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-Alphes)

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

$$
\begin{array}{rcll}
P, Q & ::= & \overline{a}\langle P \rangle & \text{output} \\
 & \mid & a(x).\, P & \text{input prefix} \\
 & \mid & x & \text{process variable} \\
 & \mid & P \parallel Q & \text{parallel composition} \\
 & \mid & \mathbf{0} & \text{nil}
\end{array}
$$

HOCORE: a core calculus for higher-order concurrency

Semantics

Labeled Transition System

$$\textsc{Inp}\quad a(x).\,P \xrightarrow{a(x)} P \qquad\qquad \textsc{Out}\quad \overline{a}\langle P\rangle \xrightarrow{\overline{a}\langle P\rangle} \mathbf{0}$$

$$\textsc{Act1}\quad \frac{P_1 \xrightarrow{\alpha} P_1' \qquad \mathrm{bv}(\alpha) \cap \mathrm{fv}(P_2) = \emptyset}{P_1 \parallel P_2 \xrightarrow{\alpha} P_1' \parallel P_2}$$

$$\textsc{Tau1}\quad \frac{P_1 \xrightarrow{\overline{a}\langle P\rangle} P_1' \qquad P_2 \xrightarrow{a(x)} P_2'}{P_1 \parallel P_2 \xrightarrow{\tau} P_1' \parallel P_2'\{^{P}\!/x\}}$$

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;
- $\overline{a}\langle Q\rangle$: 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).

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

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 $\widehat{a_i}$:

$$(a_1.\, P_1 + a_2.\, P_2) \parallel \widehat{a_i} \longrightarrow P_i$$

This is encoded as

$$\begin{aligned} [\![a_1.\, P_1 + a_2.\, P_2]\!]_+ &= \overline{a_1}\langle [\![P_1]\!]_+ \rangle \parallel \overline{a_2}\langle [\![P_2]\!]_+ \rangle \\ [\![\widehat{a_i}]\!]_+ &= a_1(x_1).\, a_2(x_2).\, x_i \end{aligned}$$

and an extra communication is introduced:

$$[\![(a_1.\, P_1 + a_2.\, P_2) \parallel \widehat{a_i}]\!]_+ \longrightarrow\longrightarrow [\![P_i]\!]_+$$

Expressivity of HOcore

Input-guarded replication

Divergence-free adaptation of the usual encoding of replication:

$$[\![!a(z).\, P]\!]_{\mathrm{i!}} = a(z).\,(Q_c \parallel P) \parallel \overline{c}\langle a(z).\,(Q_c \parallel P)\rangle$$

where

- $Q_c = c(x).\,(x \parallel \overline{c}\langle x\rangle)$
- P contains no replications (nested replications are forbidden)
- $[\![\cdot]\!]_{\mathrm{i!}}$ 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	$r_j == 0$	$r_j > 0$
$\mathtt{INC}(r_j)$	$r_j = r_j + 1$	$r_j = r_j + 1$
$\mathtt{DECJ}(r_j, k)$	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:

$$(\!| \; k+1 \; |\!)_j = \overline{r_j^S}\langle (\!| \; k \; |\!)_j \rangle \parallel \widehat{n_j}$$

Similarly, $(\!| \; 0 \; |\!)_j = \overline{r_j^0} \parallel \widehat{z_j}$ (the "zero" channel and the "zero" flag).

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:

$$(\!|\ k+1\ |\!)_j = \overline{r_j^S}\langle (\!|\ k\ |\!)_j \rangle \parallel \widehat{n_j}$$

Similarly, $(\!|\ 0\ |\!)_j = \overline{r_j^0} \parallel \widehat{z_j}$ (the "zero" channel and the "zero" flag).

Example: Encoding 2

$$(\!|\ 0\ |\!)_j = \overline{r_j^0} \parallel \widehat{z_j}$$

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 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:

$$(\!|\ k+1\ |\!)_j = \overline{r_j^S}\langle(\!|\ k\ |\!)_j\rangle \parallel \widehat{n_j}$$

Similarly, $(\!|\ 0\ |\!)_j = \overline{r_j^0} \parallel \widehat{z_j}$ (the "zero" channel and the "zero" flag).

Example: Encoding 2

$$(\!|\ 1\ |\!)_j = \overline{r_j^S}\langle(\!|\ 0\ |\!)_j\rangle \parallel \widehat{n_j}$$

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:

$$\llparenthesis\, k+1 \,\rrparenthesis_j = \overline{r_j^S}\langle \llparenthesis\, k \,\rrparenthesis_j \rangle \parallel \widehat{n_j}$$

Similarly, $\llparenthesis\, 0 \,\rrparenthesis_j = \overline{r_j^0} \parallel \widehat{z_j}$ (the "zero" channel and the "zero" flag).

Example: Encoding 2

$$\llparenthesis\, 1 \,\rrparenthesis_j = \overline{r_j^S}\langle \overline{r_j^0} \parallel \widehat{z_j} \rangle \parallel \widehat{n_j}$$

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:

$$(\!|\ k+1\ |\!)_j = \overline{r_j^S}\langle (\!|\ k\ |\!)_j \rangle \parallel \widehat{n_j}$$

Similarly, $(\!|\ 0\ |\!)_j = \overline{r_j^0} \parallel \widehat{z_j}$ (the "zero" channel and the "zero" flag).

Example: Encoding 2

$$(\!|\ 2\ |\!)_j = \overline{r_j^S}\langle (\!|\ 1\ |\!)_j \rangle \parallel \widehat{n_j}$$

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:

$$(\!|\ k+1\ |\!)_j = \overline{r_j^S}\langle (\!|\ k\ |\!)_j \rangle \parallel \widehat{n_j}$$

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

Example: Encoding 2

$$(\!|\ 2\ |\!)_j = \overline{r_j^S}\langle \overline{r_j^S}\langle \overline{r_j^0} \parallel \widehat{z_j} \rangle \parallel \widehat{n_j} \rangle \parallel \widehat{n_j}$$

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 ($\widehat{z_j}$ or $\widehat{n_j}$) is spawned.

Encoding Minsky machines into HOCORE

Instructions: encoded hand-in-hand with registers.

An instruction $(i : I_i)$ 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

$$
\begin{aligned}
&\textsc{Instructions}\ (i : I_i) \\
&[\![(i : \mathtt{INC}(r_j))]\!]_{\mathsf{M}} &&= \ !p_i.\,(\widehat{inc_j} \parallel ack.\,\overline{p_{i+1}}) \\
&[\![(i : \mathtt{DECJ}(r_j, k))]\!]_{\mathsf{M}} &&= \ !p_i.\,(\widehat{dec_j} \parallel ack.\,(z_j.\,\overline{p_k} + n_j.\,\overline{p_{i+1}})
\end{aligned}
$$

Encoding Minsky machines into HOcore

Instructions $(i : I_i)$

$$
\begin{aligned}
[\![(i : \mathtt{INC}(r_j))]\!]_{\mathsf{M}} &= \,!p_i.\,(\widehat{inc_j} \parallel ack.\,\overline{p_{i+1}}) \\
[\![(i : \mathtt{DECJ}(r_j, k))]\!]_{\mathsf{M}} &= \,!p_i.\,(\widehat{dec_j} \parallel ack.\,(z_j.\,\overline{p_k} + n_j.\,\overline{p_{i+1}})
\end{aligned}
$$

Registers r_j

$$
\begin{aligned}
[\![r_j = 0]\!]_{\mathsf{M}} &= (inc_j.\,(\overline{r_j^S}\langle(\!|\, 1 \,|\!)_j\rangle \parallel \overline{ack}) \;+\; dec_j.\,((\!|\, 0 \,|\!)_j \parallel \overline{ack})) \parallel \mathsf{REG}_j \\
[\![r_j = m]\!]_{\mathsf{M}} &= (inc_j.\,(\overline{r_j^S}\langle(\!|\, m \,|\!)_j\rangle \parallel \overline{ack}) \;+\; dec_j.\,((\!|\, m - 1 \,|\!)_j \parallel \overline{ack})) \parallel \mathsf{REG}_j
\end{aligned}
$$

Encoding Minsky machines into HOCORE

Instructions $(i : I_i)$

$$\begin{aligned}
\llbracket (i : \mathtt{INC}(r_j)) \rrbracket_{\mathsf{M}} &= \,!p_i.\,(\widehat{inc_j} \parallel ack.\,\overline{p_{i+1}}) \\
\llbracket (i : \mathtt{DECJ}(r_j, k)) \rrbracket_{\mathsf{M}} &= \,!p_i.\,(\widehat{dec_j} \parallel ack.\,(z_j.\,\overline{p_k} + n_j.\,\overline{p_{i+1}}))
\end{aligned}$$

Registers r_j

$$\begin{aligned}
\llbracket r_j = 0 \rrbracket_{\mathsf{M}} &= (inc_j.\,(\overline{r_j^S}\langle (\!| 1 |\!)_j \rangle \parallel \overline{ack}) \,+\, dec_j.\,((\!| 0 |\!)_j \parallel \overline{ack})) \parallel \mathsf{REG}_j \\
\llbracket r_j = m \rrbracket_{\mathsf{M}} &= (inc_j.\,(\overline{r_j^S}\langle (\!| m |\!)_j \rangle \parallel \overline{ack}) \,+\, dec_j.\,((\!| m-1 |\!)_j \parallel \overline{ack})) \parallel \mathsf{REG}_j
\end{aligned}$$

where:

$$\begin{aligned}
\mathsf{REG}_j \;=\; &!r_j^0.\,(inc_j.\,(\overline{r_j^S}\langle (\!| 1 |\!)_j \rangle \parallel \overline{ack}) \,+\, dec_j.\,((\!| 0 |\!)_j \parallel \overline{ack})) \parallel \\
&!r_j^S(Y).\,(inc_j.\,(\overline{r_j^S}\langle \overline{r_j^S}\langle Y \rangle \parallel \widehat{n_j} \rangle \parallel \overline{ack}) \,+\, dec_j.\,(Y \parallel \overline{ack}))
\end{aligned}$$

$$(\!| k |\!)_j = \begin{cases} \overline{r_j^0} \parallel \widehat{z_j} & \text{if } k = 0 \\ \overline{r_j^S}\langle (\!| k-1 |\!)_j \rangle \parallel \widehat{n_j} & \text{if } k > 0. \end{cases}$$

Encoding Minsky machines into HOcore

Lemma

Let $[\![\cdot]\!]_{\mathsf{M}}$ *represent the encoding of Minsky machines into* HOcore*. Given a Minsky machine* N*, we have:*

- $N \longrightarrow N'$ *if and only iff* $[\![N]\!]_{\mathsf{M}} \longrightarrow^{*} [\![N']\!]_{\mathsf{M}}$*;*
- $N \not\longrightarrow$ *if and only iff* $[\![N]\!]_{\mathsf{M}} \not\longrightarrow$*;*
- N *diverges if and only iff* $[\![N]\!]_{\mathsf{M}}$ *diverges.*

Encoding Minsky machines into HOCORE

Lemma

Let $[\![\cdot]\!]_{\mathsf{M}}$ represent the encoding of Minsky machines into HOCORE*. Given a Minsky machine N, we have:*

- $N \longrightarrow N'$ *if and only iff* $[\![N]\!]_{\mathsf{M}} \longrightarrow^{*} [\![N']\!]_{\mathsf{M}}$;
- $N \not\longrightarrow$ *if and only iff* $[\![N]\!]_{\mathsf{M}} \not\longrightarrow$;
- N *diverges if and only iff* $[\![N]\!]_{\mathsf{M}}$ *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 HOCORE

HOCORE 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 $\parallel$:

$$\overline{a}\langle P \parallel Q\rangle \not\sim \overline{a}\langle Q \parallel P\rangle$$

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 $\parallel$:

  $$\overline{a}\langle P \parallel Q\rangle \not\sim \overline{a}\langle Q \parallel P\rangle$$

- In Higher-order bisimilarity ($\sim_{\mathtt{HO}}$) 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 ($\sim^{\circ}_{\mathtt{IO}}$) is the largest symmetric relation $\mathcal{R}$ on open processes such that whenever $P \,\mathcal{R}\, Q$:

$$\begin{tikzcd}
P \arrow[r, dash, "\mathcal{R}"] \arrow[d, "\bar{a}\langle P''\rangle"'] & Q \\
P' &
\end{tikzcd}$$

Input/Output bisimilarity

Input/Output bisimilarity ($\sim^{\circ}_{\mathtt{IO}}$) is the largest symmetric relation $\mathcal{R}$ on open processes such that whenever $P \,\mathcal{R}\, Q$:

$$\begin{tikzcd}
P \arrow[r, dash, "\mathcal{R}"] \arrow[d, "\bar{a}\langle P''\rangle"'] \arrow[dr, phantom, "\mathcal{R}"] & Q \arrow[d, dotted, "\bar{a}\langle Q''\rangle"] \\
P' \arrow[r, dotted, dash, "\mathcal{R}"] & Q'
\end{tikzcd}$$

Input/Output bisimilarity

Input/Output bisimilarity ($\sim^{\circ}_{\mathtt{IO}}$) is the largest symmetric relation $\mathcal{R}$ on open processes such that whenever $P \,\mathcal{R}\, Q$:

$$\begin{tikzcd}
P \arrow[r, dash, "\mathcal{R}"] \arrow[d, "\bar{a}\langle P''\rangle"'] \arrow[dr, phantom, "\mathcal{R}"] & Q \arrow[d, dotted, "\bar{a}\langle Q''\rangle"] \\
P' \arrow[r, dotted, dash, "\mathcal{R}"] & Q'
\end{tikzcd}
\qquad
\begin{tikzcd}
P \arrow[r, dash, "\mathcal{R}"] \arrow[d, "a(x)"'] & Q \\
P' &
\end{tikzcd}$$

Input/Output bisimilarity

Input/Output bisimilarity ($\sim^{\circ}_{\mathtt{IO}}$) is the largest symmetric relation $\mathcal{R}$ on open processes such that whenever $P \,\mathcal{R}\, Q$:

$$\begin{tikzcd}
P \arrow[r, dash, "\mathcal{R}"] \arrow[d, "\bar{a}\langle P''\rangle"'] \arrow[dr, phantom, "\mathcal{R}"] & Q \arrow[d, dotted, "\bar{a}\langle Q''\rangle"] \\
P' \arrow[r, dotted, dash, "\mathcal{R}"] & Q'
\end{tikzcd}
\qquad
\begin{tikzcd}
P \arrow[r, dash, "\mathcal{R}"] \arrow[d, "a(x)"'] & Q \arrow[d, dotted, "a(x)"] \\
P' \arrow[r, dotted, dash, "\mathcal{R}"] & Q'
\end{tikzcd}$$

Input/Output bisimilarity

Input/Output bisimilarity ($\sim^{\circ}_{\mathtt{IO}}$) is the largest symmetric relation $\mathcal{R}$ on open processes such that whenever $P \,\mathcal{R}\, Q$:

$$\begin{tikzcd}
& P \arrow[rr, dash, "\mathcal{R}"] \arrow[dd] & & Q \arrow[dd, dotted] & \\
\bar{a}\langle P''\rangle \arrow[rrrr, dotted, dash, bend left, "\mathcal{R}"] & & & & \bar{a}\langle Q''\rangle \\
& P' \arrow[rr, dotted, dash, "\mathcal{R}"] & & Q' &
\end{tikzcd}$$

$$\begin{tikzcd}
P \arrow[rr, dash, "\mathcal{R}"] \arrow[dd, "a(x)"'] & & Q \arrow[dd, dotted, "a(x)"] \\
& & \\
P' \arrow[rr, dotted, dash, "\mathcal{R}"] & & Q'
\end{tikzcd}$$

$$\begin{tikzcd}
P' \parallel x \arrow[r, dash, "\equiv"] & P \arrow[r, dash, "\mathcal{R}"] & Q
\end{tikzcd}$$

Input/Output bisimilarity

Input/Output bisimilarity ($\sim^{\circ}_{\mathtt{IO}}$) is the largest symmetric relation $\mathcal{R}$ on open processes such that whenever $P \mathrel{\mathcal{R}} Q$:

$$\begin{tikzcd}
P \arrow[r, dash, "\mathcal{R}"] \arrow[d, "\bar{a}\langle P''\rangle"'] \arrow[dr, phantom, "\mathcal{R}"] & Q \arrow[d, dotted, "\bar{a}\langle Q''\rangle"] \\
P' \arrow[r, dash, dotted, "\mathcal{R}"] & Q'
\end{tikzcd}
\qquad
\begin{tikzcd}
P \arrow[r, dash, "\mathcal{R}"] \arrow[d, "a(x)"'] & Q \arrow[d, dotted, "a(x)"] \\
P' \arrow[r, dash, dotted, "\mathcal{R}"] & Q'
\end{tikzcd}$$

$$\begin{tikzcd}
P' \parallel x \arrow[r, dash, "\equiv"] & P \arrow[r, dash, "\mathcal{R}"] & Q \arrow[r, dash, dotted, "\equiv"] & Q' \parallel x \\
P' \arrow[rrr, dash, dotted, "\mathcal{R}"] & & & Q'
\end{tikzcd}\ .$$

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

$$\begin{tikzcd}
P \arrow[r, dash, "\mathcal{R}"] \arrow[d, "\bar{a}\langle P''\rangle"'] \arrow[dr, phantom, "\mathcal{R}"] & Q \arrow[d, dotted, "\bar{a}\langle Q''\rangle"] \\
P' \arrow[r, dotted, dash, "\mathcal{R}"] & Q'
\end{tikzcd}
\qquad
\begin{tikzcd}
P \arrow[r, dash, "\mathcal{R}"] \arrow[d, "a(x)"'] & Q \arrow[d, dotted, "a(x)"] \\
P' \arrow[r, dotted, dash, "\mathcal{R}"] & Q'
\end{tikzcd}$$

$$\begin{tikzcd}
P' \parallel x \arrow[r, dash, "\equiv"] & P \arrow[r, dash, "\mathcal{R}"] & Q \arrow[r, dotted, dash, "\equiv"] & Q' \parallel x \\
P' \arrow[rrr, dotted, dash, "\mathcal{R}"] & & & Q'
\end{tikzcd}.$$

τ-bisimilarity

$$\begin{tikzcd}
P \arrow[r, dash, "\mathcal{R}"] \arrow[d, "\tau"'] & Q \arrow[d, dotted, "\tau"] \\
P' \arrow[r, dash, dashed, "\mathcal{R}"] & Q'
\end{tikzcd}$$

IO bisimilarity implies τ-bisimilarity

Lemma

If $P \sim^{\circ}_{\mathtt{IO}} Q$ and $P \xrightarrow{\tau} P'$ then $\exists Q'$ s.t. $Q \xrightarrow{\tau} Q'$ and $P' \sim^{\circ}_{\mathtt{IO}} Q'$.

IO bisimilarity implies τ-bisimilarity

Lemma

If $P \sim^{\circ}_{\mathtt{IO}} Q$ *and* $P \xrightarrow{\tau} P'$ *then* $\exists Q'$ *s.t.* $Q \xrightarrow{\tau} Q'$ *and* $P' \sim^{\circ}_{\mathtt{IO}} Q'$.

Proof (Sketch)

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

IO bisimilarity implies τ-bisimilarity

Lemma

If $P \sim^{\circ}_{\mathtt{IO}} Q$ and $P \xrightarrow{\tau} P'$ then $\exists Q'$ s.t. $Q \xrightarrow{\tau} Q'$ and $P' \sim^{\circ}_{\mathtt{IO}} Q'$.

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 $P \xrightarrow{\overline{a}\langle R\rangle} P_1$ followed by an input $P_1 \xrightarrow{a(x)} P_2$, with $P' = P_2\{{}^{R}\!/x\}$.

IO bisimilarity implies τ-bisimilarity

Lemma

If $P \sim^{\circ}_{\mathtt{IO}} Q$ and $P \xrightarrow{\tau} P'$ then $\exists Q'$ s.t. $Q \xrightarrow{\tau} Q'$ and $P' \sim^{\circ}_{\mathtt{IO}} Q'$.

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 $P \xrightarrow{\overline{a}\langle R \rangle} P_1$ 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 $Q_2 \sim^{\circ}_{\mathtt{IO}} P_2$.

IO bisimilarity implies τ-bisimilarity

Lemma

If $P \sim^{\circ}_{\mathtt{IO}} Q$ and $P \xrightarrow{\tau} P'$ then $\exists Q'$ s.t. $Q \xrightarrow{\tau} Q'$ and $P' \sim^{\circ}_{\mathtt{IO}} Q'$.

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 $P \xrightarrow{\overline{a}\langle R \rangle} P_1$ 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 $Q_2 \sim^{\circ}_{\mathtt{IO}} P_2$.
- 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 $P \sim^{\circ}_{\mathtt{IO}} Q$ and $P \xrightarrow{\tau} P'$ then $\exists Q'$ s.t. $Q \xrightarrow{\tau} Q'$ and $P' \sim^{\circ}_{\mathtt{IO}} Q'$.

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 $P \xrightarrow{\overline{a}\langle R\rangle} P_1$ 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 $Q_2 \sim^{\circ}_{\mathtt{IO}} P_2$.
- 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 $\sim^{\circ}_{\mathtt{IO}}$ 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 $P \downarrow_{\overline{a}}$, then also $Q \downarrow_{\overline{a}}$).*

(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.*

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

Axiomatization

Consider $\equiv_{\mathrm{E}}$, 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 $\equiv_{\mathrm{E}}$.

Theorem

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

Corollary

$\equiv_{\mathrm{E}}$ 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:

$$\begin{array}{rcl} T & ::= & \nu a.\, T \;\big|\; P \\ P & ::= & \overline{a}\langle P\rangle \;\big|\; a(x).\, P \;\big|\; x \;\big|\; P \parallel Q \;\big|\; \mathsf{nil} \end{array}$$

(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 $T_1, \ldots, T_n$ 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 $i_1, \ldots, i_k$, $1 \leq i_j \leq n$ ($j \in 1 \cdots k$), such that $u_{i_1} \cdots u_{i_k} = l_{i_1} \cdots l_{i_k}$.

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	T2	T3	T4
aba	bbb	aab	bb
a	aaa	abab	babba

A PCP instance

Given the following four tiles

T1	T2	T3	T4
aba	bbb	aab	bb
a	aaa	abab	babba

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

A PCP instance

Given the following four tiles

T1	T2	T3	T4
aba	bbb	aab	bb
a	aaa	abab	babba

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

aba	bb	aab	aba
a	babba	abab	a
T1	T4	T3	T1

Encoding PCP into HOCORE

Encoding Strategy

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

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

Encoding PCP into HOCORE

Processes $P_1, \ldots, P_n$: 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:

$$\begin{array}{ll}
\textsc{Letters} & [\![a_1, P]\!]_u = [\![a_2, P]\!]_l = \overline{a}\langle P\rangle \\
 & [\![a_2, P]\!]_u = [\![a_1, P]\!]_l = a(x).\,(x \parallel P) \\
\textsc{Strings} & [\![a_i \cdot s, P]\!]_w = [\![a_i, [\![s, P]\!]_w]\!]_w \\
 & [\![\epsilon, P]\!]_w = P \qquad (\epsilon \text{ is the empty word})
\end{array}$$

Encoding PCP into HOCORE

Encoding P_j

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

$$
\begin{array}{ll}
\textsc{Letters} & [\![a_1, P]\!]_u = [\![a_2, P]\!]_l = \overline{a}\langle P\rangle \\
 & [\![a_2, P]\!]_u = [\![a_1, P]\!]_l = a(x).\,(x \parallel P) \\
\textsc{Strings} & [\![a_i \cdot s, P]\!]_w = [\![a_i, [\![s, P]\!]_w]\!]_w \\
 & [\![\epsilon, P]\!]_w = P \qquad (\epsilon \text{ is the empty word}) \\
 & \\
\textsc{Starter} & S_{u_i,l_i} = \overline{up}\langle [\![u_i, \overline{b}]\!]_u\rangle \parallel \overline{low}\langle [\![l_i, b.\,\overline{success}]\!]_l\rangle \\
\textsc{Creators} & C_i = up(x).\,low(y).\,(\overline{up}\langle [\![u_i, x]\!]_u\rangle \parallel \overline{low}\langle [\![l_i, y]\!]_l\rangle) \\
\textsc{Executor} & E = up(x).\,low(y).\,(x \parallel y) \\
\textsc{System} & P_j = \nu up\, \nu low\, \nu a\, \nu b\, (\, S_{u_j,l_j} \parallel !\prod_i C_i \parallel E)
\end{array}
$$

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!