincides with contextual equivalence (i.e., barbed congruence).

Drawback: It involves a universal quantification over contexts in the clause for output actions.

Bisimilarities for HO calculi in a nutshell

- **Context bisimilarity** explicitly takes into account every possible context the emitted process could go to.

It yields more satisfactory process equalities, and it coincides with contextual equivalence (i.e., barbed congruence).

Drawback: It involves a universal quantification over contexts in the clause for output actions.

- **Normal bisimilarity** simplifies context bisimilarity by replacing universal quantifications in the output clause with a single process.

Drawback: Its definition may depend on the operators in the calculus. Also, the correspondence with context bisimilarity may be hard to prove.

Strong bisimilarity is decidable in HOCORE

We define **Input/Output bisimilarity** as an auxiliary bisimilarity.

- it is a congruence and is decidable.
- τ -actions do not participate in the bisimulation game but ...

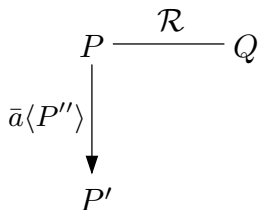
Strong bisimilarity is decidable in HOCORE

We define **Input/Output bisimilarity** as an auxiliary bisimilarity.

- it is a congruence and is decidable.
- τ -actions do not participate in the bisimulation game but ...
- ...they are preserved by the bisimilarity, which allows to relate it to other bisimilarities.

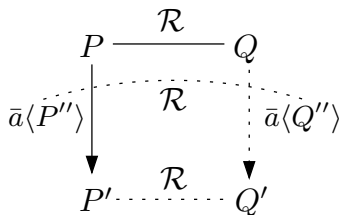
Input/Output bisimilarity

Input/Output bisimilarity (\sim_{IO}°) is the largest symmetric relation \mathcal{R} on open processes such that whenever $P \mathcal{R} Q$:



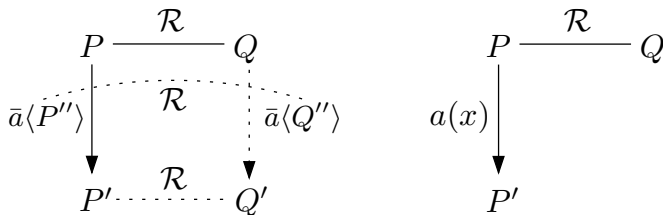
Input/Output bisimilarity

Input/Output bisimilarity (\sim_{IO}°) is the largest symmetric relation \mathcal{R} on open processes such that whenever $P \mathcal{R} Q$:



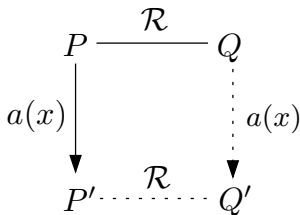
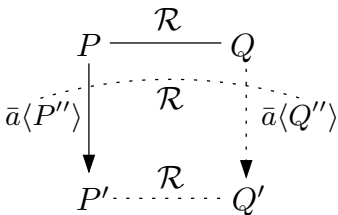
Input/Output bisimilarity

Input/Output bisimilarity (\sim_{IO}°) is the largest symmetric relation \mathcal{R} on open processes such that whenever $P \mathcal{R} Q$:



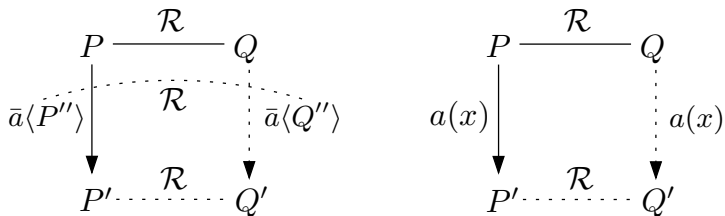
Input/Output bisimilarity

Input/Output bisimilarity (\sim_{IO}°) is the largest symmetric relation \mathcal{R} on open processes such that whenever $P \mathcal{R} Q$:



Input/Output bisimilarity

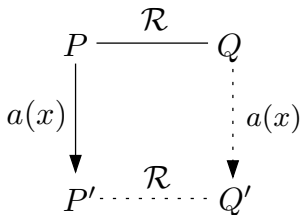
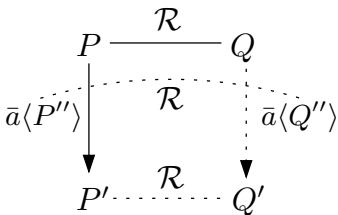
Input/Output bisimilarity (\sim_{IO}°) is the largest symmetric relation \mathcal{R} on open processes such that whenever $P \mathcal{R} Q$:



$$P' \parallel x \equiv P \mathcal{R} Q$$

Input/Output bisimilarity

Input/Output bisimilarity (\sim_{IO}°) is the largest symmetric relation \mathcal{R} on open processes such that whenever $P \mathcal{R} Q$:



$$P' \parallel x \equiv P \xrightarrow{\mathcal{R}} Q \equiv Q' \parallel x$$

$$P' \text{---} \mathcal{R} \text{---} Q'$$

Input/Output bisimilarity

Lemma

In HOCORE, IO bisimilarity

- *is preserved by substitutions*
- *is a congruence*
- *is decidable.*

Input/Output bisimilarity

Lemma

In HOCORE, IO bisimilarity

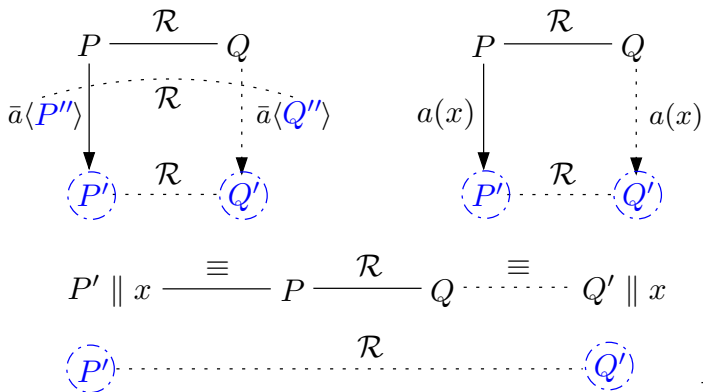
- *is preserved by substitutions*
- *is a congruence*
- *is decidable.*

Key in the proofs are:

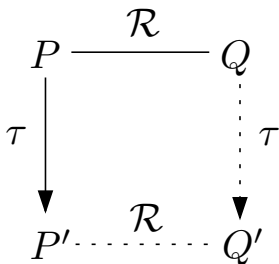
- The fact that it does not require to match τ -actions.
- The size of processes **always decreases** during the bisimulation game (because it's open and has no τ , so no process copying).

Input/Output bisimilarity

The size of processes **always decreases** during the bisimulation game



τ -bisimilarity



IO bisimilarity implies τ -bisimilarity

Lemma

If $P \sim_{\text{IO}}^{\circ} Q$ and $P \xrightarrow{\tau} P'$ then $\exists Q'$ s.t. $Q \xrightarrow{\tau} Q'$ and $P' \sim_{\text{IO}}^{\circ} Q'$.

IO bisimilarity implies τ -bisimilarity

Lemma

If $P \sim_{\text{IO}}^{\circ} Q$ and $P \xrightarrow{\tau} P'$ then $\exists Q'$ s.t. $Q \xrightarrow{\tau} Q'$ and $P' \sim_{\text{IO}}^{\circ} Q'$.

Proof (Sketch)

Suppose $P \xrightarrow{\tau} P'$: we have to find a matching transition from Q .

IO bisimilarity implies τ -bisimilarity

Lemma

If $P \sim_{\text{IO}}^{\circ} Q$ and $P \xrightarrow{\tau} P'$ then $\exists Q'$ s.t. $Q \xrightarrow{\tau} Q'$ and $P' \sim_{\text{IO}}^{\circ} Q'$.

Proof (Sketch)

Suppose $P \xrightarrow{\tau} P'$: we have to find a matching transition from Q .

- P 's transition can be decomposed into an output $P \xrightarrow{\bar{a}(R)} P_1$ followed by an input $P_1 \xrightarrow{a(x)} P_2$, with $P' = P_2\{R/x\}$.

IO bisimilarity implies τ -bisimilarity

Lemma

If $P \sim_{\text{IO}}^{\circ} Q$ and $P \xrightarrow{\tau} P'$ then $\exists Q'$ s.t. $Q \xrightarrow{\tau} Q'$ and $P' \sim_{\text{IO}}^{\circ} Q'$.

Proof (Sketch)

Suppose $P \xrightarrow{\tau} P'$: we have to find a matching transition from Q .

- P 's transition can be decomposed into an output $P \xrightarrow{\bar{a}(R)} P_1$ followed by an input $P_1 \xrightarrow{a(x)} P_2$, with $P' = P_2\{R/x\}$.
- Q is capable of matching these transitions, and the final derivative is a process Q_2 with $Q_2 \sim_{\text{IO}}^{\circ} P_2$.

IO bisimilarity implies τ -bisimilarity

Lemma

If $P \sim_{\text{IO}}^{\circ} Q$ and $P \xrightarrow{\tau} P'$ then $\exists Q'$ s.t. $Q \xrightarrow{\tau} Q'$ and $P' \sim_{\text{IO}}^{\circ} Q'$.

Proof (Sketch)

Suppose $P \xrightarrow{\tau} P'$: we have to find a matching transition from Q .

- P 's transition can be decomposed into an output $P \xrightarrow{\bar{a}(R)} P_1$ followed by an input $P_1 \xrightarrow{a(x)} P_2$, with $P' = P_2\{R/x\}$.
- Q is capable of matching these transitions, and the final derivative is a process Q_2 with $Q_2 \sim_{\text{IO}}^{\circ} P_2$.
- Since HOCORE is asynchronous, these two transitions from Q can be combined into a τ -transition that matches that of P .

IO bisimilarity implies τ -bisimilarity

Lemma

If $P \sim_{\text{IO}}^{\circ} Q$ and $P \xrightarrow{\tau} P'$ then $\exists Q'$ s.t. $Q \xrightarrow{\tau} Q'$ and $P' \sim_{\text{IO}}^{\circ} Q'$.

Proof (Sketch)

Suppose $P \xrightarrow{\tau} P'$: we have to find a matching transition from Q .

- P 's transition can be decomposed into an output $P \xrightarrow{\bar{a}(R)} P_1$ followed by an input $P_1 \xrightarrow{a(x)} P_2$, with $P' = P_2\{R/x\}$.
- Q is capable of matching these transitions, and the final derivative is a process Q_2 with $Q_2 \sim_{\text{IO}}^{\circ} P_2$.
- Since HOCORE is asynchronous, these two transitions from Q can be combined into a τ -transition that matches that of P .
- We conclude using the fact \sim_{IO}° is a congruence and preserved by substitutions.

Equivalence collapsing in HOCORE

In HOCORE, a number of sensible equivalences coincide with IO bisimilarity and inherit its properties:

- Higher-order bisimilarity
- Context bisimilarity
- Normal bisimilarity

Barbed Congruence

Strong bisimulation also coincides with barbed congruence, the contextual equivalence in concurrency.

We consider an **asynchronous** version:

- it implies the result for the synchronous version;
- barbs are only produced by output messages; this fits better with HOCORE asynchrony.

(Asynchronous) Barbed Congruence

Definition

Asynchronous barbed congruence, \simeq , is the largest symmetric relation on closed processes that

- 1** *is a τ -bisimilarity;*
- 2** *is context-closed (i.e., $P \simeq Q$ implies $C[P] \simeq C[Q]$, for all closed contexts $C[\cdot]$);*
- 3** *is barb preserving (i.e., if $P \simeq Q$ and $P \downarrow_{\bar{a}}$, then also $Q \downarrow_{\bar{a}}$).*

(Asynchronous) Barbed Congruence

Lemma

Asynchronous barbed congruence coincides with normal bisimilarity.

In **synchronous** barbed congruence, input barbs are also observable (condition 3).

Corollary

In HOCORE asynchronous and synchronous barbed congruence coincide.

- 1 Motivation
- 2 A core calculus for higher-order concurrency
- 3 Focus on some of the results
 - Absolute Expressiveness
 - Behavioral Equivalences in HOCORE
- 4 A bird-eye view on other results
 - Axiomatization
 - Limits of decidability
- 5 Final Remarks

Axiomatization

Consider \equiv_E , the extension of \equiv with the **distribution law**:

$$a(x). (P \parallel \prod_1^{k-1} a(x). P) = \prod_1^k a(x). P$$

A process P is in **normal form** $n(P)$ if it can't be further simplified in \equiv_E .

Theorem

For any processes P and Q , we have $P \sim Q$ iff $n(P) \equiv n(Q)$.

Corollary

\equiv_E is a sound and complete axiomatization of bisimilarity in HOCORE .

Roadmap

- 1 Motivation
- 2 A core calculus for higher-order concurrency
- 3 Focus on some of the results
 - Absolute Expressiveness
 - Behavioral Equivalences in HOCORE
- 4 A bird-eye view on other results
 - Axiomatization
 - Limits of decidability
- 5 Final Remarks

Limits of decidability

Consider the following extension of HOCORE , with **static restrictions**:

$$\begin{aligned} T & ::= \nu a. T \mid P \\ P & ::= \bar{a}\langle P \rangle \mid a(x).P \mid x \mid P \parallel Q \mid \text{nil} \end{aligned}$$

(Recursion can not be encoded.)

- **Four** static restrictions are enough to break decidability.
- The proof uses a reduction from the Post correspondence problem (PCP).

Post correspondence problem (PCP)

Definition

A PCP instance consists of

- an alphabet A containing at least two symbols
- a finite list T_1, \dots, T_n of *tiles*, each tile being a pair of words over A .

$T_i = (u_i, l_i)$ is the tile with upper word u_i and lower word l_i .

A solution is a non-empty sequence of indices i_1, \dots, i_k , $1 \leq i_j \leq n$ ($j \in 1 \dots k$), such that $u_{i_1} \dots u_{i_k} = l_{i_1} \dots l_{i_k}$.

Decision problem: to determine whether a solution for PCP exists or not.
This is known to be undecidable.

A PCP instance

Given the following four tiles

T₁

aba
a

T₂

bbb
aaa

T₃

aab
abab

T₄

bb
babba

A PCP instance

Given the following four tiles

 T_1

aba
a

 T_2

bbb
aaa

 T_3

aab
abab

 T_4

bb
babba

The sequence (1,4,3,1) is a solution:

A PCP instance

Given the following four tiles

 T_1

aba
a

 T_2

bbb
aaa

 T_3

aab
abab

 T_4

bb
babba

The sequence (1,4,3,1) is a solution:

aba	bb	aab	aba
a	babba	abab	a
T_1	T_4	T_3	T_1

Encoding PCP into HOCORE

Encoding Strategy

Let D be the divergent process that makes τ transitions indefinitely.

- Build a set of processes P_1, \dots, P_n for each tile T_1, \dots, T_n .
- P_j is bisimilar to D iff the PCP instance has no solution ending with tile T_j .
- Thus, PCP is solvable iff there exists a P_j not bisimilar to D .

Encoding PCP into HOCORE

Processes P_1, \dots, P_n : execute in two distinct phases:

- first they build a possible solution of PCP
- then non-deterministically they stop building the solution and execute it.

If the chosen composition is a solution then a signal on a free channel *success* is sent, thus performing a visible action, which breaks bisimilarity with D .

Encoding PCP into HOCORE

Encoding P_j

We consider words made of an alphabet of two letters, a_1 and a_2 :

LETTERS	$\llbracket a_1, P \rrbracket_u = \llbracket a_2, P \rrbracket_l = \bar{a}(P)$
	$\llbracket a_2, P \rrbracket_u = \llbracket a_1, P \rrbracket_l = a(x). (x \parallel P)$
STRINGS	$\llbracket a_i \cdot s, P \rrbracket_w = \llbracket a_i, \llbracket s, P \rrbracket_w \rrbracket_w$
	$\llbracket \epsilon, P \rrbracket_w = P \quad (\epsilon \text{ is the empty word})$

Encoding PCP into HOCORE

Encoding P_j

We consider words made of an alphabet of two letters, a_1 and a_2 :

LETTERS	$\llbracket a_1, P \rrbracket_u = \llbracket a_2, P \rrbracket_l = \bar{a} \langle P \rangle$ $\llbracket a_2, P \rrbracket_u = \llbracket a_1, P \rrbracket_l = a(x). (x \parallel P)$
STRINGS	$\llbracket a_i \cdot s, P \rrbracket_w = \llbracket a_i, \llbracket s, P \rrbracket_w \rrbracket_w$ $\llbracket \epsilon, P \rrbracket_w = P \quad (\epsilon \text{ is the empty word})$
STARTER	$S_{u_i, l_j} = \overline{up} \langle \llbracket u_i, \bar{b} \rrbracket_u \rangle \parallel \overline{low} \langle \llbracket l_j, b. \overline{success} \rrbracket_l \rangle$
CREATORS	$C_i = up(x). low(y). (\overline{up} \langle \llbracket u_i, x \rrbracket_u \rangle \parallel \overline{low} \langle \llbracket l_j, y \rrbracket_l \rangle)$
EXECUTOR	$E = up(x). low(y). (x \parallel y)$
SYSTEM	$P_j = \nu up \nu low \nu a \nu b (S_{u_i, l_j} \parallel ! \prod_i C_i \parallel E)$

Encoding PCP into HOCORE

We call **complete τ -bisimilarity** a bisimilarity with clauses for input, output, and τ -actions.

Theorem

Given an instance of PCP and one of its tiles T_j , P_j is bisimilar to D according to any complete τ -bisimilarity iff there is no solution of the instance of PCP ending with T_j .

Corollary

Barbed congruence and any complete τ -bisimilarity are undecidable in HOCORE with four static restrictions.

Final Remarks

We have studied the basic theory of HOCORE , a **minimal**, **expressive**, and **convenient** process language.

Final Remarks

We have studied the basic theory of HOCORE , a **minimal**, **expressive**, and **convenient** process language.

- Bisimilarity is shown to be **decidable** and a **congruence** with relatively simple proofs.

Final Remarks

We have studied the basic theory of HOCORE , a **minimal**, **expressive**, and **convenient** process language.

- Bisimilarity is shown to be **decidable** and a **congruence** with relatively simple proofs.
- One could argue that bisimilarity is very discriminating:

Final Remarks

We have studied the basic theory of HOCORE , a **minimal**, **expressive**, and **convenient** process language.

- Bisimilarity is shown to be **decidable** and a **congruence** with relatively simple proofs.
- One could argue that bisimilarity is very discriminating:
 - systems behavior is completely exposed;
 - one can tell whether two processes are bisimilar, but in general one cannot tell whether the processes will terminate

Final Remarks

We have studied the basic theory of HOCORE , a **minimal**, **expressive**, and **convenient** process language.

- Bisimilarity is shown to be **decidable** and a **congruence** with relatively simple proofs.
- One could argue that bisimilarity is very discriminating:
 - systems behavior is completely exposed;
 - one can tell whether two processes are bisimilar, but in general one cannot tell whether the processes will terminate
- That would also explain the collapsing of several behavioral equivalences and the simplicity of the axiomatization.

Final Remarks

We have studied the basic theory of HOCORE , a **minimal**, **expressive**, and **convenient** process language.

- Bisimilarity is shown to be **decidable** and a **congruence** with relatively simple proofs.
- One could argue that bisimilarity is very discriminating:
 - systems behavior is completely exposed;
 - one can tell whether two processes are bisimilar, but in general one cannot tell whether the processes will terminate
- That would also explain the collapsing of several behavioral equivalences and the simplicity of the axiomatization.
- The language is not trivial: it is Turing complete and allows to model basic data structures.
- In fact, our encodings of Turing complete models suggest that HOCORE could be a convenient modeling language.

Final Remarks

Current/future work on HOCORE involves:

- More on the expressiveness of *pure process passing*.
- Type systems.
- Weak equivalences.
- Orthogonal extensions for modeling.

Thanks

Thank you!