A Process Calculus I

Wolfgang Schreiner
Research Institute for Symbolic Computation
Johannes Kepler University, Linz, Austria

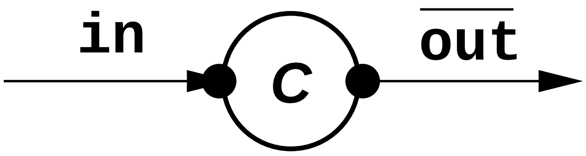
Wolfgang.Schreiner@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/schreine

A Process Calculus

- Description of process networks
  - Static communication topologies.
- History sketch
  - Robin Milner, 1980.
  - CCS: Calculus of Communicating Systems.
  - Various revisions and elaborations.
  - Extended to *mobile* processes (π-calculus).
- Algebraic approach
  - Concurrent system modeled by term.
  - Theory of term manipulations.
  - External behavior preserved.
- *Observational equivalence*
  - *External* communications follow same pattern.
  - *Internal* behavior may differ.

Modeling of communication and concurrency.

A Simple Example



- *Agent* C
  - Dynamic system is network of *agents*.
  - Each agent has own identity persisting over time.
  - Agent performs *actions* (external communications or internal actions).
  - *Behavior* of a system is its (observable) capability of communication.
- Agent has labeled *ports*.
  - Input port `in`.
  - Output port $\overline{\texttt{out}}$.
- *Behavior* of C:
  - $C := \texttt{in}(x).C'(x)$
  - $C'(x) := \overline{\texttt{out}}(x).C$

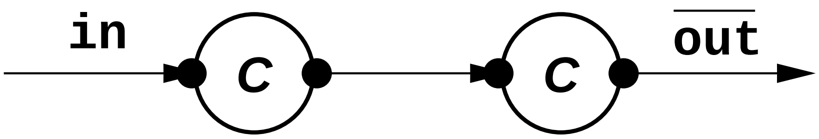
Behavior Descriptions

- Agent names can take parameters.
- Prefix $\mathtt{in}(x)$
  - Handshake in which value is received at port $\mathtt{in}$ and becomes the value of variable x.
- Agent expression $\mathtt{in}(x).C'(x)$
  - Perform handshake and proceed according to definition of C'.
- Agent expression $\overline{\mathtt{out}}(x).C$
  - Output the value of x at port $\overline{\mathtt{out}}$ and proceed according to the definition of C.
- Scope of local variables:
  - *Input* prefix introduces variable whose scope is the agent expression C.
  - Formal parameter of defining equation introduces variable whose scope is the equation.

Behavior Descriptions

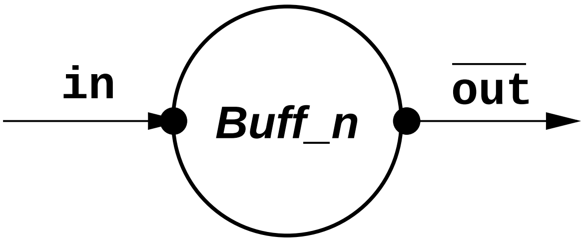
$C := \texttt{in}(x).\overline{\texttt{out}}(x).C$
$A := \texttt{in}(x).\texttt{in}(y).\overline{\texttt{out}}(x).\overline{\texttt{out}}(y).\mathsf{A}$

- How do behaviors differ?
  - A inputs two values and outputs two values.
  - C inputs and output a single value.



- Agent expression $C \frown C$.
  - *Combinator* $A_1 \frown A_2$ (defined later).
  - Agent formed by linking $\overline{\texttt{out}}$ of A_2 to $\texttt{in}$ of A_1.
  - $\frown$ is associative.

Bounded Buffer

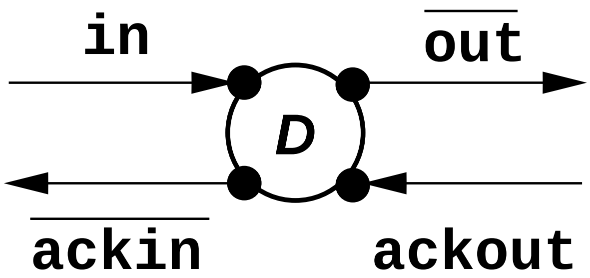


- $C^{(n)}$
  - $C^{(n)} := C \frown C \frown \ldots \frown C$
  - Behaves as bounded buffer of capacity n.
  - $C^{(n)} = \mathit{Buff}_n$
- Specification $\mathit{Buff}_n(s)$
  - $\mathit{Buff}_n\ \langle\ \rangle := \mathtt{in}(x).\mathit{Buff}_n\ \langle x \rangle$
  - $\mathit{Buff}_n\ \langle v_1, \ldots, v_n \rangle :=$
    $\overline{\mathtt{out}}(v_n).\mathit{Buff}_n\ \langle v_1, \ldots, v_{n-1} \rangle$
  - $\mathit{Buff}_n\ \langle v_1, \ldots, v_k \rangle :=$
    $\overline{\mathtt{in}}(x).\mathit{Buff}_n\ \langle x, v_1, \ldots, v_k \rangle$
    $+\ \overline{\mathtt{out}}(v_k).\mathit{Buff}_n\ \langle v_1, \ldots, v_{k-1} \rangle (0 < k < n)$
- $C^{(n)} = \mathit{Buff}_n\ \langle\ \rangle$

Summation

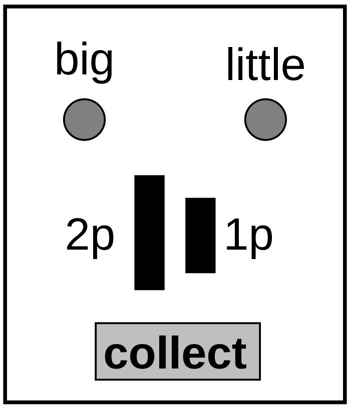
- Basic combinator '$+$'
  - $P + Q$ behaves like P or like Q.
  - When one performs its first action, other is discarded.
  - If both alternatives are allowed, selection is non-deterministic.
- Combining forms
  - *Summation* $P + Q$ of two agents.
  - *Sequencing* $\alpha.P$ of action α and agent P.
- Different levels of abstractions
  - Agent can be expressed directly in terms of its interaction with environment (C, Buff_n).
  - Agent can be expressed indirectly in terms of its composition of sammer agents ($C^{(n)}$).

Example

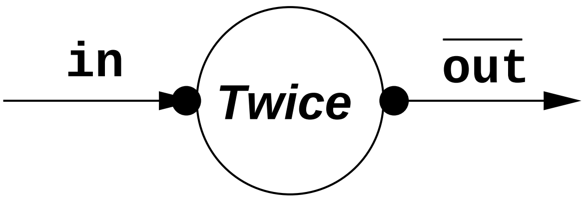


- Received values to be acknowledged.
  - $D := \texttt{in}(x).\overline{\texttt{out}}(x).\texttt{ackout}.\overline{\texttt{ackin}}.D$
  - D acknowledges input after it has delivered value as output and received acknowledgement.
  - Synchronization actions $\texttt{ackout}$, $\overline{\texttt{ackin}}$.
- Combination of n copies of D:
  - $D^{(n)} := D \frown D \frown \ldots \frown D$
  - $D^{(n)}$ behaves like *single* copy of D!
  - $D \frown D = D$!
- Alternative definition:
  - $D' := \texttt{in}(x).\overline{\texttt{ackin}}.\overline{\texttt{out}}(x).\texttt{ackout}.D'$
  - $D'^{(n)} = \mathit{Buff'}_n \langle\ \rangle$.
  - Slightly modified specification $\mathit{Buff'}_n$.

Examples

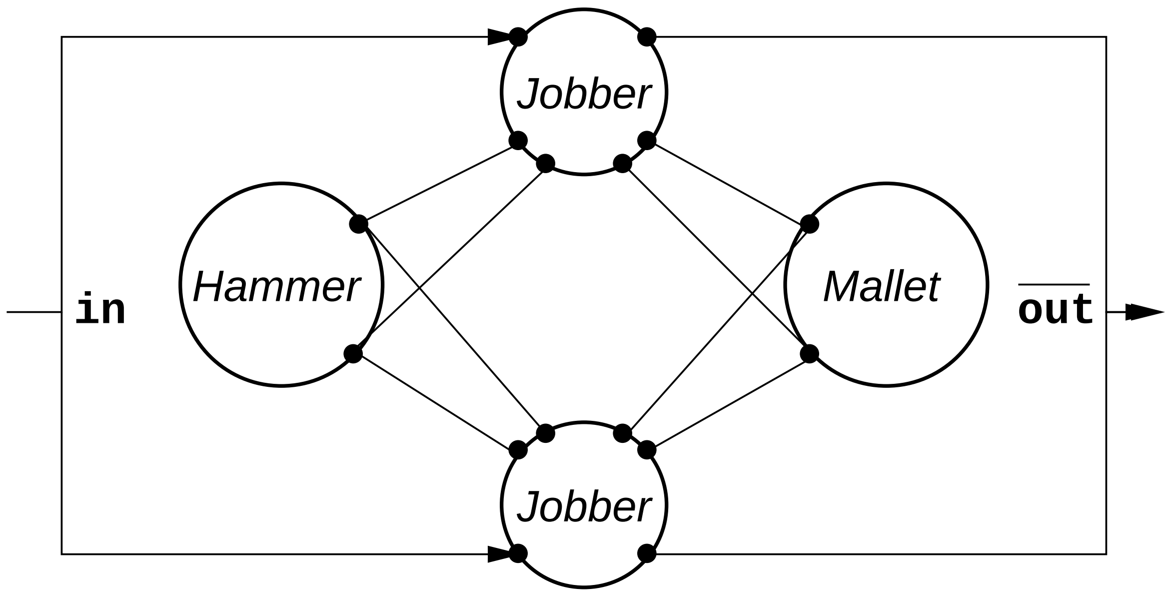


- A vending machine:
  - Big chocolade costs 2p, small one costs 1p.
  - $V := \texttt{2p.big.collect}.V$
    $+ \texttt{1p.little.collect}.V$



- A multiplier
  - $\mathit{Twice} := \texttt{in}(x).\overline{\texttt{out}}(2 * x).\mathit{Twice}.$
  - Output actions may take expressions.

A Larger Example: The Jobshop

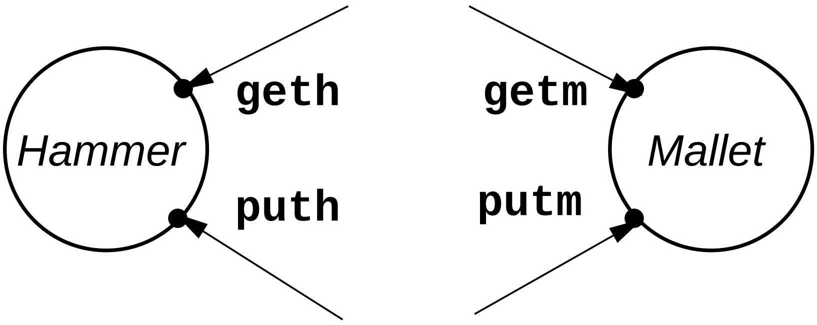


- A simple production line:
  - Two people (the *jobbers*).
  - Two tools (hammer and mallet).
  - *Jobs* arrive sequentially on a belt.
  - A job is to drive a peg into a block.

Flow Graphs

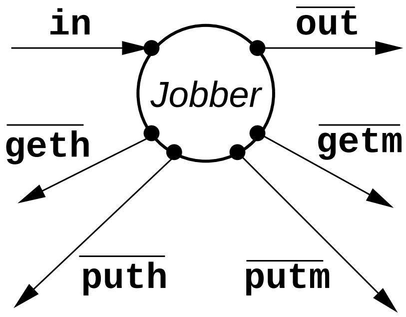
- Ports may be linked to more than one other port.
  - Jobbers compete for use of hammer.
  - Jobbers compete for use of job.
  - Source of non-determinism.
- Ports of belt are omitted from system.
  - `in` and $\overline{\texttt{out}}$ are external.
- Internal ports are not labelled:
  - Ports by which jobbers acquire and release tools.
- *Flow graph* is exact mathematical object:
  - Homomorphism from flow graph algebra to behavior algebra.

The Tools



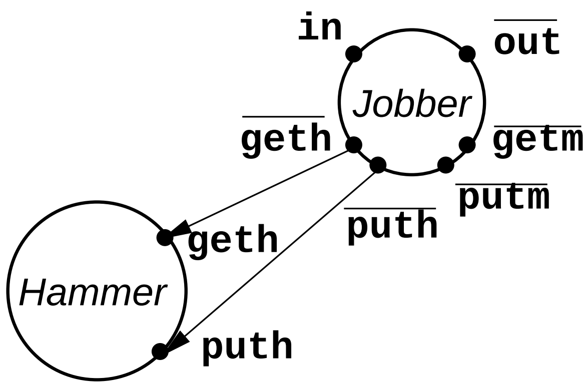
- Behaviors:
  - *Hammer* := $\mathtt{geth}.$*Busyhammer*
    Busyhammer := $\mathtt{puth}.$*Hammer*
  - *Mallet* := $\mathtt{geth}.$*Busymallet*
    Busymallet := $\mathtt{puth}.$*Mallet*
- *Sort* = set of labels
  - $P : L$ ... agent P has sort L
  - *Hammer*: $\{\mathtt{geth}, \mathtt{puth}\}$
    Mallet: $\{\mathtt{getm}, \mathtt{putm}\}$
    Jobshop: $\{\mathtt{in}, \overline{\mathtt{out}}\}$

The Jobbers



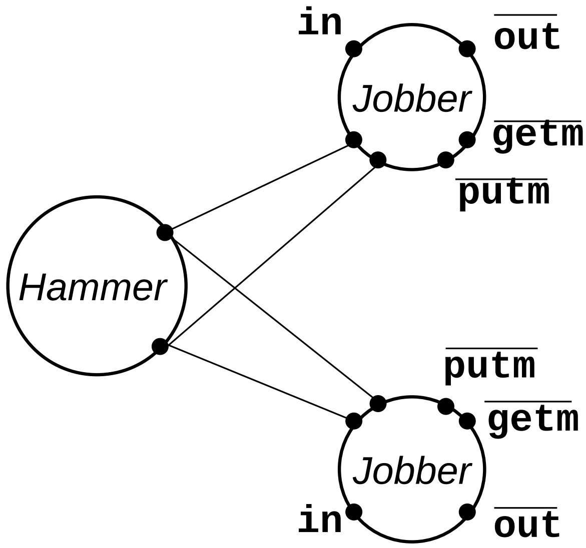
- Different kinds of jobs:
  - Easy jobs done with hands.
  - Hard jobs done with hammer.
  - Other jobs done with hammer or mallet.
- Behavior:
  - $\mathit{Jobber} := \mathtt{in}(\mathit{job}).\mathit{Start}(\mathit{job})$
  - $\mathit{Start}(\mathit{job}) :=$ **if** $\mathit{easy}(\mathit{job})$ **then** $\mathit{Finish}(\mathit{job})$
    else if $\mathit{hard}(\mathit{job})$ **then** $\mathit{Uhammer}(\mathit{job})$
    else $\mathit{Usetool}(\mathit{job})$
  - $\mathit{Usetool}(\mathit{job}) := \mathit{Uhammer}(\mathit{job}) + \mathit{Umallet}(\mathit{job})$
  - $\mathit{Uhammer}(\mathit{job}) := \overline{\mathtt{geth}}.\overline{\mathtt{puth}}.\mathit{Finish}(\mathit{job})$
  - $\mathit{Umallet}(\mathit{job}) := \overline{\mathtt{getm}}.\overline{\mathtt{putm}}.\mathit{Finish}(\mathit{job})$
  - $\mathit{Finish}(\mathit{job}) := \overline{\mathtt{out}}(\mathit{done}(\mathit{job})).\mathit{Jobber}$

Composition of Agents



- *Jobber-Hammer* subsystem
  - *Jobber* | *Hammer*
  - *Composition* operator |
  - Agents may procced independently or interact through *complementary* ports.
  - Join complementary ports.
- Two jobbers sharing hammer:
  - *Jobber* | *Hammer* | *Jobber*
  - Composition is commutative and associative.

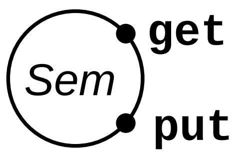
Further Compositon



- *Internalisation* of ports:
  - No further agents may be connected to ports:
  - *Restriction* operator \
  - $\backslash L$ internalizes all ports L.
  - (*Jobber* | *Jobber* | *Hammer*)$\backslash\{\texttt{geth},\texttt{puth}\}$
- Complete system:
  - *Jobshop* := (*Jobber* | *Jobber* | *Hammer* | *Mallet*)$\backslash L$
  - L := $\{\texttt{geth},\texttt{puth},\texttt{getm},\texttt{putm}\}$

Reformulations

- Alternative formulation:
  - $((\mathit{Jobber} \mid \mathit{Jobber} \mid \mathit{Hammer})\backslash\{\texttt{geth}, \texttt{puth}\} \mid \mathit{Mallet})\backslash\{\texttt{getm}, \texttt{putm}\}$
  - *Algebra* of combinators with certain laws of equivalence.
- *Relabelling* Operator
  - $P[l'_1/l_1, \ldots, l'_n/l_n]$
  - $f(\bar{l}) = \overline{f(l)}$



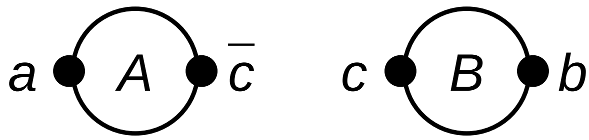
- Semaphore agent
  - $\mathit{Sem} := \mathtt{get}.\mathtt{put}.\mathit{Sem}$
- Reformulation of tools
  - $\mathit{Hammer} := \mathit{Sem}[\texttt{geth}/\texttt{get}, \texttt{puth}/\texttt{put}]$
  - $\mathit{Mallet} := \mathit{Sem}[\texttt{getm}/\texttt{get}, \texttt{putm}/\texttt{put}]$

Equality of Agents

- Five basic operators:
  - *Prefix*: $\alpha.P$
  - *Summation*: $P + Q$
  - *Composition*: $P \mid Q$
  - *Restriction*: $P\backslash\{l_1, \ldots, l_n\}$
  - *Relabelling*: $P[l'_1/l_1,\ldots,l'_n/l_n]$
- *Strongjobber* only needs hands:
  - *Strongjobber* $:=$
    `in`(*job*).$\overline{\texttt{out}}$(*done*(*job*)).*Strongjobber*
- Claim:
  - *Jobshop* $=$ *Strongjobber* $\mid$ *Strongjobber*
  - *Specification* of system *Jobshop*
  - Proof of equality required.

Action and Transition

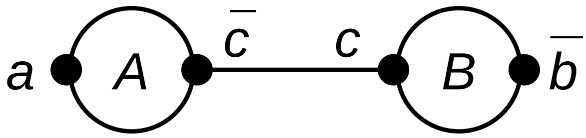
- Names and co-names
  - Set A of *names* ($\mathtt{geth}$, $\mathtt{ackin}$, ...)
  - Set $\underline{A}$ of *co-names* ($\overline{\mathtt{geth}}$, $\overline{\mathtt{ackin}}$, ...)
  - Set of *labels* $L = A \cup \overline{A}$
- Transition $P \xrightarrow{l} Q$
  - *Hammer* $\overset{\mathtt{geth}}{\to}$ *Busyhammer*
  - *Busyhammer* $\overset{\mathtt{puth}}{\to}$ *Hammer*
- Agents A and B



  - $A := a.A'$, $A' := \overline{c}.A$
  - $B := c.B'$, $B' := \overline{b}.B$

Composite Agents

- Composite Agent $A|B$



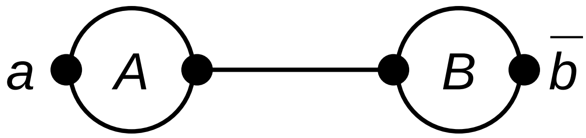
  - $A \stackrel{a}{\to} A'$ allows $A|B \stackrel{a}{\to} A'|B$
  - $A' \stackrel{\overline{c}}{\to} A$ allows $A'|B \stackrel{\overline{c}}{\to} A|B$
  - $A' \stackrel{\overline{c}}{\to} A$ and $B \stackrel{c}{\to} B'$ allows $A'|B \stackrel{\tau}{\to} A|B'$

- *Completed (perfect)* action τ.
  - Simultaneous action of both agents.
  - *Internal* to composed agent.
  - *Act* $= L \cup \{\tau\}$

- Internal versus external actions
  - Internal actions are ignored.
  - Only external actions are visible.
  - Two systems are *equivalent* if they exhibit same pattern of external actions.
  - $P \stackrel{\tau}{\to} P_1 \stackrel{\tau}{\to} \ldots \stackrel{\tau}{\to} P_n$ equivalent to $P \stackrel{\tau}{\to} P_n$

Restrictions

- Restriction $(A|B)\backslash c$



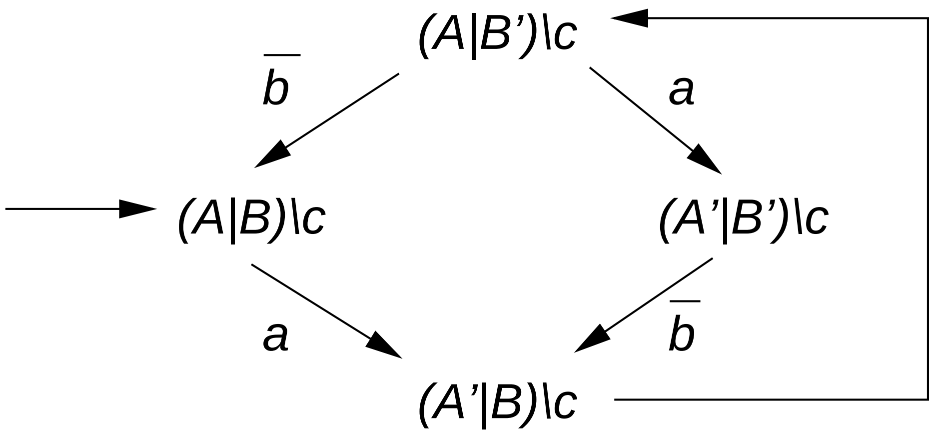
  - $P \stackrel{\alpha}{\rightarrow} P'$ allows $P\backslash L \stackrel{\alpha}{\rightarrow} P'\backslash L$ (if α, $\overline{\alpha}$ not in L)

- *Transition (derivation) tree*

$$
\begin{array}{ccc}
 & (A|B)\backslash c & \\
 & \downarrow a & \\
 & (A'|B)\backslash c & \\
 & \downarrow \tau & \\
 & (A|B')\backslash c & \\
\swarrow \overline{b} & & \searrow a \\
(A|B)\backslash c & & (A'|B')\backslash c \\
\downarrow a & & \downarrow \overline{b} \\
(A'|B)\backslash c & & (A'|B)\backslash c \\
\ldots & & \ldots
\end{array}
$$

Transition Graph

- *Transition graph*



  - $(A|B)\backslash c = a.\tau.C$
  - $C := a.\overline{b}.\tau.C + \overline{b}.a.\tau.C$

- Composite system
  - Behavior defined without use of composition combinator $|$ or restriction combinator!

- Internal communication
  - $\alpha.\tau.P = \alpha.P$
  - $(A|B)\backslash c = a.D$
  - $D := a.\overline{b}.D + \overline{b}.a.D$

The Basic Language

- Agent expressions
  - Agent constants and variables
  - *Prefix* $\alpha.E$
  - *Summation* ΣE_i
  - *Composition* $E_1|E_2$
  - *Restriction* $E\backslash L$
  - *Relabelling* $E[f]$
- No value transmission between agents
  - Just synchronization.

The Transition Rules

- Act $\quad \alpha.E \stackrel{\alpha}{\to} E$
- Sum$_j$ $\quad \dfrac{E_j \stackrel{\alpha}{\to} E'_j}{\Sigma E_i \stackrel{\alpha}{\to} E'_j}$
- Com$_1$ $\quad \dfrac{E \stackrel{\alpha}{\to} E'}{E|F \stackrel{\alpha}{\to} E'|F}$
- Com$_2$ $\quad \dfrac{F \stackrel{\alpha}{\to} F'}{E|F \stackrel{\alpha}{\to} E|F'}$
- Com$_3$ $\quad \dfrac{E \stackrel{l}{\to} E' \quad F \stackrel{\overline{l}}{\to} F'}{E|F \stackrel{\tau}{\to} E'|F'}$
- Res $\quad \dfrac{E \stackrel{\alpha}{\to} E'}{E\backslash L \stackrel{\alpha}{\to} E'\backslash L}$ $\quad$ (α, $\overline{\alpha}$ not in L)
- Rel $\quad \dfrac{E \stackrel{\alpha}{\to} E'}{E[f] \stackrel{f(\alpha)}{\to} E'[f]}$
- Con $\quad \dfrac{P \stackrel{\alpha}{\to} P'}{A \stackrel{\alpha}{\to} P'}$ $\quad$ ($A := P$)

Derivatives and Derivation Trees

- *Immediate derivative* of E
  - Pair (α, E')
  - $E \xrightarrow{\alpha} E'$
  - E' is α-*derivative* of E
- *Derivative* of E
  - Pair $(\alpha_1 \ldots \alpha_n, E')$
  - $E \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} E'$
  - E' is *($\alpha_1 \ldots \alpha_n$-)derivative* of E
- *Derivation tree* of E

$$
\begin{array}{ccccc}
 & & & & E_{11} \ldots \\
 & & & \nearrow\alpha_{11} & \\
 & & E_1 & & \\
 & \nearrow\alpha_1 & & \searrow\alpha_{12} & \\
E & & & & E_{12} \ldots \\
 & \searrow\alpha_2 & & & \\
 & & E_2 & \ldots &
\end{array}
$$

Examples of Derivation Trees

- *Partial* derivation tree

$$
\begin{array}{lll}
 & & (E|F)\backslash a \\
 & \nearrow^{\tau} & \\
((a.E + b.0)|\ \overline{a}.F)\backslash a & & \\
 & \searrow^{b} & \\
 & & (0|\ \overline{a}.F)\backslash a
\end{array}
$$

- $a.X + b.Y$

$$
\begin{array}{lll}
 & & X \\
 & \nearrow^{a} & \\
a.X + b.Y & & \\
 & \searrow^{b} & \\
 & & Y
\end{array}
$$

- *Behavioural equivalence*
  - Two agent expressions are behaviourally equivalent if they yield the same *total* derivation trees

The Value-Passing Calculus

- Values passed between agents
  - Can be reduced to basic calculus.
  - $C := \texttt{in}(x).C'(x)$
    $C'(x) := \overline{\texttt{out}}(x).C$
  - $C := \Sigma_v \ \texttt{in}_v.C'_v$
    $C'_v := \overline{\texttt{out}}_v.C \ (v \in V)$
  - *Families* of ports and agents.
- The full language
  - *Prefixes* $a(x).E$, $\overline{a}(e).E$, $\tau.E$
  - *Conditional* **if** b **then** E
- Translation
  - $a(x).E \Rightarrow \Sigma_v.E\{v/x\}$
  - $\overline{a}(e).E \Rightarrow \overline{a}_e.E$
  - $\tau.E \Rightarrow \tau.E$
  - **if** b **then** $E \Rightarrow$ (E, if b and 0, otherwise)