

Process Algebra (1)

Luís Soares Barbosa



Universidade do Minho



UNITED NATIONS
UNIVERSITY

UNU-EGOV

Interaction & Concurrency Course Unit (Lcc)

Universidade do Minho, 23.II.2018

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**

$$\alpha ::= \tau \mid a \mid \alpha \mid \alpha$$

- $a \mid b \mid \dots \mid z$ represent 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,

$$E \hat{=} a.b + a.E$$

- **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:

$$Act ::= a \mid \bar{a} \mid \tau$$

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: $A(in, out) \hat{=} in.\overline{out}.0$

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

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

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

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

Examples

n -position buffers

1-position buffer:

$$S \hat{=} (B\langle in, m \rangle \mid B\langle m, out \rangle) \setminus \{m\}$$

n -position buffer:

$$B_n \hat{=} (B\langle in, m_1 \rangle \mid B\langle m_1, m_2 \rangle \mid \cdots \mid B\langle m_{n-1}, out \rangle) \setminus \{m_i \mid i < n\}$$

Examples

mutual exclusion

$$Sem \hat{=} get.put.Sem$$

$$P_i \hat{=} \overline{get}.c_i.\overline{put}.P_i$$

$$S \hat{=} (Sem \mid (\prod_{i \in I} P_i)) \setminus_{\{get, put\}}$$

CCS Syntax

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

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

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

Abbreviations

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

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

CCS Syntax

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

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

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

Abbreviations

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

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

CCS Syntax

Process declaration

$$A(\vec{x}) \hat{=} E_A$$

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

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

Process declaration: fixed point expression

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

- syntactic substitution over \mathbb{P} , cf.,
 - $\{c/b\} a.b.0$
 - (internal variables renaming) $\{x/y\} y.x.0 \setminus_{\{x\}} = x.x'.0 \setminus_{\{x'\}}$

CCS Syntax

Process declaration

$$A(\vec{x}) \hat{=} E_A$$

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

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

Process declaration: fixed point expression

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

- syntactic **substitution** over \mathbb{P} , cf.,
 - $\{c/b\} a.b.\mathbf{0}$
 - (internal variables renaming) $\{x/y\} y.x.\mathbf{0} \setminus_{\{x\}} = x.x'.\mathbf{0} \setminus_{\{x'\}}$

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

Semantics

$$\frac{}{a.p \longrightarrow p} \text{ (prefix)}$$

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

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

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

Semantics

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

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

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

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 transition is defined from transitions in its sup-processes

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

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

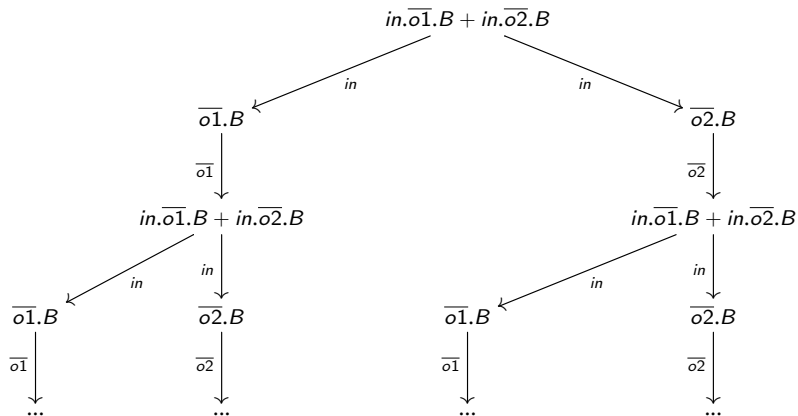
- represent interfaces of processes
- static combinators are an **algebra** of synchronization diagrams

Transition graph

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

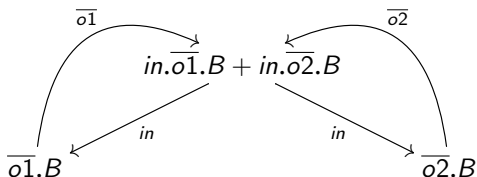
Transition tree

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

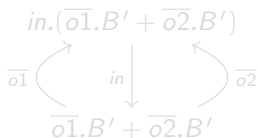


Transition graph

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

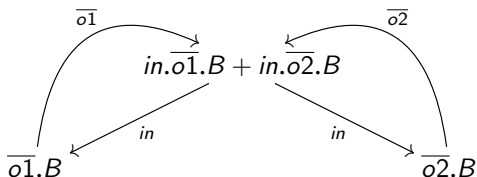


compare with $B' \hat{=} in.(\overline{o1}.B' + \overline{o2}.B')$

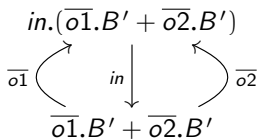


Transition graph

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



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



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 an evaluation $Val : V_e \rightarrow V$

Example

$$B \hat{=} in(x).B'_x$$

$$B'_v \hat{=} \overline{out}\langle v \rangle.B$$

- Two prefix forms: $a(x).E$ and $\bar{a}\langle e \rangle.E$ (actions as ports)
- Data parameters: $A_S(x_1, \dots, x_n) \hat{=} E_A$, 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

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

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

$$B \hat{=} in(x).B'_x$$

$$B'_v \hat{=} \overline{out}\langle v \rangle.B$$

- Two prefix forms: $a(x).E$ and $\bar{a}\langle e \rangle.E$ (**actions** as **ports**)
- Data parameters: $A_S(x_1, \dots, x_n) \hat{=} E_A$, 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

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

Data parameters

Additional semantic rules

$$\frac{}{a(x).E \xrightarrow{a(v)} \{v/x\}E} \text{ (prefix}_i\text{)} \quad \text{for } v \in V$$

$$\frac{}{\bar{a}\langle e \rangle.E \xrightarrow{\bar{a}\langle v \rangle} E} \text{ (prefix}_o\text{)} \quad \text{for } \text{Val}(e) = v$$

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

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

Back to \mathbb{P}

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

$$T(a(x).E) = \sum_{v \in V} a_v \cdot T(\{v/x\}E)$$

$$T(\bar{a}\langle e \rangle.E) = \bar{a}_e \cdot T(E)$$

$$T\left(\sum_{i \in I} E_i\right) = \sum_{i \in I} T(E_i)$$

$$T(E \mid F) = T(E) \mid T(F)$$

$$T(E \setminus K) = T(E) \setminus_{\{a_v \mid a \in K, v \in V\}}$$

and

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

EX1: Canonical concurrent form

$$P \hat{=} (E_1 \mid E_2 \mid \dots \mid E_n) \setminus_K$$

The chance machine

$$IO \hat{=} m.\overline{bank}.(lost.\overline{loss}.IO + rel(x).\overline{win}\langle x \rangle.IO)$$

$$B_n \hat{=} bank.\overline{max}\langle n+1 \rangle.left(x).B_x$$

$$Dc \hat{=} max(z).(\overline{lost}.\overline{left}\langle z \rangle.Dc + \sum_{1 \leq x \leq z} \overline{rel}\langle x \rangle.\overline{left}\langle z-x \rangle.Dc)$$

$$M_n \hat{=} (IO \mid B_n \mid Dc) \setminus_{\{bank, max, left, lost, rel\}}$$

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

$$A \hat{=} rq.B$$

$$B \hat{=} out0.C + out1.\overline{odd}.A$$

$$C \hat{=} out0.D + out1.\overline{even}.A$$

$$D \hat{=} out0.\overline{zero}.A + out1.\overline{even}.A$$

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 $a \in Act$,

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

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

i.e.,

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

Processes are 'prototypical' transition systems

Example: $S \sim M$

$$T \hat{=} i.\bar{k}.T$$

$$R \hat{=} k.j.R$$

$$S \hat{=} (T \mid R) \setminus \{k\}$$

$$M \hat{=} i.\tau.N$$

$$N \hat{=} j.i.\tau.N + i.j.\tau.N$$

through **bisimulation**

$$R = \{ \langle S, M \rangle, \langle (\bar{k}.T \mid R) \setminus \{k\}, \tau.N \rangle, \langle (T \mid j.R) \setminus \{k\}, N \rangle, \\ \langle (\bar{k}.T \mid j.R) \setminus \{k\}, j.\tau.N \rangle \}$$

Example: Semaphores

A semaphore

$$Sem \hat{=} get.put.Sem$$

n -semaphores

$$Sem_n \hat{=} Sem_{n,0}$$

$$Sem_{n,0} \hat{=} get.Sem_{n,1}$$

$$Sem_{n,i} \hat{=} get.Sem_{n,i+1} + put.Sem_{n,i-1}$$

(for $0 < i < n$)

$$Sem_{n,n} \hat{=} put.Sem_{n,n-1}$$

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

$$Sem^n \hat{=} Sem \mid Sem \mid \dots \mid Sem$$

Example: Semaphores

A semaphore

$$Sem \hat{=} get.put.Sem$$

n -semaphores

$$Sem_n \hat{=} Sem_{n,0}$$

$$Sem_{n,0} \hat{=} get.Sem_{n,1}$$

$$Sem_{n,i} \hat{=} get.Sem_{n,i+1} + put.Sem_{n,i-1}$$

(for $0 < i < n$)

$$Sem_{n,n} \hat{=} put.Sem_{n,n-1}$$

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

$$Sem^n \hat{=} Sem \mid Sem \mid \dots \mid Sem$$

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 $a \in Act$,

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

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

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 $a \in Act$,

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

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

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 $a \in Act$,

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

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

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

$$(E \setminus_K) \setminus_{K'} \sim E \setminus_{(K \cup K')}$$

$$E \setminus_K \sim E$$

$$\text{if } \mathbb{L}(E) \cap (K \cup \bar{K}) = \emptyset$$

$$(E \mid F) \setminus_K \sim E \setminus_K \mid F \setminus_K$$

$$\text{if } \mathbb{L}(E) \cap \overline{\mathbb{L}(F)} \cap (K \cup \bar{K}) = \emptyset$$

- **proof idea:** discuss whether S is a **bisimulation**:

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

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

$$(E \setminus_K) \setminus_{K'} \sim E \setminus_{(K \cup K')}$$

$$E \setminus_K \sim E \quad \text{if } \mathbb{L}(E) \cap (K \cup \bar{K}) = \emptyset$$

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

- **proof idea:** discuss whether S is a **bisimulation**:

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

\sim is a congruence

congruence is the name of **modularity** in Mathematics

- **process combinators** preserve \sim

Lemma

Assume $E \sim F$. Then,

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

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

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

$$E \setminus \kappa \sim F \setminus \kappa$$

- **recursive definition** preserves \sim

\sim is a congruence

congruence is the name of **modularity** in Mathematics

- **process combinators** preserve \sim

Lemma

Assume $E \sim F$. Then,

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

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

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

$$E \setminus \kappa \sim F \setminus \kappa$$

- **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

- $\tilde{P} \hat{=} \tilde{E} \Rightarrow \tilde{P} \sim \tilde{E}$
where \tilde{E} is a family of process expressions and \tilde{P} a family of process **identifiers**.
- 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:

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

Then

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

The expansion theorem

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

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

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**):

$$\begin{aligned}
 E \sim & \sum \{ f_i(a).(E_1[f_1] \mid \dots \mid E_i'[f_i] \mid \dots \mid E_n[f_n]) \setminus_K \mid \\
 & E_i \xrightarrow{a} E_i' \wedge f_i(a) \notin K \cup \bar{K} \} \\
 + & \\
 & \sum \{ \tau.(E_1[f_1] \mid \dots \mid E_i'[f_i] \mid \dots \mid E_j'[f_j] \mid \dots \mid E_n[f_n]) \setminus_K \mid \\
 & E_i \xrightarrow{a} E_i' \wedge E_j \xrightarrow{b} E_j' \wedge f_i(a) = \overline{f_j(b)} \}
 \end{aligned}$$

for $E \hat{=} (E_1[f_1] \mid \dots \mid E_n[f_n]) \setminus_K$, with $n \geq 1$

The expansion theorem

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

$$(E + F) \setminus_K \sim E \setminus_K + F \setminus_K$$
$$(a.E) \setminus_K \sim \begin{cases} \mathbf{0} & \text{if } a \in (K \cup \bar{K}) \\ a.(E \setminus_K) & \text{otherwise} \end{cases}$$

Example

$$S \sim M$$

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

Let $N' = (T \mid j.R) \setminus \{k\}$.

This expands into $N' \sim i.j.\tau.(T \mid j.R) \setminus \{k\} + j.i.\tau.(T \mid j.R) \setminus \{k\}$,

Therefore $N' \sim N$ and $S \sim i.\tau.N \sim M$

- requires result on **unique** solutions for recursive process equations