Process Algebra

Luís Soares Barbosa

Interaction & Concurrency Course Unit (Lcc)

Universidade do Minho

Motivation: composition and interaction

Recall from a previous exercise

From $T_1 = \langle S_1, N, \longrightarrow_1 \rangle$ and $T_2 = \langle S_2, N, \longrightarrow_2 \rangle$, define

- Asynchronous composition: $T_1 \mid\mid\mid T_2$ as $(S_1 \times S_2, N, \longrightarrow)$, where

$$(s_1, s_2) \xrightarrow{a} (s_1', s_2) \Leftarrow s_1 \xrightarrow{a} s_1'$$
$$(s_1, s_2) \xrightarrow{a} (s_1, s_2') \Leftarrow s_2 \xrightarrow{a} s_2'$$

- Synchronous composition: $T_1 \mid\mid\mid_{sy} T_2$ as $(S_1 \times S_2, N \times N, \longrightarrow)$, where

$$(s_1, s_2) \xrightarrow{(a,b)} (s_1', s_2') \Leftarrow s_1 \xrightarrow{a} s_1' \wedge s_2 \xrightarrow{b} s_2'$$

Motivation: composition and interaction

A process algebra, i.e. an algebra of reactive systems,

- ... is driven by a discipline of interaction
- and provides a specification notation for reactive systems

Actions & processes

Action

- elementary unit of behaviour that can execute itself atomically in time (no duration), after which it terminates successfully
- is a latency for interaction

$$\alpha \quad ::= \quad \tau \mid a \mid (\alpha \mid \alpha)$$

- $a \mid b \mid \cdots \mid z$ represents a collection of actions that occur at the same time instant
- τ is the empty action, which contains no actions and as such cannot be observed
- $\langle N, |, \tau \rangle$ forms a monoid

Actions & processes

Process

is a description of how the interaction capacities of a system evolve, *i.e.*, its behaviour
for example,

$$E \mathrel{\widehat{=}} a.b + a.E$$

- analogy: regular expressions vs finite automata

The framework

Process

... abstract representation of a system's behaviour

Algebra

... a mathematical structure satisfying a particular set of axioms

Process Algebra

... a framework for the specification and manipulation of process terms as induced by a collection of operator symbols, encompassing an operational and an axiomatic theory

The framework

Transition systems operational representation of system's behaviour through labelled graphs

Behavioural equivalences to distinguished states in transition systems

Process terms algebraic representation of transition systems (for the purpose of mathematical reasoning)

Structural operational semantics inductive proof rules to provide each process term with its intended transition system

Equational theory Axiomatic theory of processes, expressed in an equational logic on process terms, that is sound and complete wrt bisimilarity.

Instantiating the framework

CCS: a prototypical process algebra

- *Calculus of Communicating Systems* [Milner, 1980]
- Actions:

$$Act \quad ::= \quad a \mid \overline{a} \mid \tau$$

  for $a \in N$, N denoting a set of names
- Processes:
  - No sequential composition: but action prefix $a.$
  - No distinction between termination and deadlock (why?)
  - Communication by binary handshake (of complementary actions)

Examples

Buffers

1-position buffer: $A(in, out) \widehat{=}\ in.\overline{out}.\mathbf{0}$

... non terminating: $B(in, out) \widehat{=}\ in.\overline{out}.B$

... with two output ports: $C(in, o_1, o_2) \widehat{=}\ in.(\overline{o_1}.C + \overline{o_2}.C)$

... non deterministic: $D(in, o_1, o_2) \widehat{=}\ in.\overline{o_1}.D + in.\overline{o_2}.D$

... with parameters: $B(in, out) \widehat{=}\ in(x).\overline{out}\langle x \rangle.B$

Examples

n-position buffers

1-position buffer:

$$S \mathrel{\widehat{=}} (B\langle in, m\rangle \mid B\langle m, out\rangle)\backslash_{\{m\}}$$

n-position buffer:

$$Bn \mathrel{\widehat{=}} (B\langle in, m_1\rangle \mid B\langle m_1, m_2\rangle \mid \cdots \mid B\langle m_{n-1}, out\rangle)\backslash_{\{m_i \mid i<n\}}$$

Examples

mutual exclusion

$$Sem \widehat{=} \; get.put.Sem$$

$$P_i \widehat{=} \; \overline{get}.c_i.\overline{put}.P_i$$

$$S \widehat{=} \; (Sem \mid (|_{i \in I} \; P_i)) \backslash_{\{get, put\}}$$

CCS Syntax

The set $\mathbb{P}$ of processes is the set of all terms generated by the following BNF:

$$E \ ::= \ A(x_1, ..., x_n) \ | \ a.E \ | \ \sum_{i \in I} E_i \ | \ E_0 \mid E_1 \ | \ E\backslash_K$$

for $a \in Act$ and $K \subseteq L$

Abbreviatures

$$E_0 + E_1 \stackrel{\text{abv}}{=} \sum_{i \in \{0,1\}} E_i$$

$$\mathbf{0} \stackrel{\text{abv}}{=} \sum_{i \in \emptyset} E_i$$

CCS Syntax

The set $\mathbb{P}$ of processes is the set of all terms generated by the following BNF:

$$E \ ::= \ A(x_1, ..., x_n) \ | \ a.E \ | \ \sum_{i \in I} E_i \ | \ E_0 \,|\, E_1 \ | \ E \backslash_K$$

for $a \in Act$ and $K \subseteq L$

Abbreviatures

$$E_0 + E_1 \stackrel{\text{abv}}{=} \sum_{i \in \{0,1\}} E_i$$

$$\mathbf{0} \stackrel{\text{abv}}{=} \sum_{i \in \emptyset} E_i$$

CCS Syntax

Process declaration

$$A(\vec{x}) \mathrel{\widehat{=}} E_A$$

with $fn(E_A) \subseteq \vec{x}$ (where $fn(P)$ is the set of free variables of P).

- used as, *e.g.*, $\boxed{A(a, b, c) \mathrel{\widehat{=}} a.b.\mathbf{0} + c.A\langle d, e, f\rangle}$

Process declaration: fixed point expression

$$\underline{fix}\ (X = E_X)$$

- syntactic substitution over $\mathbb{P}$, *cf.*
  - $\{c/b\}\, a.b.\mathbf{0}$

CCS Syntax

Process declaration

$$A(\vec{x}) \;\widehat{=}\; E_A$$

with $fn(E_A) \subseteq \vec{x}$ (where $fn(P)$ is the set of free variables of P).

- used as, *e.g.*, $\boxed{A(a, b, c) \;\widehat{=}\; a.b.\,\mathbf{0} + c.A\langle d, e, f\rangle}$

Process declaration: fixed point expression

$$\underline{fix}\;(X = E_X)$$

- syntactic substitution over $\mathbb{P}$, *cf.*
  - $\{c/b\}\, a.b.\mathbf{0}$

Semantics

Two-level semantics

- arquitectural, expresses a notion of similar assembly configurations and is expressed through a structural congruence relation;
- behavioural given by transition rules which express how system's components interact

Semantics

Structural congruence

$\equiv$ over $\mathbb{P}$ is given by the closure of the following conditions:

- for all $A(\vec{x}) \widehat{=} E_A$, $A(\vec{y}) \equiv \{\vec{y}/\vec{x}\} E_A$, (*i.e.*, folding/unfolding preserve $\equiv$)
- α-conversion (*i.e.*, replacement of bounded variables).
- both $|$ and $+$ originate, with $\mathbf{0}$, Abelian monoids
- forall $a \notin fn(P)$ $(P \mid Q)\backslash_{\{a\}} \equiv P \mid Q\backslash_{\{a\}}$
- $\mathbf{0}\backslash_{\{a\}} \equiv \mathbf{0}$

Semantics

$$\frac{}{a.p \xrightarrow{a} p}\,(prefix)$$

$$\frac{\{\vec{k}/\vec{x}\}\, p_A \xrightarrow{a} p'}{A(\vec{k}) \xrightarrow{a} p'}\,(ident) \quad (\text{if } A(\vec{x}) \widehat{=}\ p_A)$$

$$\frac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'}\,(sum - l) \qquad \frac{q \xrightarrow{a} q'}{p + q \xrightarrow{a} q'}\,(sum - r)$$

Semantics

$$\frac{p \xrightarrow{a} p'}{p \mid q \xrightarrow{a} p' \mid q}\,(par - l) \qquad \frac{q \xrightarrow{a} q'}{p \mid q \xrightarrow{a} p \mid q'}\,(par - r)$$

$$\frac{p \xrightarrow{a} p' \quad q \xrightarrow{\bar{a}} q'}{p \mid q \xrightarrow{\tau} p' \mid q'}\,(react)$$

$$\frac{p \xrightarrow{a} p'}{p\backslash_{\{k\}} \xrightarrow{a} p'\backslash_{\{k\}}}\,(res) \quad (\text{if } a \notin \{k, \overline{k}\})$$

Compatibility

Lemma

Structural congruence preserves transitions:

if $p \xrightarrow{a} p'$ and $p \equiv q$ there exists a process q' such that $q \xrightarrow{a} q'$ and $p' \equiv q'$.

Semantics

These rules define a LTS

$$\{\xrightarrow{a} \subseteq \mathbb{P} \times \mathbb{P} \mid a \in Act\}$$

Relation $\xrightarrow{a}$ is defined inductively over process structure entailing a semantic description which is

Structural *i.e.*, each process shape (defined by the most external combinator) has a type of transitions

Modular *i.e.*, a process trasition is defined from transitions in its sup-processes

Complete *i.e.*, all possible transitions are infered from these rules

$\boxed{\text{static vs dynamic combinators}}$

Graphical representations

Synchronization diagram

- represent interfaces of processes
- static combinators are an algebra of synchronization diagrams

Transition graph

- derivative, *n*-derivative, transition tree
- folds into a transition graph

Graphical representations

Synchronization diagram

- represent interfaces of processes
- static combinators are an algebra of synchronization diagrams

Transition graph

- derivative, *n*-derivative, transition tree
- folds into a transition graph

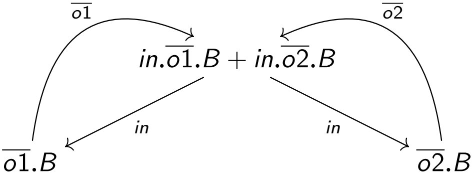
Transition tree

$$B \widehat{=}\ in.\overline{o1}.B + in.\overline{o2}.B$$

$$\begin{tikzcd}
 & & in.\overline{o1}.B + in.\overline{o2}.B \arrow[dl, "in"'] \arrow[dr, "in"] & \\
 & \overline{o1}.B \arrow[d, "\overline{o1}"'] & & \overline{o2}.B \arrow[d, "\overline{o2}"'] \\
 & in.\overline{o1}.B + in.\overline{o2}.B \arrow[dl, "in"] \arrow[d, "in"'] & & in.\overline{o1}.B + in.\overline{o2}.B \arrow[dl, "in"] \arrow[d, "in"'] \\
\overline{o1}.B \arrow[d, "\overline{o1}"'] & \overline{o2}.B \arrow[d, "\overline{o2}"'] & \overline{o1}.B \arrow[d, "\overline{o1}"'] & \overline{o2}.B \arrow[d, "\overline{o2}"'] \\
\dots & \dots & \dots & \dots
\end{tikzcd}$$

Transition graph

$B \widehat{=}\ in.\overline{o1}.B + in.\overline{o2}.B$



compare with $B' \widehat{=}\ in.(\overline{o1}.B' + \overline{o2}.B')$

Transition graph

$B \widehat{=}\ in.\overline{o1}.B + in.\overline{o2}.B$

$$
\begin{tikzcd}
 & in.\overline{o1}.B + in.\overline{o2}.B \arrow[dl, "in"'] \arrow[dr, "in"] & \\
\overline{o1}.B \arrow[ur, bend left, "\overline{o1}"] & & \overline{o2}.B \arrow[ul, bend right, "\overline{o2}"']
\end{tikzcd}
$$

compare with $B' \widehat{=}\ in.(\overline{o1}.B' + \overline{o2}.B')$

$$
\begin{tikzcd}
in.(\overline{o1}.B' + \overline{o2}.B') \arrow[d, "in"'] \\
\overline{o1}.B' + \overline{o2}.B' \arrow[u, bend left=60, "\overline{o1}"] \arrow[u, bend right=60, "\overline{o2}"']
\end{tikzcd}
$$

Data parameters

Language $\mathbb{P}$ is extended to $\mathbb{P}_V$ over a data universe V, a set V_e of expressions over V and a evaluation $Val : V_e \rightarrow V$

Example

$$B \widehat{=} in(x).B'_x$$
$$B'_v \widehat{=} \overline{out}\langle v\rangle.B$$

- Two prefix forms: $a(x).E$ and $\overline{a}\langle e\rangle.E$ (actions as ports)
- Data parameters: $A_S(x_1, ..., x_n) \widehat{=} E_A$, with $S \in V$ and each $x_i \in L$
- Conditional combinator: $if\ b\ then\ P$, $if\ b\ then\ P_1\ else\ P_2$

Clearly

$$if\ b\ then\ P_1\ else\ P_2 \stackrel{\text{abv}}{=} (if\ b\ then\ P_1) + (if\ \neg b\ then\ P_2)$$

Data parameters

Language $\mathbb{P}$ is extended to $\mathbb{P}_V$ over a data universe V, a set V_e of expressions over V and a evaluation $Val : V_e \rightarrow V$

Example

$$B \widehat{=} \; in(x).B'_x$$
$$B'_v \widehat{=} \; \overline{out}\langle v\rangle.B$$

- Two prefix forms: $a(x).E$ and $\overline{a}\langle e\rangle.E$ (actions as ports)
- Data parameters: $A_S(x_1, ..., x_n) \widehat{=} \; E_A$, with $S \in V$ and each $x_i \in L$
- Conditional combinator: *if b then P*, *if b then P_1 else P_2*

Clearly

$$if \; b \; then \; P_1 \; else \; P_2 \; \stackrel{\text{abv}}{=} \; (if \; b \; then \; P_1) + (if \; \neg b \; then \; P_2)$$

Data parameters

Additional semantic rules

$$\frac{}{a(x).E \xrightarrow{a(v)} \{v/x\}E}\ (prefix_i) \qquad \text{for } v \in V$$

$$\frac{}{\overline{a}\langle e\rangle.E \xrightarrow{\overline{a}\langle v\rangle} E}\ (prefix_o) \qquad \text{for } Val(e) = v$$

$$\frac{E_1 \xrightarrow{a} E'}{\mathit{if}\ b\ \mathit{then}\ E_1\ \mathit{else}\ E_2 \xrightarrow{a} E'}\ (\mathit{if}_1) \qquad \text{for } Val(b) = \mathit{true}$$

$$\frac{E_2 \xrightarrow{a} E'}{\mathit{if}\ b\ \mathit{then}\ E_1\ \mathit{else}\ E_2 \xrightarrow{a} E'}\ (\mathit{if}_2) \qquad \text{for } Val(b) = \mathit{false}$$

Back to $\mathbb{P}$

Encoding in the basic language: $T(\ ) : \mathbb{P}_V \longrightarrow \mathbb{P}$

$$
\begin{aligned}
T(a(x).E) &= \sum_{v \in V} a_v . T(\{v/x\}E) \\
T(\overline{a}\langle e \rangle .E) &= \overline{a}_e . T(E) \\
T(\sum_{i \in I} E_i) &= \sum_{i \in I} T(E_i) \\
T(E \mid F) &= T(E) \mid T(F) \\
T(E \backslash_K) &= T(E) \backslash_{\{a_v \mid a \in K, v \in V\}}
\end{aligned}
$$

and

$$
T(\mathit{if}\ b\ \mathit{then}\ E) = \begin{cases} T(E) & \text{if } Val(b) = \mathit{true} \\ \mathbf{0} & \text{if } Val(b) = \mathit{false} \end{cases}
$$

EX1: Canonical concurrent form

$$P \mathrel{\widehat{=}} (E_1 \mid E_2 \mid \ldots \mid E_n)\backslash_K$$

The chance machine

$$\begin{aligned}
IO &\mathrel{\widehat{=}} m.\overline{bank}.(lost.\overline{loss}.IO + rel(x).\overline{win}\langle x\rangle.IO) \\
B_n &\mathrel{\widehat{=}} bank.\overline{max}\langle n+1\rangle.left(x).B_x \\
Dc &\mathrel{\widehat{=}} max(z).(\overline{lost}.\overline{left}\langle z\rangle.Dc + \sum_{1\leq x\leq z} \overline{rel}\langle x\rangle.\overline{left}\langle z-x\rangle.Dc)
\end{aligned}$$

$$M_n \mathrel{\widehat{=}} (IO \mid B_n \mid Dc)\backslash_{\{bank,max,left,lost,rel\}}$$

EX2: Sequential patterns

1. List all states (configurations of variable assignments)
2. Define an order to capture systems's evolution
3. Specify an expression in $\mathbb{P}$ to define it

A 3-bit converter

$$
\begin{aligned}
A &\widehat{=} \; rq.B \\
B &\widehat{=} \; out0.C + out1.\overline{odd}.A \\
C &\widehat{=} \; out0.D + out1.\overline{even}.A \\
D &\widehat{=} \; out0.\overline{zero}.A + out1.\overline{even}.A
\end{aligned}
$$

Processes are 'prototypical' transition systems

... hence all definitions apply:

$E \sim F$

- Processes E, F are bisimilar if there exist a bisimulation S st $\{\langle E, F\rangle\} \in S$.
- A binary relation S in $\mathbb{P}$ is a (strict) bisimulation iff, whenever $(E, F) \in S$ and $a \in Act$,

$$\begin{array}{ll} \text{i)} & E \xrightarrow{a} E' \;\Rightarrow\; F \xrightarrow{a} F' \wedge (E', F') \in S \\ \text{ii)} & F \xrightarrow{a} F' \;\Rightarrow\; E \xrightarrow{a} E' \wedge (E', F') \in S \end{array}$$

I.e.,

$$\sim \;=\; \bigcup \{S \subseteq \mathbb{P} \times \mathbb{P} \mid S \quad \text{is a (strict) bisimulation}\}$$

Processes are 'prototipycal' transition systems

Example: $S \sim M$

$$
\begin{aligned}
T &\widehat{=}\ i.\overline{k}.T \\
R &\widehat{=}\ k.j.R \\
S &\widehat{=}\ (T \mid R)\backslash_{\{k\}}
\end{aligned}
$$

$$
\begin{aligned}
M &\widehat{=}\ i.\tau.N \\
N &\widehat{=}\ j.i.\tau.N + i.j.\tau.N
\end{aligned}
$$

through bisimulation

$$
\begin{aligned}
R = &\{\langle S, M)\rangle, \langle(\overline{k}.T \mid R)\backslash_{\{k\}}, \tau.N\rangle, \langle(T \mid j.R)\backslash_{\{k\}}, N\rangle, \\
&\langle(\overline{k}.T \mid j.R)\backslash_{\{k\}}, j.\tau.N\rangle\}
\end{aligned}
$$

Example: Semaphores

A semaphore

$$Sem \mathrel{\widehat{=}} get.put.Sem$$

n-semaphores

$$\begin{aligned} Sem_n &\mathrel{\widehat{=}} Sem_{n,0} \\ Sem_{n,0} &\mathrel{\widehat{=}} get.Sem_{n,1} \\ Sem_{n,i} &\mathrel{\widehat{=}} get.Sem_{n,i+1} + put.Sem_{n,i-1} \\ & \quad (\text{for } 0 < i < n) \\ Sem_{n,n} &\mathrel{\widehat{=}} put.Sem_{n,n-1} \end{aligned}$$

Sem_n can also be implemented by the parallel composition of n Sem processes:

$$Sem^n \mathrel{\widehat{=}} Sem \mid Sem \mid ... \mid Sem$$

Example: Semaphores

A semaphore

$$Sem \mathrel{\widehat{=}} get.put.Sem$$

n-semaphores

$$\begin{aligned}
Sem_n &\mathrel{\widehat{=}} Sem_{n,0}\\
Sem_{n,0} &\mathrel{\widehat{=}} get.Sem_{n,1}\\
Sem_{n,i} &\mathrel{\widehat{=}} get.Sem_{n,i+1} + put.Sem_{n,i-1}\\
&\quad (\text{for } 0 < i < n)\\
Sem_{n,n} &\mathrel{\widehat{=}} put.Sem_{n,n-1}
\end{aligned}$$

Sem_n can also be implemented by the parallel composition of n Sem processes:

$$Sem^n \mathrel{\widehat{=}} Sem \mid Sem \mid \ldots \mid Sem$$

Example: Semaphores

Is $Sem_n \sim Sem^n$?

For $n = 2$:

$$\{\langle Sem_{2,0}, Sem \mid Sem\rangle, \langle Sem_{2,1}, Sem \mid put.Sem\rangle,$$
$$\langle Sem_{2,1}, put.Sem \mid Sem\rangle \langle Sem_{2,2}, put.Sem \mid put.Sem\rangle\}$$

is a bisimulation.

- but can we get rid of structurally congruent pairs?

Example: Semaphores

Is $Sem_n \sim Sem^n$?

For $n = 2$:

$$\{\langle Sem_{2,0}, Sem \mid Sem\rangle, \langle Sem_{2,1}, Sem \mid put.Sem\rangle, \\ \langle Sem_{2,1}, put.Sem \mid Sem\rangle \langle Sem_{2,2}, put.Sem \mid put.Sem\rangle\}$$

is a bisimulation.

- but can we get rid of structurally congruent pairs?

Bisimulation up to $\equiv$

Definition

A binary relation S in $\mathbb{P}$ is a (strict) bisimulation up to $\equiv$ iff, whenever $(E, F) \in S$ and $a \in Act$,

$$\text{i)}\quad E \xrightarrow{a} E' \;\Rightarrow\; F \xrightarrow{a} F' \wedge (E', F') \in \equiv \cdot S \cdot \equiv$$

$$\text{ii)}\quad F \xrightarrow{a} F' \;\Rightarrow\; E \xrightarrow{a} E' \wedge (E', F') \in \equiv \cdot S \cdot \equiv$$

Lemma

If S is a (strict) bisimulation up to $\equiv$, then $S \subseteq \sim$

- To prove $Sem_n \sim Sem^n$ a bisimulation will contain 2^n pairs, while a bisimulation up to $\equiv$ only requires $n+1$ pairs.

Bisimulation up to $\equiv$

Definition

A binary relation S in $\mathbb{P}$ is a (strict) bisimulation up to $\equiv$ iff, whenever $(E,F) \in S$ and $a \in Act$,

$$\text{i)}\quad E \xrightarrow{a} E' \;\Rightarrow\; F \xrightarrow{a} F' \wedge (E',F') \in \equiv \cdot S \cdot \equiv$$

$$\text{ii)}\quad F \xrightarrow{a} F' \;\Rightarrow\; E \xrightarrow{a} E' \wedge (E',F') \in \equiv \cdot S \cdot \equiv$$

Lemma

If S is a (strict) bisimulation up to $\equiv$, then $S \subseteq \sim$

- To prove $Sem_n \sim Sem^n$ a bisimulation will contain 2^n pairs, while a bisimulation up to $\equiv$ only requires $n+1$ pairs.

Bisimulation up to $\equiv$

Definition

A binary relation S in $\mathbb{P}$ is a (strict) bisimulation up to $\equiv$ iff, whenever $(E, F) \in S$ and $a \in Act$,

$$\text{i)}\quad E \xrightarrow{a} E' \;\Rightarrow\; F \xrightarrow{a} F' \wedge (E', F') \in \equiv \cdot S \cdot \equiv$$

$$\text{ii)}\quad F \xrightarrow{a} F' \;\Rightarrow\; E \xrightarrow{a} E' \wedge (E', F') \in \equiv \cdot S \cdot \equiv$$

Lemma

If S is a (strict) bisimulation up to $\equiv$, then $S \subseteq \sim$

- To prove $Sem_n \sim Sem^n$ a bisimulation will contain 2^n pairs, while a bisimulation up to $\equiv$ only requires $n + 1$ pairs.

A $\sim$-calculus

Lemma

$$E \equiv F \;\Rightarrow\; E \sim F$$

- proof idea: show that $\{(E + E, E) \mid E \in \mathbb{P}\} \cup Id_{\mathbb{P}}$ is a bisimulation

Lemma

$$(E\backslash_K)\backslash_{K'} \sim E\backslash_{(K \cup K')}$$

$$E\backslash_K \sim E \qquad \text{if } \mathbb{L}(E) \cap (K \cup \overline{K}) = \emptyset$$

$$(E \mid F)\backslash_K \sim E\backslash_K \mid F\backslash_K \qquad \text{if } \mathbb{L}(E) \cap \overline{\mathbb{L}(F)} \cap (K \cup \overline{K}) = \emptyset$$

- proof idea: discuss whether S is a bisimulation:

$$S \;=\; \{(E\backslash_K, E) \mid E \in \mathbb{P} \wedge \mathbb{L}(E) \cap (K \cup \overline{K}) = \emptyset\}$$

A $\sim$-calculus

Lemma

$$E \equiv F \;\Rightarrow\; E \sim F$$

- proof idea: show that $\{(E + E, E) \mid E \in \mathbb{P}\} \cup Id_{\mathbb{P}}$ is a bisimulation

Lemma

$$(E\backslash_K)\backslash_{K'} \sim E\backslash_{(K \cup K')}$$

$$E\backslash_K \sim E \qquad \text{if } \mathbb{L}(E) \cap (K \cup \overline{K}) = \emptyset$$

$$(E \mid F)\backslash_K \sim E\backslash_K \mid F\backslash_K \qquad \text{if } \mathbb{L}(E) \cap \overline{\mathbb{L}(F)} \cap (K \cup \overline{K}) = \emptyset$$

- proof idea: discuss whether S is a bisimulation:

$$S \;=\; \{(E\backslash_K, E) \mid E \in \mathbb{P} \wedge \mathbb{L}(E) \cap (K \cup \overline{K}) = \emptyset\}$$

$\sim$ is a congruence

$\boxed{\text{congruence is the name of modularity in Mathematics}}$

- process combinators preserve $\sim$

Lemma

Assume $E \sim F$. Then,

$$a.E \sim a.F$$

$$E + P \sim F + P$$

$$E \mid P \sim F \mid P$$

$$E \backslash_K \sim F \backslash_K$$

- recursive definition preserves $\sim$

$\sim$ is a congruence

$\boxed{\text{congruence is the name of modularity in Mathematics}}$

- process combinators preserve $\sim$

Lemma

Assume $E \sim F$. Then,

$$a.E \sim a.F$$
$$E + P \sim F + P$$
$$E \mid P \sim F \mid P$$
$$E\backslash_K \sim F\backslash_K$$

- recursive definition preserves $\sim$

$\sim$ is a congruence

- First $\sim$ is extended to processes with variables:

$$E \sim F \;\equiv\; \forall_{\tilde{P}} \,.\;\; E[\tilde{P}/\tilde{X}] \sim F[\tilde{P}/\tilde{X}]$$

- Then prove:

Lemma

i) $\tilde{P} \widehat{=} \tilde{E} \;\Rightarrow\; \tilde{P} \sim \tilde{E}$
where $\tilde{E}$ is a family of process expressions and $\tilde{P}$ a family of process identifiers.

ii) Let $\tilde{E} \sim \tilde{F}$, where $\tilde{E}$ and $\tilde{F}$ are families of recursive process expressions over a family of process variables $\tilde{X}$, and define:

$$\tilde{A} \widehat{=} \tilde{E}[\tilde{A}/\tilde{X}] \quad \text{and} \quad \tilde{B} \widehat{=} \tilde{F}[\tilde{B}/\tilde{X}]$$

Then

$$\tilde{A} \sim \tilde{B}$$

The expansion theorem

$\boxed{\text{Every process is equivalent to the sum of its derivatives}}$

$$E \sim \sum\{a.E' \mid E \xrightarrow{a} E'\}$$

understood?

$$E \sim \sum\{a.E' \mid E \xrightarrow{a} E'\}$$

clear?

$$E \sim \sum\{a.E' \mid E \xrightarrow{a} E'\}$$

The expansion theorem

$\boxed{\text{Every process is equivalent to the sum of its derivatives}}$

$$E \sim \sum \{a.E' \mid E \xrightarrow{a} E'\}$$

understood?

$$E \sim \sum \{a.E' \mid E \xrightarrow{a} E'\}$$

clear?

$$E \sim \sum \{a.E' \mid E \xrightarrow{a} E'\}$$

The expansion theorem

$\boxed{\text{Every process is equivalent to the sum of its derivatives}}$

$$E \sim \sum \{a.E' \mid E \xrightarrow{a} E'\}$$

understood?

$$E \sim \sum \{a.E' \mid E \xrightarrow{a} E'\}$$

clear?

$$E \sim \sum \{a.E' \mid E \xrightarrow{a} E'\}$$

The expansion theorem

The usual definition (based on the concurrent canonical form):

$$E \sim \sum \{ f_i(a).(E_1[f_1] \mid \ldots \mid E_i'[f_i] \mid \ldots \mid E_n[f_n])\backslash_K \mid E_i \xrightarrow{a} E_i' \wedge f_i(a) \notin K \cup \overline{K} \} + \sum \{ \tau.(E_1[f_1] \mid \ldots \mid E_i'[f_i] \mid \ldots \mid E_j'[f_j] \mid \ldots \mid E_n[f_n])\backslash_K \mid E_i \xrightarrow{a} E_i' \wedge E_j \xrightarrow{b} E_j' \wedge f_i(a) = \overline{f_j(b)} \}$$

for $E \widehat{=} (E_1[f_1] \mid \ldots \mid E_n[f_n])\backslash_K$, with $n \geq 1$

The expansion theorem

Corollary (for $n = 1$ and $f_1 = id$)

$$(E + F)\backslash_K \sim E\backslash_K + F\backslash_K$$

$$(a.E)\backslash_K \sim \begin{cases} \mathbf{0} & \text{if } a \in (K \cup \overline{K}) \\ a.(E\backslash_K) & \text{otherwise} \end{cases}$$

Example

$S \sim M$

$$
\begin{aligned}
S &\sim (T \mid R)\backslash_{\{k\}} \\
&\sim i.(\overline{k}.T \mid R)\backslash_{\{k\}} \\
&\sim i.\tau.(T \mid j.R)\backslash_{\{k\}} \\
&\sim i.\tau.(i.\,(\overline{k}.T \mid j.R)\backslash_{\{k\}} + j.(T \mid R)\backslash_{\{k\}}) \\
&\sim i.\tau.(i.j.\,(\overline{k}.T \mid R)\backslash_{\{k\}} + j.i.(\overline{k}.T \mid R)\backslash_{\{k\}}) \\
&\sim i.\tau.(i.j.\tau.\,(T \mid j.R)\backslash_{\{k\}} + j.i.\tau.(T \mid j.R)\backslash_{\{k\}})
\end{aligned}
$$

Let $N' = (T \mid j.R)\backslash_{\{k\}}$.
This expands into $N' \sim i.j.\tau.\,(T \mid j.R)\backslash_{\{k\}} + j.i.\tau.(T \mid j.R)\backslash_{\{k\}}$,
Therefore $N' \sim N$ and $S \sim i.\tau.N \sim M$

- requires result on unique solutions for recursive process equations