A Note on Scope and Infinite Behaviour in CCS-like Calculi

Gerardo Schneider

Uppsala University

Department of Information Technology

Uppsala, Sweden

Joint work with **Pablo Giambiagi** and **Frank Valencia**

Motivation: Scoping

- Consider $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

Motivation: Scoping

- Consider $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

- Question: Will action b ever be executed?

UPPSALA UNIVERSITET

Motivation: Scoping

- Consider $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

- Question: Will action b ever be executed?
- Answer: It depends... (!?)

$\Longrightarrow$ Static vs Dynamic Scoping

Motivation: Infiniteness

- Parametric vs. Constant definitions

1. CCS-like calculus, with $A \stackrel{\text{def}}{=} P$
2. CCS-like calculus, with $A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P$

Motivation: Infiniteness

- Parametric vs. Constant definitions
  1. CCS-like calculus, with $A \stackrel{\text{def}}{=} P$
  2. CCS-like calculus, with $A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P$
  - Can we encode (2) into (1)?

UPPSALA UNIVERSITET

Motivation: Infiniteness

- Parametric vs. Constant definitions

1. CCS-like calculus, with $A \stackrel{\text{def}}{=} P$
2. CCS-like calculus, with $A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P$

- Can we encode (2) into (1)?
- Do we need relabelling?

Motivation: Infiniteness

- Parametric vs. Constant definitions
  1. CCS-like calculus, with $A \stackrel{\mathrm{def}}{=} P$
  2. CCS-like calculus, with $A(x_1, \ldots, x_n) \stackrel{\mathrm{def}}{=} P$
  - Can we encode (2) into (1)?
  - Do we need relabelling?
- What happens with other forms of introducing infinite behaviour? For instance, Replication

Motivation and Contributions

These are important issues when comparing CCS variants

- Static vs Dynamic Scoping?
- Parametric vs. Constant definitions?
- Recursion vs Replication

Motivation and Contributions

These are important issues when comparing CCS variants

- Static vs Dynamic Scoping?
- Parametric vs. Constant definitions?
- Recursion vs Replication

We will show that these issues affect

- Expressiveness
- Analysis of certain properties

Overview of the presentation

- The finite core
- Static vs Dynamic scoping
- Infinite behaviour
- Expressiveness
- Concluding Remarks

Overview of the presentation

- The finite core
- Static vs Dynamic scoping
- Infinite behaviour
- Expressiveness
- Concluding Remarks

The Finite Core: Syntax

- Given:
  - A set of *names*, $\mathcal{N}$ $(a, b, x, y \ldots)$
  - A set of *co-names*, $\overline{\mathcal{N}} = \{\overline{a} \mid a \in \mathcal{N}\}$
  - A set of *actions*, $Act = \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$ (α, β)

UPPSALA UNIVERSITET

The Finite Core: Syntax

- Given:
  - A set of *names*, $\mathcal{N}$ $(a, b, x, y \ldots)$
  - A set of *co-names*, $\overline{\mathcal{N}} = \{\overline{a} \mid a \in \mathcal{N}\}$
  - A set of *actions*, $Act = \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$ (α, β)
- Processes specifying finite behaviour:

$$P ::= \sum_{i \in I} \alpha_i . P_i \mid P \backslash a \mid P \parallel P$$

The Finite Core: Semantics

SUM $\dfrac{}{\sum_{i\in I}\alpha_i.P_i \xrightarrow{\alpha_j} P_j}$ if $j \in I$

RES $\dfrac{P \xrightarrow{\alpha} P'}{P\backslash a \xrightarrow{\alpha} P'\backslash a}$ if $\alpha \notin \{a,\overline{a}\}$

PAR$_1$ $\dfrac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}$

PAR$_2$ $\dfrac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'}$

COM $\dfrac{P \xrightarrow{l} P' \quad Q \xrightarrow{\bar{l}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$

Overview of the presentation

- The finite core
- Static vs Dynamic scoping
- Infinite behaviour
- Expressiveness
- Concluding Remarks

Scoping: Example

- Consider $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

UPPSALA UNIVERSITET

Scoping: Example

- Consider $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

Consider the following rule:

$$\text{REC}\ \frac{P[\mu X.P/X] \xrightarrow{\alpha} P'}{\mu X.P \xrightarrow{\alpha} P'}$$

(without name α-conversion)

Scoping: Example

- Consider $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

Then, $P[\mu X.P/X]$
$= a \parallel (\overline{a}.b \parallel \mu X.P)\backslash a$

Scoping: Example

- Consider $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

Then, $P[\mu X.P/X]$
$= a \parallel (\overline{a}.b \parallel \mu X.P)\backslash a$
$= a \parallel (\overline{a}.b \parallel \mu X.(a \parallel (\overline{a}.b \parallel X)\backslash a))\backslash a$

Scoping: Example

- Consider $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

Then, $P[\mu X.P/X]$
$= a \parallel (\overline{a}.b \parallel \mu X.P)\backslash a$
$= a \parallel (\overline{a}.b \parallel \mu X.(a \parallel (\overline{a}.b \parallel X)\backslash a))\backslash a$

Scoping: Example

- Consider $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

Then, $P[\mu X.P/X]$
$= a \parallel (\overline{a}.b \parallel \mu X.P)\backslash a$
$= a \parallel (\overline{a}.b \parallel \mu X.(a \parallel (\overline{a}.b \parallel X)\backslash a))\backslash a$

Then b may be executed!

Scoping: Example 2

- Consider again $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

Scoping: Example 2

- Consider again $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

Consider now the following rule:

$$\text{REC}\ \frac{P[\mu X.P/X] \xrightarrow{\alpha} P'}{\mu X.P \xrightarrow{\alpha} P'}$$

(applying name α-conversion when necessary)

Scoping: Example 2

- Consider again $\mu X.P$ with

$$P = a \parallel (\bar{a}.b \parallel X)\backslash a$$

Then, $P[\mu X.P/X]$
$= a \parallel (\bar{a}.b \parallel \mu X.P)\backslash a$

Scoping: Example 2

- Consider again $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

Then, $P[\mu X.P/X]$
$= a \parallel (\overline{a}.b \parallel \mu X.P)\backslash a$

Scoping: Example 2

- Consider again $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

Then, $P[\mu X.P/X]$
$= a \parallel (\overline{c}.b \parallel \mu X.P)\backslash c$

Scoping: Example 2

- Consider again $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

Then, $P[\mu X.P/X]$
$= a \parallel (\bar{c}.b \parallel \mu X.P)\backslash c$
$= a \parallel (\bar{c}.b \parallel \mu X.(a \parallel (\bar{a}.b \parallel X)\backslash a))\backslash c$

UPPSALA UNIVERSITET

Scoping: Example 2

- Consider again $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

Then, $P[\mu X.P/X]$
$= a \parallel (\overline{c}.b \parallel \mu X.P)\backslash c$
$= a \parallel (\overline{c}.b \parallel \mu X.(a \parallel (\bar{a}.b \parallel X)\backslash a))\backslash c$

Scoping: Example 2

- Consider again $\mu X.P$ with

$$P = a \parallel (\overline{a}.b \parallel X)\backslash a$$

Then, $P[\mu X.P/X]$
$= a \parallel (\overline{c}.b \parallel \mu X.P)\backslash c$
$= a \parallel (\overline{c}.b \parallel \mu X.(a \parallel (\overline{a}.b \parallel X)\backslash a))\backslash c$

Then b will never be executed!

UPPSALA UNIVERSITET

Static vs Dynamic Scoping

- Name α-conversion to avoid name capture $\Longrightarrow$ static scoping
- Otherwise, $\Longrightarrow$ dynamic scoping

Dynamic scoping: the occurrence of a name may get *dynamically* (i.e. during execution) captured under the scope of some restriction

Overview of the presentation

- The finite core
- Static vs Dynamic scoping
- Infinite behaviour
- Expressiveness
- Concluding Remarks

Infinite Behaviour

There are at least four manners of introducing infinite behaviour

Infinite Behaviour

There are at least four manners of introducing infinite behaviour

- CCS$_k$: Infinite behavior given by a *finite* set of *constant* (i.e., parameterless) definitions of the form $A \stackrel{\mathrm{def}}{=} P$. The calculus is essentially CCS (Milner's book'1989) without relabelling nor infinite summations.

UPPSALA UNIVERSITET

Infinite Behaviour

There are at least four manners of introducing infinite behaviour

- $\mathrm{CCS_k}$: $A \stackrel{\mathrm{def}}{=} P$
- $\mathrm{CCS_p}$: Like $\mathrm{CCS_k}$ but using *parametric definitions* of the form $A(x_1, \ldots, x_n) \stackrel{\mathrm{def}}{=} P$. The calculus is the variant in Milner's book on the π-calculus

Infinite Behaviour

There are at least four manners of introducing infinite behaviour

- CCS_k: $A \stackrel{\text{def}}{=} P$
- CCS_p: $A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P$
- $\mathsf{CCS}_!$: Infinite behavior given by *replication* of the form $!P$. This variant is presented, e.g. in a paper by Busi, Gabbrielli and Zavattaro.

UPPSALA UNIVERSITET

Infinite Behaviour

There are at least four manners of introducing infinite behaviour

- CCS$_k$: $A \stackrel{\mathrm{def}}{=} P$
- CCS$_p$: $A(x_1, \ldots, x_n) \stackrel{\mathrm{def}}{=} P$
- CCS$_!$: $!P$
- CCS$_\mu$: Infinite behavior given by *recursive expressions* of the form $\mu X.P$. However, we adopt *static scoping* of channel names.

Infinite Behaviour

There are at least four manners of introducing infinite behaviour

- CCS$_\mathsf{k}$: $A \stackrel{\text{def}}{=} P$
- CCS$_\mathsf{p}$: $A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P$
- CCS$_!$: $!P$
- CCS$_\mu$: $\mu X.P$

Parametric Definitions: $\mathsf{CCS_p}$

Syntax:

$$P ::= \ldots \mid A(y_1, \ldots, y_n)$$

where $A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P_A$, $fn(P_A) \subseteq \{x_1, \ldots, x_n\}$.

UPPSALA UNIVERSITET

Parametric Definitions: $\mathbf{CCS_p}$

Syntax:

$$P ::= \ldots \mid A(y_1, \ldots, y_n)$$

where $A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P_A$, $fn(P_A) \subseteq \{x_1, \ldots, x_n\}$.

Semantics:

$$\text{CALL}\ \frac{P_A[y_1, \ldots, y_n / x_1, \ldots, x_n] \xrightarrow{\alpha} P'}{A(y_1, \ldots, y_n) \xrightarrow{\alpha} P'}\ \text{if } A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P_A$$

(name α-conversion when necessary)

Constant Definitions: CCS_k

Syntax:

$$P ::= \ldots \mid A$$

where $A \stackrel{\mathrm{def}}{=} P_A$

Constant Definitions: $\mathsf{CCS_k}$

Syntax:

$$P ::= \ldots \mid A$$

where $A \stackrel{\mathrm{def}}{=} P_A$

Semantics:

$$\text{CONS}\ \frac{P_A \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'}\ \text{if } A \stackrel{\mathtt{def}}{=} P_A$$

Constant Definitions: CCS_k

Syntax:

$$P ::= \ldots \mid A$$

where $A \stackrel{\text{def}}{=} P_A$

Semantics (alternative):

$$\text{REC}\ \frac{P[\mu X.P/X] \xrightarrow{\alpha} P'}{\mu X.P \xrightarrow{\alpha} P'}$$

(without name α-conversion)

Recursion Expressions: CCS$_\mu$

Syntax:

$$P ::= \ldots \mid X \mid \mu X.P$$

Recursion Expressions: CCS$_\mu$

Syntax:

$$P ::= \ldots \mid X \mid \mu X.P$$

Semantics:

$$\text{REC}\ \frac{P[\mu X.P/X] \xrightarrow{\alpha} P'}{\mu X.P \xrightarrow{\alpha} P'}$$

(name α-conversion when necessary)

UPPSALA UNIVERSITET

Replication: $\text{CCS}_!$

Syntax:

$$P ::= \ldots \mid\, !P$$

UPPSALA UNIVERSITET

Replication: CCS$_!$

Syntax:

$$P ::= \ldots \mid\, !P$$

Semantics:

$$\text{REP}\ \frac{P \parallel !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$$

UPPSALA UNIVERSITET

Overview of the presentation

- The finite core
- Static vs Dynamic scoping
- Infinite behaviour
- Expressiveness
- Concluding Remarks

Expressiveness and Classification Criteria

- Bisimilarity $\rightarrow$

Expressiveness and Classification Criteria

- Bisimilarity
- CCS$_\sigma$ is *as expressive as* CCS$_{\sigma'}$ iff for every $P \in Proc_\sigma$, there exists $Q \in Proc_{\sigma'}$ such that $P \approx Q$

Expressiveness and Classification Criteria

- Bisimilarity
- CCS$_\sigma$ is *as expressive as* CCS$_{\sigma'}$ iff for every $P \in Proc_\sigma$, there exists $Q \in Proc_{\sigma'}$ such that $P \approx Q$
- Divergence

Expressiveness and Classification Criteria

- Bisimilarity
- CCS$_\sigma$ is *as expressive as* CCS$_{\sigma'}$ iff for every $P \in Proc_\sigma$, there exists $Q \in Proc_{\sigma'}$ such that $P \approx Q$
- Divergence

  P is *divergent* iff $P(\xrightarrow{\tau})^\omega$, i.e., there exists an infinite sequence $P = P_0 \xrightarrow{\tau} P_1 \xrightarrow{\tau} \ldots$.

UPPSALA UNIVERSITET

Expressiveness and Classification Criteria

- Bisimilarity
- CCS$_\sigma$ is *as expressive as* CCS$_{\sigma'}$ iff for every $P \in Proc_\sigma$, there exists $Q \in Proc_{\sigma'}$ such that $P \approx Q$
- Divergence

We will study:

1. The relative expressiveness w.r.t. weak bisimilarity
2. The decidability of divergence

UPPSALA UNIVERSITET

Expressiveness Results

Encodings: *(weak) bisimulation preserving* mappings $[\![\cdot]\!] : Proc_\sigma \rightarrow Proc_{\sigma'}$

Expressiveness Results

Encodings: *(weak) bisimulation preserving* mappings $[\![\cdot]\!] : Proc_{\sigma} \rightarrow Proc_{\sigma'}$

- Encoding CCS_{p} into CCS_{k}
- Encoding CCS_{k} into CCS_{p}
- Encoding CCS_{μ} into $\text{CCS}_{!}$
- Encoding $\text{CCS}_{!}$ into CCS_{μ}

UPPSALA UNIVERSITET

Expressiveness Results

Encodings: *(weak) bisimulation preserving* mappings $[\![\cdot]\!] : Proc_\sigma \to Proc_{\sigma'}$

- Encoding CCS_p into CCS_k
- Encoding CCS_k into CCS_p
- Encoding CCS_μ into $\text{CCS}_!$
- Encoding $\text{CCS}_!$ into CCS_μ

Encoding $\mathrm{CCS_p}$ into $\mathrm{CCS_k}$

$$[\![\cdot]\!] : Proc_p \rightarrow Proc_k$$

UPPSALA UNIVERSITET

Encoding CCS_p into CCS_k

$[\![\cdot]\!] : Proc_p \rightarrow Proc_k$

Idea:

- Assume a definition of the form $A(x) \stackrel{\text{def}}{=} P_A$
- Generate as many constants A_y as occurrences of $A(y)$ in P_A

UPPSALA UNIVERSITET

Encoding $\mathrm{CCS_p}$ into $\mathrm{CCS_k}$

$[\![\cdot]\!] : Proc_p \longrightarrow Proc_k$

Idea:

- Assume a definition of the form $A(x) \stackrel{\text{def}}{=} P_A$
- Generate as many constants A_y as occurrences of $A(y)$ in P_A

Problem: Potentially infinitely many definitions!

UPPSALA UNIVERSITET

Encoding $\mathbf{CCS_p}$ into $\mathbf{CCS_k}$

$[\![\cdot]\!] : Proc_p \longrightarrow Proc_k$

Idea:

- Assume a definition of the form $A(x) \stackrel{\text{def}}{=} P_A$
- Generate as many constants A_y as occurrences of $A(y)$ in P_A

Problem: Potentially infinitely many definitions!

- Due to name α-conversion a possible infinite number of fresh names can be generated

Encoding $\mathsf{CCS_p}$ into $\mathsf{CCS_k}$: Example

Let $A(x) \stackrel{\text{def}}{=} (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A(z))\backslash z$

UPPSALA UNIVERSITET

Encoding $\mathsf{CCS_p}$ into $\mathsf{CCS_k}$: Example

Let $A(x) \stackrel{\text{def}}{=} (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A(z))\backslash z$

1. $A_x \stackrel{\text{def}}{=} (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A_z)\backslash z$

Encoding CCS_p into CCS_k: Example

Let $A(x) \stackrel{\text{def}}{=} (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A(z))\backslash z$

1. $A_x \stackrel{\text{def}}{=} (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A_z)\backslash z$
2. $A_z \stackrel{\text{def}}{=} (z.z.\mathbf{0} \parallel \overline{z}.\mathbf{0} \parallel A_z)\backslash z$

Encoding CCS$_p$ into CCS$_k$: Example

Let $A(x) \stackrel{\text{def}}{=} (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A(z))\backslash z$

1. $A_x \stackrel{\text{def}}{=} (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A_z)\backslash z$
2. $A_z \stackrel{\text{def}}{=} (z.z.\mathbf{0} \parallel \overline{z}.\mathbf{0} \parallel A_z)\backslash z$

UPPSALA UNIVERSITET

Encoding $\mathsf{CCS_p}$ into $\mathsf{CCS_k}$: Example

Let $A(x) \stackrel{\text{def}}{=} (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A(z))\backslash z$

1. $A_x \stackrel{\text{def}}{=} (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A_z)\backslash z$
2. $A_z \stackrel{\text{def}}{=} (z_1.z.\mathbf{0} \parallel \overline{z}.\mathbf{0} \parallel A_{z_1})\backslash z_1$

UPPSALA UNIVERSITET

Encoding CCS_p into CCS_k: Example

Let $A(x) \stackrel{\text{def}}{=} (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A(z))\backslash z$

1. $A_x \stackrel{\text{def}}{=} (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A_z)\backslash z$
2. $A_z \stackrel{\text{def}}{=} (z_1.z.\mathbf{0} \parallel \overline{z}.\mathbf{0} \parallel A_{z_1})\backslash z_1$
3. $A_{z_1} \stackrel{\text{def}}{=} (z.z_1.\mathbf{0} \parallel \overline{z_1}.\mathbf{0} \parallel A_z)\backslash z$

Encoding CCS$_p$ into CCS$_k$: Example

Let $A(x) \stackrel{\text{def}}{=} (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A(z))\backslash z$

1. $A_x \stackrel{\text{def}}{=} (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A_z)\backslash z$
2. $A_z \stackrel{\text{def}}{=} (z_1.z.\mathbf{0} \parallel \overline{z}.\mathbf{0} \parallel A_{z_1})\backslash z_1$
3. $A_{z_1} \stackrel{\text{def}}{=} (z.z_1.\mathbf{0} \parallel \overline{z_1}.\mathbf{0} \parallel A_z)\backslash z$

Remark: The generation of fresh names could continue forever!

UPPSALA UNIVERSITET

Encoding $\mathsf{CCS_p}$ into $\mathsf{CCS_k}$

Theorem: For any $P \in \mathsf{CCS_p}$ with a finite set of definitions, one can effectively construct the associated set of definitions of $[\![P]\!]$.

Encoding $\mathsf{CCS_p}$ into $\mathsf{CCS_k}$

Theorem: For any $P \in \mathsf{CCS_p}$ with a finite set of definitions, one can effectively construct the associated set of definitions of $[\![P]\!]$.

Theorem: Given a process $P \in \mathsf{CCS_p}$, $P \sim [\![P]\!]$.

UPPSALA UNIVERSITET

Encoding $\mathsf{CCS_p}$ into $\mathsf{CCS_k}$

Theorem: For any $P \in \mathsf{CCS_p}$ with a finite set of definitions, one can effectively construct the associated set of definitions of $[\![P]\!]$.

Theorem: Given a process $P \in \mathsf{CCS_p}$, $P \sim [\![P]\!]$.

Corollary: Injective relabellings are redundant in CCS.

Encoding CCS$_\mu$ into CCS$_!$

$[\![\cdot]\!] : Proc_\mu \rightarrow Proc_!$

Idea:

$$
\begin{array}{lcl}
[\![X_i]\!] & = & \overline{x_i}.\mathbf{0} \\
[\![\mu X_i.P]\!] & = & (!x_i.[\![P]\!] \parallel \overline{x_i}.\mathbf{0})\backslash x_i
\end{array}
$$

UPPSALA UNIVERSITET

Encoding CCS$_\mu$ into CCS$_!$: Example

Let be the following CCS$_\mu$ process:

$$P = \mu X.(a.X)$$

UPPSALA UNIVERSITET

Encoding CCS$_\mu$ into CCS$_!$: Example

Let be the following CCS$_\mu$ process:

$$P = \mu X.(a.X)$$

Then the corresponding encoding is:

$$[\![P]\!] = (!x.a.\bar{x} \parallel \bar{x})\backslash x$$

Encoding CCS$_\mu$ into CCS$_!$: Example

Let be the following CCS$_\mu$ process:

$$P = \mu X.(a.X)$$

Then the corresponding encoding is:

$$[\![P]\!] = (!x.a.\bar{x} \parallel \bar{x})\backslash x$$

They are clearly not strongly bisimilar:

$$\mu X.a.X \xrightarrow{a}_\mu \mu X.a.X \xrightarrow{a}_\mu \mu X.a.X \ldots$$

$$(!x.a.\bar{x} \parallel \bar{x})\backslash x \xrightarrow{\tau}_! (!x.a.\bar{x} \parallel a.\bar{x})\backslash x \xrightarrow{a}_! (!x.a.\bar{x} \parallel \bar{x})\backslash x \ldots$$

UPPSALA UNIVERSITET

Encoding CCS$_\mu$ into CCS$_!$

Theorem: For $P \in Proc_\mu$, $P \approx [\![P]\!]$. Moreover, P diverges iff $[\![P]\!]$ diverges.

Overview of the presentation

- The finite core
- Static vs Dynamic scoping
- Infinite behaviour
- Expressiveness
- Concluding Remarks

Conclusions

$\mathrm{CCS_p} \sim \mathrm{CCS_k}$	$\mathrm{CCS}_\mu \approx \mathrm{CCS_!}$
Divergence: Undecidable	Divergence: Decidable

Conclusions

$\mathrm{CCS_p} \sim \mathrm{CCS_k}$	$\mathrm{CCS}_\mu \approx \mathrm{CCS}_!$
Divergence: Undecidable	Divergence: Decidable

- Injective relabellings are redundant in CCS
- Interpretation of Rule REC leads to important differences
- CCS exhibits dynamic name scope and it does not preserve α-conversion

Related Work

- The CCS variant in Milner's book π-calculus uses parametric definitions with static scope
- Edinburgh Concurrency Workbench tool (CWB) uses dynamic scoping for parametric definitions
- ECCS advocates the static scoping of names
- CHOCS uses dynamic name scoping in the context of higher-order CCS

Auxiliary Slides

Bisimilarity

A relation $\mathcal{S} \subseteq \mathit{Proc} \times \mathit{Proc}$ is a **(strong) simulation** if for all $(P, Q) \in \mathcal{S}$:

$$\begin{array}{ccc} P & \xrightarrow{\alpha} & P' \\ \mathcal{S} & & \\ Q & & \end{array}$$

UPPSALA UNIVERSITET

Bisimilarity

A relation $\mathcal{S} \subseteq Proc \times Proc$ is a **(strong) simulation** if for all $(P, Q) \in \mathcal{S}$:

$$\begin{array}{ccc} P & \xrightarrow{\alpha} & P' \\ \mathcal{S} & & \mathcal{S} \\ Q & \xrightarrow{\alpha} & Q' \end{array}$$

Bisimilarity

A relation $\mathcal{S} \subseteq Proc \times Proc$ is a **(strong) simulation** if for all $(P, Q) \in \mathcal{S}$:

$$\begin{array}{ccc} P & \xrightarrow{\alpha} & P' \\ \mathcal{S} & & \mathcal{S} \\ Q & \xrightarrow{\alpha} & Q' \end{array}$$

$\mathcal{S}$ is a ***(strong) bisimulation*** if both $\mathcal{S}$ and its converse are (strong) simulations: $P \sim Q$.

Bisimilarity

A relation $\mathcal{S} \subseteq Proc \times Proc$ is a weak simulation if for all $(P, Q) \in \mathcal{S}$:

$$\begin{array}{ccc} P & \stackrel{s}{\Longrightarrow} & P' \\ \mathcal{S} & & \mathcal{S} \\ Q & \stackrel{s}{\Longrightarrow} & Q' \end{array}$$

$\mathcal{S}$ is a *weak bisimulation* if both $\mathcal{S}$ and its converse are weak simulations: $P \approx Q$.

- "$\stackrel{s}{\Longrightarrow}$" (where $s = \alpha_1.\alpha_2.\ldots$) is $(\stackrel{\tau}{\longrightarrow})^* \stackrel{\alpha_1}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^* \ldots (\stackrel{\tau}{\longrightarrow})^* \stackrel{\alpha_n}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^*$

$\longleftarrow$

Encoding $\mathsf{CCS_p}$ into $\mathsf{CCS_k}$

$[\![\cdot]\!] : Proc_p \to Proc_k$

Idea:

- For each $P \in$ $\mathsf{CCS_p}$, let $\widehat{P} \in$ $\mathsf{CCS_k}$ replacing in P each occurrence of $B(y)$ with B_y
- For each definition $A(x) \stackrel{\mathtt{def}}{=} P_A$, generate a constant definition $A_x \stackrel{\mathtt{def}}{=} \widehat{P_A}$

UPPSALA UNIVERSITET