

A Process Calculus I

A Process Calculus I

Wolfgang Schreiner

Research Institute for Symbolic Computation

Johannes Kepler University, Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at

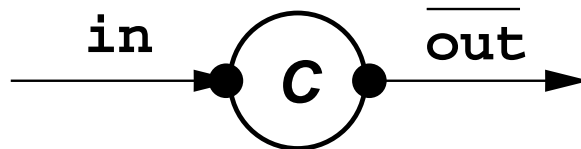
<http://www.risc.uni-linz.ac.at/people/schreine>

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 $\overline{\text{out}}$.
- *Behavior* of *C*:
 - $C := \text{in}(x).C'(x)$
 - $C'(x) := \overline{\text{out}}(x).C$

Behavior Descriptions

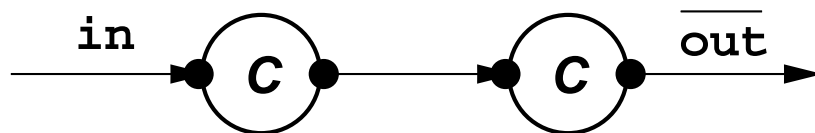
- Agent names can take parameters.
- Prefix $\text{in}(x)$
 - Handshake in which value is received at port in and becomes the value of variable x .
- Agent expression $\text{in}(x).C'(x)$
 - Perform handshake and proceed according to definition of C' .
- Agent expression $\overline{\text{out}}(x).C$
 - Output the value of x at port $\overline{\text{out}}$ 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

$C := \text{in}(x).\overline{\text{out}}(x).C$

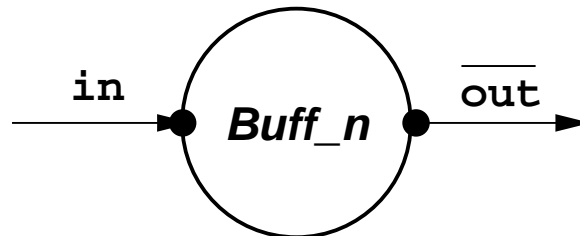
$A := \text{in}(x).\text{in}(y).\overline{\text{out}}(x).\overline{\text{out}}(y).A$

- 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 $\overline{\text{out}}$ of A_2 to in of A_1 .
 - \frown is associative.

Bounded Buffer

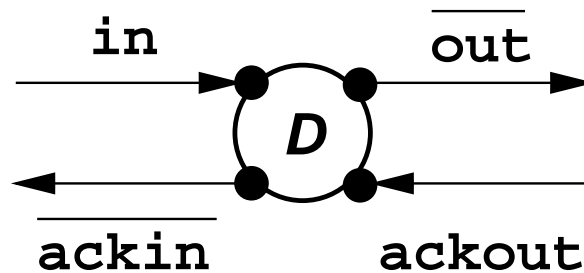


- $C^{(n)}$
 - $C^{(n)} := C \frown C \frown \dots \frown C$
 - Behaves as bounded buffer of capacity n .
 - $C^{(n)} = Buff_n$
- Specification $Buff_n(s)$
 - $Buff_n \langle \rangle := in(x).Buff_n \langle x \rangle$
 - $Buff_n \langle v_1, \dots, v_n \rangle := \overline{out}(v_n).Buff_n \langle v_1,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

Summation

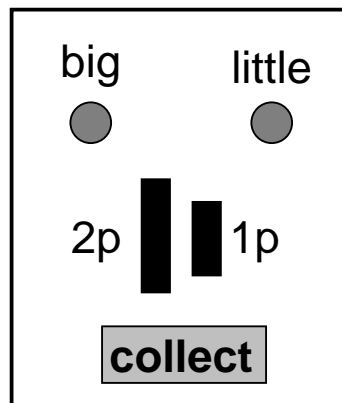
- Basic combinator '+'
 - $P + Q$ behaves like P or like Q .
 - When one performs its first action, other is discarded.
 - If both alternatives are allowed, selection is non-deterministic.
- Combining forms
 - *Summation* $P + Q$ of two agents.
 - *Sequencing* $\alpha.P$ of action α and agent P .
- Different 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

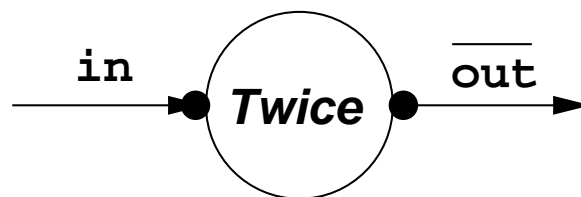


- Received values to be acknowledged.
 - $D := \text{in}(x).\overline{\text{out}}(x).\text{ackout}.\overline{\text{ackin}}.D$
 - D acknowledges input after it has delivered value as output and received acknowledgement.
 - Synchronization actions ackout , $\overline{\text{ackin}}$.
- Combination of n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

Examples

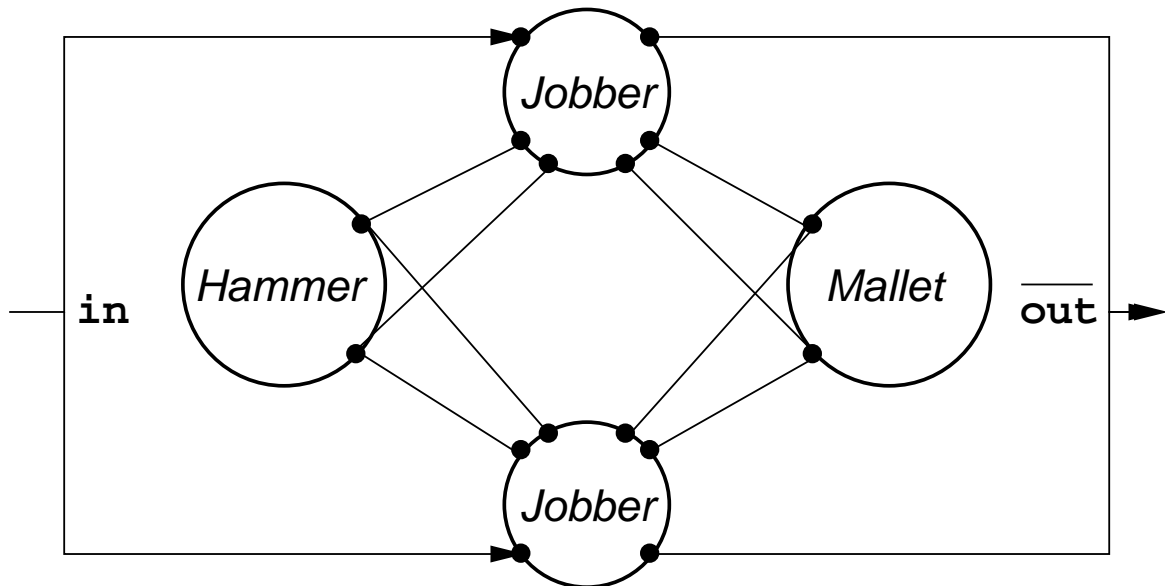


- A vending machine:
 - Big chocolate costs 2p, small one costs 1p.
 - $V := 2p.\text{big}.\text{collect}.V + 1p.\text{little}.\text{collect}.V$



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

A Larger Example: The Jobshop

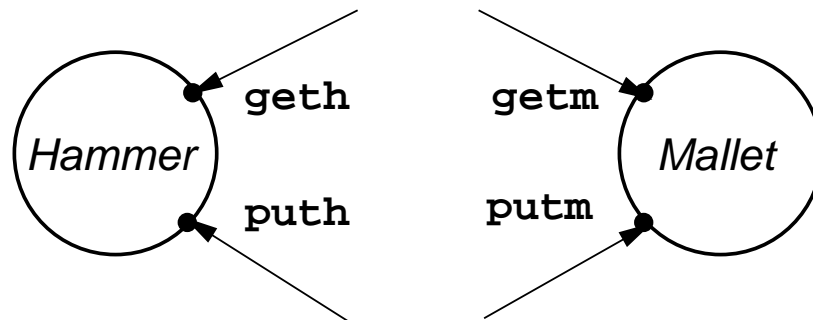


- A simple production line:
 - Two people (the *jobbers*).
 - Two tools (hammer and mallet).
 - *Jobs* arrive sequentially on a belt.
 - A job is to drive a peg into a block.

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 $\overline{\text{out}}$ 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



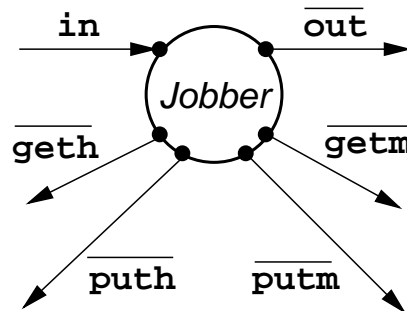
- Behaviors:

- $\text{Hammer} := \text{geth}.\text{Busyhammer}$
 $\text{Busyhammer} := \text{puth}.\text{Hammer}$
- $\text{Mallet} := \text{getm}.\text{Busymallet}$
 $\text{Busymallet} := \text{putm}.\text{Mallet}$

- Sort = set of labels

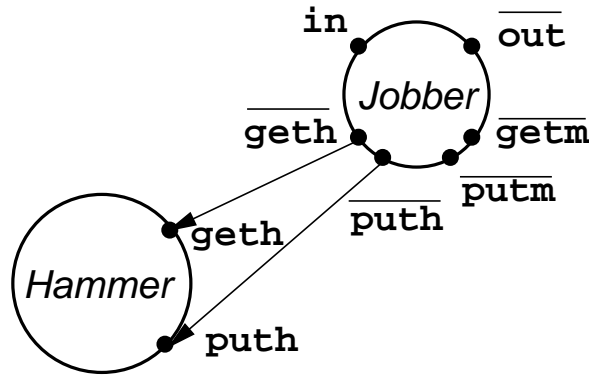
- $P : L \dots$ agent P has sort L
- $\text{Hammer}: \{\text{geth}, \text{puth}\}$
 $\text{Mallet}: \{\text{getm}, \text{putm}\}$
 $\text{Jobshop}: \{\text{in}, \overline{\text{out}}\}$

The Jobbers



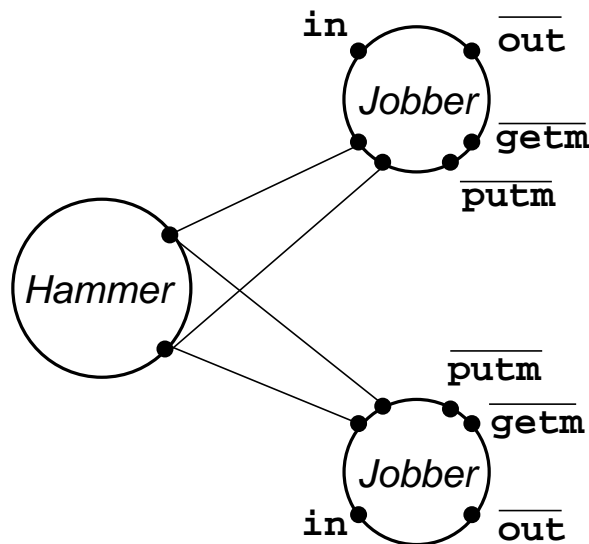
- Different kinds of jobs:
 - Easy jobs done with hands.
 - Hard jobs done with hammer.
 - Other jobs done with hammer or mallet.
- Behavior:
 - $Jobber := in(job).Start(job)$
 - $Start(job) := \mathbf{if\ } easy(job) \mathbf{\ then\ } Finish(job)$
 $\mathbf{else\ if\ } hard(job) \mathbf{\ then\ } Uhammer(job)$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

Composition of Agents



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

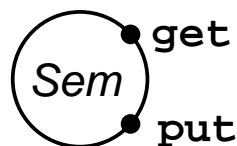
Further Composition



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

Reformulations

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



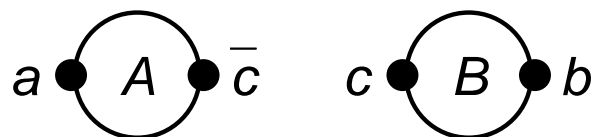
- Semaphore agent
 - $Sem := get.put.Sem$
- Reformulation of tools
 - $Hammer := Sem[geth/get, puth/put]$
 - $Mallet := Sem[getm/get, putm/put]$

Equality of Agents

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

Action and Transition

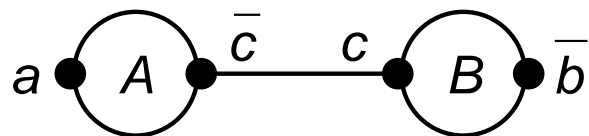
- Names and co-names
 - Set A of *names* (*geth*, *ackin*, ...)
 - Set \bar{A} of *co-names* ($\overline{\text{geth}}$, $\overline{\text{ackin}}$, ...)
 - Set of *labels* $L = A \cup \bar{A}$
- Transition $P \xrightarrow{l} Q$
 - $\text{Hammer} \xrightarrow{\text{geth}} \text{Busyhammer}$
 - $\text{Busyhammer}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



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

Composite Agents

- Composite Agent $A|B$



- $A \xrightarrow{a} A'$ allows $A|B \xrightarrow{a} A'|B$
- $A' \xrightarrow{\bar{c}} A$ allows $A'|B \xrightarrow{\bar{c}} A|B$
- $A' \xrightarrow{\bar{c}} A$ and $B \xrightarrow{c} B'$ allows $A'|B \xrightarrow{\tau} A|B'$

- *Completed (perfect) action* τ .

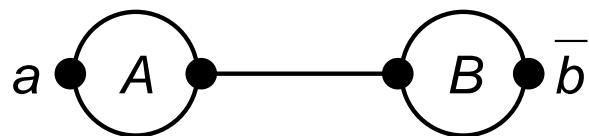
- Simultaneous action of both agents.
- *Internal* to composed agent.
- $Act = L \cup \{\tau\}$

- 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.
- $P \xrightarrow{\tau} P_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} P_n$ equivalent to $P \xrightarrow{\tau} P_n$

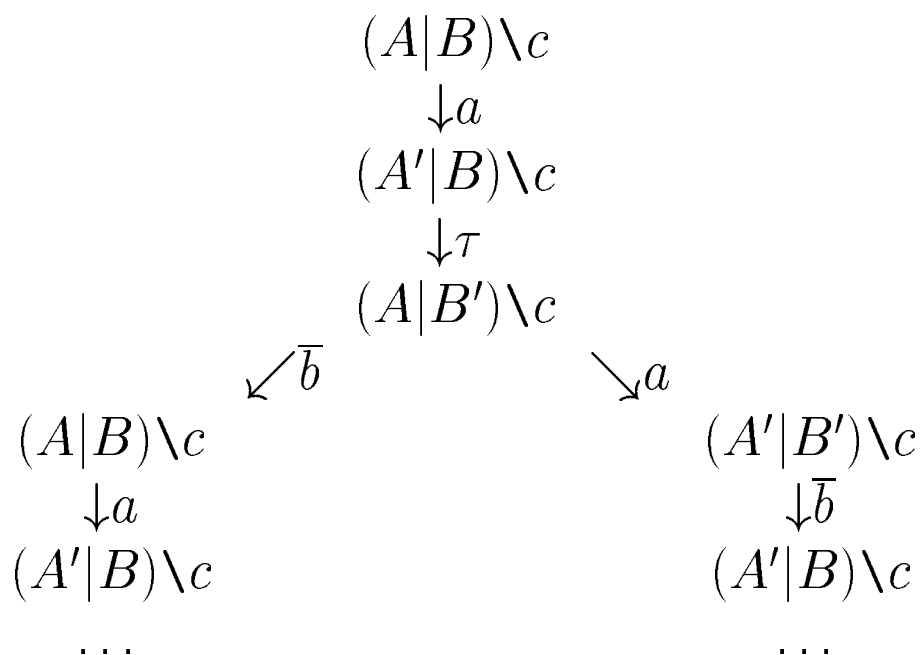
Restrictions

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



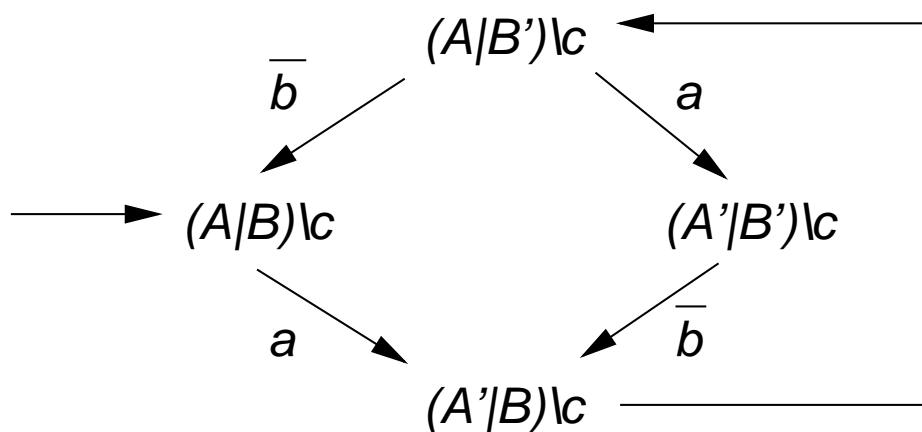
- $P \xrightarrow{\alpha} P'$ allows $P\backslash L \xrightarrow{\alpha} P'\backslash L$
(if $\alpha, \bar{\alpha}$ not in L)

- *Transition (derivation) tree*



Transition Graph

- *Transition graph*



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

- **Composite system**

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

- **Internal communication**

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

The Basic Language

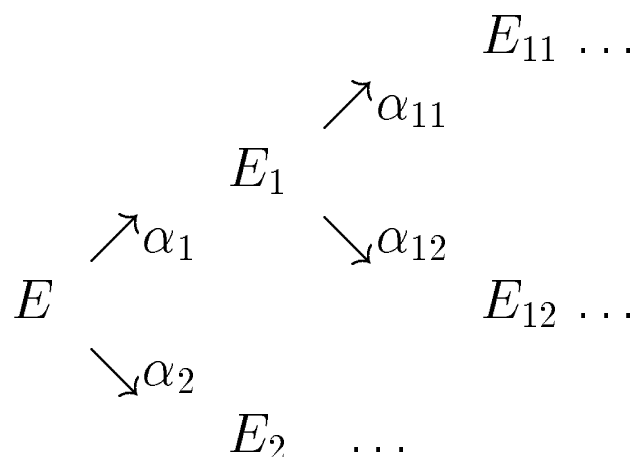
- Agent expressions
 - Agent constants and variables
 - *Prefix* $\alpha.E$
 - *Summation* ΣE_i
 - *Composition* $E_1|E_2$
 - *Restriction* $E \setminus L$
 - *Relabelling* $E[f]$
- No value transmission between agents
 - Just synchronization.

The Transition Rules

- Act $\alpha.E \xrightarrow{\alpha} E$
- Sum_j
$$\frac{E_j \xrightarrow{\alpha} E'_j}{\Sigma E_i \xrightarrow{\alpha} E'_j}$$
- Com₁
$$\frac{E \xrightarrow{\alpha} E'}{E|F \xrightarrow{\alpha} E'|F}$$
- Com₂
$$\frac{F \xrightarrow{\alpha} F'}{E|F \xrightarrow{\alpha} E|F'}$$
- Com₃
$$\frac{E \xrightarrow{l} E' \quad F \xrightarrow{\bar{l}} F'}{E|F \xrightarrow{\tau} E'|F'}$$
- Res
$$\frac{E \xrightarrow{\alpha} E'}{E \setminus L \xrightarrow{\alpha} E' \setminus L} \quad (\alpha, \bar{\alpha} \text{ not in } L)$$
- Rel
$$\frac{E \xrightarrow{\alpha} E'}{E[f] \xrightarrow{f(\alpha)} E'[f]}$$
- Con
$$\frac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'} \quad (A := P)$$

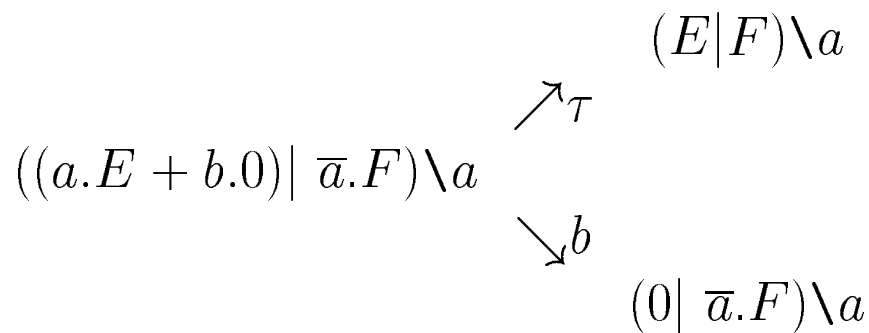
Derivatives and Derivation Trees

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

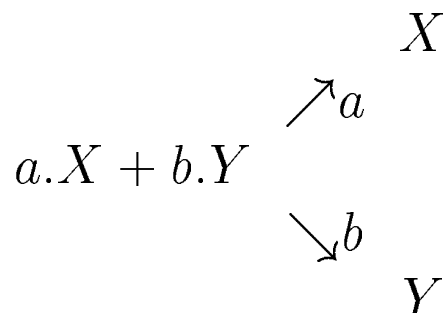


Examples of Derivation Trees

- *Partial* derivation tree



- $a.X + b.Y$



- *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.
 - $C := \text{in}(x).C'(x)$
 $C'(x) := \overline{\text{out}}(x).C$
 - $C := \Sigma_v \text{in}_v.C'_v$
 $C'_v := \overline{\text{out}}_v.C \ (v \in V)$
 - *Families* of ports and agents.
- The full language
 - *Prefixes* $a(x).E, \bar{a}(e).E, \tau.E$
 - *Conditional* **if** b **then** E
- Translation
 - $a(x).E \Rightarrow \Sigma_v.E\{v/x\}$
 - $\bar{a}(e).E \Rightarrow \bar{a}_e.E$
 - $\tau.E \Rightarrow \tau.E$
 - **if** b **then** $E \Rightarrow (E, \text{ if } b \text{ and } 0, \text{ otherwise})$