Expressiveness in Concurrency

Jorge A. Pérez

university of groningen

University of Brasilia
July 21, 2015

Acknowledgment

Some of the following slides are based on the lecture notes prepared by Frank Valencia for his course at the Master Parisien de Recherche en Informatique (MPRI).

I have adapted them for the purposes of this course.

Outline

Motivation: Expressiveness

Given a particular process calculus, it is reasonable to ask:

- What kind of behavior can we express on it?
- What is its minimal set of operators?
- Are there fragments of it in which interesting properties (e.g. bisimilarity, reachability) are decidable?
  If so, what kind of systems can we model in such fragment?
- When is it more expressive than another calculus?

Expressiveness studies aim at providing answers to these kind of questions.

Motivation: Expressiveness

Given a particular process calculus, it is reasonable to ask:

- What kind of behavior can we express on it?
- What is its minimal set of operators?
- Are there fragments of it in which interesting properties (e.g. bisimilarity, reachability) are decidable?
  If so, what kind of systems can we model in such fragment?
- When is it more expressive than another calculus?

Expressiveness studies aim at providing answers to these kind of questions.

Motivation: Expressiveness

Expressiveness is a well-known criterion for assessing the significance of a paradigm.

- In areas such as automata theory the notion of expressiveness is well-understood and settled
- In concurrency theory there is yet no agreement on a formal characterization of the expressive power of a language

Motivation: Encodings

Expressiveness claims are based on the notion of encoding

- We have seen several such claims along the course
- Encodings are translations that enjoy certain correctness criteria (structural/syntactic and semantic criteria)
- Devising formal characterizations of expressiveness entails understanding precisely such criteria
- Depending on the purpose and on the given language(s), the set of applicable criteria might vary and/or there might be criteria more adequate than others
- A single, unifying theory for comparing concurrent languages does not exist

Motivation: In This Lecture

We will overview main ideas and approaches towards a meta-theory for language comparison

We present two classic expressiveness results in process calculi:

- Recursion vs. replication in CCS
- Synchronous vs. asynchronous π-calculus

Notational conventions:

- We use $\mathcal{L}_1, \mathcal{L}_2, \ldots$ to range over process languages
- We use $\approx$ (possibly decorated) to denote a suitable behavioral equivalence.
- Also, $\longrightarrow$ and $\Longrightarrow$ denote some (reduction) semantics and its reflexive, transitive closure, respectively.

Motivation: In This Lecture

We will overview main ideas and approaches towards a meta-theory for language comparison

We present two classic expressiveness results in process calculi:

- Recursion vs. replication in CCS
- Synchronous vs. asynchronous π-calculus

Notational conventions:

- We use $\mathcal{L}_1, \mathcal{L}_2, \ldots$ to range over process languages
- We use $\approx$ (possibly decorated) to denote a suitable behavioral equivalence.
- Also, $\longrightarrow$ and $\Longrightarrow$ denote some (reduction) semantics and its reflexive, transitive closure, respectively.

Outline

Expressiveness Studies: Goals

Broadly, expressiveness studies aim at two kinds of results:

- Encodability results: is there an encoding $[\![\cdot]\!]$ from $\mathcal{L}_1$ to $\mathcal{L}_2$?
- Non-encodability (or impossibility) results: prove that such an encoding does not exist

Expressiveness Studies: Goals

To assert that $\mathcal{L}_1$ is more expressive than $\mathcal{L}_2$ one needs to show an encodability and an impossibility result:

1. An encoding $[\![\cdot]\!]_e : \mathcal{L}_2 \to \mathcal{L}_1$
   (All the behaviors expressible in $\mathcal{L}_2$ can be expressed in $\mathcal{L}_1$)
2. A proof that an encoding $[\![\cdot]\!]_i : \mathcal{L}_1 \to \mathcal{L}_2$ does not exist
   (There are behaviors that $\mathcal{L}_1$ can express but that $\mathcal{L}_2$ cannot)

Correctness criteria for $[\![\cdot]\!]_e$ should be different from those for $[\![\cdot]\!]_i$

- $[\![\cdot]\!]_e$ should satisfy the most demanding criteria possible
  (Ensuring a tight relationship between terms of $\mathcal{L}_1$ and $\mathcal{L}_2$)
- $[\![\cdot]\!]_i$ should satisfy the least demanding criteria possible
  (Ensuring the most general result possible)

Proof techniques and methods are very different, too.

Expressiveness Studies: Goals

To assert that $\mathcal{L}_1$ is more expressive than $\mathcal{L}_2$ one needs to show an encodability and an impossibility result:

1. An encoding $[\![\cdot]\!]_e : \mathcal{L}_2 \to \mathcal{L}_1$
   (All the behaviors expressible in $\mathcal{L}_2$ can be expressed in $\mathcal{L}_1$)
2. A proof that an encoding $[\![\cdot]\!]_i : \mathcal{L}_1 \to \mathcal{L}_2$ does not exist
   (There are behaviors that $\mathcal{L}_1$ can express but that $\mathcal{L}_2$ cannot)

Correctness criteria for $[\![\cdot]\!]_e$ should be different from those for $[\![\cdot]\!]_i$

- $[\![\cdot]\!]_e$ should satisfy the most demanding criteria possible
  (Ensuring a tight relationship between terms of $\mathcal{L}_1$ and $\mathcal{L}_2$)
- $[\![\cdot]\!]_i$ should satisfy the least demanding criteria possible
  (Ensuring the most general result possible)

Proof techniques and methods are very different, too.

More Demanding? Less Demanding? (1)

We illustrate the meaning of demanding correctness criteria for encodings.

A syntactic criteria is compositionality: the encoding of a complex term should depend on the encoding of its parts.

1. Homomorphism wrt parallel: $[\![P \parallel Q]\!] = [\![P]\!] \parallel [\![Q]\!]$
2. Full compositionality: $[\![P \parallel Q]\!] = C([\![P]\!], [\![Q]\!])$, where C is a context of the target language

Clearly, (2) induces a wider class of "good encodings":

- For impossibility results, (2) is more demanding than (1)
  [It entails ensuring the non existence of more encodings]
- For encodability results, (2) is less demanding than (1)
  [Accepts more encodings as good encodings. Not always ideal, e.g. centralized encodings]

More Demanding? Less Demanding? (1)

We illustrate the meaning of demanding correctness criteria for encodings.

A syntactic criteria is compositionality: the encoding of a complex term should depend on the encoding of its parts.

1. Homomorphism wrt parallel: $[\![P \parallel Q]\!] = [\![P]\!] \parallel [\![Q]\!]$
2. Full compositionality: $[\![P \parallel Q]\!] = C([\![P]\!], [\![Q]\!])$, where C is a context of the target language

Clearly, (2) induces a wider class of "good encodings":

- For impossibility results, (2) is more demanding than (1) [It entails ensuring the non existence of more encodings]
- For encodability results, (2) is less demanding than (1) [Accepts more encodings as good encodings. Not always ideal, e.g. centralized encodings]

More Demanding? Less Demanding? (2)

We illustrate the meaning of demanding correctness criteria for encodings.

A semantic criteria is completeness: the behavior of a source term should be preserved by the encoding.

1. if $P \xrightarrow{\alpha} P'$ then $[\![P]\!] \xrightarrow{\alpha} [\![P']\!]$
2. if $P \xrightarrow{\alpha} P'$ then $[\![P]\!] \overset{\alpha}{\Longrightarrow}\approx [\![P']\!]$

Again, (2) induces a wider class of “good encodings”:

- For an impossibility result, (2) is more demanding than (1) [It entails ensuring the non existence of more encodings]
- For an encodability result, (2) is less demanding than (1) [The translation can be hidden via internal behaviors]

More Demanding? Less Demanding? (2)

We illustrate the meaning of demanding correctness criteria for encodings.

A semantic criteria is completeness: the behavior of a source term should be preserved by the encoding.

1. if $P \xrightarrow{\alpha} P'$ then $[\![P]\!] \xrightarrow{\alpha} [\![P']\!]$
2. if $P \xrightarrow{\alpha} P'$ then $[\![P]\!] \overset{\alpha}{\Longrightarrow}\approx [\![P']\!]$

Again, (2) induces a wider class of "good encodings":

- For an impossibility result, (2) is more demanding than (1) [It entails ensuring the non existence of more encodings]
- For an encodability result, (2) is less demanding than (1) [The translation can be hidden via internal behaviors]

Absolute Expressiveness

We classify expressiveness results as absolute or as relative depending on whether or not the expressive power of a given language is analyzed with respect to another language.

Results of absolute expressiveness aim at assessing the expressive power that is intrinsic to the language and its associated semantics

- Focus on the expressiveness of the terms of the language, and on the kind of operators that are expressible in it
- A widespread approach is computational expressiveness: relate calculi to some standard model of computation

Absolute Expressiveness

Computational expressiveness shows the absolute expressiveness of a language by encoding well-known computability models.

The use of Turing complete models is quite frequent:

- Under certain conditions, such an encoding suffices to show that decision problems such as termination are undecidable.
- Models typically used include Random Access Machines (RAMs), Minsky machines (MMs), and Turing machines. [Key: finding ways of modeling counters and operations over them.]

Calculi have been also related to models of computability strictly less expressive than Turing machines

- E.g., context-sensitive, context-free, and regular languages [The language generated by a process: the set of its labelled traces]

Absolute Expressiveness

Computational expressiveness shows the absolute expressiveness of a language by encoding well-known computability models.

The use of Turing complete models is quite frequent:

- Under certain conditions, such an encoding suffices to show that decision problems such as termination are undecidable.
- Models typically used include Random Access Machines (RAMs), Minsky machines (MMs), and Turing machines. [Key: finding ways of modeling counters and operations over them.]

Calculi have been also related to models of computability strictly less expressive than Turing machines

- E.g., context-sensitive, context-free, and regular languages [The language generated by a process: the set of its labelled traces]

Relative Expressiveness

Results of relative expressiveness involve two languages, $\mathcal{L}_1$ and $\mathcal{L}_2$

- A measure of the expressiveness of $\mathcal{L}_1$ taking $\mathcal{L}_2$ as a reference
- For instance, to show that they have the same expressive power (two encodability results) or that one is less expressive than the other (as described before)

It is common to investigate the influence that a particular operator has on the overall expressiveness of $\mathcal{L}_1$

- In this case, the reference language $\mathcal{L}_2$ is the fragment of $\mathcal{L}_1$ without that operator
- One then aims at showing that $\mathcal{L}_1$ cannot be encoded into $\mathcal{L}_2$
- If successful then one has a separation result: the given construct separates the expressiveness $\mathcal{L}_1$ from that of $\mathcal{L}_2$

Relative Expressiveness

Results of relative expressiveness involve two languages, $\mathcal{L}_1$ and $\mathcal{L}_2$

- A measure of the expressiveness of $\mathcal{L}_1$ taking $\mathcal{L}_2$ as a reference
- For instance, to show that they have the same expressive power (two encodability results) or that one is less expressive than the other (as described before)

It is common to investigate the influence that a particular operator has on the overall expressiveness of $\mathcal{L}_1$

- In this case, the reference language $\mathcal{L}_2$ is the fragment of $\mathcal{L}_1$ without that operator
- One then aims at showing that $\mathcal{L}_1$ cannot be encoded into $\mathcal{L}_2$
- If successful then one has a separation result: the given construct separates the expressiveness $\mathcal{L}_1$ from that of $\mathcal{L}_2$

Separation via Representative Problems

An approach to separation results investigates the decidability of some decision problem in the two languages.

Similarly, one can separate two languages based on their ability of solving some well-established problem.
[Useful when the languages are both Turing complete.]

Hence, $\mathcal{L}_1$ is considered to be more expressive than $\mathcal{L}_2$ if the problem is undecidable (or can be solved) in $\mathcal{L}_1$ but not in $\mathcal{L}_2$.

Examples: (We will describe them later.)

- The (un)decidability of termination and convergence separates CCS with recursive definitions from CCS with replication
- Solving the leader election problem separates the (synchronous) π-calculus from its asynchronous fragment

Outline

The Notion of Encoding

We present a number of proposals for precisely characterizing the correctness criteria that encodings should enjoy

- We review them chronologically, in the sequence in which they were proposed
- This helps understanding how the notions evolved, and the reasons underlying such evolution
- Later we will collect some of those criteria to study correctness of known encodings

Sangiorgi (1/3)

1992

In his study on the relationship between first-order and higher-order π-calculus, Sangiorgi identifies three steps to determine that a source language $\mathcal{L}_s$ can be representable into a target language $\mathcal{L}_t$:

1. Formal definition of the semantics of $\mathcal{L}_s$ and $\mathcal{L}_t$;
2. Definition of the encoding from $\mathcal{L}_s$ to $\mathcal{L}_t$;
3. Proof of correctness of the encoding with respect to the semantics given.

Sangiorgi (2/3)

Sangiorgi defines only a syntactic criteria: compositionality.

Given an encoding $[\![\cdot]\!] : \mathcal{L}_s \to \mathcal{L}_t$, and an n-adic construct op of $\mathcal{L}_s$, compositionality can be expressed as follows:

$$[\![\mathsf{op}(P_1, \ldots, P_n)]\!] = C^{\mathsf{op}}[[\![P_1]\!], \ldots, [\![P_n]\!]] \tag{1}$$

where C^{op} is a valid process context in $\mathcal{L}_t$.

His main semantic criteria is full abstraction, i.e., two terms in $\mathcal{L}_s$ should be equivalent if and only if their translations are equivalent:

$$S_1 \approx_s S_2 \text{ if and only } [\![S_1]\!] \approx_t [\![S_2]\!] \,. \tag{2}$$

That is, full-abstraction enforces both preservation and reflection of the equivalence of source terms.

Sangiorgi (3/3)

- Sangiorgi admits that full-abstraction represents a strong approach to representability.
- It is OK, as his goal is to transfer reasoning techniques from the first-order setting to the higher-order one.
- In this sense, full abstraction turns out to be necessary:
    - Target terms should be usable in any context
    - Indistinguishability of two source terms should imply that of their translations
- Still, Sangiorgi acknowledges that full-abstraction alone is not informative enough wrt the relationship between source and target terms.
- To that end, he argues that full-abstraction should be complemented with some form of operational correspondence relating a term and its translation.

Nestmann (1/4)

1996

Based on his works on the encodability of choice operators into the (choice-free) π-calculus, Nestmann collected a number of desirable correctness criteria.

He argues that full abstraction might not applicable in those cases in which $\mathcal{L}_s$ doesn't have a notion of equivalence.

Then, a suitable notion of operational correspondence gains relevance. It is usually expressed as two complementary criteria:

1. Completeness ensures the preservation of execution steps:

$$S_1 \longrightarrow_s S_2 \text{ implies } [\![S_1]\!] \Longrightarrow_t [\![S_2]\!] \,.$$

2. Soundness ensures the reflection of execution steps:

$$[\![S_1]\!] \Longrightarrow_t [\![S_2]\!] \text{ implies } S_1 \Longrightarrow_s S_2 \,.$$

Nestmann (2/4)

However, soundness as in

$$[\![S_1]\!] \Longrightarrow_t [\![S_2]\!] \text{ implies } S_1 \Longrightarrow_s S_2 \,.$$

is not satisfactory: it disregards the intermediate processes the translation of a source term might need to go through in order to simulate its behavior. A refinement is the following:

$$\text{if } [\![S]\!] \longrightarrow_t [\![T]\!] \text{ then there is } S \longrightarrow_s S' \text{ such that } [\![T]\!] \approx_t [\![S']\!] \,.$$

A further refinement to soundness is the one that takes into account the administrative steps that an encoding might have to perform before simulating a step of the source term:

$$\text{if } [\![S]\!] \Longrightarrow_t [\![T]\!] \text{ then there is } S \Longrightarrow_s S' \text{ such that } [\![T]\!] \Longrightarrow_t [\![S']\!] \,.$$

Nestmann (3/4)

Besides full-abstraction and operational correspondence, Nestmann considers preservation/reflection of deadlocks and divergence.

Reflecting deadlocks is quite natural: the translation of a term should not deadlock if the given source term does not deadlock.

Preserving deadlocks is also reasonable as long as administrative steps in the target side that might precede deadlock are taken into account.

Nestmann (4/4)

As for divergence, Nestmann finds an encoding that adds divergence perfectly acceptable.

This is motivated by his two encodings of the π-calculus with input-guarded choice into the choice-free fragment:

- one encoding is atomic wrt choice but introduces divergence;
- the other encoding is divergence-free but replaces atomic commitment of choice with gradual commitment.

Therefore, there could be scenarios in which correct encodings that add divergence might still be worth having.

Palamidessi (1/3)

Around 1996–2000

As part of her work on the comparison of the expressive power of synchronous and asynchronous communication in the π-calculus, Palamidessi proposed a notion of uniformity to capture syntactic criteria:

1. homomorphism with respect to parallel composition, i.e., $[\![P \parallel Q]\!] = [\![P]\!] \parallel [\![Q]\!]$;
2. preservation of renaming, i.e. for any permutation of names σ in the domain of $\mathcal{L}_s$, there exists a permutation θ in the domain of $\mathcal{L}_t$ such that, for all name i,

$$\sigma(i) = \theta(i) \text{ and } [\![\sigma(P)]\!] = \theta([\![P]\!])$$

[Intuition: encodings should not depend on the choice of names.]

Palamidessi (2/3)

Requiring homomorphism with respect to parallel composition rather than full compositionality (as Sangiorgi's) can be considered too strong a criterion.

Palamidessi argues that such a criterion is essential to ensure that the encoding preserves the degree of distribution of the system, i.e. encodings of distributed system do not add global coordinating processes (or sites).

Palamidessi (3/3)

In Palamidessi's results, encodings are required to be semantically reasonable. That is, encodings are required to preserve

> *a semantics which distinguishes two processes P and Q whenever there exists a (finite or infinite) computation of P in which the intended observables (some visible actions) are different from the observables in any (maximal) computation of Q.*

This is quite a liberal way of capturing requirements such as operational correspondence and the reflection/preservation of deadlocks and divergence, discussed above.

"Good encodings" are both uniform and semantically reasonable.

Palamidessi (3/3)

In Palamidessi's results, encodings are required to be semantically reasonable. That is, encodings are required to preserve

> *a semantics which distinguishes two processes P and Q whenever there exists a (finite or infinite) computation of P in which the intended observables (some visible actions) are different from the observables in any (maximal) computation of Q.*

This is quite a liberal way of capturing requirements such as operational correspondence and the reflection/preservation of deadlocks and divergence, discussed above.

"Good encodings" are both uniform and semantically reasonable.

Gorla (1/2)

2006

Gorla defined an abstract meta-theory for reasoning about relative expressiveness, which synthesizes most of the criteria just discussed.

His syntactic criteria include

- Full compositionality, with the context parametrized by the set of free names of the source terms
- A generalization of the name invariance condition by Palamidessi
  [It considers so-called renaming policies]

Gorla (2/2)

Semantic criteria include

- a form of operational correspondence that is defined up to the "garbage terms" that an encoding might produce;
- divergence reflection, that is, that the encoding does not add divergence;
- success sensitiveness, i.e., a criteria which, based on some notion of "success" ensures that a successful source term is mapped into a successful target term.
  [A sensible notion of success is barbs.]

Advantage: this proposal can be exploited by diverse concurrent languages (with different behavioral equivalences).

The proposal has been instantiated so as to obtain simplified proofs for known results, and to obtain new ones.

Outline

Encodability Results

We now revisit (or present) some encodings

- Polyadic into monadic π-calculus
- π with recursive definitions into π with replication
- Synchronous π into asynchronous π

and analyze their correctness in the light of some of the criteria just presented, namely

- Preservation of behavioral equivalence
- Preservation of observations (barbs)
- Operational correspondence
- Full abstraction
- Compositionality

Encodability Results

We now revisit (or present) some encodings

- Polyadic into monadic π-calculus
- π with recursive definitions into π with replication
- Synchronous π into asynchronous π

and analyze their correctness in the light of some of the criteria just presented, namely

- Preservation of behavioral equivalence
- Preservation of observations (barbs)
- Operational correspondence
- Full abstraction
- Compositionality

Encodings: $[\![\cdot]\!]_{\mathsf{pc}} : \text{poly}\pi \to \text{mona}\pi$

[Milner, 91]

The encoding $[\![\cdot]\!]_{\mathsf{pc}} : \text{poly}\pi \to \text{mona}\pi$ is defined as

$$\begin{array}{rcl} [\![x(y_1 \cdots y_n).P]\!]_{\mathsf{pc}} & = & x(w).w(y_1).\cdots.w(y_n).[\![P]\!]_{\mathsf{pc}} \\ [\![\overline{x}\langle z_1 \cdots z_n\rangle.Q]\!]_{\mathsf{pc}} & = & (\nu w)(\overline{x}\langle w\rangle.\overline{w}\langle z_1\rangle.\cdots.\overline{x}\langle z_n\rangle.[\![Q]\!]_{\mathsf{pc}}) \end{array}$$

and the other operators are defined homomorphically

Encodings: $[\![\cdot]\!]_{\mathsf{rd}} : \pi_{\mathsf{rd}} \to \pi_{!}$

The encoding $[\![\cdot]\!]_{\mathsf{rd}} : \pi_{\mathsf{rd}} \to \pi_{!}$ is defined as

$$[\![P]\!]_{\mathsf{rd}} \;=\; (\nu a_1, \ldots, a_k)\Big([\![P]\!]_{\mathsf{rd}_0} \parallel \prod_{i \in \{1,\ldots,k\}} [\![A_i(\tilde{x}_i) \overset{\text{def}}{=} Q_i]\!]_{\mathsf{rd}_0}\Big)$$

where $a_1, \ldots, a_k \notin \mathsf{fn}(P)$ and

$$\begin{aligned}
[\![A_i(\tilde{x}_i) \overset{\text{def}}{=} Q_i]\!]_{\mathsf{rd}_0} &= \; !a_i(\tilde{x}_i).[\![Q_i]\!]_{\mathsf{rd}_0} \\
[\![A\langle\tilde{z}\rangle]\!]_{\mathsf{rd}_0} &= \; \overline{a_i}\langle\tilde{z}\rangle
\end{aligned}$$

$[\![\cdot]\!]_{\mathsf{rd}_0}$ is an homomorphism for the other cases.

Encodings: $[\![\cdot]\!]_{\mathsf{sy}} : A\pi \to \pi$

[Boudol, 92]

The encoding $[\![\cdot]\!]_{\mathsf{sy1}} : A\pi \to \pi$ is defined as

$$\begin{array}{rcl} [\![\overline{x}\langle z\rangle.P]\!]_{\mathsf{sy1}} & = & (\nu w)(\overline{x}\langle w\rangle \parallel w(u).(\overline{u}\langle z\rangle \parallel [\![P]\!]_{\mathsf{sy1}})) \\ [\![x(y).Q]\!]_{\mathsf{sy1}} & = & x(w).(\nu u)(\overline{w}\langle u\rangle \parallel u(y).[\![Q]\!]_{\mathsf{sy1}}) \end{array}$$

[Honda and Tokoro, 92]

The encoding $[\![\cdot]\!]_{\mathsf{sy2}} : A\pi \to \pi$ is defined as

$$\begin{array}{rcl} [\![\overline{x}\langle z\rangle.P]\!]_{\mathsf{sy2}} & = & x(w).(\overline{w}\langle z\rangle \parallel [\![P]\!]_{\mathsf{sy2}}) \\ [\![x(y).Q]\!]_{\mathsf{sy2}} & = & (\nu w)(\overline{x}\langle w\rangle \parallel w(y).[\![Q]\!]_{\mathsf{sy2}}) \end{array}$$

Encodings: $[\![\cdot]\!]_{\mathsf{sy}} : A\pi \to \pi$

[Boudol, 92]

The encoding $[\![\cdot]\!]_{\mathsf{sy1}} : A\pi \to \pi$ is defined as

$$\begin{array}{rcl} [\![\overline{x}\langle z\rangle.P]\!]_{\mathsf{sy1}} & = & (\nu w)(\overline{x}\langle w\rangle \parallel w(u).(\overline{u}\langle z\rangle \parallel [\![P]\!]_{\mathsf{sy1}})) \\ [\![x(y).Q]\!]_{\mathsf{sy1}} & = & x(w).(\nu u)(\overline{w}\langle u\rangle \parallel u(y).[\![Q]\!]_{\mathsf{sy1}}) \end{array}$$

[Honda and Tokoro, 92]

The encoding $[\![\cdot]\!]_{\mathsf{sy2}} : A\pi \to \pi$ is defined as

$$\begin{array}{rcl} [\![\overline{x}\langle z\rangle.P]\!]_{\mathsf{sy2}} & = & x(w).(\overline{w}\langle z\rangle \parallel [\![P]\!]_{\mathsf{sy2}}) \\ [\![x(y).Q]\!]_{\mathsf{sy2}} & = & (\nu w)(\overline{x}\langle w\rangle \parallel w(y).[\![Q]\!]_{\mathsf{sy2}}) \end{array}$$

Correctness Criteria: Beh. Equivalences

Strong and Weak Barbs

We write $P \downarrow \mu$ ($\mu \in \{x, \overline{x}\}$) iff $\exists \tilde{z}, Q, R$ such that $x \notin \tilde{z}$ and $P \equiv (\nu \tilde{z})(\pi.Q \parallel R)$ and $\pi = x(y)$ (if $\mu = x$) and $\pi = \overline{x}\langle y \rangle$ (otherwise).

Also, we write $P \Downarrow \mu$ iff $\exists Q$ such that $P \longrightarrow^* Q$ and $Q \downarrow \mu$.

Barbed Bisimilarity and Congruence

- A relation $\mathcal{R}$ is a barbed bisimulation if for every $(P, Q) \in \mathcal{R}$:
  - if $P \longrightarrow P'$ then $\exists Q'$ such that $Q \Longrightarrow Q'$ and $(P', Q') \in \mathcal{R}$;
  - if $P \downarrow \mu$ then $Q \Downarrow \mu$
- Processes P, Q are barbed bisimilar, written $P \approx Q$, if $P \mathcal{R} Q$, for some barbed bisimilarity $\mathcal{R}$.
- Processes P, Q are barbed congruent, written $P \simeq Q$, if $C(P) \approx C(Q)$, for every context C.

Preservation of behavioral equivalence

Preservation wrt $\bowtie$

For all $\mathcal{L}_s$, it should hold that $P \bowtie [\![P]\!]$

- Typically $\bowtie$ is some bisimilarity relation.
- It could be a very strong correspondence depending on the chosen $\bowtie$
- But it presupposes that both $\mathcal{L}_s$ and $\mathcal{L}_t$ are equipped with $\bowtie$

We have that

- $[\![\cdot]\!]_{\mathsf{pc}} : \text{poly}\pi \to \text{mona}\pi$ satisfies it when $\bowtie$ is weak barbed bisimulation
- $[\![\cdot]\!]_{\mathsf{rd}} : \pi_{\mathsf{rd}} \to \pi_{!}$ satisfies it when $\bowtie$ is barbed congruence

Preservation of observations (barbs)

Preservation of observations (cf. Gorla's success sensitiveness)

For all $P \in \mathcal{L}_s$, it should hold that $\mathsf{Obs}(P) = \mathsf{Obs}([\![P]\!])$

Above, we use $\mathsf{Obs}(\cdot)$ to denotes a set of observations that can be made of processes in $\mathcal{L}_s, \mathcal{L}_t$.
A simple instance of such observations are barbs.

While helpful, preservation of observables is not enough to describe preservation of behavior.

We have that both $[\![\cdot]\!]_{\mathsf{pc}} : \mathsf{poly}\pi \to \mathsf{mona}\pi$ and $[\![\cdot]\!]_{\mathsf{rd}} : \pi_{\mathsf{rd}} \to \pi_{!}$ satisfy this criteria, for barbs.

Operational correspondence

Operational Correspondence

For all $P, Q \in \mathcal{L}_s$,

- If $P \longrightarrow Q$ then $\exists Q' \in \mathcal{L}_t$ such that $[\![P]\!] \longrightarrow^* Q' \bowtie [\![Q]\!]$
- If $[\![P]\!] \longrightarrow R$ then $\exists R' \in \mathcal{L}_s$ such that $R \bowtie [\![R']\!]$

- Very operational flavor: preservation and reflection of reduction steps
- It can be also defined with labeled transitions. But combining the above definition with preservation of observables is usually enough.
- We have that both $[\![\cdot]\!]_{\mathsf{pc}}$: poly$\pi \to$ monaπ and $[\![\cdot]\!]_{\mathsf{rd}} : \pi_{\mathsf{rd}} \to \pi_!$ satisfy this criteria, when $\bowtie$ is barbed congruence

Full abstraction

Full abstraction

For all $P, Q \in \mathcal{L}_s$, $P \bowtie_s Q$ if and only if $[\![P]\!] \bowtie_t [\![Q]\!]$.

- Equivalent processes are translated into equivalent processes
- If direction is called soundness (or adequacy); only if direction is called completeness.
- We have that $[\![\cdot]\!]_{\mathsf{pc}} : \mathsf{poly}\pi \to \mathsf{mona}\pi$ is only sound, when $\bowtie$ is barbed congruence
- We have that $[\![\cdot]\!]_{\mathsf{rd}} : \pi_{\mathsf{rd}} \to \pi_{!}$ is fully abstract, also when $\bowtie$ is barbed congruence

Compositionality

Compositionality

Let $[\![\cdot]\!]_c : \mathcal{L}_s \to \mathcal{L}_t$ be an encoding. We have

1. $[\![\cdot]\!]_c$ is compositional wrt an n-ary operator op if and only if there exists a context $C \in \mathcal{L}_t$ with n-holes such that $[\![op(P_1, \ldots, P_n)]\!] = C([\![P_1]\!], \ldots, [\![P_n]\!])$
2. $[\![\cdot]\!]_c$ is weakly compositional iff $\exists C$ such that $\forall P$ we have $[\![P]\!] = C([\![P]\!]_0)$ where $[\![\cdot]\!]_0$ is compositional.
3. $[\![\cdot]\!]_c$ is homomorphic wrt an n-ary operator op in $\mathcal{L}_s$ if and only if $[\![op(P_1, \ldots, P_n)]\!] = op([\![P_1]\!], \ldots, [\![P_n]\!])$

- We have already discussed drawbacks and possible justifications for using homomorphism wrt parallel
- $[\![\cdot]\!]_{\mathsf{pc}} : \text{poly}\pi \to \text{mona}\pi$ is compositional for all operators
- $[\![\cdot]\!]_{\mathsf{rd}} : \pi_{\mathsf{rd}} \to \pi_{!}$ is only weakly compositional

Outline

Separation Results

We overview two “classic” separation results in process calculi:

- The (synchronous) π-calculus is strictly more expressive than its asynchronous variant
- CCS with recursive definitions is strictly more expressive than CCS with replication

Asynchronous π-calculus

In the mid 90s there was a great interest about the expressive power and behavioral theory of the asynchronous π-calculus ($A\pi$).

- The simplest, non trivial fragment of the π-calculus
- The choice-free π-calculus is encodable in $A\pi$ (cf. $[\![\cdot]\!]_{\mathsf{sy1}}$)
- The combination of guarded choices and synchronous communication proved to be tricky to represent
- In particular, the fact that the (standard, full) π-calculus implements mixed choices. That is, in a sum

$$\sum_{i \in I} \alpha_i . P_i$$

each α_i can be an input $x_i(y_i)$ or an output $\overline{x_j}\langle z_j \rangle$.

Asynchronous π-calculus

Two variants of guarded choices were then explored:

- π_{imp}, π with input-guarded choice: sums are only of the form

$$\sum_{i \in I} x_i(y_i).P_i$$

- π_{sep}, π with separate choice: sums are either of the form

$$\sum_{i \in I} x_i(y_i).P_i \quad \text{or} \quad \sum_{j \in J} \overline{x_j}\langle z_j \rangle.P_j$$

Separation between π and $A\pi$

The most famous separation result, by Palamidessi:
There is no "good" encoding of π_{mix} (full π) into π_{sep} and $A\pi$

The proof uses the leader election problem

- A problem related to consensus in distributed setting
- She proved that there is no symmetric network in π_{sep} that solves leader election, whereas there are such networks in π_{mix}
- Her (intricate) proof exploits the ability that π_{mix} has for breaking symmetries; an ability that π_{sep} lacks
  Example: a parallel composition of symmetric choices:

$$P \parallel Q = \overline{y_0}\langle 0\rangle.P_0 + y_1(x).P_1 \parallel y_0(x).Q_0 + \overline{y_1}\langle 1\rangle.Q_1$$

  (P and Q are identical up to $\equiv$ and renamings.)
- In π_{inp}, π_{sep} such scenarios correspond to confluent executions.
  "Good encodings" preserve this property; symmetries are never broken.

Separation between π and $A\pi$

The most famous separation result, by Palamidessi:

There is no "good" encoding of π_{mix} (full π) into π_{sep} and $A\pi$

The proof uses the leader election problem

- A problem related to consensus in distributed setting
- She proved that there is no symmetric network in π_{sep} that solves leader election, whereas there are such networks in π_{mix}
- Her (intricate) proof exploits the ability that π_{mix} has for breaking symmetries; an ability that π_{sep} lacks
  Example: a parallel composition of symmetric choices:

$$P \parallel Q = \overline{y_0}\langle 0\rangle.P_0 + y_1(x).P_1 \parallel y_0(x).Q_0 + \overline{y_1}\langle 1\rangle.Q_1$$

  (P and Q are identical up to $\equiv$ and renamings.)
- In π_{inp}, π_{sep} such scenarios correspond to confluent executions. "Good encodings" preserve this property; symmetries are never broken.

Separation is in the eye of the beholder (1)

Recall that Palamidessi requires "good encodings" which, among other things, translate $\parallel$ homomorphically and preserve divergence.

Nestmann showed that there is a good encoding from π_{sep} to $A\pi$

- The encoding, based on locks to handle sums, combines independent encodings for output- and input-guarded sums
- The encoding is "good", in particular it does not add divergences

How to break symmetries? Nestmann proposes two not-so-good encodings of π_{mix} into $A\pi$:

- The first extends the encoding of π_{sep} into $A\pi$ with randomized choices. It is uniform but it may add divergences.
- The second introduces a centralized way of enforcing a total ordering. It is divergence-free but not uniform.

Separation is in the eye of the beholder (2)

Recall that Palamidessi requires “good encodings” which, among other things, translate $\parallel$ homomorphically and preserve divergence.

Recently, Peters and Nestmann have shown an encodability result of π_{mix} into $A\pi$, by adopting Gorla's definition of compositionality for parallel (i.e., $[\![P \parallel Q]\!] = C([\![P]\!], [\![Q]\!])$, for some $A\pi$ context C).

- The encoding is quite sophisticated, and relies on implementing distributed mixed choices using local coordination
- If the definition of encoding is extended to include a criteria on distribution degree of parallel components, then a separation result is again recovered.

Infinite Behavior in CCS

Historically, CCS had been presented and used with different forms of representing infinite behavior. The most relevant two are

- Recursive definitions
- Replication

Busi, Gabbrielli, and Zavattaro compared the expressiveness of CCS^d and $\mathrm{CCS}^!$, the variants of CCS which use recursive definitions and replication, respectively.

Infinite Behavior in CCS

Their main result is that CCS^{d} is strictly more expressive than $\mathrm{CCS}^{!}$. It is based on the (un)decidability of two decision problems:

Termination: the non existence of diverging computations (Universal termination)

Convergence: the existence of a terminating computation (Existential termination)

While convergence is undecidable in both CCS^{d} and $\mathrm{CCS}^{!}$, termination is undecidable in CCS^{d} but decidable in $\mathrm{CCS}^{!}$.

This defines a gap between recursive definitions and replication. Intuitively, the "in depth" infinite behavior of CCS^{d} is more expressive than the "in width" behavior expressible with $\mathrm{CCS}^{!}$.

Convergence and Termination

We denote with $\longrightarrow^*$ the reflexive and transitive closure of $\longrightarrow$.
We use $P \not\longrightarrow$ to denote that there is no P' such that $P \longrightarrow P'$

Definition

Let P be a process.

- P converges iff there exists a P' such that $P \longrightarrow^* P'$ and $P' \not\longrightarrow$.
- P terminates iff there exist no $\{P_i\}_{i \in \mathbb{N}}$ such that $P_0 = P$ and $P_j \longrightarrow P_{j+1}$ for any j.

Note: Termination implies convergence, but the opposite doesn't hold.

Separating Infinite Behavior in CCS

In other words, BGZ proved the following:

	CCS^d	$\mathrm{CCS}^!$
termination	undecidable	decidable
convergence	undecidable	undecidable

[They also proved other results, concerning barbs and weak bisimulation.]

- The undecidability results are obtained by using (termination preserving) encodings of Turing machines
- The decidability of termination is derived via the theory of well-structured transition systems

The CCS variants

We define finite core CCS via the following grammar:

$$\begin{array}{rcl} P & ::= & \mathbf{0} \mid \alpha.P \mid P+P \mid P \parallel P \mid (\nu x)P \\ \alpha & ::= & \tau \mid x \mid \overline{x} \end{array}$$

[The LTS is as we have discussed before.]

1. CCS^d extends this grammar with the production $P ::= D$. Each D is assumed to have a defining equation $D \stackrel{\text{def}}{=} P$. The LTS is extended with the rule:

$$\frac{P \xrightarrow{\alpha} P' \quad D \stackrel{\text{def}}{=} P}{D \xrightarrow{\alpha} P'}$$

2. $\mathrm{CCS}^!$ extends this grammar with the production $P ::= !P$. The LTS is extended with the rule:

$$P \parallel !P \xrightarrow{\alpha} P'$$

The CCS variants

We define finite core CCS via the following grammar:

$$\begin{array}{rcl} P & ::= & \mathbf{0} \mid \alpha.P \mid P + P \mid P \parallel P \mid (\nu x)P \\ \alpha & ::= & \tau \mid x \mid \overline{x} \end{array}$$

[The LTS is as we have discussed before.]

1. CCS^d extends this grammar with the production $P ::= D$. Each D is assumed to have a defining equation $D \overset{\mathrm{def}}{=} P$. The LTS is extended with the rule:

$$\frac{P \xrightarrow{\alpha} P' \quad D \overset{\mathrm{def}}{=} P}{D \xrightarrow{\alpha} P'}$$

2. $\mathrm{CCS}^!$ extends this grammar with the production $P ::= !P$. The LTS is extended with the rule:

$$P \parallel !P \xrightarrow{\alpha} P'$$

Minsky machines

Rather than encoding "full" Turing machines (a tape, symbols, etc.), a relationship with Turing complete models is easier to obtain by encoding Minsky machines (MMs).

Minsky machines

A counter machine 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 instruction being executed.

Encoding MMs in CCS^d

We describe how termination of MMs can be reduced to convergence and termination of CCS^d processes.

The key idea is to encode numbers as a chain of nested restrictions, with a length corresponding to the content of the register.

Encoding MMs in CCS^d

Each register r_j is represented by a constant Z_j, which uses other two constants, noted O_j and E_j:

$$\begin{aligned}
Z_j &\overset{\text{def}}{=} zero_j.Z_j + inc_j.(\nu x)(O_j \parallel x.\overline{ack}.Z_j)\\
O_j &\overset{\text{def}}{=} dec_j.\overline{x} + inc_j.(\nu y)(E_j \parallel y.\overline{ack}.O_j)\\
E_j &\overset{\text{def}}{=} dec_j.\overline{y} + inc_j.(\nu x)(O_j \parallel x.\overline{ack}.E_j)
\end{aligned}$$

Intuitively, O_j (resp. E_j) models the state of r_j when it holds an odd (resp. even) number.

Instructions $(i : I_i)$ are modeled with a definition $Inst_i$, as follows:

$$\begin{aligned}
Inst_i &\overset{\text{def}}{=} \overline{inc_j}.Inst_{i+1} && \text{if } I_i = \mathsf{INC}(r_j))\\
Inst_i &\overset{\text{def}}{=} \overline{dec_j}.ack.Inst_{i+1} + \overline{zero_j}.Inst_s && \text{if } I_i = \mathsf{DECJ}(r_j))
\end{aligned}$$

Encoding MMs in CCS^d: Example (1/3)

Consider a MM with registers r_1, r_2 and the following program:

$(1 : \mathsf{INC}(r_1)) \quad (2 : \mathsf{INC}(r_1)) \quad (3 : \mathsf{DECJ}(r_1, 5)) \quad (4 : \mathsf{DECJ}(r_2, 3))$

[Note: 5 is an undefined instruction]

BGZ formalize MMs with the following process:

$$\begin{aligned} P &= Inst_1 \parallel Z_1 \parallel Z_2 \\ &= \overline{inc_1}.Inst_2 \parallel zero_1.Z_1 + inc_j.(\nu x)(O_1 \parallel x.\overline{ack}.Z_1) \parallel Z_2 \end{aligned}$$

We have that P has the following 2 deterministic reduction steps corresponding to the two increment instructions:

$$\begin{aligned} P &\longrightarrow Inst_2 \parallel (\nu x)(O_1 \parallel x.\overline{ack}.Z_1) \parallel Z_2 \\ &\longrightarrow Inst_3 \parallel (\nu x)((\nu y)(E_1 \parallel y.\overline{ack}.O_1) \parallel x.\overline{ack}.Z_1) \parallel Z_2 \; = \; Q \end{aligned}$$

At this point, r_1 contains the value 2; this is represented by the nesting of the two restrictions on the names x and y.

Encoding MMs in CCS^d: Example (2/3)

Let $\equiv_R$ be a congruence that discards $\mathbf{0}$s and unused restrictions.

The computation continues with the following 3 deterministic reduction steps corresponding to the first decrement:

$$\begin{aligned}
Q \;&\longrightarrow\; ack.Inst_4 \parallel (\nu x)((\nu y)(\overline{y} \parallel y.\overline{ack}.O_1) \parallel x.\overline{ack}.Z_1) \parallel Z_2 \\
&\longrightarrow\; ack.Inst_4 \parallel (\nu x)((\nu y)(\mathbf{0} \parallel \overline{ack}.O_1) \parallel x.\overline{ack}.Z_1) \parallel Z_2 \\
&\longrightarrow\; Inst_4 \parallel (\nu x)((\nu y)(\mathbf{0} \parallel O_1) \parallel x.\overline{ack}.Z_1) \parallel Z_2 \\
&\equiv_R\; Inst_4 \parallel (\nu x)(O_1 \parallel x.\overline{ack}.Z_1) \parallel Z_2 \;=\; R
\end{aligned}$$

At this point, r_j contains the value 1; this is represented by the fact that the inner restriction on y can be removed by $\equiv_R$.

Encoding MMs in CCS^d: Example (3/3)

The computation is then completed by the following 6 deterministic reduction steps:

$$\begin{array}{rl}
R & \longrightarrow\ Inst_3 \parallel (\nu x)(O_1 \parallel x.\overline{ack}.Z_1) \parallel Z_2 \\
 & \longrightarrow\ ack.Inst_4 \parallel (\nu x)(\overline{x} \parallel x.\overline{ack}.Z_1) \parallel Z_2 \\
 & \longrightarrow\ ack.Inst_4 \parallel (\nu x)(\mathbf{0} \parallel \overline{ack}.Z_1) \parallel Z_2 \\
 & \longrightarrow\ Inst_4 \parallel (\nu x)(\mathbf{0} \parallel Z_1) \parallel Z_2 \ \equiv_R Inst_4 \parallel Z_1 \parallel Z_2 \\
 & \longrightarrow\ Inst_3 \parallel Z_1 \parallel Z_2 \\
 & \longrightarrow\ Inst_5 \parallel Z_1 \parallel Z_2
\end{array}$$

where this last process is dead (i.e., it cannot reduce) because $Inst_5 \overset{\text{def}}{=} \mathbf{0}$.

Undecidability of Convergence for CCS^d

It is not difficult to prove a strong operational correspondence between MMs and their encoding in CCS^d.

Precisely, it can be shown that

1. If the program counter holds the label of an inexistent instruction, then the encoding cannot reduce
2. If the MM can reduce, then the encoding can reduce as well and the resulting states are related

Based on this correspondence, it can be shown that a MM terminates if and only if its encoding in CCS^d converges.

- Since termination for MMs is undecidable, this means that convergence for CCS^d process is undecidable
- Since the encoding preserves terminating computations (item (1) above), termination is undecidable as well

Encoding MMs in $\mathrm{CCS}^!$

The proof that convergence is undecidable for $\mathrm{CCS}^!$ process is also based on a reduction from termination of MMs.

However, the encoding is much more involved, as a deterministic encoding is not possible.

- Counting based on "units" placed on parallel (not nested).
- A non deterministic encoding of MMs, which introduces computations that the MM does not feature.
- All such wrong computations (jumps when the register is not zero), however, are guaranteed to be infinite
- This ensures that, given a MM, its encoding has a terminating computation if and only if the MM terminates.
- This proves that convergence is undecidable

We now give the main ideas of the encoding.

Encoding MMs in $\mathrm{CCS}^!$ (1/4)

Let R be a MM with registers r_1, r_2, and instructions $(1 : I_1), \ldots, (m : I_m)$.

- The program counter is modeled with a process $\overline{p_i}$ indicating that the i-th instruction is the next to be executed.
- For each $1 \leq i \leq m$, the i-th instruction $(i : I_i)$ of R is a replicated process, guarded by an input on p_i.
- Once active, the instruction first performs its operation on the register. Then, it waits for an acknowledgment indicating that the operation has been performed. Finally, it updates the program counter by producing $\overline{p_{i+1}}$ (or $\overline{p_k}$ in case of jump).

Encoding MMs in $\mathrm{CCS}^{!}$ (2/4)

The instruction $(i : I_i)$ is modeled by $[\![(i : I_i)]\!]$ which is a shorthand notation for the following processes.

$$[\![(i : I_i)]\!] \quad = \quad !p_i.(\overline{inc_j} \parallel ack.\overline{p_{i+1}}) \qquad \text{if } I_i = \mathsf{INC}(r_j)$$

$$[\![(i : I_i)]\!] \quad = \quad !p_i.(\overline{dec_j} \parallel (\underbrace{ack.\overline{p_{i+1}} \; + jmp.ack.\overline{p_k}})) \qquad \text{if } I_i = \mathsf{DECJ}(r_j,$$

Encoding MMs in $\mathrm{CCS}^!$ (2/4)

The instruction $(i : I_i)$ is modeled by $[\![(i : I_i)]\!]$ which is a shorthand notation for the following processes.

$$[\![(i : I_i)]\!] = !p_i.(\overline{inc_j} \parallel ack.\overline{p_{i+1}}) \qquad \text{if } I_i = \mathsf{INC}(r_j)$$

$$[\![(i : I_i)]\!] = !p_i.(\overline{dec_j} \parallel (\underbrace{ack.\overline{p_{i+1}} + jmp.ack.\overline{p_k}}_{\text{source of non-determinism}})) \qquad \text{if } I_i = \mathsf{DECJ}(r_j,$$

Encoding MMs in $\mathrm{CCS}^{!}$ (3/4)

- The content of the registers is modeled by the parallel composition of a corresponding number of processes $(\overline{u} \parallel d.u.\overline{m})$.
- An $\overline{u}$ stands for a unit inside the register. The term $d.u.\overline{m}$ is responsible for removing the unit in the case of a decrement.
- Non-determinism arises in the possibility of wrong jumps, which may occur even if the register is not empty.

Encoding MMs in $\mathrm{CCS}^!$ (4/4)

The encoding of register r_j with content c_j is as follows:

$$
\begin{aligned}
[\![r_j = c_j]\!] = (\nu\ m, u)\ \big(& \\
& \textstyle\prod_{c_j} \overline{u} \parallel \\
& inc_j.(\overline{m} \parallel \overline{u}) + dec_j.(u.\overline{m} + \overline{jmp}.(u.DIV \parallel \overline{nr_j})) \parallel
\end{aligned}
$$

Above:

- DIV stands for $\overline{w'} \parallel !w'.\overline{w'}$.
- After an instruction, the register needs to be recreated. Increment and decrement operations do it by an output $\overline{m}$; jump operations do it by an output $\overline{nr_1}$.

Encoding MMs in CCS$^!$ (4/4)

The encoding of register r_j with content c_j is as follows:

$$
\begin{aligned}
[\![r_j = c_j]\!] = (\nu\, m, u)\ \big(& \\
& \textstyle\prod_{c_j} \overline{u} \parallel \\
& inc_j.(\overline{m} \parallel \overline{u}) + dec_j.(u.\overline{m} + \overline{jmp}.(u.DIV \parallel \overline{nr_j})) \parallel \\
& !m.(\overline{ack} \parallel inc_j.(\overline{m} \parallel \overline{u}) + dec_j.(u.\overline{m} + \overline{jmp}.(u.DIV \parallel \overline{nr_j})))\ \big) \parallel \\
& !nr_j.(\nu\, m, u)\ \big(\\
& \overline{m} \parallel !m.(\overline{ack} \parallel inc_j.(\overline{m} \parallel \overline{u}) + dec_j.(u.\overline{m} + \overline{jmp}.(u.DIV \parallel \overline{nr_j})))
\end{aligned}
$$

Above:

- DIV stands for $\overline{w'} \parallel !w'.\overline{w'}$.
- After an instruction, the register needs to be recreated. Increment and decrement operations do it by an output $\overline{m}$; jump operations do it by an output $\overline{nr_1}$.

Encoding MMs in $\mathrm{CCS}^!$ (4/4)

The encoding of register r_j with content c_j is as follows:

$$\begin{aligned}\llbracket r_j = c_j \rrbracket = (\nu\ m, u)\ (&\\ &\textstyle\prod_{c_j} \overline{u} \parallel \\ &inc_j.(\overline{m} \parallel \overline{u}) + dec_j.(u.\overline{m} + \overline{jmp}.(u.DIV \parallel \overline{nr_j})) \parallel\end{aligned}$$

Observe:

- DIV is guarded by an input on u. This ensures that it is only launched when the jump is incorrectly invoked.
- The encoding of jumps always leaves some garbage, even in the case in which the jump is correctly done.
  This garbage can be ignored using structural congruence.

Encoding MMs in CCS$^!$ (4/4)

The encoding of register r_j with content c_j is as follows:

$$
\begin{aligned}
[\![r_j = c_j]\!] = (\nu\, m, u) \ \big(& \\
& \textstyle\prod_{c_j} \overline{u} \parallel \\
& inc_j.(\overline{m} \parallel \overline{u}) + dec_j.(u.\overline{m} + \overline{jmp}.(u.DIV \parallel \overline{nr_j})) \parallel \\
& !m.(\overline{ack} \parallel inc_j.(\overline{m} \parallel \overline{u}) + dec_j.(u.\overline{m} + \overline{jmp}.(u.DIV \parallel \overline{nr_j}))) \ \big) \parallel \\
& !nr_j.(\nu\, m, u) \ \big(\\
& \overline{m} \parallel !m.(\overline{ack} \parallel inc_j.(\overline{m} \parallel \overline{u}) + dec_j.(u.\overline{m} + \overline{jmp}.(u.DIV \parallel \overline{nr_j}))) \big)
\end{aligned}
$$

Observe:

- DIV is guarded by an input on u. This ensures that it is only launched when the jump is incorrectly invoked.
- The encoding of jumps always leaves some garbage, even in the case in which the jump is correctly done.
  This garbage can be ignored using structural congruence.

Undecidability of Convergence for $\mathrm{CCS}^{!}$

Based on the previous observations, an operational correspondence result relating a MM and its encoding into $\mathrm{CCS}^{!}$ can be stated.

In turn, such a result allows to conclude that convergence is undecidable in $\mathrm{CCS}^{!}$.

However, differently from the encoding of MMs into CCS^{d}, we cannot conclude that also termination is undecidable.

- This is because the presented encoding adds infinite computations even to processes encoding a terminating MM.
- Hence, it is no longer true that for these processes convergence and termination coincide.

Decidability of Termination for CCS$^!$

BGZ prove decidability of termination by exploiting the theory of well-structured transition systems (WSTSs).

The proof is very technical and requires quite a lot of background.
We give some very high-level ideas.

Decidability of Termination for CCS$^!$

Intuitively, well-structured transition system is a transition system enriched with an ordering relation over the set of states.

Definition (Well-structured transition system)

A well-structured transition system with strong compatibility is a transition system $TS = (S, \rightarrow, \leq)$ such that:

1. $(S, \rightarrow)$ is a transition system
2. $\leq \subseteq S \times S$ is a simulation relation and a well-quasi-order (wqo):
   - Every infinite sequence of S has two comparable states
   - S has no infinite antichains

If $\leq$ is a simulation, then it's said to be strongly compatible wrt $\rightarrow$.

Decidability of Termination for CCS$^!$

Intuitively, well-structured transition system is a transition system enriched with an ordering relation over the set of states.

Definition (Well-structured transition system)

A well-structured transition system with strong compatibility is a transition system $TS = (S, \rightarrow, \leq)$ such that:

1. $(S, \rightarrow)$ is a transition system
2. $\leq \subseteq S \times S$ is a simulation relation and a well-quasi-order (wqo):
   - Every infinite sequence of S has two comparable states
   - S has no infinite antichains

If $\leq$ is a simulation, then it's said to be strongly compatible wrt $\rightarrow$.

Decidability of Termination for CCS$^!$

Intuitively, well-structured transition system is a transition system enriched with an ordering relation over the set of states.

Definition (Well-structured transition system)

A well-structured transition system with strong compatibility is a transition system $TS = (S, \rightarrow, \leq)$ such that:

1. $(S, \rightarrow)$ is a transition system
2. $\leq \subseteq S \times S$ is a simulation relation and a well-quasi-order (wqo):
   - Every infinite sequence of S has two comparable states
   - S has no infinite antichains

If $\leq$ is a simulation, then it's said to be strongly compatible wrt $\rightarrow$.

Decidability of Termination for CCS$^!$

Intuitively, well-structured transition system is a transition system enriched with an ordering relation over the set of states.

Definition (Well-structured transition system)

A well-structured transition system with strong compatibility is a transition system $TS = (S, \rightarrow, \leq)$ such that:

1. $(S, \rightarrow)$ is a transition system
2. $\leq \subseteq S \times S$ is a simulation relation and a well-quasi-order (wqo):
   - Every infinite sequence of S has two comparable states
   - S has no infinite antichains

If $\leq$ is a simulation, then it's said to be strongly compatible wrt $\rightarrow$.

Decidability of Termination for CCS$^!$

Intuitively, well-structured transition system is a transition system enriched with an ordering relation over the set of states.

Definition (Well-structured transition system)

A well-structured transition system with strong compatibility is a transition system $TS = (S, \rightarrow, \leq)$ such that:

1. $(S, \rightarrow)$ is a transition system
2. $\leq \subseteq S \times S$ is a simulation relation and a well-quasi-order (wqo):
   - Every infinite sequence of S has two comparable states
   - S has no infinite antichains

If $\leq$ is a simulation, then it's said to be strongly compatible wrt $\rightarrow$.

Well-structured transition systems

The proof proceeds by instantiating the following theorem:

Theorem (Finkel and Schnoebelen, 2001)

Let $TS = (S, \rightarrow, \leq)$ be a finitely branching, well-structured transition system with strong compatibility, and decidable $\leq$. Then the existence of an infinite computation starting from a state in S is decidable.

Termination is Decidable in $\mathrm{CCS}^!$

The proof scheme can be summarized in the following steps:

1. Define a normal form for $\mathrm{CCS}^!$ processes.
2. Define an alternative finitely branching LTS for $\mathrm{CCS}^!$, and show it corresponds with the usual one.
3. First, characterize an upper bound for the derivatives of a process in normal form.
   Then, define an ordering $\preceq$ over these derivatives.
4. Show that $\preceq$ is a wqo strongly compatible wrt the alternative LTS of $\mathrm{CCS}^!$, defined in (2)

Some insights:

- The lack of recursive definitions eases the characterization of processes into normal forms
- It also eases decomposition of processes and the characterization of the upper bound

Outline

Challenges

Expressiveness for typed process languages

- Most encodability/separation results are for untyped processes
- Types add discipline to concurrent interactions: thus rules out plausible encodings in the untyped setting
- Types abstract communication structures: this calls for "better" encodings in which also types are encoded (type correspondences)
- Can we revisit established results, from a typed perspective?
- Can we formulate a theory of typed expressiveness, for instance, refining Gorla's proposal?

Challenges

Understanding Curry-Howard for Concurrency (CHoCO)

- CHoCO identifies a canonical class of confluent, terminating, well-behaved processes
- But several classes of typed concurrent processes were proposed before CHoCO
- Can we explain all these developments, taking CHoCO as a formal yardstick?
- Preliminary results for deadlock-free processes: CHoCO induces "highly parallel" process specifications

Expressiveness in Concurrency

Jorge A. Pérez

university of groningen

University of Brasilia

July 21, 2015