Encodings into Asynchronous Pi Calculus

Uwe Nestmann
(Technische Universität Berlin)
Mobility in Pi-Calculus

\[(\nu z)(P \mid R) \mid Q\]

\[P = \bar{x}\langle z \rangle.P'\]

\[Q = x(y).Q'\]

\[P' \mid (\nu z)(R \mid Q')\]

\[(x \text{ not in } P')\]
Pi-Calculus

“I reject the idea that there can be a unique conceptual model, or one preferred formalism, for all aspects of something as large as concurrent computation.”

(Robin Milner, 1993)
Pi-Calculus

“I reject the idea that there can be a unique conceptual model, or one preferred formalism, for all aspects of something as large as concurrent computation.”

(Robin Milner, 1993)

“Pi Calculus is better than Process Algebra”

(Bill Gates, 2003?)
A Jungle?

many members of the family

Paola Quaglia’s note

“Which Pi Calculus are you talking about?”

\( \pi, \nu, \gamma, \Lambda \pi, \Lambda \nu, \Pi \pi, \Pi \nu, \Pi \gamma, \Pi \Lambda, \Pi \chi, \chi, \rho, s\pi, k, \)

Blue, Fusion, Applied, ...

polyadic, polymorphic, polynomic, polarized, dyadic, ...
A Jungle?

many members of the family
Paola Quaglia’s note
“Which Pi Calculus are you talking about?”
π, ν, γ, Λπ, Lπ Pπ, πI, ΗΟπ, λπ, πξ, χ, ρ, sπ, k,
Blue, Fusion, Applied, ...
polyadic, polymorphic, polynomic, polarized, dyadic, ...

many dimensions
communication models (applicability, minimality, ... )
comparisons / encodings
implementations
semantics / models / types / proof techniques / tools
applications
Overview

Part 0a: Encodings vs Full Abstraction
Comparison of Languages/Calculi
Correctness of Encodings

Part 0b: Asynchronous Pi Calculus

Part 1: Input-Guarded Choice
  Encoding (Distributed Implementation)
  Decoding (Correctness Proof)

Part 2: Output-Guarded Choice
  Encoding Separate Choice
  Encoding Mixed Choice

Conclusions
Overview

Part 0a: Encodings vs Full Abstraction
Comparison of Languages/Calculi
Correctness of Encodings

Part 0b: Asynchronous Pi Calculus

Part 1: Input-Guarded Choice
Encoding (Distributed Implementation)
Decoding (Correctness Proof)

Part 2: Output-Guarded Choice
Encoding Separate Choice
Encoding Mixed Choice

Conclusions
Comparison of Languages/Calculi
Absolute Expressiveness

(* see also Joachim Parrow @ LIX Colloquium 2006 *)

Given a single process calculus, what can it express?

What objects are expressible
(as closed terms)?

What operators are expressible
(as contexts, or open terms)?

What problems can be solved?
(leader election, matching systems, …)
Relative Expressiveness (I)

Given two (process) calculi $S$ and $T$, say that “$T$ is as least as expressive as $S$” to mean that

“$T$ can express anything that $S$” can.
Relative Expressiveness (II)

This can also be formulated without actually saying what is being expressed by exhibiting a \textit{(syntactic) encoding} $[[\cdot]] : S \rightarrow T$

Such an encoding shall be good/reasonable/compositional/…
Relative Expressiveness (II)

This can also be formulated without actually saying what is being expressed by exhibiting a (syntactic) encoding \([[-]] : S \rightarrow T\)

Such an encoding shall be good/reasonable/compositional/…

Daniele Gorla @ EXPRESS 2006: “Everybody seems to have his/her own idea about which properties to check for.”
Relative Expressiveness (II)

This can also be formulated without actually saying what is being expressed by exhibiting a (syntactic) encoding $[[-]] : S \rightarrow T$

Such an encoding shall be good/reasonable/compositional/…

_Daniele Gorla @ EXPRESS 2006:_ “Everybody seems to have his/her own idea about which properties to check for.”

both for **positive** statements (correctness, “goodness”, ...) and **negative** statements (separation, “badness”, ...)
The \( \pi \)-calculus hierarchy.

- **\( \pi \)-calculus with mixed choice**
  - Palamidessi 97

- **\( \pi \)-calculus with separate choice**
  - Identity encoding
  - **\( \pi \)-calculus with input choice**
    - (no output prefix)
    - Nestmann–Pierce 96
  - **\( \pi \)-calculus without choice**
    - (output prefix)
    - **Identity encoding**
    - Nestmann 97
    - Nestmann–Pierce 96
    - Honda–Tokoro 91, Boudol 92

- **Asynchronous \( \pi \)-calculus**
  - (no choice, no output prefix)
IMHO

All of the current proposals of *goodness* are ad-hoc; we do not yet have a proper theory of encodings.
Encodings
Encodings
An encoding is a (total) function

\[
[[-]] : S \rightarrow T
\]

that translates
the syntax of language $S$ (the source) into
the syntax of language $T$ (the target).
Encodings

An encoding is a (total) function

\[ [[-]] : S \rightarrow T \]

that translates
the syntax of language \( S \) (the source) into
the syntax of language \( T \) (the target).

Many encodings are injective, i.e,

\[ P \neq Q \quad \text{implies} \quad [[P]] \neq [[Q]] \]
Encodings

An encoding is a (total) function

\[
[[\cdot]] : \mathcal{S} \rightarrow \mathcal{T}
\]

that translates
the syntax of language $\mathcal{S}$ (the source) into
the syntax of language $\mathcal{T}$ (the target).

Many encodings are injective, i.e.,

\[ P \neq Q \quad \text{implies} \quad [[P]] \neq [[Q]] \]

and we only consider compositional definitions,
Encodings

An encoding is a (total) function

\[[\dash] \] : S \rightarrow T

that translates
the syntax of language S (the source) into
the syntax of language T (the target).

Many encodings are injective, i.e,

\text{P} \neq \text{Q} \quad \text{implies} \quad [[\text{P}]] \neq [[\text{Q}]]

and we only consider compositional definitions,
following the syntactic structure of source terms.
Correctness of Encodings
Indistinguishability (I)
Indistinguishability (I)

Let $P$ and $[[P]]$ live in the same calculus.

The encoding shall be “unnoticable”: 
Indistinguishability (I)

Let $P$ and $[[P]]$ live in the same calculus. The encoding shall be “unnoticeable”:

$$P \equiv [[P]]$$

- The choice of $\equiv$ captures the expressible artifacts that one considers worth comparing …
Indistinguishability (I)

Let $P$ and $[[P]]$ live in the same calculus. The encoding shall be “unnoticeable”:

$$P \equiv [[P]]$$

- The choice of $\equiv$ captures the expressible artifacts that one considers worth comparing …
- $[[\cdot]]$ and $\equiv$ are closely related.
Indistinguishability (I)

Let $P$ and $[[P]]$ live in the same calculus. The encoding shall be “unnoticeable”:

$$P \equiv [[P]]$$

- The choice of $\equiv$ captures the expressible artifacts that one considers worth comparing …
- $[[-]]$ and $\equiv$ are closely related.
- Different results are often comparable.
  $\Rightarrow$ Seek the strongest equivalence that holds.
Indistinguishability (I)

Let $P$ and $[[P]]$ live in the same calculus. The encoding shall be “unnoticable”:

$$P \equiv [[P]]$$

• The choice of $\equiv$ captures the expressible artifacts that one considers worth comparing …

• $[[\cdot]]$ and $\equiv$ are closely related.

• Different results are often comparable.
  $\Rightarrow$ Seek the strongest equivalence that holds.

• Encodings are often not injective.
Indistinguishability (II)
Indistinguishability (II)

Let $P$ and $[[P]]$ live in two completely different calculi. Then

$$P \equiv [[[P]]]$$

is no longer possible as a requirement.
Indistinguishability (II)

Let $P$ and $[[P]]$ live in two completely different calculi. Then

$$P \cong [[P]]$$

is no longer possible as a requirement.

*Full abstraction* helps !?
Full Abstraction (I)

Notion to capture the quality of denotational models (of programming languages) [Plotkin, TCS 1977].
Full Abstraction (I)

Notion to capture the quality of denotational models (of programming languages) [Plotkin, TCS 1977].

Let $P$ be the syntax of a programming language. Let $D$ be some mathematical domain. Let $\llbracket \cdot \rrbracket : P \to D$ be the denotational semantics of $P$. Let $\equiv_P$ be an (operational) equivalence on $P$. 
Full Abstraction (I)

Notion to capture the quality of denotational models (of programming languages) [Plotkin, TCS 1977].

Let $P$ be the syntax of a programming language. Let $D$ be some mathematical domain. Let $\llbracket - \rrbracket : P \rightarrow D$ be the denotational semantics of $P$. Let $\equiv_P$ be an (operational) equivalence on $P$.

Then, $\llbracket - \rrbracket$ is called **fully abstract** w.r.t. $\equiv_P$, if for all $P,Q$ in $P$: $P \equiv_P Q$ iff $\llbracket P \rrbracket = \llbracket Q \rrbracket$.
Full Abstraction (II)

Let $[[\cdot]] : S \to T$.
Let $\equiv_S$ and $\equiv_T$ be respective equivalences on $S$ and $T$. 
Full Abstraction (II)

Let $[[\cdot]] : S \rightarrow T$.
Let $\equiv_S$ and $\equiv_T$ be respective equivalences on $S$ and $T$.

Then, take $(T, \equiv_T)$ as the respective denotational model, and take the encoding $[[\cdot]]$ as the denotation function.
Full Abstraction (II)

Let \([\llbracket - \rrbracket] : S \to T\).
Let \(\equiv_S\) and \(\equiv_T\) be respective equivalences on \(S\) and \(T\).

Then, take \((T, \equiv_T)\) as the respective denotational model, and take the encoding \([\llbracket - \rrbracket]\) as the denotation function.

Then \([\llbracket - \rrbracket]\) is called \textit{fully abstract} w.r.t. \(\equiv_S\) and \(\equiv_T\), if it \textit{preserves and reflects} the equivalences of \(S\) and \(T\):

\[
\text{for all } P, Q \text{ in } S: \quad P \equiv_S Q \iff [\llbracket P \rrbracket] \equiv_T [\llbracket Q \rrbracket]
\]
Full Abstraction (III)
Full Abstraction (III)

Problems:

- on what basis to choose $\approx_S$ and $\approx_T$
  (cf. Rob van Glabbeek’s talk)

- various ways to have results for congruences
  - all target contexts
  - only translated contexts (respecting the protocol)
  - only well-typed contexts (w.r.t. a target type system)
Full Abstraction (III)

Problems:
• on what basis to choose $\equiv_s$ and $\equiv_T$ (cf. Rob van Glabbeek’s talk)
• various ways to have results for congruences
  – all target contexts
  – only translated contexts (respecting the protocol)
  – only well-typed contexts (w.r.t. a target type system)

Observation:
• full abstraction results are “easy to get”
• full abstraction results are hard to compare
Operational Correspondence (I)

Completeness (Preservation of execution step).

if $S \rightarrow_s S'$, then $\llbracket S \rrbracket \Rightarrow_t \llbracket S' \rrbracket$
Operational Correspondence (I)

Completeness (Preservation of execution step).

if $S \rightarrow_s S'$, then $\llbracket S \rrbracket \Rightarrow_t \llbracket S' \rrbracket$

Soundness (Reflection of execution steps).

if $\llbracket S \rrbracket \Rightarrow_t \llbracket S' \rrbracket$ then $S \Rightarrow_s S'$
Operational Correspondence (I)

Completeness (Preservation of execution step).

if $S \rightarrow_s S'$, then $\llbracket S \rrbracket \Rightarrow_t \llbracket S' \rrbracket$

Soundness (Reflection of execution steps).

if $\llbracket S \rrbracket \Rightarrow_t \llbracket S' \rrbracket$ then $S \Rightarrow_s S'$

if $\llbracket S \rrbracket \rightarrow_t T$ then there is $S \rightarrow_s S'$ such that $T \preceq_t \llbracket S' \rrbracket$
Operational Correspondence (I)

**Completeness** (Preservation of execution step).

\[
\text{if } S \rightarrow_s S', \text{ then } [S] \Rightarrow_t [S']
\]

**Soundness** (Reflection of execution steps)

\[
\begin{align*}
\text{if } [S] \Rightarrow_t [S'] & \text{ then } S \Rightarrow_s S' \\
\text{if } [S] \rightarrow_t T & \text{ then there is } S \rightarrow_s S' \text{ such that } T \preceq_t [S'] \\
\text{if } [S] \Rightarrow_t T & \text{ then there is } S \Rightarrow_s S' \text{ such that } T \Rightarrow_t [S']
\end{align*}
\]
Operational Correspondence (II)

Obviously (?),
some form of operational correspondence
is often employed as a means to support the proof of
full abstraction w.r.t. bisimulation-based equivalences.

Think about how to prove:
Operational Correspondence (II)

Obviously (?), some form of operational correspondence is often employed as a means to support the proof of full abstraction w.r.t. bisimulation-based equivalences.

Think about how to prove:

\[ P \approx_s Q \iff [[P]] \approx_T [[Q]] \]
Question

Assume an encoding $[[ - ]] : S \rightarrow T$.
Assume $\equiv_{S_1}$ and $\equiv_{S_2}$ are equivalences on $S$.
Assume $\equiv_{T_1}$ and $\equiv_{T_2}$ are equivalences on $T$.
Assume $\equiv_{S_1} \subseteq \equiv_{S_2}$ and $\equiv_{T_1} \subseteq \equiv_{T_2}$.
Question

Assume an encoding \([\cdot - \cdot] : S \rightarrow T\).
Assume \(\equiv_{S_1}\) and \(\equiv_{S_2}\) are equivalences on \(S\).
Assume \(\equiv_{T_1}\) and \(\equiv_{T_2}\) are equivalences on \(T\).
Assume \(\equiv_{S_1} \subseteq \equiv_{S_2}\) and \(\equiv_{T_1} \subseteq \equiv_{T_2}\).
Question

Assume an encoding $[[ - ]] : S \rightarrow T$.
Assume $\equiv_{S1}$ and $\equiv_{S2}$ are equivalences on $S$.
Assume $\equiv_{T1}$ and $\equiv_{T2}$ are equivalences on $T$.
Assume $\equiv_{S1} \subseteq \equiv_{S2}$ and $\equiv_{T1} \subseteq \equiv_{T2}$.

Does F.A.w.r.t. $(\equiv_{S1}, \equiv_{T1})$ imply F.A.w.r.t. $(\equiv_{S2}, \equiv_{T2})$?
Question

Assume an encoding $[[ - ]] : S \rightarrow T$.
Assume $\equiv_{S1}$ and $\equiv_{S2}$ are equivalences on $S$.
Assume $\equiv_{T1}$ and $\equiv_{T2}$ are equivalences on $T$.
Assume $\equiv_{S1} \subseteq \equiv_{S2}$ and $\equiv_{T1} \subseteq \equiv_{T2}$.

Does $\text{F.A.w.r.t. } (\equiv_{S1}, \equiv_{T1})$ imply $\text{F.A.w.r.t. } (\equiv_{S2}, \equiv_{T2})$?

- **Special case**: consider universal relations
- **Special case**: identity embeddings
Assume an arbitrary encoding $\llbracket \cdot \rrbracket : S \rightarrow T$.

Then, $\llbracket \cdot \rrbracket$ is fully abstract w.r.t. $(\text{Ker}(\llbracket \cdot \rrbracket), \text{Id}_T)$. 
Task

Assume an encoding \[ \text{[[ - ]] : S \rightarrow T} \].
Assume \( \equiv_{S_1} \) and \( \equiv_{S_2} \) are equivalences on \( S \).
Assume \( \equiv_{T_1} \) and \( \equiv_{T_2} \) are equivalences on \( T \).
Task

Assume an encoding $[[ - ]] : S \rightarrow T$.
Assume $\equiv_{S1}$ and $\equiv_{S2}$ are equivalences on $S$.
Assume $\equiv_{T1}$ and $\equiv_{T2}$ are equivalences on $T$.

Identify “reasonable” conditions to state that:

$\text{F.A.w.r.t.} \ (\equiv_{S1}, \equiv_{T1}) \quad \text{is better than} \quad \text{F.A.w.r.t.} \ (\equiv_{S2}, \equiv_{T2})$
Example Encodings
Tuples

\[
\overline{\langle x \langle a_1 \ldots a_n \rangle \cdot Q \rangle} = \nu w \overline{xw} \overline{wa_1} \ldots \overline{wan} \cdot [Q]
\]

\[
\overline{x \langle z_1 \ldots z_n \rangle \cdot P \rangle} = x(w) \cdot w(z_1) \ldots w(z_n) \cdot [P]
\]

... all other operators translated homomorphically
Tuples

\[
\left[ x \langle a_1 \ldots a_n \rangle . Q \right] = \nu w \; \overline{xw}. \overline{wa_1} . \ldots . \overline{wa_n}. \left[ Q \right]
\]

\[
\left[ x(z_1 \ldots z_n) . P \right] = x(w). w(z_1). \ldots . w(z_n). \left[ P \right]
\]

... all other operators translated homomorphically

- operational correspondence
- no “obvious” F.A.
- F.A. w.r.t. translated contexts
- F.A. w.r.t. typed barbed congruence
Functions as Processes

\[ [\lambda x. M]_p \overset{\text{def}}{=} \nu y \bar{p}\langle y \rangle \cdot !y(x, q) \cdot [M]_q \]

\[ [x]_p \overset{\text{def}}{=} \bar{x}\langle p \rangle \]

\[ [MN]_p \overset{\text{def}}{=} \nu q ( [M]_q \mid q(v) \cdot \nu x (\overline{v}\langle x, p \rangle \cdot !x(r) \cdot [N]_r)) \]
Functions as Processes

$$\begin{align*}
\boxed{\lambda x. M}_p & \triangleq \nu y \bar{p}\langle y \rangle. !y(x, q). \boxed{M}_q \\
\boxed{x}_p & \triangleq \bar{x}\langle p \rangle \\
\boxed{MN}_p & \triangleq \nu q (\quad \boxed{M}_q \quad | \\
& \quad q(u). \nu x (\bar{v}\langle x, p \rangle. !x(r). \boxed{N}_r))
\end{align*}$$

various 1/2 F.A. results for various variants
used to compare Lambda and Pi Calculi
inspired Lambda Calculus theory
used to prove new equations
## Objects as Processes

\[
\begin{align*}
\text{let } x = a \text{ in } b & \overset{\text{def}}{=} (\nu q) \left( \left[ a \right]_q^k \mid q(x, k') \cdot \bar{y} \langle \text{cln}_p, k' \rangle \right) \\
\text{fork}(a) & \overset{\text{def}}{=} (\nu t) \left( \left[ a \right]_q^{k'} \mid \bar{p}(t, k) \mid q(x, k').t(r, k'').\bar{r}(x, k'') \right) \\
\text{join}(b) & \overset{\text{def}}{=} (\nu q) \left( \left[ b \right]_q^k \mid q(t, k') \cdot \bar{t}(p, k') \right)
\end{align*}
\]

Table 6: Translational semantics of Øjeblik — Clients, Scoping, Concurrency
\[
\text{newO}_0(s, \tilde{t}) \overset{\text{def}}{=} (\nu s) \left( \bar{r}(s, k) \mid \text{newO}_0(s, \tilde{t}, \prod_{j \in J} t_j(s_j, \bar{x}_j, r, k'), [b_j]_{k'}) \right)
\]

\[
\text{newA}_0(s, s_a) \overset{\text{def}}{=} (\nu m_a m_b k_c k_d) \left( \bar{m}_c \mid \text{OM}_0(s, m_e, m_i, k_e, k_d, \tilde{t}) \right)
\]

\[
\text{OM}_0(s, \bar{m}, k_e, k_d, \tilde{t}) \overset{\text{def}}{=} s(l, k), (\nu k^*)
\]

\[
\begin{align*}
\text{if } [k = k_1] \text{ then} \\
\text{case } l \text{ of } \text{cln}_{-}(r) : \text{OM}_0(s, \bar{m}, k_e, k^*, \tilde{t}) \mid (\nu s^*) \left( \bar{r}(s^*, k^*) \mid \text{newO}_0(s^*, \tilde{t}) \right) ; \\
\text{ali}_{-}(s_a, r) : \text{AM}_0(s, \bar{m}, k_e, k^*, s_a) \mid \bar{r}(s_a, k^*) ; \\
\text{upd}_{-}(l', r) : \text{OM}_0(s, \bar{m}, k_e, k^*, t_1, \ldots, t_{j-1}, l', t_{j+1} \ldots) \mid \bar{r}(s, k^*) ; \\
\text{inv}_{-}(\bar{x}, r) : \text{OM}_0(s, \bar{m}, k_e, k^*, \tilde{t}) \mid \bar{r}(s, \bar{x}, r, k^*) ; \\
\text{sur}_{-}(r) : \text{OM}_0(s, \bar{m}, k_e, k^*, \tilde{t}) \mid [s \text{ alias } (s \text{ clone})]_{k^*} \\
\text{png}_{-}(r) : \text{OM}_0(s, \bar{m}, k_e, k^*, \tilde{t}) \mid [s]_{k^*}
\end{align*}
\]

\[
\begin{align*}
\text{elif } [k = k_2] \text{ then} \\
\text{OM}_0(s, \bar{m}, k_e, k^*, \tilde{t}) \mid \text{case } l \text{ of } \text{cln}_{-}(r) : m_k(k), \bar{m}_c ; \\
\text{ali}_{-}(s_a, r) : m_k(k), \bar{m}_c ; \\
\text{upd}_{-}(l', r) : m_k(k), \bar{m}_c ; \\
\text{inv}_{-}(\bar{x}, r) : \text{CM}[\bar{r}(s, \bar{x}, r^*, k^*)] ; \\
\text{sur}_{-}(r) : \text{CM}[\text{[s alias (s clone)]}]_{k^*} ; \\
\text{png}_{-}(r) : \text{CM}[\text{[s]}]_{k^*}
\end{align*}
\]

\[
\begin{align*}
\text{else OM}_0(s, \bar{m}, k_e, k_d, \tilde{t}) \mid m_e, (\bar{r}(l, k) \mid \bar{m}_k)
\end{align*}
\]

\[
\text{CM}[\cdot] \overset{\text{def}}{=} (\nu s^*) \left( \cdot \mid r^*(y, k'), m_i(k''), (\bar{r}(y, k'') \mid \bar{m}_e) \right)
\]

\[
\text{AM}_0(s, \bar{m}, k_e, k_d, s_a) \overset{\text{def}}{=} s(l, k), (\nu k^*)
\]

\[
\begin{align*}
\text{if } [k = k_1] \text{ then} \\
\text{case } l \text{ of } \text{cln}_{-}(r) : \text{AM}_0(s, \bar{m}, k_e, k^*, s_a) \mid (\nu s^*) \left( \bar{r}(s^*, k^*) \mid \text{newA}_0(s^*, s_a) \right) ; \\
\text{ali}_{-}(s'_a, r) : \text{AM}_0(s, \bar{m}, k_e, k^*, s'_a) \mid \bar{r}(s'_a, k^*) ; \\
\text{upd}_{-}(l', r) : \text{AM}_0(s, \bar{m}, k_e, k^*, s_a) \mid \bar{s}_a(l, k) ; \\
\text{inv}_{-}(\bar{x}, r) : \text{AM}_0(s, \bar{m}, k_e, k^*, s_a) \mid \bar{s}_a(l, k) ; \\
\text{sur}_{-}(r) : \text{AM}_0(s, \bar{m}, k_e, k^*, s_a) \mid \bar{s}_a(l, k) ; \\
\text{png}_{-}(r) : \text{AM}_0(s, \bar{m}, k_e, k^*, s_a) \mid \bar{s}_a(l, k)
\end{align*}
\]

\[
\begin{align*}
\text{elif } [k = k_2] \text{ then } \text{AM}_0(s, \bar{m}, k_e, k^*, s_a) \mid m_k(k), (\bar{s}_a(l, k) \mid \bar{m}_e) \\
\text{else } \text{AM}_0(s, \bar{m}, k_e, k_d, s_a) \mid m_e, (\bar{s}(l, k) \mid \bar{m}_k)
\end{align*}
\]
\[ \text{newO}_O(s, \tilde{t}) \overset{\text{def}}{=} (\nu s^*)(\text{newO}_O(s^*, \tilde{t})) \]
\[ \text{newA}_O(s, s_a) \overset{\text{def}}{=} (\nu m_k)(\text{newA}_O(s, m_k, s_a)) \]
\[ \text{OM}_O(s, m, k_e, k_0, \tilde{t}) \overset{\text{def}}{=} \text{OM}_O(s, m, k_e, k_0, \tilde{t}) \]

If \([k=0]\) then

\[ \text{case } l \text{ of } \text{cln}(r) : \text{OM}_O(s, m, k_e, k^*, \tilde{t}) \mid (\nu s^*)(\text{OM}_O(s^*, \tilde{t})) \]
\[ \quad \text{ali.(s_a, r) : AM}(s, m, k_e, k^*, s_a) \mid \text{OM}(s, k^*) \]
\[ \quad \text{upd.j}(l', r) : \text{OM}(s, m, k_e, k^*, l, l', \ldots, l_n) \mid \text{OM}(s, k^*) \]
\[ \quad \text{inv.j}(x, r) : \text{CM}[(\text{OM}(s, m, k_e, k^*, \tilde{t})) \mid \text{OM}(s, m, k_e, k^*, \tilde{t})] \]
\[ \quad \text{sur.(r) : CM}[(s, \text{alias}(s_{\text{clone}}))] \]
\[ \quad \text{png}(r) : \text{CM}[(s, k^*)] \]

Else if \([k=0]\) then

\[ \text{OM}_O(s, m, k_e, k^*, \tilde{t}) \mid \text{OM}(s, m, k_e, k^*, \tilde{t}) \mid (\nu s^*)(\text{OM}(s^*, \tilde{t})) \]
\[ \quad \text{ali.(s_a, r) : AM}(s, m, k_e, k^*, s_a) \mid \text{OM}(s, k^*) \]
\[ \quad \text{upd.j}(l', r) : \text{OM}(s, m, k_e, k^*, l, l', \ldots, l_n) \mid \text{OM}(s, k^*) \]
\[ \quad \text{inv.j}(x, r) : \text{CM}[(\text{OM}(s, m, k_e, k^*, \tilde{t})) \mid \text{OM}(s, m, k_e, k^*, \tilde{t})] \]
\[ \quad \text{sur.(r) : CM}[(s, \text{alias}(s_{\text{clone}}))] \]
\[ \quad \text{png}(r) : \text{CM}[(s, k^*)] \]

Else \(\text{OM}_O(s, m, k_e, k_0, \tilde{t}) \mid m_e \mid \text{OM}(s, k_0) \mid \text{OM}(s, k_0) \)

\[ \text{CM}[(\nu s^*)] \overset{\text{def}}{=} (\nu s^*)[(\nu \text{in}(y, k'), \text{in}(y, k'')) \mid (\nu \text{in}(y, k')) \mid (\nu \text{in}(y, k''))] \]

\[ \text{AM}_O(s, m, k_e, k_0, s_a) \overset{\text{def}}{=} \text{AM}_O(s, m, k_e, k_0, s_a) \]

If \([k=0]\) then

\[ \text{case } l \text{ of } \text{cln}(r) : \text{AM}_O(s, m, k_e, k^*, s_a) \mid (\nu s^*)(\text{AM}_O(s^*, s_a)) \]
\[ \quad \text{ali.(s_a, r) : AM}_O(s, m, k_e, k^*, s_a) \mid \text{OM}(s, k^*) \]
\[ \quad \text{upd.j}(l', r) : \text{AM}_O(s, m, k_e, k^*, l, l', \ldots, l_n) \mid \text{OM}(s, k^*) \]
\[ \quad \text{inv.j}(x, r) : \text{CM}[(\text{AM}_O(s, m, k_e, k^*, \tilde{t})) \mid \text{AM}_O(s, m, k_e, k^*, \tilde{t})] \]
\[ \quad \text{sur.(r) : CM}[(s, \text{alias}(s_{\text{clone}}))] \]
\[ \quad \text{png}(r) : \text{CM}[(s, k^*)] \]

Else if \([k=0]\) then \(\text{AM}_O(s, m, k_e, k^*, s_a) \mid m_k \mid (\text{OM}(s, k_0) \mid \text{OM}(s, k_0)) \)

Else \(\text{AM}_O(s, m, k_e, k_0, s_a) \mid m_e \mid (\text{OM}(s, k_0) \mid \text{OM}(s, k_0)) \)
used to provide a formal semantics used to prove an equation correct used for debugging an existing compiler
Overview

Part 0a: Encodings vs Full Abstraction
Comparison of Languages/Calculi
Correctness of Encodings

Part 0b: Asynchronous Pi Calculus

Part 1: Input-Guarded Choice
Encoding (Distributed Implementation)
Decoding (Correctness Proof)

Part 2: Output-Guarded Choice
Encoding Separate Choice
Encoding Mixed Choice

Conclusions
Intuition

Send operations are *non-blocking*.

Thus, there is neither need nor use to consider send prefixes.

Instead, include just *messages ... without continuation*.
Syntax

Let \( \mathbb{N} \) be a countable set of names.
Then, the set \( \mathcal{P} \) of processes \( P \) is defined by

\[
P ::= (x) P \mid P \mid P \mid \bar{y}z \mid 0 \mid R \mid !R
\]

\[
R ::= y(x).P
\]
Operational Semantics (I)

\[ \mu ::= y(z) \mid \bar{y}z \mid yz \mid \tau \]

<table>
<thead>
<tr>
<th>OUT:</th>
<th>( \bar{y}z \xrightarrow{\bar{y}z} 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>INP:</td>
<td>( y(x).P \xrightarrow{yz} P{z/x} )</td>
</tr>
<tr>
<td>R-INP:</td>
<td>( !y(x).P \xrightarrow{yz} P{z/x} \mid !y(x).P )</td>
</tr>
</tbody>
</table>
| COM\(^*\):| \[
\begin{align*}
P_1 \xrightarrow{\bar{y}z} P'_1 & \quad P_2 \xrightarrow{yz} P'_2 \\
\tau & \quad P_1 \mid P_2 \xrightarrow{\tau} P'_1 \mid P'_2
\end{align*}
\] |
**Operational Semantics (II)**

**Table 1**

<table>
<thead>
<tr>
<th>Transition Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OUT</strong>:</td>
</tr>
<tr>
<td>$y / DC4 z$</td>
</tr>
<tr>
<td>$y / DC4 z$</td>
</tr>
<tr>
<td><strong>INP</strong>:</td>
</tr>
<tr>
<td>$y (x). P w / DC4$</td>
</tr>
<tr>
<td>$y (x). P w / DC4$</td>
</tr>
<tr>
<td><strong>R-INP</strong>:</td>
</tr>
<tr>
<td>$y (x). P w / DC4$</td>
</tr>
<tr>
<td>$y (x). P w / DC4$</td>
</tr>
<tr>
<td><strong>COM</strong>:</td>
</tr>
<tr>
<td>$P 1 w / DC4 y z$</td>
</tr>
<tr>
<td>$P 1 w / DC4 y z$</td>
</tr>
<tr>
<td><strong>OPEN</strong>:</td>
</tr>
<tr>
<td>$P \overset{\bar{y}z}{\rightarrow} P'$</td>
</tr>
<tr>
<td>$(z) P \overset{\bar{y}(z)}{\rightarrow} P'$</td>
</tr>
<tr>
<td><strong>CLOSE</strong></td>
</tr>
<tr>
<td>$P_1 \overset{\bar{y}(z)}{\rightarrow} P'_1$</td>
</tr>
<tr>
<td>$P_2 \overset{yz}{\rightarrow} P'_2$</td>
</tr>
<tr>
<td>$P_1</td>
</tr>
</tbody>
</table>

**2.3. Bisimulation**

Two process systems are equivalent when they allow us to observe the same operational behavior. Bisimulation equivalence is defined as the mutual simulation of single computation steps resulting in equivalent system states. In the standard literature on bisimulations, e.g. [Mil89, MPW92], a simulation is a binary relation $S$ on processes such that $(P, Q) \in S$ implies, for arbitrary label $v$:

- if $P w / DC4 P'$, then there is $Q w / DC4 Q'$ such that $(P', Q') \in S$.

---

**DECODING CHOICE ENCODINGS**

---

**Wednesday, October 14, 2009**
Operational Semantics (III)

\[ PAR^*_1: \quad \begin{array}{c} P_1 \xrightarrow{\mu} P'_1 \\ P_1 \parallel P_2 \xrightarrow{\mu} P'_1 \parallel P_2 \end{array} \quad \text{if} \quad \text{bn}(\mu) \cap \text{fn}(P_2) = \emptyset \]

\[ RES: \quad \begin{array}{c} P \xrightarrow{\mu} P' \\ (x) P \xrightarrow{\mu} (x) P' \end{array} \quad \text{if} \quad x \notin \text{n}(\mu) \]

\[ ALPHA: \quad \begin{array}{c} \hat{P} \xrightarrow{\mu} \hat{P}' \\ P \xrightarrow{\mu} \hat{P}' \end{array} \quad \text{if} \quad P = \alpha \hat{P} \]
Asynchronous Bisimulation

**Definition**

A binary relation $\mathcal{S}$ on processes is a *strong simulation* if $(P, Q) \in \mathcal{S}$ implies:
Asynchronous Bisimulation

**Definition**

A binary relation $\mathcal{S}$ on processes is a *strong simulation* if $(P, Q) \in \mathcal{S}$ implies:

- if $P \xrightarrow{\mu} P'$, where $\mu$ is either $\tau$ or output with $\text{bn}(\mu) \cap \text{fn}(P | Q) = \emptyset$, there is $Q'$ such that $Q \xrightarrow{\mu} Q'$ and $(P', Q') \in \mathcal{S}$
Asynchronous Bisimulation

**Definition**

A binary relation $\mathcal{S}$ on processes is a **strong simulation** if $(P, Q) \in \mathcal{S}$ implies:

- if $P \xrightarrow{\mu} P'$, where $\mu$ is either $\tau$ or output with $\text{bn}(\mu) \cap \text{fn}(P \mid Q) = \emptyset$, there is $Q'$ such that $Q \xrightarrow{\mu} Q'$ and $(P', Q') \in \mathcal{S}$

- $(\bar{a}z \mid P, \bar{a}z \mid Q) \in \mathcal{S}$ for arbitrary messages $\bar{a}z$. 

---

Certain laws on processes have been recognized as having merely "structural" implications. Write corresponding simulations. Unless otherwise stated, we implicitly assume an asynchronous interpretation of observation throughout the paper. Here, we follow the latter approach. Unless otherwise stated, we implicitly assume an asynchronous interlabeled semantics with asynchronous observers [ACS98].

Fact 2.3.2. Listed in Table 2 in order to simplify the presentation of some derivation.

In calculi with synchronous message-passing, output-and input-transitions are dealt with symmetrically in the definition of bisimulation. In contrast, the concept of asynchronous messages suggests a nonstandard way of observing processes. Since it may be captured by a strong simulation $\mathcal{S}$, there is $Q'$ such that $Q \xrightarrow{\mu} Q'$ and $(P', Q') \in \mathcal{S}$ implies:

- if $P \xrightarrow{\mu} P'$, where $\mu$ is either $\tau$ or output with $\text{bn}(\mu) \cap \text{fn}(P \mid Q) = \emptyset$, there is $Q'$ such that $Q \xrightarrow{\mu} Q'$ and $(P', Q') \in \mathcal{S}$

- $(\bar{a}z \mid P, \bar{a}z \mid Q) \in \mathcal{S}$ for arbitrary messages $\bar{a}z$. 

---

Fact 1. Certain laws on processes have been recognized as having merely "structural" implications. Write corresponding simulations.
Asynchronous Bisimulation

**Definition**

A binary relation $\mathcal{I}$ on processes is a *strong simulation* if $(P, Q) \in \mathcal{I}$ implies:

- if $P \xrightarrow{\mu} P'$, where $\mu$ is either $\tau$ or output with $\text{bn}(\mu) \cap \text{fn}(P \mid Q) = \emptyset$, there is $Q'$ such that $Q \xrightarrow{\mu} Q'$ and $(P', Q') \in \mathcal{I}$

- $(\bar{a}z \mid P, \bar{a}z \mid Q) \in \mathcal{I}$ for arbitrary messages $\bar{a}z$.

$\mathcal{B}$ is called a *strong bisimulation* if both $\mathcal{B}$ and $\mathcal{B}^{-1}$ are strong simulations.
Asynchronous Bisimulation

**Definition**

A binary relation $\mathcal{S}$ on processes is a *strong simulation* if $(P, Q) \in \mathcal{S}$ implies:

- if $P \xrightarrow{\mu} P'$, where $\mu$ is either $\tau$ or output with $\text{bn}(\mu) \cap \text{fn}(P \mid Q) = \emptyset$, there is $Q'$ such that $Q \xrightarrow{\mu} Q'$ and $(P', Q') \in \mathcal{S}$

- $(\bar{a}z \mid P, \bar{a}z \mid Q) \in \mathcal{S}$ for arbitrary messages $\bar{a}z$.

$\mathcal{B}$ is called a *strong bisimulation* if both $\mathcal{B}$ and $\mathcal{B}^{-1}$ are strong simulations.

Replacing $Q \xrightarrow{\mu} Q'$ with $Q \xrightarrow{\hat{\mu}} Q'$ ... yields the *weak* versions.
Asynchronous Bisimulation

**Definition**

A binary relation $\mathcal{I}$ on processes is a *strong simulation* if $(P, Q) \in \mathcal{I}$ implies:

- if $P \xrightarrow{\mu} P'$, where $\mu$ is either $\tau$ or output with $\text{bn}(\mu) \cap \text{fn}(P | Q) = \emptyset$, there is $Q'$ such that $Q \xrightarrow{\mu} Q'$ and $(P', Q') \in \mathcal{I}$
- $((\bar{a}z | P, \bar{a}z | Q) \in \mathcal{I})$ for arbitrary messages $\bar{a}z$.

$\mathcal{B}$ is called a *strong bisimulation* if both $\mathcal{B}$ and $\mathcal{B}^{-1}$ are strong simulations.

Replacing $Q \xrightarrow{\mu} Q'$ with $Q \xrightarrow{\hat{\mu}} Q'$...

... yields the *weak* versions.
When Weak is Too Weak …

(* better control on T-moves *)
When Weak is Too Weak ...

(* better control on T-moves *)

**Definition**

A weak simulation $\mathcal{I}$ is called
When Weak is Too Weak ...  

(∗ better control on T-moves ∗)

**Definition**

A weak simulation $\mathcal{S}$ is called

- *progressing*, if $P \xrightarrow{\mu} P'$ implies that there is $Q'$ with $Q \xRightarrow{\mu} Q'$ such that $(P', Q') \in \mathcal{S}$
When Weak is Too Weak ...

(* better control on \( T \)-moves *)

**Definition**

A weak simulation \( \mathcal{S} \) is called

- **progressing**, if \( P \xrightarrow{\mu} P' \) implies that there is \( Q' \) with \( Q \xrightarrow{\mu} Q' \) at least one \( T \)

Table 2

\begin{tabular}{|c|c|}
\hline
1 & 2 \\
\hline
\end{tabular}
When Weak is Too Weak …

(* better control on T-moves *)

**Definition**

A weak simulation $S$ is called

- **progressing**, if $P \xrightarrow{\mu} P'$ implies that there is $Q'$ such that $Q \xrightarrow{\mu} Q'$ at least one $T$

- **strict**, if $P \xrightarrow{\mu} P'$ implies that there is $Q'$ such that $(P', Q') \in S$
When Weak is Too Weak …

(* better control on T-moves *)

**Definition**

A weak simulation $\mathcal{S}$ is called

- **progressing**, if $P \xrightarrow{\mu} P'$ implies that there is $Q'$ with $Q \xrightarrow{\mu} Q'$ at least one $T$

- **strict**, if $P \xrightarrow{\mu} P'$ implies that there is $Q'$ with $Q \xrightarrow{\mu} Q'$ at most one $T$
Definition

A weak simulation $\mathcal{S}$ is called

- **progressing**, if $P \xrightarrow{\mu} P'$ implies that there is $Q'$ with $Q \xrightarrow{\mu} Q'$ at least one $\tau$

- **strict**, if $P \xrightarrow{\mu} P'$ implies that there is $Q'$ with $Q \xrightarrow{\mu} Q'$ at most one $\tau$

for all $(P, Q) \in \mathcal{S}$ and for all $\mu$ being $\tau$ or output with $\text{bn}(\mu) \cap \text{fn}(P \mid Q) = \emptyset$. 
Too Weak (I): Expansion

[Arun-Kumar, Hennessy 1991]
Too Weak (I): Expansion

[Arun-Kumar, Hennessy 1991]

**Definition**

A binary relation $\mathcal{E}$ on processes is an *expansion* if $\mathcal{E}$ is a *progressing* simulation and $\mathcal{E}^{-1}$ is a *strict* simulation.
Too Weak (I): Expansion

[Arun-Kumar, Hennessy 1991]

**Definition**

A binary relation $\mathcal{E}$ on processes is an *expansion* if $\mathcal{E}$ is a *progressing* simulation and $\mathcal{E}^{-1}$ is a *strict* simulation.
**Too Weak (I): Expansion**

[Arun-Kumar, Hennessy 1991]

**Definition**

A binary relation $\mathcal{E}$ on processes is an *expansion* if $\mathcal{E}$ is a *progressing* simulation and $\mathcal{E}^{-1}$ is a *strict* simulation.

$\sim \subset \preceq \preceq$.
Too Weak (II): Eventual Progress

Eventually progressing simulations preserve divergence.
Too Weak (II): Eventual Progress

**Definition**

*Eventually progressing simulations preserve divergence.*
**Too Weak (II): Eventual Progress**

**Definition**

A weak simulation $\mathcal{S}$ is **eventually progressing** if, for all $(P, Q) \in \mathcal{S}$, there is a natural number $k_P \in \mathbb{N}$ such that $P \xrightarrow{\tau}^n P'$ with $n > k_P$ implies that there is $Q'$ with $Q \xrightarrow{\tau} + Q'$ such that $(P', Q') \in \mathcal{S}$.

*Eventually progressing simulations preserve divergence.*
## Structural Laws

<table>
<thead>
<tr>
<th>Law</th>
<th>Equation</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$-conversion</td>
<td>$P \equiv Q$</td>
<td>$P = \alpha Q$</td>
</tr>
<tr>
<td>Associativity</td>
<td>$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$</td>
<td></td>
</tr>
<tr>
<td>Commutativity</td>
<td>$P \mid Q \equiv Q \mid P$</td>
<td></td>
</tr>
<tr>
<td>Neutrality</td>
<td>$P \mid 0 \equiv P$</td>
<td></td>
</tr>
<tr>
<td>Scope Extrusion</td>
<td>$(y) P \mid Q \equiv (y)(P \mid Q)$</td>
<td>$y \notin \text{fn}(Q)$</td>
</tr>
<tr>
<td>Scope Elimination</td>
<td>$(y) Q \equiv Q$</td>
<td>$y \notin \text{fn}(Q)$</td>
</tr>
</tbody>
</table>
When Weak is Too Strong ...

\[ \begin{align*}
  P & \quad \tau \quad \tau \\
  A & \quad B & \quad C
\end{align*} \]

\[ \begin{align*}
  Q & \quad \tau \quad \tau \\
  A & \quad BC \\
  B & \quad \tau & \quad \tau \\
  C
\end{align*} \]
When Weak is Too Strong ...

\[ P \overset{\text{def}}{=} (i) \quad (\bar{i} \mid i.A \mid i.B \mid i.C) \]
\[ Q \overset{\text{def}}{=} (i_1)(i_2) \quad (\bar{i_1} \mid \bar{i_2} \mid i_1.A \mid i_1.(i_2.B \mid i_2.C)) \]
When Weak is Too Strong ...

\[ P \overset{\text{def}}{=} (i) \quad (\bar{i} \mid i.A \mid i.B \mid i.C) \]

\[ Q \overset{\text{def}}{=} (i_1)(i_2) \quad (\bar{i}_1 \mid \bar{i}_2 \mid i_1.A \mid i_1.(i_2.B \mid i_2.C)) \]
When Weak is Too Strong ...

\[ P \overset{\text{def}}{=} (i) \quad (\bar{i} \mid i.A \mid i.B \mid i.C) \]
\[ Q \overset{\text{def}}{=} (i_1)(i_2) \quad (\bar{i}_1 \mid \bar{i}_2 \mid i_1.A \mid i_1.(i_2.B \mid i_2.C)) \]

\( P \) and \( Q \) are **not weakly bisimilar**.
They just **simulate each other**.
However, we can do much better ...

---

Wednesday, October 14, 2009
Coupled Simulation (I)
[Parrow, Sjödin 1994]
**Coupled Simulation (I)**

**Definition**

A *mutual simulation* is a pair \((S_1, S_2)\), where \(S_1\) and \(S_2^{-1}\) are weak simulations.
**Coupled Simulation (I)**

[Parrow, Sjödin 1994]

**Definition**

A *mutual simulation* is a pair \((L_1, L_2)\), where \(L_1\) and \(L_2^{-1}\) are weak simulations.

A *coupled simulation* is a mutual simulation \((L_1, L_2)\) satisfying...
**Coupled Simulation (I)**

**Definition**

A *mutual simulation* is a pair \((\mathcal{S}_1, \mathcal{S}_2)\), where \(\mathcal{S}_1\) and \(\mathcal{S}_2^{-1}\) are weak simulations.

A *coupled simulation* is a mutual simulation \((\mathcal{S}_1, \mathcal{S}_2)\) satisfying

- if \((P, Q) \in \mathcal{S}_1\), then there is some \(Q'\) such that \(Q \Rightarrow Q'\) and \((P, Q') \in \mathcal{S}_2\)

[Parrow, Sjödin 1994]
**Coupled Simulation (I)**

[Parrow, Sjödin 1994]

**Definition**

A *mutual simulation* is a pair \((\mathcal{S}_1, \mathcal{S}_2)\), where \(\mathcal{S}_1\) and \(\mathcal{S}_2^{-1}\) are weak simulations.

A *coupled simulation* is a mutual simulation \((\mathcal{S}_1, \mathcal{S}_2)\) satisfying

- if \((P, Q) \in \mathcal{S}_1\), then there is some \(Q'\) such that \(Q \Rightarrow Q'\) and \((P, Q') \in \mathcal{S}_2\)
- if \((P, Q') \in \mathcal{S}_2\), then there is some \(P'\) such that \(P \Rightarrow P'\) and \((P', Q') \in \mathcal{S}_1\)
Coupled Simulation (I)

[Parrow, Sjödin 1994]

**Definition**

A *mutual simulation* is a pair \((S_1, S_2)\), where \(S_1\) and \(S_2^{-1}\) are weak simulations.

A *coupled simulation* is a mutual simulation \((S_1, S_2)\) satisfying

- if \((P, Q) \in S_1\), then there is some \(Q'\) such that \(Q \Rightarrow Q'\) and \((P, Q') \in S_2\)
- if \((P, Q') \in S_2\), then there is some \(P'\) such that \(P \Rightarrow P'\) and \((P', Q') \in S_1\)

\[P \leftrightarrow Q\]

*coupled simulation equivalent* (or *coupled similar*)
Coupled Simulation (II)

- if \((P, Q) \in \mathcal{S}_1\), then there is some \(Q'\) such that \(Q \Rightarrow Q'\) and \((P, Q') \in \mathcal{S}_2\)
- if \((P, Q') \in \mathcal{S}_2\), then there is some \(P'\) such that \(P \Rightarrow P'\) and \((P', Q') \in \mathcal{S}_1\)
Coupled Simulation (III)

\[ P \sim P' \sim Q' \sim Q \]

\[ \simeq \subset \Leftarrow \Rightarrow \]
Coupled Simulation (III)

\[ P \leftrightarrow Q \]

\[ Q' \leftrightarrow Q \]

\[ \approx \subseteq \cong \].

\[ \cong \text{ is a congruence on } \mathbb{P}. \]
Overview

Part 0a: Encodings vs Full Abstraction
Comparison of Languages/Calculi
Correctness of Encodings

Part 0b: Asynchronous Pi Calculus

Part 1: Input-Guarded Choice
Encoding (Distributed Implementation)
Decoding (Correctness Proof)

Part 2: Output-Guarded Choice
Encoding Separate Choice
Encoding Mixed Choice

Conclusions
Choice in Asynchronous Contexts

The intuition of asynchronous send operations is that the send actions happen instantaneously, without interference with the environment.

Having such a send compete with an receive, which itself is to wait for the availability of a message, will always make the send action win over the receive action.

So, better not mix up send and receive within a choice ...
Syntax + Semantics

The set $\mathcal{P}^\Sigma$ of processes with *input-guarded choice*

$$P ::= (x) P \mid P \mid P \mid \bar{y}z \mid 0 \mid R \mid !R \mid \sum_{j \in J} R_j$$

$$R ::= y(x).P$$

$$\sum_{j \in J} y_j(x).P_j \xrightarrow{y_kz} P_k \quad \text{if} \quad k \in J$$
Encoding Choice
Two Choice Encodings (I)

\[ S ::= P^\Sigma \]

\[ \Downarrow \]

\[ \Downarrow P \]

\[ =: T \]
Two Choice Encodings (I)

\[ S ::= \quad P^\Sigma \]

For notational convenience, the target language contains:

\[ P ::= \cdots \mid \text{test } y \text{ then } P \text{ else } P. \]

with:

\[
\begin{align*}
\text{test } l \text{ then } P_1 \text{ else } P_2 & \xrightarrow{lt} P_1 \\
\text{test } l \text{ then } P_1 \text{ else } P_2 & \xrightarrow{lf} P_2 \\
\end{align*}
\]

(* for the special names t and f *)
Two Choice Encodings (I)

For notational convenience, the target language contains:

\[ P ::= \cdots | \text{test } y \text{ then } P \text{ else } P. \]

with:

\[ \text{test } l \text{ then } P_1 \text{ else } P_2 \xrightarrow{lt} P_1 \]
\[ \text{test } l \text{ then } P_1 \text{ else } P_2 \xrightarrow{lf} P_2 \]

(* for the special names t and f *)
Two Choice Encodings (II)

\[ C[ ], D[ ] : S \rightarrow T \]
Two Choice Encodings (II)

\[ \mathcal{C} \] \( \rightarrow \) \( \mathcal{D} \): \( S \rightarrow T \)

\[
\begin{align*}
\llbracket (x) \ P \rrbracket & \overset{\text{def}}{=} (x) \llbracket P \rrbracket \\
\llbracket yz \rrbracket & \overset{\text{def}}{=} yz \\
\llbracket y(x) \ P \rrbracket & \overset{\text{def}}{=} y(x) \llbracket P \rrbracket \\
\llbracket 0 \rrbracket & \overset{\text{def}}{=} 0 \\
\llbracket ! R \rrbracket & \overset{\text{def}}{=} ! \llbracket R \rrbracket.
\end{align*}
\]
Two Choice Encodings (II)

\[ C[ ] \), \( D[ ] \]: \( S \rightarrow T \)

\[
\begin{align*}
\llbracket (x) \, P \rrbracket &\triangleq (x) \llbracket P \rrbracket \\
\llbracket yz \rrbracket &\triangleq yz \\
\llbracket y(x) \cdot P \rrbracket &\triangleq y(x) \cdot \llbracket P \rrbracket \\
\llbracket 0 \rrbracket &\triangleq 0 \\
\llbracket ! \, R \rrbracket &\triangleq ! \llbracket R \rrbracket \\
\llbracket \sum_{j \in J} R_j \rrbracket &\triangleq (l) \left( \bar{l}t \mid \prod_{j \in J} \text{Branch}_l\llbracket R_j \rrbracket \right)
\end{align*}
\]
Two Choice Encodings (II)

\[ C[\ ] \text{, } D[\ ] : S \rightarrow T \]

\[
\begin{align*}
[(x) \ P] & \overset{\text{def}}{=} (x)[P] \\
[yz] & \overset{\text{def}}{=} yz \\
y(x).P & \overset{\text{def}}{=} y(x).[P] \\
[P_1 | P_2] & \overset{\text{def}}{=} [P_1] | [P_2] \\
[0] & \overset{\text{def}}{=} 0 \\
[! R] & \overset{\text{def}}{=} ![R].
\end{align*}
\]

\[
\left[ \sum_{j \in J} R_j \right] \overset{\text{def}}{=} (l) \left( \bar{\ell} t \mid \prod_{j \in J} \text{Branch}_l \left< [R_j] \right> \right)
\]
Two Choice Encodings (III)

\[
\mathcal{C} \left[ \sum_{j \in J} R_j \right] \defeq (l) \left( \bar{l}t \mid \prod_{j \in J} \text{Read}_l\langle \mathcal{C}[R_j] \rangle \right)
\]

\begin{align*}
\text{Read}_l\langle R \rangle & \defeq y(x) \cdot \text{Test}_l\langle R \rangle \\
\text{Test}_l\langle R \rangle & \defeq \text{test } l \text{ then } \text{Commit}_l\langle R \rangle \text{ else } \text{Abort}_l\langle R \rangle \\
\text{Commit}_l\langle R \rangle & \defeq \bar{l}f \mid P \\
\text{Abort}_l\langle R \rangle & \defeq \bar{l}f \mid \bar{y}x
\end{align*}
Two Choice Encodings (IV)

\[
\mathcal{D} \left[ \sum_{j \in J} R_j \right] \overset{\text{def}}{=} (l) \left( \bar{l} t \mid \prod_{j \in J} ! \text{Read}_i \langle \mathcal{D}[R_j] \rangle \right)
\]

\[
\text{Read}_i \langle R \rangle \overset{\text{def}}{=} y(x). \text{Test}_i \langle R \rangle
\]

\[
\text{Test}_i \langle R \rangle \overset{\text{def}}{=} \text{test } l \text{ then } \text{Commit}_i \langle R \rangle \text{ else } \text{Abort}_i \langle R \rangle
\]

\[
\text{Commit}_i \langle R \rangle \overset{\text{def}}{=} \bar{l} f \mid P
\]

\[
\text{Abort}_i \langle R \rangle \overset{\text{def}}{=} \bar{l} f \mid \bar{y} x
\]
Two Choice Encodings (IV)

\[ \mathcal{D} \left[ \sum_{j \in J} R_j \right] \overset{\text{def}}{=} (l) \left( \bar{l}t \mid \prod_{j \in J} !\text{Read}_i \langle \mathcal{D} [R_j] \rangle \right) \]

\[ \text{Read}_i \langle R \rangle \overset{\text{def}}{=} y(x) \cdot \text{Test}_i \langle R \rangle \]

\[ \text{Test}_i \langle R \rangle \overset{\text{def}}{=} \text{test } l \text{ then Commit}_i \langle R \rangle \text{ else Abort}_i \langle R \rangle \]

\[ \text{Commit}_i \langle R \rangle \overset{\text{def}}{=} \bar{l}f \mid P \]

\[ \text{Abort}_i \langle R \rangle \overset{\text{def}}{=} \bar{l}f \mid \bar{y}x \]
Two Choice Encodings (IV)

\[
\mathcal{D} \left[ \sum_{j \in J} R_j \right] \overset{\text{def}}{=} (l) \left( \bar{t} | \prod_{j \in J} !Read_l(\mathcal{D} [R_j]) \right)
\]

\[
\text{Read}_l(R) \overset{\text{def}}{=} y(x) . \text{Test}_l(R)
\]

\[
\text{Test}_l(R) \overset{\text{def}}{=} \text{test } l \text{ then Commit}_l(R) \text{ else Abort}_l(R)
\]

\[
\text{Commit}_l(R) \overset{\text{def}}{=} \bar{f} \mid P
\]

\[
\text{Abort}_l(R) \overset{\text{def}}{=} \bar{f} \mid \bar{y}x
\]

\[
\text{Test}_l(R) \overset{\text{def}}{=} \text{test } l \text{ then } \text{Commit}_l(R) \oplus \text{Undo}_l(R) \text{ else } \text{Abort}_l(R)
\]

\[
\text{Undo}_l(R) \overset{\text{def}}{=} \bar{t} \mid \bar{y}x.
\]
Two Choice Encodings (IV)

\[
\mathcal{D} \left[ \sum_{j \in J} R_j \right] \overset{\text{def}}{=} (l) \left( \bar{t} \mid \prod_{j \in J} \text{Read}_l \langle \mathcal{D} \left[ R_j \right] \rangle \right)
\]

\[
\text{Read}_l \langle R \rangle \overset{\text{def}}{=} y(x). \text{Test}_l \langle R \rangle
\]

\[
\text{Test}_l \langle R \rangle \overset{\text{def}}{=} \text{test } l \text{ then Commit}_l \langle R \rangle \text{ else Abort}_l \langle R \rangle
\]

\[
\text{Commit}_l \langle R \rangle \overset{\text{def}}{=} \bar{t}f \mid P
\]

\[
\text{Abort}_l \langle R \rangle \overset{\text{def}}{=} \bar{t}f \mid \bar{y}x
\]

\[
\text{Test}_l \langle R \rangle \overset{\text{def}}{=} \text{test } l \text{ then Commit}_l \langle R \rangle \oplus \text{Undo}_l \langle R \rangle \text{ else Abort}_l \langle R \rangle
\]

\[
\text{Undo}_l \langle R \rangle \overset{\text{def}}{=} \bar{t} \mid \bar{y}x.
\]
Encoding an Example Process (I)

\[ S = \overline{y_2}z \mid N \quad \text{where} \quad N = R_1 + R_2 \quad \text{with} \quad R_i = y_i(x).P_i \]
Encoding an Example Process (I)

\[
S = \overline{y_2}z \mid N \quad \text{where} \quad N = R_1 + R_2 \quad \text{with} \quad R_i = y_i(x).P_i
\]

\[
S : \quad N \xrightarrow{\overline{y_2}z} S \xrightarrow{\tau} P_2\{z/x\}
\]
Encoding an Example Process (I)

\[ S = \overline{y_2z} \mid N \quad \text{where} \quad N = R_1 + R_2 \quad \text{with} \quad R_i = y_i(x).P_i \]

\[ S : \quad N \xleftarrow{\overline{y_2z}} S \xrightarrow{\tau} P_2\{z/x\} \]

\[ T : \quad \mathcal{C}[N] \xleftarrow{\overline{y_2z}} \mathcal{C}[S] \xrightarrow{\tau} C \xrightarrow{\tau} C_2 \]
Encoding an Example Process (I)

\[ S = \overline{y_2 z} \mid N \quad \text{where} \quad N = R_1 + R_2 \quad \text{with} \quad R_i = y_i(x).P_i \]

For some value, since there might still be a suitable message provided by the context.

\[ S : \quad N \xleftarrow{\overline{y_2 z}} S \xrightarrow{\tau} P_2 \{ z/x \} \]

\[ T : \quad \mathcal{C}[N] \xleftarrow{\overline{y_2 z}} \mathcal{C}[S] \xrightarrow{\tau} C \xrightarrow{\tau} \mathcal{C}_2 \]

\[ \equiv (l)(\overline{\ell f} \mid B_1) \mid \mathcal{C}[P_2] \{ z/x \} \]
Encoding an Example Process (I)

\[ S = \overline{y_2}z \mid N \quad \text{where} \quad N = R_1 + R_2 \quad \text{with} \quad R_i = y_i(x).P_i \]

\[ S : \quad N \xleftarrow{\overline{y_2}z} S \xrightarrow{\tau} P_2 \{ \frac{z}{x} \} \]

\[ T : \quad \mathcal{C}[N] \xleftarrow{\overline{y_2}z} \mathcal{C}[S] \xrightarrow{\tau} C \xrightarrow{\tau} C_2 \]

\[ \equiv (l)(\overline{f} \mid B_1) \mid \mathcal{C}[P_2] \{ \frac{z}{x} \} \approx 0 \]
Encoding an Example Process (I)

\[ S = \overline{y_2 z} \mid N \quad \text{where} \quad N = R_1 + R_2 \quad \text{with} \quad R_i = y_i(x) \cdot P_i \]

\[ S : \quad N \xleftarrow{\overline{y_2 z}} S \xrightarrow{\tau} P_2 \{ z/x \} \]

\[ T : \quad \mathcal{C}[N] \xleftarrow{\overline{y_2 z}} \mathcal{C}[S] \xrightarrow{\tau} C \xrightarrow{\tau} C_2 \]

\[ \equiv (l)(\overline{lf} \mid B_1) \mid \mathcal{C}[P_2] \{ z/x \} \approx 0 \]
Encoding an Example Process (I)

\[ S = \overline{y_2}z \mid N \quad \text{where} \quad N = R_1 + R_2 \quad \text{with} \quad R_i = y_i(x).P_i \]

\[ S : \quad N \leftarrow \overline{y_2}z \quad S \xrightarrow{\tau} P_2\{z/x\} \]

\[ T : \quad C[N] \leftarrow \overline{y_2}z \quad C[S] \xrightarrow{\tau} C \xrightarrow{\tau} C_2 \]

\[ S \not\approx C[S] \]

\[ \equiv (l)(\overline{l}f \mid B_1) \mid C[P_2]\{z/x\} \approx 0 \]
Encoding an Example Process (II)

\[ S = \overline{y_2}z \mid N \quad \text{where} \quad N = R_1 + R_2 \quad \text{with} \quad R_i = y_i(x).P_i \]

\[ S \cong \mathcal{D}[S] \]

\[ \mathcal{D}[N] \xleftarrow{\overline{y_2}z} \mathcal{D}[S] \overset{\tau}{\longrightarrow} D \overset{\tau}{\longrightarrow} D' \overset{\tau}{\longrightarrow} D_2 \]

\[ \equiv (l)(\overline{\bar{f}} \mid B_1 \mid B_2) \mid \mathcal{D}[P_2] \{z/x\} \]

\[ \approx 0 \]
Expanded Encodings

\[ S = \text{def} \ P \Sigma \]

\[ C \leq [\ ], D \leq [\ ] \]

\[ C [\ ], D [\ ] \]

\[ P \downarrow \triangleleft \]

\[ B [\ ] \]

\[ P_{\text{test}} \]

\[ T \]

essentially the “classical” encoding of booleans

for all \( S \in S \):

\[ C [S] \rightarrow^* T : T \preceq B [T] \]
Correctness Proof
Correctness Proof Strategy (I)

Provide a notational framework that allows to **decode** any derivative of an encoded term to “the” corresponding state in the source.

\[ S := \text{ } P^\Sigma \triangleleft \rightarrow P_{\text{test}} \Rightarrow := T \]
Correctness Proof Strategy (I)

Provide a notational framework that allows to **decode** any derivative of an encoded term to “the” corresponding state in the source.

\[ S := P^Σ \]

\[ P^Σ \rightarrow \]

\[ P^↓ \rightarrow \]

\[ P \rightarrow \]

\[ P^{test} =: T \]
Correctness Proof Strategy (II)

For the encoding with partial commitments, we have to denote two corresponding states!
Factorization
The Decoding Problem

Once a source term is translated into the target language, its source-level structure is often completely lost.
The Decoding Problem

Once a source term is translated into the target language, its source-level structure is often completely lost.

\[
\left[ \sum_{j \in J} R_j \right] \overset{\text{def}}{=} (l) \left( \overline{l}t \mid \prod_{j \in J} \text{Branch}_l \langle \overline{[R_j]} \rangle \right)
\]
The Decoding Problem

Once a source term is translated into the target language, its source-level structure is often completely lost.

\[
\left[ \sum_{j \in J} R_j \right] \overset{\text{def}}{=} \left( l \mid \prod_{j \in J} \text{Branch}_l \langle \left[ R_j \right] \rangle \right)
\]

To preserve—and track—this source-level structure under target-level computation, we introduce an \textit{annotated source language}. 
Once a source term is translated into the target language, its source-level structure is often completely lost.

\[
\left[ \sum_{j \in J} R_j \right] \overset{\text{def}}{=} (l) \left( \tilde{t} \mid \prod_{j \in J} \text{Branch}_j \langle [R_j] \rangle \right)
\]
The Decoding Problem

Once a source term is translated into the target language, its source-level structure is often completely lost.

To preserve—and track—this source-level structure under target-level computation, we introduce an annotated source language.

\[
\left[ \sum_{j \in J} R_j \right] \overset{\text{def}}{=} (l) \left( \bar{lt} \mid \prod_{j \in J} \text{Branch}_{i} \langle [R_j] \rangle \right)
\]

To preserve—and track—this source-level structure under target-level computation, we introduce an annotated source language.
The Decoding Problem

Once a source term is translated into the target language, its source-level structure is often completely lost.

\[
\left[ \sum_{j \in J} R_j \right] \overset{\text{def}}{=} (l) \left( \bar{l}t \mid \prod_{j \in J} \text{Branch}_l \langle [R_j] \rangle \right)
\]

To preserve—and track—this source-level structure under target-level computation, we introduce an annotated source language.

Consequently, the decoding is conveniently definable by exploiting the annotation structure.
The Decoding Problem

Once a source term is translated into the target language, its source-level structure is often completely lost.

\[
\left[ \sum_{j \in J} R_j \right] \overset{\text{def}}{=} (l) \left( \bar{t} | \prod_{j \in J} \text{Branch}_j \langle \left[ R_j \right] \rangle \right)
\]
The Decoding Problem

Once a source term is translated into the target language, its source-level structure is often completely lost.

\[
\left[ \sum_{j \in J} R_j \right] \overset{\text{def}}{=} (l) \left( \bar{t} \mid \prod_{j \in J} \text{Branch}_l\langle [R_j] \rangle \right)
\]
Intermediate Language (I)

annotated choice

\[
\left( \sum_{j \in J} R_j \right)^v_B
\]

- A partial function \( v : J \rightarrow V \)
- A possibly empty set \( B \subseteq J \)
  \( B \cap \text{dom}(v) = \emptyset \)
Intermediate Language (I)

**annotated choice**

\[
\left( \sum_{j \in J} R_j \right)^{v}_{B} \\
\]

- a partial function \( v: J \rightarrow V \)
- a possibly empty set \( B \subseteq J \)

\( B \cap \text{dom}(v) = \emptyset \)

corresponding to decoded source-level choice states:

- **Read-state** if \( k \in J \setminus (V \cup B) \)
- **Test-state** if \( k \in V \)
- **Commit/Abort-state** if \( k \in B \).
Definition 5.2.1 (Annotated choice). Let $J$ be a set of indices. Let $R_j = y_j(x)$. Let $P_j$ be input prefixes for $j \in J$. Let $v : J \rightarrow V$ and $B = \{k\}$. Then, $v$ and $B$ are referred to as bare and annotated choice, respectively.

Annotated choice is given the operational semantics in Table 5. The dynamics of annotated choice mimic precisely the behavior of the intended low-level process.

**READ** allows a branch $k$ in Read-state ($k \notin J \setminus (V \cup B)$) to optimistically consume a message. If the choice is not yet resolved ($B = \{\}$), **COMMIT** specifies that an arbitrary branch $k$ in Test-state ($k \in V$) can immediately evolve into its Commit-state, i.e., trigger its continuation process $P_k\{v_k/x\}$. After the choice is resolved ($B = \{\}$), **ABORT** allows any branch $k$ in Commit-state ($k \in V$) to evolve into its Abort-state to release their consumed messages. Intuitively, by reading the lock, a branch immediately leaves the choice system and exits. Therefore, annotated choice only contains branches in either Read- or Test-state.

We distinguish three cases for choice constructors that are important enough to give them names:

- **initial** for $V = \{\} = B$,
- **partial** for $V = \{\}$ and $B = \{\}$, and
- **committed** for $B = \{\}$. Note that both initial and partial choice contain all branches, whereas committed choice never does; it will even become empty, once all branches have reached their final state ($B = J$).

Committed choice exhibits a particularly interesting property: its branches in Test-state already have consumed a message which they will return after recognizing, by internally testing the lock, that the choice is already committed; its branches in Read-state are still waiting for values to be consumed and NAK/NAK since the choice is resolved and the lock carries /NAK/NAK immediately resent after an internal step. Processes with such receive-and-resend behavior are weakly bisimilar to $0$ and were called...
Annotated Choice States

(* as composed by the states of its branches *)

initial for \( V = \emptyset = B \)

partial for \( V \neq \emptyset \) and \( B = \emptyset \)

committed for \( B \neq \emptyset \)
The Key Lemma

(* the essence of asynchrony *)

\[
\left( \sum_{j \in J} R_j \right)^v \geq \prod_{j \in V} \overline{y}_j v_j.
\]

Messages consumed (see \(v\)) by committed choices (see \(B\)) are still fully available to the environment.

\[
\left( \sum_{j \in J} R_j \right) \not\equiv \emptyset B \neq \emptyset \geq 0
\]

“asynchronous garbage”
30 for choice according to the scheme in Section 4. The latter case is given by the encoding functions considered as relations.

An intermediate sublanguage restricted form, which can be made precise by characterizing the possible shape of transitions.

The proof is by structural induction on processes. The set \( \mathcal{A} \) of processes belongs to the same high-level choice, we introduce a language of (synchronous) observer, which may detect inputs, would be able to tell the difference, except that it involves additional internal computation.

We now introduce the components of a factorization for the encoding. Since the purpose of annotated choice is to keep track of which low-level actions are relevant for a process to reach an observed state, we derive an internal step and afterward release the message.

The operational semantics of \( \mathcal{A} \)-expressions to the grammar of \( \mathcal{P}(\Sigma) \). We show the proof for weak bisimulation behavior for that case is trivial. LHS does not exhibit further interaction, hence for \( n \) restricted, LHS simulates transitions of RHS. We observe that (LHS, RHS) \( \sim \) (LHS', RHS').

Due to Lemma 5.2.2, where the expansion relation holds, we conclude.

We now introduce the components of a factorization for the encoding. We show the proof for weak bisimulation behavior for that case is trivial. LHS does not exhibit further interaction, hence for \( n \) restricted, LHS simulates transitions of RHS. We observe that (LHS, RHS) \( \sim \) (LHS', RHS').

Due to Lemma 5.2.2, where the expansion relation holds, we conclude.
Intermediate Encoding

$A \left[ \sum_{j \in J} R_j \right] \overset{\text{def}}{=} \left( \sum_{j \in J} A[R_j] \right) \emptyset$

$A$ is a weak simulation up to expansion.
Intermediate Encoding

\[ A \left[ \sum_{j \in J} R_j \right] \overset{\text{def}}{=} \left( \sum_{j \in J} A[R_j] \right) \emptyset \]

\( A \) is a weak simulation up to expansion.
Flattening

\[
\mathcal{F} \left[ \left( \sum_{j \in J} R_j \right)^v \right] \overset{\text{def}}{=} (l) \left( \bar{l} b \left| \prod_{j \in J \setminus (V \cup B)} \text{Read}_1(\mathcal{F} \left[ R_j \right]) \right. \right)
\]

\[
\left| \prod_{j \in V} \text{Test}_1(\mathcal{F} \left[ R_j \right]) \left\{ v(j)/x \right\} \right.
\]

where \( b \) is \( t \), if \( B \neq \emptyset \), and \( f \), otherwise.
\[ \mathcal{F}^v \left[ \left( \sum_{j \in J} R_j \right)_B \right] \overset{\text{def}}{=} (l) \left( \bar{l}b \right) \prod_{j \in J \setminus (V \cup B)} \text{Read}_l \left( \mathcal{F} [ R_j ] \right) \prod_{j \in V} \text{Test}_l \left( \mathcal{F} [ R_j ] \right) \left\{ v(j)/x \right\} \]

where \( b = \text{t} \), if \( B \neq \emptyset \), and \( f \), otherwise

\( \mathcal{F} \) is a strong bisimulation.
Decoding
Decoding (I)

Decoding (Section 5.4) Annotated source terms deal with partial commitments explicitly. We define two decoding functions $U_{#}$ of annotated terms back into source term where $U_{#}/GS$ resets and $U_{>}$ completes partial commitments.

Expansion (Section 5.5) Since the target language $T$ is a language extended with Booleans, we show that their use in our setting is rather well behaved according to the expanding encoding $B$.

The factorization, the decodings, and the expansion enjoy several nice properties:

1. $F$ is a strong bisimulation between abbreviations and Boolean target terms.
2. $(U_{#}/GS, U_{>})$ is a coupled simulation between abbreviations and source terms.
3. $B$, i.e., a variant of it, is an expansion for Booleans in target terms.

Those can be combined to provide a coupled simulation on $S_{-}P$. The observation that every source term $S$ and its translation $C_{ETB}/ETB/S_{EM}$ are related by this relation concludes the proof of coupled-simulation-correctness of the $C_{ETB}$-encoding (Section 5.6).

Simplifications due to homomorphic encodings and decodings. Many of the proofs in this section have in common that they exhibit particular transitions of terms by constructing appropriate inference trees either

- from the inductive structure of (annotated) terms, or
- by simply replacing some leafs in the inference trees of their encodings or decodings.

Since the $A, F, U_{#}/GS, U_{>}, B$-functions are each defined homomorphically on every constructor of $P$ according to the scheme in Section 4, there is a strong syntactic correspondence between terms and their respective translations, and, as a consequence, there is also a strong correspondence between transition inference trees. More precisely, since in transitions involving choice

- there is at most one application of a choice rule, and
- an application of the choice rule always represents a leaf in the inference tree,

it suffices for all proofs to regard choice terms in isolation.
Decoding (I)

Annotated source terms deal with partial commitments explicitly. We define two decoding functions $U$ of annotated terms back into source term where $U_{\text{GS}}$ resets and $U_{\text{completes}}$ completes partial commitments.

Expansion (Section 5.5) Since the target language $T$ is a language extended with Booleans, we show that their use in our setting is rather well behaved according to the expanding encoding $B$.

The factorization, the decodings, and the expansion enjoy several nice properties:
1. $F$ is a strong bisimulation between abbreviations and Boolean target terms.
2. $(U_{\text{GS}}, U_{\text{completes}})$ is a coupled simulation between abbreviations and source terms.
3. $B$, i.e., a variant of it, is an expansion for Booleans in target terms.

Those can be combined to provide a coupled simulation on $S_P$. The observation that every source term $S$ and its translation $C_{\text{ETB}}$/$ETB$/$SUB$/$S_{\text{EM}}$ are related by this relation concludes the proof of coupled-simulation-correctness of the $C_{\text{ETB}}$-encoding (Section 5.6).

Simplifications due to homomorphic encodings and decodings. Many of the proofs in this section have in common that they exhibit particular transitions of terms by constructing appropriate inference trees either from the inductive structure of (annotated) terms, or by simply replacing some leafs in the inference trees of their encodings or decodings.

Since the $A$, $F$, $U_{\text{GS}}$, $U_{\text{completes}}$, $B$-functions are each defined homomorphically on every constructor of $P$ according to the scheme in Section 4, there is a strong syntactic correspondence between terms and their respective translations, and, as a consequence, there is also a strong correspondence between transition inference trees. More precisely, since in transitions involving choice there is at most one application of a choice rule, and an application of the choice rule always represents a leaf in the inference tree, it suffices for all proofs to regard choice terms in isolation.

25 Decoding Choice Encodings

Wednesday, October 14, 2009
Annotated Choice States

(* as composed by the states of its branches *)

\[
\text{initial for } V = \emptyset = B,
\]

\[
\text{partial for } V \neq \emptyset \text{ and } B = \emptyset
\]

\[
\text{committed for } B \neq \emptyset
\]
Decoding (II)

**initial:** \[ \mathcal{U} \left[ \left( \sum_{j \in J} R_j \right)^{\emptyset} \right] \overset{\text{def}}{=} \sum_{j \in J} \mathcal{U}[R_j] \]

**committed:** \[ \mathcal{U} \left[ \left( \sum_{j \in J} R_j \right)^{B \neq \emptyset} \right] \overset{\text{def}}{=} \prod_{j \in V} \overline{y_j} v_j \]
Decoding (II)

**initial:**
\[ \mathcal{U} \left[ \left( \sum_{j \in J} R_j \right)^0 \right] \overset{\text{def}}{=} \sum_{j \in J} \mathcal{U} \left[ R_j \right] \]

**committed:**
\[ \mathcal{U} \left[ \left( \sum_{j \in J} R_j \right)^v \right] \overset{\text{def}}{=} \prod_{j \in V} \overline{y}_j v_j \]

**partial:**
\[ \mathcal{U}_b \left[ \left( \sum_{j \in J} R_j \right)^v \right] \overset{\text{def}}{=} \prod_{j \in V} \overline{y}_j v_j \sum_{j \in J} \mathcal{U}_b \left[ R_j \right] \]
Decoding (II)

\[
\begin{align*}
\text{initial:} & \quad \mathcal{U} \left[ \left( \sum_{j \in J} R_j \right)^{\emptyset} \right] \overset{\text{def}}{=} \sum_{j \in J} \mathcal{U} [R_j] \\
\text{committed:} & \quad \mathcal{U} \left[ \left( \sum_{j \in J} R_j \right)^{B \neq \emptyset} \right] \overset{\text{def}}{=} \prod_{j \in V} \overline{y}_j v_j \\
\text{partial:} & \quad \mathcal{U}_b \left[ \left( \sum_{j \in J} R_j \right)^{\emptyset \neq \emptyset} \right] \overset{\text{def}}{=} \prod_{j \in V} \overline{y}_j v_j \mid \sum_{j \in J} \mathcal{U}_b [R_j] \\
\text{partial:} & \quad \mathcal{U}_\# \left[ \left( \sum_{j \in J} R_j \right)^{\emptyset \neq \emptyset} \right] \overset{\text{def}}{=} \overline{y}_j v_j \mid \mathcal{U}_\# [P_k] \{v_k/x\} \\
\end{align*}
\]

\[k := \text{take}(V)\]
Results (I)

\[ \mathcal{U}_p \] is a weak simulation.

\[ \mathcal{U}_#^{-1} \] is a weak simulation.
Results (I)

\( \mathcal{U}_b \) is a weak simulation.
\( \mathcal{U}^{-1} \) is a weak simulation.

\( \mathcal{U}_b \) is strict; \( \mathcal{U}^{-1} \) is progressing.
Results (I)

\[ \mathcal{U}_b \] is a weak simulation.
\[ \mathcal{U}^{-1} \] is a weak simulation.

\[ \mathcal{U}_b \] is strict; \[ \mathcal{U}^{-1} \] is progressing.

Both \[ \mathcal{U}_b \] and \[ \mathcal{U}^{-1} \] are eventually progressing.
Results (II)

Since \( U/GS/SUB/A/EM/A \) for all \( A \neq A \), the first coupling property requires the existence of transition \( U/GS/SUB/A/EM/U > SUB/A/EM \) (in \( S \), thus called \( S \)-coupling).

Lemma 5.4.6 (\( S \)-coupling).

For all \( A \neq A \), \( U/GS/SUB/A/EM/O U > SUB/A/EM \).

Proof. The proof is by structural induction on \( A \neq A \). By the simplification discussed in Section 5.1, it suffices to regard the case \( A = \emptyset \): \( j \neq J \neq R j + v B \), where, by definition of \( U > SUB/EM \), there are three subcases.

Case (initial). \( V = < = B \) or (committed) \( B \{ < : \) Immediate by Fact 5.4.5.

Case (partial). \( B = < \{ : \) Let \( k = \) take \( (V) \). Then, \( U/GS/SUB/A/EM = ` j \neq V y j v j = ` j \neq U U/GS/SUB/R j/EM w/DC4 \{ ` j \neq V & k y j v j = ` U > SUB/P k/EM _k = U > SUB/P k/EM _k = U > SUB/A/EM \) since \( P k _k \) is fully committed.

There may be several occurrences of partially committed choices in a term \( A \), but, by definition, they only occur unguarded. We may simply collect the corresponding internal steps in either order which leads to \( A O A $.

Since \( A/p U > SUB/A/EM \) for all \( A \neq A \), the second coupling property addresses \( U > -related \) terms. In this case, it is not as simple as for the \( S \)-coupling to denote what coupling means, so we explain it a bit more carefully: For all \( A \neq A \), whenever \( (A, S)# U > \), i.e., \( S = U/GS/SUB/A/EM/S \), there is an internal sequence \( A O A $ (in \( A \), thus called \( A \)-coupling), such that \( (A$, \( S)# U/GS ), i.e., \( S = U/GS/SUB/A $/EM/S $). If we link the two equations for \( S \), we get the coupling requirement \( U > SUB/A/EM = U/GS/SUB/A/EM $ for \( A O A $.

\[ U_b[A] \xrightarrow{\sim} U[A] \xrightarrow{\sim} U[A] \]
Results (II)

\[ U_b[A] \longrightarrow U_h[A] \]

\[ A \quad \Rightarrow \quad A \]

\[ U_h[A] = U_h[A'] = U_b[A'] \]

\[ A \quad \Rightarrow \quad A' \]
Results (II)

\[(\mathcal{U}_b, \mathcal{U}_\#) \text{ is a coupled simulation.}\]

\[(\mathcal{U}^{-1}_\#, \mathcal{U}^{-1}_b) \text{ is a coupled simulation.}\]
Results (III)

Proof. Directly from the operational correspondence Lemma 5.5.1. The eventually progressing property for \((B/DC4)\&1\) comes with an upper bound for trivially simulating \(-\text{steps as determined by the (finite) number of "active" test-expressions, multiplied with 5 as the worst case that all of the active test-expressions have just done the first step.}

Main Result

In this section, we establish a coupled simulation (cf. Definition 2.4.1) between source terms and their \(C/ETB\)-translations by exploiting the results for the \(A\)-encoding via the decodings \(U/\text{GS}\) and \(U/\text{EM}\). Reasoning about the annotated versions of choice allowed us to use their high-level structure for the decoding functions. We argued that we could safely concentrate on the annotated language \(A\), since \(F/\text{EM}\) flattens abbreviation terms correctly (up to \(t\)) into terms of \(P/\text{test}\), whereas \(B/\text{EM}\) expands \(\text{test}\)-expressions correctly (up to \(/ETB\)) into terms of \(P\). In order to combine those ideas, let the simulations \(C\) (completeness) and \(S\) (soundness) be defined by

\[
\begin{align*}
C &= \text{def} \ U/\text{SUB/EM} F/\text{DC4} \quad \text{and} \\
S &= \text{def} \ U/\text{SUB/EM} F/\text{DC4} \quad \text{according to the diagram}
\end{align*}
\]

where the relations \(U/\text{GS}\) and \(U/\text{EM}\) are only defined on the subset \(A\) of \(P\). The results for annotated terms carry over smoothly to the expanded versions.

Theorem 5.6.1 \((C,S)\) is a coupled simulation.

Proof. By Corollary 5.4.10, Proposition 5.3.4, and Lemma 2.5.3 twice.

Observe that \(C\) is constructed from the committing decoding \(U/\text{SUB/EM}\), so derivatives of target terms are at most as committed as their \(C\)-related source terms. Analogously, \(S\) is constructed from the resetting decoding \(U/\text{GS}/\text{SUB/EM}\), so derivatives of target terms are at least as committed as their \(S\)-related source terms. By construction, the relations \(C\) and \(S\) are big enough to contain all source and target terms and, in particular, to relate all source terms and their \(C/ETB\)-translations.

Lemma 5.6.2

For all \(S:\ (S,C)\&1\): \((S,C)\&1\)
Results (III)

\[ \begin{align*}
\mathcal{C} & \overset{\text{def}}{=} \mathcal{U} \uparrow^{-1} \bar{F} \quad \mathcal{B} \\
\mathcal{C} & \overset{\text{def}}{=} \mathcal{U} \downarrow^{-1} \bar{F} \quad \mathcal{B} \\
\end{align*} \]
Results (III)

(\mathcal{C}, \mathcal{G}) \text{ is a coupled simulation.}
Results (III)

\[
\begin{align*}
\mathcal{C} & \overset{\text{def}}{=} \mathcal{U}^{-1} \mathcal{F} \mathcal{B} \\
\mathcal{S} & \overset{\text{def}}{=} \mathcal{U}_b^{-1} \mathcal{F} \mathcal{B} \\
\end{align*}
\]

\( (\mathcal{C}, \mathcal{S}) \) is a coupled simulation.

For all \( S \in \mathcal{S} : (S, \mathcal{C} \preceq [S]) \in \mathcal{C} \cap \mathcal{S}. \)
Results (III)

$(C, \mathcal{S})$ is a coupled simulation.

For all $S \in \mathcal{S}$: $(S, C \preceq [S]) \in C \cap \mathcal{S}$.

For all $S \in \mathcal{S}$: $S \Rightarrow C \Rightarrow [S]$. 
Results (IV)

Running the same proof strategy for the atomically committing encoding, we may use a single decoding functions.

\[ \mathcal{U} \text{ is a weak bisimulation.} \]

For all \( S \in \mathcal{S} : S \approx \mathcal{D}[S]. \)
Results (V)

\[ \mathcal{C}^{-1} \text{ is eventually progressing.} \]

\[ \mathcal{C} \preceq \mathcal{C} \text{ is divergence-free.} \]
Operational Correspondence

(* for free *)

![Diagram](image)

\[ S \xrightarrow{a_1 \ldots a_n} S' \xrightarrow{S} S'' \]

\[ \mathcal{C} \left[ S \right] \xrightarrow{a_1 \ldots a_n} T \xrightarrow{\mathcal{C} \left[ S'' \right]} \]

Wednesday, October 14, 2009
Intermediate Conclusions
A bit of cheating: encodings depend on n-ary choice.

The role of name-passing.

Asynchronous Pi is “sufficiently” expressive ... ... for programming. (Who needs mixed choice?)

The role of coupled simulation.

There is no theory of encodings yet.
Overview

Part 0a: Encodings vs Full Abstraction
Comparison of Languages/Calculi
Correctness of Encodings

Part 0b: Asynchronous Pi Calculus

Part 1: Input-Guarded Choice
Encoding (Distributed Implementation)
Decoding (Correctness Proof)

Part 2: Output-Guarded Choice
Encoding Separate Choice
Encoding Mixed Choice

Conclusions
Calculus
Choice in Synchronous(!) Contexts

As motivated by Catuscia Palamidessi’s talk, we focus now on encodings of choice operators that may contain (synchronous) send prefixes.
Choice in Synchronous(!) Contexts

As motivated by Catuscia Palamidessi’s talk, we focus now on encodings of choice operators that may contain (synchronous) send prefixes.

We try to investigate some limits of encodability. In contrast to Catuscia Palamidessi, we seek positive correctness results.
\[ \pi\text{-calculus with mixed choice} \]

\[ \pi\text{-calculus with separate choice} \]

- Identity encoding
- \( \pi\text{-calculus with input choice} \)
  - (no output prefix)
  - Nestmann–Pierce 96

- Identity encoding
- \( \pi\text{-calculus without choice} \)
  - (output prefix)
  - Nestmann 97
  - Honda–Tokoro 91, Boudol 92

- Asynchronous \( \pi\text{-calculus} \)
  - (no choice, no output prefix)

Palamidessi 97

Nestmann 97

Identity encoding

Honda

Tokoro 91, Boudol 92

Pierce 96
\[ \pi \text{- calculus with mixed choice} \]
\[ \text{Palamidessi 97} \]
\[ \pi \text{- calculus with separate choice} \]
\[ \text{Identity encoding} \]
\[ \pi \text{- calculus with input choice} \]
\[ \text{Nestmann-Pierce 96} \]
\[ \pi \text{- calculus without choice} \]
\[ \text{(output prefix)} \]
\[ \text{Honda-Tokoro 91,Boudol 92} \]
\[ \text{Asynchronous } \pi \text{- calculus} \]
\[ \text{(no choice, no output prefix)} \]
Fig. 2.2. The π-calculus hierarchy.

The dashed line represents an identity encoding.

π-calculus with mixed choice

Palamidessi 97

π-calculus with separate choice

Identity encoding

Identity encoding

π-calculus with input choice
(no output prefix)

Nestmann 97

π-calculus without choice
(output prefix)

Honda–Tokoro 91, Boudol 92

Nestmann–Pierce 96

Asynchronous π-calculus
(no choice, no output prefix)
Syntax + Semantik

\[ \pi : ::= y!\{\tilde{z}\} \mid y?[\tilde{x}] \]

\[ \mathcal{S}_{\text{inp}}: P ::= \ldots \mid \sum_{i \in I} y_i?[\tilde{x}_i].P_i \]

\[ \mathcal{T}: P ::= \ldots \mid y?[\tilde{x}].P \mid y!\{\tilde{z}\} \]
Syntax + Semantik

\[ \pi : \triangleq y!\tilde{z} \mid y?\tilde{x} \]

\[ \sum_{i \in I} \pi_i \cdot P_i \]

\[ \sum_{i \in I} y_i?\tilde{x}_i \cdot P_i \]

\[ y?\tilde{x} \cdot P \mid y!\tilde{z} \]
\[ \pi : ::= y!\{ \tilde{z} \} \mid y?\{ \tilde{x} \} \]

\[ \mathcal{S}^{\text{mix}}: P : ::= \ldots \mid \sum_{i \in I} \pi_i \cdot P_i \]

\[ \mathcal{S}^{\text{sep}}: P : ::= \ldots \mid \sum_{i \in I} y_i ?\{ \tilde{x}_i \} \cdot P_i \mid \sum_{i \in I} y_i !\{ \tilde{z}_i \} \cdot P_i \]

\[ \mathcal{S}^{\text{inp}}: P : ::= \ldots \mid \sum_{i \in I} y_i ?\{ \tilde{x}_i \} \cdot P_i \]

\[ \top : P : ::= \ldots \mid y?\{ \tilde{x} \} \cdot P \mid y!\{ \tilde{z} \} \]
Reduction Semantics

\[ S^{\text{mix}}, S^{\text{sep}} : \quad (\cdots + y?[\tilde{x}].P) \mid (y![\tilde{z}].Q + \cdots) \rightarrow P\{\tilde{z}/\tilde{x}\} \mid Q \]

\[ S^{\text{mix}}, S^{\text{sep}} : \quad y?*[\tilde{x}].P \mid (y![\tilde{z}].Q + \cdots) \rightarrow P\{\tilde{z}/\tilde{x}\} \mid Q \mid y?*[\tilde{x}].P \]

\[ S^{\text{inp}} : \quad (\cdots + y?[\tilde{x}].P) \mid y![\tilde{z}] \rightarrow P\{\tilde{z}/\tilde{x}\} \]

\[ S^{\text{inp}}, T : \quad y?*[\tilde{x}].P \mid y![\tilde{z}] \rightarrow P\{\tilde{z}/\tilde{x}\} \mid y?*[\tilde{x}].P \]

\[ T : \quad y?[\tilde{x}].P \mid y![\tilde{z}] \rightarrow P\{\tilde{z}/\tilde{x}\} \]

\[ T : \quad \text{test } y \text{ then } P \text{ else } Q \mid y![t] \rightarrow P \]

\[ T : \quad \text{test } y \text{ then } P \text{ else } Q \mid y![f] \rightarrow Q \]

\[ \text{if } P \rightarrow P' \text{ then } (\nu x) P \rightarrow (\nu x) P' \]

\[ \text{if } P \rightarrow P' \text{ then } Q \mid P \rightarrow Q \mid P' \]

\[ \text{if } P \equiv Q \rightarrow Q' \equiv P' \text{ then } P \mid Q \equiv Q \mid P \]

\[ (\nu y) (\nu x) P \equiv (\nu x) (\nu y) P \]

\[ P \mid (Q|R) \equiv (P|Q) \mid R \]

\[ (\nu y) P \mid Q \equiv (\nu y)(P \mid Q) \text{ if } y \not\in \text{fn}(Q) \]
Encodings
Homomorphic Encoding Scheme

\[ [P_1 \mid P_2] \overset{\text{def}}{=} [P_1] \mid [P_2] \]
\[ [(\nu x) P] \overset{\text{def}}{=} (\nu x) [P] \]
\[ \left[ \sum_{i \in I} \pi_i.P_i \right] \overset{\text{def}}{=} (\nu l) \left( l![t] \mid \prod_{i \in I} [\pi_i.P_i]_l \right) \]
Homomorphic Encoding Scheme

\[
\begin{align*}
[ P_1 | P_2 ] & \overset{\text{def}}{=} [ P_1 ] | [ P_2 ] \\
[ (\nu x) P ] & \overset{\text{def}}{=} (\nu x) [ P ] \\
[ \sum_{i \in I} \pi_i.P_i ] & \overset{\text{def}}{=} (\nu l) \ ( l! [t] | \prod_{i \in I} [ \pi_i.P_i ]_l )
\end{align*}
\]

Encoding Input-Guarded Choice:

\[
\begin{align*}
[ y! [\tilde{z}] ] & \overset{\text{def}}{=} y! [\tilde{z}] \\
[ y?[\tilde{x}].P ]_l & \overset{\text{def}}{=} y?[\tilde{x}] \ . \ \text{test } l \ \text{then } ( l! [f] | [ P ] ) \ \text{else } ( l! [f] | y! [\tilde{x}] ) \\
[ y*[\tilde{x}].P ] & \overset{\text{def}}{=} y*[\tilde{x}]. [ P ]
\end{align*}
\]

The **locking game** on \( l \) does the trick.
Homomorphc Encoding Scheme

\[
\begin{align*}
[P_1 \mid P_2] & \overset{\text{def}}{=} \ [P_1] \mid [P_2] \\
(\nu x) P & \overset{\text{def}}{=} (\nu x) [P] \\
\sum_{i \in I} \pi_i . P_i & \overset{\text{def}}{=} (\nu l) ( l! [t] \mid \prod_{i \in I} [\pi_i . P_i]_l )
\end{align*}
\]

Encoding Input-Guarded Choice:

\[
\begin{align*}
[y! [\tilde{z}]] & \overset{\text{def}}{=} y! [\tilde{z}] \\
[y? [\tilde{x}] . P] & \overset{\text{def}}{=} y? [\tilde{x}] \cdot \text{test } l \text{ then } ( l! [f] \mid [P] ) \text{ else } ( l! [f] \mid y! [\tilde{x}] ) \\
[y* [\tilde{x}] . P] & \overset{\text{def}}{=} y* [\tilde{x}] . [P]
\end{align*}
\]

The **locking game** on \( l \) does the trick.
Homomorphism Encoding Scheme

\[
\begin{align*}
[ P_1 \mid P_2 ] & \overset{\text{def}}{=} [ P_1 ] \mid [ P_2 ] \\
[(\nu x) P] & \overset{\text{def}}{=} (\nu x) [ P ] \\
[ \sum_{i \in I} \pi_i . P_i ] & \overset{\text{def}}{=} (\nu l) (l! [t] \mid \prod_{i \in I} [ \pi_i . P_i ]_l)
\end{align*}
\]

Encoding Input-Guarded Choice:

\[
\begin{align*}
[y! [\tilde{z}]] & \overset{\text{def}}{=} y! [\tilde{z}] \\
[y? [\tilde{x}] . P]_l & \overset{\text{def}}{=} y? [\tilde{x}] . \text{test } l \text{ then } (l! [t] \mid [P]) \text{ else } (l! [t] \mid y! [\tilde{x}]) \\
[y* [\tilde{x}] . P] & \overset{\text{def}}{=} y* [\tilde{x}] . [P]
\end{align*}
\]

The **locking game** on \( l \) does the trick.
Homomorphic Encoding Scheme

\[
\begin{align*}
[ P_1 \mid P_2 ] & \overset{\text{def}}{=} [ P_1 ] \mid [ P_2 ] \\
[ (\nu x) P ] & \overset{\text{def}}{=} (\nu x) [ P ] \\
[ \sum_{i \in I} \pi_i . P_i ] & \overset{\text{def}}{=} (\nu l) ( l![t] \mid \prod_{i \in I} [ \pi_i . P_i ]_l )
\end{align*}
\]

Encoding Input-Guarded Choice:

\[
\begin{align*}
[ y![\tilde{z}] ] & \overset{\text{def}}{=} y![\tilde{z}]
\\
[ y?[^{\tilde{x}}] . P ]_l & \overset{\text{def}}{=} y?[^{\tilde{x}}] . \text{test } l \text{ then } ( l![f] \mid [ P ] ) \text{ else } ( l![f] \mid y?[\tilde{x}] )
\\
[ y?[^{\tilde{x}}] . P ] & \overset{\text{def}}{=} y?[^{\tilde{x}}] . [ P ]
\end{align*}
\]

The **locking game** on \( l \) does the trick.
Encoding Separate Choice

\[ [y![\bar{z}].P]_r \overset{\text{def}}{=} (\nu a) ( y![r, a, \bar{z}] | \text{test } a \text{ then } [P] \text{ else } 0 ) \]
Encoding Separate Choice

\[
[y!][\bar{z}].P_r \quad \overset{\text{def}}{=} \quad (\nu a)(y![r, a, \bar{z}] | \text{test } a \text{ then } [P] \text{ else } 0)
\]
Encoding Separate Choice

\[
[y![\tilde{z}],P]_r \overset{\text{def}}{=} \nu a \left( y![r,a,\tilde{z}] \mid \text{test } a \text{ then } [P] \text{ else } 0 \right)
\]

\[
[y?[\tilde{x}],P]_l \overset{\text{def}}{=} \nu b \left( b![] \mid b?*[] \right).
\]

\[
y?[r,a,\tilde{x}].
\]

\textit{l: local} lock

\textit{r: remote} lock

\text{test } l

\text{then test } r

\text{then} \ l![f] \mid r![f] \mid a![t] \mid [P]

\text{else} \ l![t] \mid r![f] \mid a![f] \mid b![]

\text{else} \ l![f] \mid y![r,a,\tilde{x}] \)

\text{Evaluation.} \text{ An encoding is deadlock-free, if it does not add deadlock loops to the behavior of terms; a deadlock loop that occurs in (some derivative of) an encoded term necessarily results from a deadlock loop already occurring in (some derivative of) the original term. Note that divergence-freedom implies livelock-freedom. To prove deadlock-freedom, we take advantage of type information for the channels that are added in the encoding. We refine channel types according to Kobayashi's classification [Kob97], which distinguishes between reliable and unreliable channels. The following three types of channels are reliable: linear channels, which are used just once (like our acknowledgement channels a), } y?[r,a,\tilde{x}].}
Encoding Separate Choice

\[
\begin{align*}
[y!\tilde{z}.P]_r & \overset{\text{def}}{=} (\nu a) (y![r,a,\tilde{z}] \mid \text{test } a \text{ then } [P] \text{ else } 0 ) \\
[y?\tilde{x}.P]_l & \overset{\text{def}}{=} (\nu b) (b![] \mid b?[]).
\end{align*}
\]

\begin{itemize}
\item \textit{l: local} lock
\item \textit{r: remote} lock
\end{itemize}

\[
\begin{align*}
[y!*\tilde{x}.P] & \overset{\text{def}}{=} y?[r,a,\tilde{x}].\text{test } r \text{ then } r![f] \mid a![t] \mid [P] \text{ else } r![f] \mid a![f] \\
\end{align*}
\]
Encoding Separate Choice

\[ S^{\text{sep}} \rightarrow T \] is divergence-free.

Proof: only sketched, but reasonably straightforward.
Encoding Separate Choice

\[ \mathbb{S}^{\text{sep}} \rightarrow \top \text{ is divergence-free.} \]

Proof: only sketched, but reasonably straightforward.

\[ \mathbb{S}^{\text{sep}} \rightarrow \top \text{ is deadlock-free.} \]

Proof: using a sophisticated type system by Kobayashi et.al.
Encoding Mixed Choice

So, we seem to know how to implement input-guards and how to implement output-guards.

Why not reuse the same encoding also for the case of mixed choice?
Symmetric Cyclic Wait

\[
P \parallel Q \overset{\text{def}}{=} y_0!\{0\}.P_0 + y_1?[x].P_1
\]
\[
| y_0?[x].Q_0 + y_1!\{1\}.Q_1
\]
Symmetric Cyclic Wait

\[ P | Q \overset{\text{def}}{=} y_0 ![0]. P_0 + y_1 ?[x] . P_1 \]
\[ \quad | y_0 ?[x] . Q_0 + y_1 ![1] . Q_1 \]

\( P \) and \( Q \)'s code is symmetric.

\[
\begin{array}{c}
\text{\( y_0 \)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\( y_1 \)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\( y_0 \)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\( y_1 \)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\( y_0 \)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\( y_1 \)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\( y_0 \)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\( y_1 \)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\( y_0 \)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\( y_1 \)} \\
\end{array}
\]
Symmetric Cyclic Wait

\[ P \mid Q \overset{\text{def}}{=} y_0!\langle 0 \rangle . P_0 + y_1?\langle x \rangle . P_1 \]
\[ \mid y_0?\langle x \rangle . Q_0 + y_1!\langle 1 \rangle . Q_1 \]

\( P \) and \( Q \)'s code is symmetric.

Mixed guarded choice can break the symmetry!
But the encodings of separate choice may deadlock ...
Symmetric Cyclic Wait

\[ P \mid Q \stackrel{\text{def}}{=} y_0!\[0].P_0 + y_1?[x].P_1 \]
\[ \mid y_0?[x].Q_0 + y_1!\[1].Q_1 \]

\[ [ y?[\bar{x}].P ]_l \stackrel{\text{def}}{=} (\nu b) (b![] \mid b?*[]). \]
\[ y?[r, a, \bar{x}]. \]
\[ \text{test } l \]
\[ \text{then test } r \]
\[ \text{then } l![f] \mid r![f] \mid a![t] \mid [ P ] \]
\[ \text{else } l![t] \mid r![f] \mid a![f] \mid b[] \]
\[ \text{else } l![f] \mid y?[r, a, \bar{x}] \) \]

Wednesday, October 14, 2009
Symmetric Cyclic Wait

\[ P \parallel Q \overset{\text{def}}{=} y_0!\{0\}.P_0 + y_1?[x].P_1 \]
\[ | y_0?[x].Q_0 + y_1![1].Q_1 \]

\( P \) and \( Q \)'s code is symmetric.

Mixed guarded choice can break the symmetry!
But the encodings of separate choice may deadlock...
Incest

\[ I := y!\,[z]\cdot P + y?\,[x]\cdot Q. \]
Incest

\[ I := y![z].P + y?[x].Q \]
Incest

\[ I := y!\{z\}.P + y?\{x\}.Q \]

No Intra-Communication possible on the source!
But the encoding of separate choice may deadlock ...
Nondeterminism/Randomization

\[
\left[y?\bar{x}\right].P \downarrow \overset{\text{def}}{=} (\nu b) \left( b![] \mid b?[] \cdot y?[r, a, \bar{x}] \right).
\]

\[
(\nu s, lcl, rmt, rnd, bth) \cdot lcl?[] \cdot l?[b_L] \cdot s?[tag, b] \cdot
\]

- \text{if } tag=\text{R} \text{ then } bth![b_L, b] \mid s![N, f] \text{ else } s![L, b_L] \mid \text{rnd}![]
- \text{if } tag=\text{L} \text{ then } bth![b, b_R] \mid s![N, f] \text{ else } s![R, b_R] \mid \text{rnd}![]

\[
rnd?[] \cdot s?[tag, b] \cdot ( s![N, f] \mid \text{if } tag=\text{L} \text{ then } l![b] \mid lcl[] \text{ else }
\]

- \text{if } tag=\text{R} \text{ then } r![b] \mid \text{rmt}[] \text{ else } 0

\[
| bth?[b_L, b_R] \cdot \text{if } b_L \land b_R \text{ then } l![f] \mid r![f] \mid a![t] \mid [ P ] \text{ else }
\]

- \text{if } b_L \text{ then } l![t] \mid r![f] \mid a![f] \mid b[] \text{ else }
- \text{if } b_R \text{ then } l![f] \mid r![t] \mid y![r, a, \bar{x}] \text{ else } l![f] \mid r![f] \mid a![f]

\[
| lcl[] \mid rmt[] \mid s![N, f] )
\]
The encoding is:

• loosely inspired by [Rabin, Lehmann 94]
• (strongly) compositional
• obeys a name discipline
• deadlock-free
• ... but not livelock-free ... it introduces divergence
• fully abstract in a very restricted way ...
• a candidate for an explicit probabilistic treatment
Mixed-Guarded Choice

\[
\begin{align*}
[\nu x P] &= \nu x [P] \\
[P_1 | P_2] &= [P_1] | [P_2] \\
[X] &= X \\
[\text{rec}_X P] &= \text{rec}_X [P]
\end{align*}
\]

\[
\left[ \sum_i \alpha_i P_i + \sum_j \tau Q_j + \sum_k \beta_k R_k \right] = \nu l (\tilde{t} \mid \nu h (\bar{h} \mid \prod_i [\alpha_i P_i]_l h) \mid \prod_j [\tau Q_j]_l \mid \prod_k [\beta_k R_k]_l)
\]

\[
[\bar{x}y. P]_{rh} = \nu a (\bar{x} \langle r, a, h, y \rangle \mid a(b). \text{if } b \text{ then } [P] \text{ else } 0)
\]

\[
[\tau Q]_l = l(b). (\tilde{f} \mid \text{if } b \text{ then } [Q] \text{ else } 0)
\]

\[
[x(y). R]_l = \text{rec}_X (x(r, a, h, y). h. \text{rec}_Y (1/2 \tau l(b_L).((1 - \varepsilon) r(b_R). B + \varepsilon \tau. (\bar{r} b_L \mid Y)) + 1/2 \tau r(b_R).((1 - \varepsilon) l(b_L). B + \varepsilon \tau. (\bar{r} b_R \mid Y)) ))
\]

where

\[
B = \begin{cases} 
\text{if } b_L \wedge b_R & \text{then } \bar{h} | \tilde{f} | \bar{f} | \bar{a} t | [R] \\
\text{else if } b_L & \text{then } \bar{h} | \tilde{f} | \bar{f} | \bar{a} f | X \\
\text{else if } b_R & \text{then } \bar{h} | \tilde{f} | \bar{f} | \bar{a} \langle r, a, h, y \rangle \\
\text{else} & \bar{h} | \tilde{f} | \bar{f} | \bar{a} f
\end{cases}
\]
Mixed-Guarded Choice

\[
[\nu x P] = \nu x [P]
\]
\[
[P_1 | P_2] = [P_1] | [P_2]
\]
\[
[X] = X
\]
\[
[\text{rec}_X P] = \text{rec}_X [P]
\]
\[
\left[ \sum_i \alpha_i \cdot P_i + \sum_j \tau \cdot Q_j + \sum_k \beta_k \cdot R_k \right] = \nu l (\bar{t} | \nu h (\bar{h} | \prod_i [\alpha_i \cdot P_i]_{lh}) | \prod_j [\tau \cdot Q_j]_l | \prod_k [\beta_k \cdot R_k]_l)
\]
\[
[\bar{x} y. P]_{rh} = \nu a (\bar{x} \langle r, a, h, y \rangle | a(b). \text{if } b \text{ then } [P] \text{ else } 0)
\]
\[
[\tau \cdot Q]_l = l(b). (\bar{t} | \text{if } b \text{ then } [Q] \text{ else } 0)
\]
\[
[x(y). R]_l = \text{rec}_X (x(r, a, h, y). h. \text{rec}_Y (1/2 \tau. l(b_L). ((1 - \varepsilon) r(b_R). B + \varepsilon \tau. (\bar{b}_L | Y))
+ 1/2 \tau. r(b_R). ((1 - \varepsilon) l(b_L). B + \varepsilon \tau. (\bar{b}_R | Y)) ))
\]

where
\[
B = \begin{cases} 
  \text{if } b_L \land b_R & \text{then } \bar{h} | \bar{t} | \bar{f} | \bar{t} \text{ a } [R] \\
  \text{else if } b_L & \text{then } \bar{h} | \bar{t} | \bar{f} | \bar{f} \text{ a } X \\
  \text{else if } b_R & \text{then } \bar{h} | \bar{t} | \bar{f} | \bar{t} \text{ a } \bar{x} \langle r, a, h, y \rangle \\
  \text{else} & \bar{h} | \bar{t} | \bar{f} | \bar{f}
\end{cases}
\]

co-stimulated the development of probabilistic Pi Calculus
Mixed-Guarded Choice

\[
\begin{align*}
[\nu x P] &= \nu x[P] \\
[P_1 | P_2] &= [P_1] | [P_2] \\
[X] &= X \\
[\text{rec}_X P] &= \text{rec}_X[P]
\end{align*}
\]

\[
\begin{align*}
\left[\sum_i \alpha_i \cdot P_i + \sum_j \tau \cdot Q_j + \sum_k \beta_k \cdot R_k \right] &= \nu l (\bar{t} | \nu h (\bar{h} | \Pi_i[\alpha_i \cdot P_i]_{lh}) | \Pi_j[\tau \cdot Q_j]_l | \Pi_k[\beta_k \cdot R_k]_l) \\
[\bar{xy}.P]_{rh} &= \nu a (\bar{x}\langle r, a, h, y \rangle | a(b). \text{if } b \text{ then } [P] \text{ else } 0) \\
[\tau.Q]_l &= l(b).(\bar{f} | \text{if } b \text{ then } [Q] \text{ else } 0) \\
[x(y).R]_l &= \text{rec}_X (x(r, a, h, y).h.\text{rec}_Y (\frac{1}{2} \tau.l(b_L).(b_R).((1 - \varepsilon) r(b_R).B + \varepsilon \tau.\bar{b}_L | Y)) + \frac{1}{2} \tau.r(b_R).(b_L).B + \varepsilon \tau.\bar{r}b_R | Y))
\end{align*}
\]

where

\[
B = \begin{cases} 
\text{if } b_L \wedge b_R & \text{then } \bar{h} | \bar{f} | \bar{t} | \bar{a} | [R] \\
\text{else if } b_L & \text{then } \bar{h} | \bar{t} | \bar{r} | \bar{a} | X \\
\text{else if } b_R & \text{then } \bar{h} | \bar{f} | \bar{t} | \bar{x}\langle r, a, h, y \rangle \\
\text{else} & \text{else } \bar{h} | \bar{f} | \bar{r} | \bar{a}
\end{cases}
\]
A "Bakery" Algorithm (I)

\[
\begin{align*}
\llbracket P \rrbracket & \overset{\text{def}}{=} (\nu c) \left( c!^{[42]} \mid \llbracket P \rrbracket^c \right) \\
\llbracket P_1 \mid P_2 \rrbracket^c & \overset{\text{def}}{=} \llbracket P_1 \rrbracket^c \mid \llbracket P_2 \rrbracket^c \\
\llbracket (\nu x) P \rrbracket^c & \overset{\text{def}}{=} (\nu x) \llbracket P \rrbracket^c \\
\llbracket \sum_{i \in I} \pi_i . P_i \rrbracket^c & \overset{\text{def}}{=} c?[n]. \left( c![n+1] \mid (\nu l) \left( l![t] \mid \prod_{i \in I} \llbracket \pi_i . P_i \rrbracket^c_{n,l} \right) \right) \\
\llbracket y!^{[\tilde{z}]} . P \rrbracket^c_{n,l} & \overset{\text{def}}{=} (\nu a) \left( y![n, l, a, \tilde{z}] \mid \text{test } a \text{ then } \llbracket P \rrbracket^c \text{ else } 0 \right)
\end{align*}
\]
A “Bakery” Algorithm (I)

The following encoding is not strongly compositional!

\[
\begin{align*}
[ P ] & \overset{\text{def}}{=} (\nu c) ( c![42] \mid [ P ]^c ) \\
[ P_1 \mid P_2 ]^c & \overset{\text{def}}{=} [ P_1 ]^c \mid [ P_2 ]^c \\
[ (\nu x) P ]^c & \overset{\text{def}}{=} (\nu x) [ P ]^c \\
\sum_{i \in I} \pi_i . P_i ]^c & \overset{\text{def}}{=} c?[n].( c![n+1] \mid (\nu l) ( l![t] \mid \prod_{i \in I} [ \pi_i . P_i ]_{n,l}^c ) ) \\
y! [\tilde{z}] . P ]_{n,l}^c & \overset{\text{def}}{=} (\nu a) ( y![n, l, a, \tilde{z}] \mid \text{test } a \text{ then } [ P ]^c \text{ else } 0 )
\end{align*}
\]
A “Bakery” Algorithm (I)

The following encoding is not strongly compositional! In fact, it is centralized ...

\[
\begin{align*}
\llbracket P \rrbracket & \overset{\text{def}}{=} (\nu c) \left( c!\llbracket 42 \rrbracket \mid \llbracket P \rrbracket^c \right) \\
\llbracket P_1 \mid P_2 \rrbracket^c & \overset{\text{def}}{=} \llbracket P_1 \rrbracket^c \mid \llbracket P_2 \rrbracket^c \\
\llbracket (\nu x) P \rrbracket^c & \overset{\text{def}}{=} (\nu x) \llbracket P \rrbracket^c \\
\llbracket \sum_{i \in I} \pi_i . P_i \rrbracket^c & \overset{\text{def}}{=} c?[n] . \left( c!\llbracket n+1 \rrbracket \mid (\nu l) \left( l!\llbracket t \rrbracket \mid \prod_{i \in I} \llbracket \pi_i . P_i \rrbracket^c_{n,l} \right) \right) \\
\llbracket y!\tilde{z} . P \rrbracket^c_{n,l} & \overset{\text{def}}{=} (\nu a) \left( y![n, l, a, \tilde{z}] \mid \text{test } a \text{ then } \llbracket P \rrbracket^c \text{ else } 0 \right)
\end{align*}
\]
A “Bakery” Algorithm (I)

The following encoding is **not strongly compositional**! In fact, it is **centralized** ...

$$
\begin{align*}
\left[ P \right] & \quad \overset{\text{def}}{=} \quad (\nu c) \left( c!\left[ 42 \right] \mid \left[ P \right]^c \right) \\
\left[ P_1 \mid P_2 \right]^c & \quad \overset{\text{def}}{=} \quad \left[ P_1 \right]^c \mid \left[ P_2 \right]^c \\
\left[ (\nu x) P \right]^c & \quad \overset{\text{def}}{=} \quad (\nu x) \left[ P \right]^c \\
\left[ \sum_{i \in I} \pi_i.P_i \right]^c & \quad \overset{\text{def}}{=} \quad c?[n]. \left( c![n+1] \mid (\nu l) \left( l![t] \mid \prod_{i \in I} \left[ \pi_i.P_i \right]^c_{n,l} \right) \right) \\
\left[ y!\tilde{z}.P \right]_{n,l}^c & \quad \overset{\text{def}}{=} \quad (\nu a) \left( y![n, l, a, \tilde{z}] \mid \text{test } a \text{ then } \left[ P \right]^c \text{ else } 0 \right)
\end{align*}
$$
A “Bakery” Algorithm (I)

The following encoding is **not strongly compositional**! In fact, it is **centralized** ...
A “Bakery” Algorithm (I)

The following encoding is not strongly compositional! In fact, it is centralized ...

\[
\begin{align*}
\llbracket P \rrbracket & \quad \text{def} \quad (\forall c)(c!\llbracket 42 \rrbracket \mid \llbracket P \rrbracket^c) \\
\llbracket P_1 \mid P_2 \rrbracket^c & \quad \text{def} \quad \llbracket P_1 \rrbracket^c \mid \llbracket P_2 \rrbracket^c \\
\llbracket (\forall x)P \rrbracket^c & \quad \text{def} \quad (\forall x)\llbracket P \rrbracket^c \\
\llbracket \sum_{i \in I} \pi_i.P_i \rrbracket^c & \quad \text{def} \quad c?[n].(c![n+1] \mid (\forall l)(l![t] \mid \prod_{i \in I} \llbracket \pi_i.P_i \rrbracket_{n,l}^c)) \\
\llbracket y!\tilde{z}.P \rrbracket^c_{n,l} & \quad \text{def} \quad (\forall a)(y![n,l,a,\tilde{z}] \mid \text{test } a \text{ then } \llbracket P \rrbracket^c \text{ else 0})
\end{align*}
\]

The top-level adds another layer to the encoding, so it cannot be composed with other instances.
A “Bakery” Algorithm (I)

The following encoding is **not strongly compositional**!
In fact, it is **centralized** ...

![Encoding](image)

The **top-level** adds another layer to the encoding, so it cannot be composed with other instances.
A “Bakery” Algorithm (I)

The following encoding is not strongly compositional! In fact, it is centralized ...

\[
\begin{align*}
\text{def} & \quad (\nu c) \ (c!|42] \mid [P]^c) \\
\text{def} & \quad [P_1 \mid P_2]^c \equiv [P_1]^c \mid [P_2]^c \\
\text{def} & \quad (\nu x) [P]^c \equiv (\nu x) [P]^c \\
\text{def} & \quad \sum_{i \in I} \pi_i.P_i]^c \equiv c?[n].(c![n+1] \mid (\nu l) (l![t] \mid \prod_{i \in I} [\pi_i.P_i]^c_{n,l})) \\
\text{def} & \quad (\nu a) (y![n,l,a,\tilde{z}] \mid \text{test} \ a \ \text{then} \ [P]^c \ \text{else} \ 0)
\end{align*}
\]

The top-level adds another layer to the encoding, so it cannot be composed with other instances.
A “Bakery” Algorithm (I)

The following encoding is **not strongly compositional**! In fact, it is **centralized** ...

\[
\begin{align*}
\llbracket P \rrbracket & \overset{\text{def}}{=} (\nu c) \left( c! [42] \mid \llbracket P \rrbracket^c \right) \\
\llbracket P_1 \mid P_2 \rrbracket^c & \overset{\text{def}}{=} \llbracket P_1 \rrbracket^c \mid \llbracket P_2 \rrbracket^c \\
\llbracket (\nu x) P \rrbracket^c & \overset{\text{def}}{=} (\nu x) \llbracket P \rrbracket^c \\
\llbracket \sum_{i \in I} \pi_i.P_i \rrbracket^c & \overset{\text{def}}{=} c?[n]. \left( c! [n+1] \mid (\nu l) \left( l! [t] \mid \prod_{i \in I} \llbracket \pi_i.P_i \rrbracket^c_{n, l} \right) \right) \\
\llbracket y! [\tilde{z}].P \rrbracket^c_{n, l} & \overset{\text{def}}{=} (\nu a) \left( y! [n, l, a, \tilde{z}] \mid \text{test } a \text{ then } \llbracket P \rrbracket^c \text{ else } 0 \right)
\end{align*}
\]

The **top-level** adds **another layer** to the encoding, so it cannot be composed with other instances.
A “Bakery” Algorithm (I)

The following encoding is not strongly compositional! In fact, it is centralized ...

\[
\begin{align*}
[ P ] & \overset{\text{def}}{=} (\nu c)\ (c!\{42\} \mid [ P ]^c) \\
[ P_1 \mid P_2 ]^c & \overset{\text{def}}{=} [ P_1 ]^c \mid [ P_2 ]^c \\
[ (\nu x) P ]^c & \overset{\text{def}}{=} (\nu x) [ P ]^c \\
\sum_{i \in I} \pi_i . P_i ]^c & \overset{\text{def}}{=} c?\{n\}.(c![n+1] \mid (\nu l)\ (l![t] \mid \prod_{i \in I} [\pi_i . P_i ]^c_{n,l})) \\
[y!\{\bar{z}\} . P]_{n,l}^c & \overset{\text{def}}{=} (\nu a)\ (y![n, l, a, \bar{z}] \mid \text{test } a \text{ then } [ P ]^c \text{ else } 0)
\end{align*}
\]

The top-level adds another layer to the encoding, so it cannot be composed with other instances.
A “Bakery” Algorithm (I)

The following encoding is **not strongly compositional**!

In fact, it is **centralized** ...

\[
\begin{align*}
\llbracket P \rrbracket & \overset{\text{def}}{=} (\nu c) \ (c!^{[42]} \ | \ \llbracket P \rrbracket^c) \\
\llbracket P_1 \mid P_2 \rrbracket^c & \overset{\text{def}}{=} \llbracket P_1 \rrbracket^c \mid \llbracket P_2 \rrbracket^c \\
\llbracket (\nu x) \ P \rrbracket^c & \overset{\text{def}}{=} (\nu x) \ \llbracket P \rrbracket^c \\
\llbracket \sum_{i \in I} \pi_i \cdot P_i \rrbracket^c & \overset{\text{def}}{=} c?^{[n]} \cdot (c!^{[n+1]} \ | \ (\nu l) \ (l![t] \mid \prod_{i \in I} \llbracket \pi_i \cdot P_i \rrbracket_{n,l}^c)) \\
\llbracket y![\tilde{z}] \cdot P \rrbracket_{n,l}^c & \overset{\text{def}}{=} (\nu a) \ (y![n, l, a, \tilde{z}] \ | \ \text{test } a \ \text{then } \llbracket P \rrbracket^c \ \text{else } 0)
\end{align*}
\]

The **top-level** adds another layer to the encoding, so it cannot be composed with other instances.
A “Bakery” Algorithm (I)

The following encoding is **not strongly compositional**!

In fact, it is **centralized** ...

\[
\begin{align*}
\llbracket P \rrbracket & \overset{\text{def}}{=} (\nu c) ( c! [42] \mid \llbracket P \rrbracket^c ) \\
\llbracket P_1 \mid P_2 \rrbracket^c & \overset{\text{def}}{=} \llbracket P_1 \rrbracket^c \mid \llbracket P_2 \rrbracket^c \\
\llbracket (\nu x) P \rrbracket^c & \overset{\text{def}}{=} (\nu x) \llbracket P \rrbracket^c \\
\llbracket \sum_{i \in I} \pi_i.P_i \rrbracket^c & \overset{\text{def}}{=} c?[n].( c![n+1] \mid (\nu l) ( l![t] \mid \prod_{i \in I} \llbracket \pi_i.P_i \rrbracket^c_{n,l}) ) \\
\llbracket y! [\tilde{z}] . P \rrbracket^c_{n,l} & \overset{\text{def}}{=} (\nu a) ( y![n,l,a,\tilde{z}] \mid \text{test } a \text{ then } \llbracket P \rrbracket^c \text{ else } 0 )
\end{align*}
\]

The **top-level** adds another layer to the encoding, so it cannot be composed with other instances.
A “Bakery” Algorithm (II)

\[
[y?][\tilde{x}].P]_{n,l} \overset{\text{def}}{=} (\forall b) ( b![] | b?[]).
\]

\[
y?[m, r, a, \tilde{x}].
\]

if \( n=m \) then \(( y?[m, r, a, \tilde{x}] | b![] \) \) else

if \( n<m \)

then test \( l \)

then test \( r \)

then \( l!*[f] | r!*[f] | a!*t | [P]^{c} \)

else \( l!*t | r!*f | a!*f | b!*[] \)

else \( l!*f | y*[m, r, a, \tilde{x}] \)

else test \( r \)

then test \( l \)

then \( l!*f | r!*f | a!*t | [P]^{c} \)

else \( l!*f | r!*t | y*[m, r, a, \tilde{x}] \)

else \( r!*f | a!*f | b!*[] \)
A “Bakery” Algorithm (II)

\[ y?[^x^\ldots n].P \]_n \text{ } \overset{\text{def}}{=} (\nu b) ( b![] | bаш[]. )

\begin{align*}
\text{if } & n=m \text{ then } ( y[m, r, a, \tilde{x}] | b![] ) \text{ else } \\
\text{if } & n<m \\
\text{then test } & l \\
\text{then test } & r \\
\text{then } & l!f | r!f | a[t | P]
\end{align*}

\begin{align*}
\text{else } & l!t | r!f | a[f] | b[] \\
\text{else } & l!f | y[m, r, a, \tilde{x}] \\
\text{else test } & r \\
\text{then test } & l \\
\text{then } & l!f | r[f] | a[t | P]
\end{align*}

\begin{align*}
\text{else } & l[f] | r[t | y[m, r, a, \tilde{x}] \\
\text{else } & r[f] | a[f] | b[] )
\end{align*}
A "Bakery" Algorithm (II)

\[
\begin{align*}
[y?\langle x \rangle . P]_{n!} & \overset{\text{def}}{=} (\nu b) (b![] \mid b?*[]).
\end{align*}
\]

\[
y?[m, r, a, \tilde{x}].
\]

if \( n=m \) then \((y![m, r, a, \tilde{x}] \mid b![])\) else

if \( n<m \)

then test \( l \)

then test \( r \)

then \( l![f] \mid r![f] \mid a![t] \mid \langle P \rangle^c \)

else \( l![t] \mid r![f] \mid a![f] \mid b![] \)

else \( l![f] \mid y![m, r, a, \tilde{x}] \)

else test \( r \)

then test \( l \)

then \( l![f] \mid r![f] \mid a![t] \mid \langle P \rangle^c \)

else \( l![f] \mid r![t] \mid y![m, r, a, \tilde{x}] \)

else \( r![f] \mid a![f] \mid b![] \)
A “Bakery” Algorithm (II)

\[
[ y?[^\alpha].P ]_n^c \overset{\text{def}}{=} (\nu b) \ (b![] | b?[]).
\]

\[
y?[^m, r, a, \bar{x}].
\]

\[
\text{if } \ n=m \text{ then } (y![^m, r, a, \bar{x}] | b![]) \text{ else } \n\]

\[
\text{if } \ n<m \text{ then test } l
\]

\[
\text{then test } r
\]

\[
\text{then } l![^f] | r[^f] | a[^t] | [P]^c
\]

\[
\text{else } l[^t] | r[^f] | a[^f] | b![]
\]

\[
\text{else } l[^f] | y![^m, r, a, \bar{x}]
\]

\[
\text{else test } r
\]

\[
\text{then test } l
\]

\[
\text{then } l[^f] | r[^f] | a[^t] | [P]^c
\]

\[
\text{else } l[^f] | r[^t] | y![^m, r, a, \bar{x}]
\]

\[
\text{else } r[^f] | a[^f] | b![])
\]
A "Bakery" Algorithm (II)

\[
\left[ y? [\tilde{x}].P \right]_{\text{n, l}} \overset{\text{def}}{=} (\nu b) \left( b![] \mid b?[] \right).
\]

\[
y? [m, r, a, \tilde{x}].
\]

if \( n=m \) then \( y[m, r, a, \tilde{x}] \mid b[] \) else

if \( n<m \)

then test \( l \)

then test \( r \)

then \( l[f] \mid r[f] \mid a[t] \mid \left[ P \right]^c \)

else \( l[t] \mid r[f] \mid a[f] \mid b[] \)

else \( l[f] \mid y[m, r, a, \tilde{x}] \)

else test \( r \)

then test \( l \)

then \( l[f] \mid r[f] \mid a[t] \mid \left[ P \right]^c \)

else \( l[f] \mid r[t] \mid y[m, r, a, \tilde{x}] \)

else \( r[f] \mid a[f] \mid b[] \)
A "Bakery" Algorithm (II)

\[ [y?[^{\tilde{x}}].P]_{n!}^c \overset{\text{def}}{=} (\nu b) (b![\cdot] \mid b!*[\cdot]). \]

\[ y?[m, r, a, \tilde{x}]. \]

\[ \text{if} \quad n=m \text{ then } (y![m, r, a, \tilde{x}] \mid b![]) \text{ else} \]

\[ \text{if} \quad n<m \]

\[ \text{then test } l \]

\[ \text{then test } r \]

\[ \text{then } \text{then } l![f] \mid r![f] \mid a![t] \mid [P]^c \]

\[ \text{else } \text{else } l![t] \mid r![f] \mid a![f] \mid b![] \]

\[ \text{else } \text{else } l![f] \mid y![m, r, a, \tilde{x}] \]

\[ \text{else test } r \]

\[ \text{then test } l \]

\[ \text{then } \text{then } l![f] \mid r![f] \mid a![t] \mid [P]^c \]

\[ \text{else } \text{else } l![f] \mid r![t] \mid y![m, r, a, \tilde{x}] \]

\[ \text{else } r![f] \mid a![f] \mid b![\cdot] \]
A “Bakery” Algorithm (II)

\[
[y?\bar{x}]P_{\text{Bakery}}^{c} \overset{\text{def}}{=} (\nu b) (b![] \mid b?[]).
\]

\[
y?m, r, a, \bar{x}.
\]

if \(n = m\) then \((y!m, r, a, \bar{x}) \mid b![]\) else

if \(n < m\)

then test \(l\)

then test \(r\)

then \(l!f \mid r!f \mid a![t] \mid [P]^c\)

else \(l!t \mid r!f \mid a![f] \mid b![]\)

else \(l!f \mid y!m, r, a, \bar{x}\)

else test \(r\)

then test \(l\)

then \(l!f \mid r!f \mid a![t] \mid [P]^c\)

else \(l!f \mid r![t] \mid y!m, r, a, \bar{x}\)

else \(r!f \mid a![f] \mid b![]\)
A “Bakery” Algorithm (II)

\[
\left[ y?\bar{x}.P \right]_{\text{enc}} \overset{\text{def}}{=} (\nu b) \left( b![] \mid b?[] \right).
\]

**equality check**

**prevents from incest**

(but produces divergence)

**total order property**

**prevents from deadlock**

\[
y?[m, r, a, \bar{x}] .
\]

if \( n=m \) then ( \( y?[m, r, a, \bar{x}] \mid b![] \) ) else

if \( n<m \)

then test \( l \)

then test \( r \)

then \( l![f] \mid r![f] \mid a![t] \mid \left[ P \right]_{\text{enc}} \)

else \( l![t] \mid r![f] \mid a![f] \mid b![] \)

else \( l![f] \mid y?[m, r, a, \bar{x}] \)

else test \( r \)

then test \( l \)

then \( l![f] \mid r![f] \mid a![t] \mid \left[ P \right]_{\text{enc}} \)

else \( l![f] \mid r![t] \mid y?[m, r, a, \bar{x}] \)

else \( r![f] \mid a![f] \mid b![] \)
Overview

Part 0a: Encodings vs Full Abstraction
Comparison of Languages/Calculi
Correctness of Encodings

Part 0b: Asynchronous Pi Calculus

Part 1: Input-Guarded Choice
Encoding (Distributed Implementation)
Decoding (Correctness Proof)

Part 2: Output-Guarded Choice
Encoding Separate Choice
Encoding Mixed Choice

Conclusions
A bit of cheating: encodings depend on n-ary choice.

The role of name-passing.

Asynchronous Pi is “sufficiently” expressive ... ... for programming. (Who needs mixed choice?)

The role of coupled simulation.

There is no theory of encodings yet.

The relation to distributed implementations —and to Distributed Computing in general—is not yet sufficiently understood.