x = 1

let x = 1 in ...

x(1).

!x(1)

x.set(1)

Programming Paradigms and Formal Semantics

The Calculus of Communicating Systems

Ralf Lämmel

≈1h

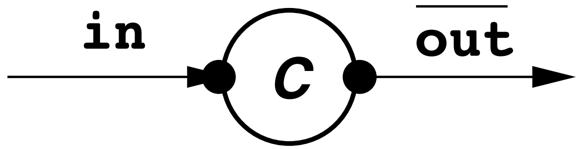
The Calculus of Communicating Systems (CCS)

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

Modeling of communication and concurrency.

These slides were obtained by copy&paste&edit from W. Schreiner's concurrency lectures (Kepler University, Linz).

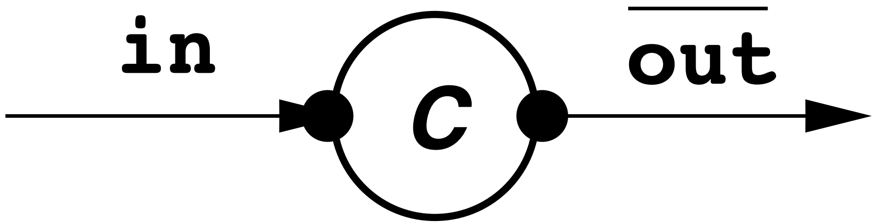
A simple example



- *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 $\overline{\texttt{out}}$.

These slides were obtained by copy&paste&edit from W. Schreiner's concurrency lectures (Kepler University, Linz).

A simple example



Behavior of C:

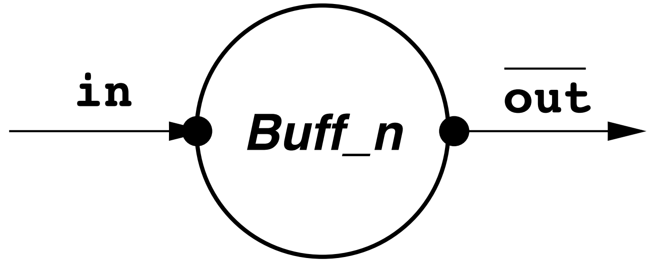
- $C := \mathtt{in}(x).C'(x)$
- $C'(x) := \overline{\mathtt{out}}(x).C$

Process behaviors are described as (mutually recursive) equations.

Behavior descriptions -- summary

- Agent names can take parameters.
- Prefix $\mathtt{in}(x)$
  - Handshake in which value is received at port `in` and becomes the value of variable x.
- Agent expression $\mathtt{in}(x).C'(x)$
  - Perform handshake and proceed as described by 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.

Another example: bounded buffers



Bounded buffer $\mathit{Buff}_n(s)$

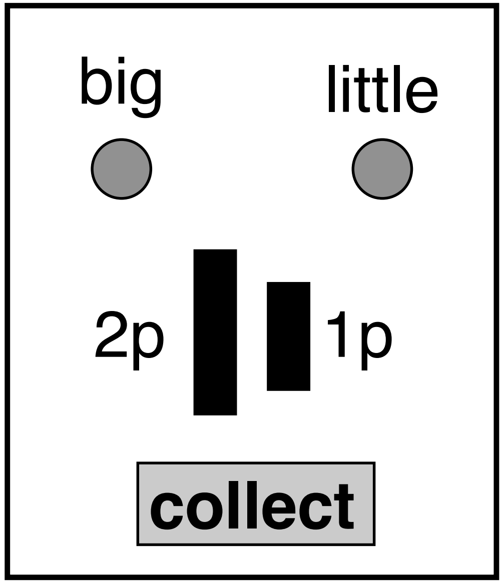
- $\mathit{Buff}_n\ \langle\ \rangle := \mathtt{in}(x).\mathit{Buff}_n\ \langle x \rangle$
- $\mathit{Buff}_n\ \langle v_1, \ldots, v_n \rangle :=$
  $\overline{\mathtt{out}}(v_n).\mathit{Buff}_n\ \langle v_1, \ldots, v_{n-1} \rangle$
- $\mathit{Buff}_n\ \langle v_1, \ldots, v_k \rangle :=$
  $\overline{\mathtt{in}}(x).\mathit{Buff}_n\ \langle x, v_1, \ldots, v_k \rangle$
  $+\ \overline{\mathtt{out}}(v_k).\mathit{Buff}_n\ \langle v_1, \ldots, v_{k-1} \rangle (0 < k < n)$

Used language elements

- 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.

Process definitions may be parameterized.

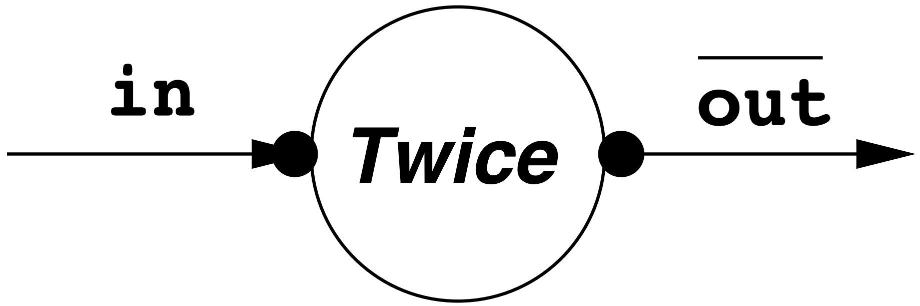
Example: a vending machine



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

Exercises:
Identify input vs. output.
What behaviors make sense for users?

Example: a multiplier

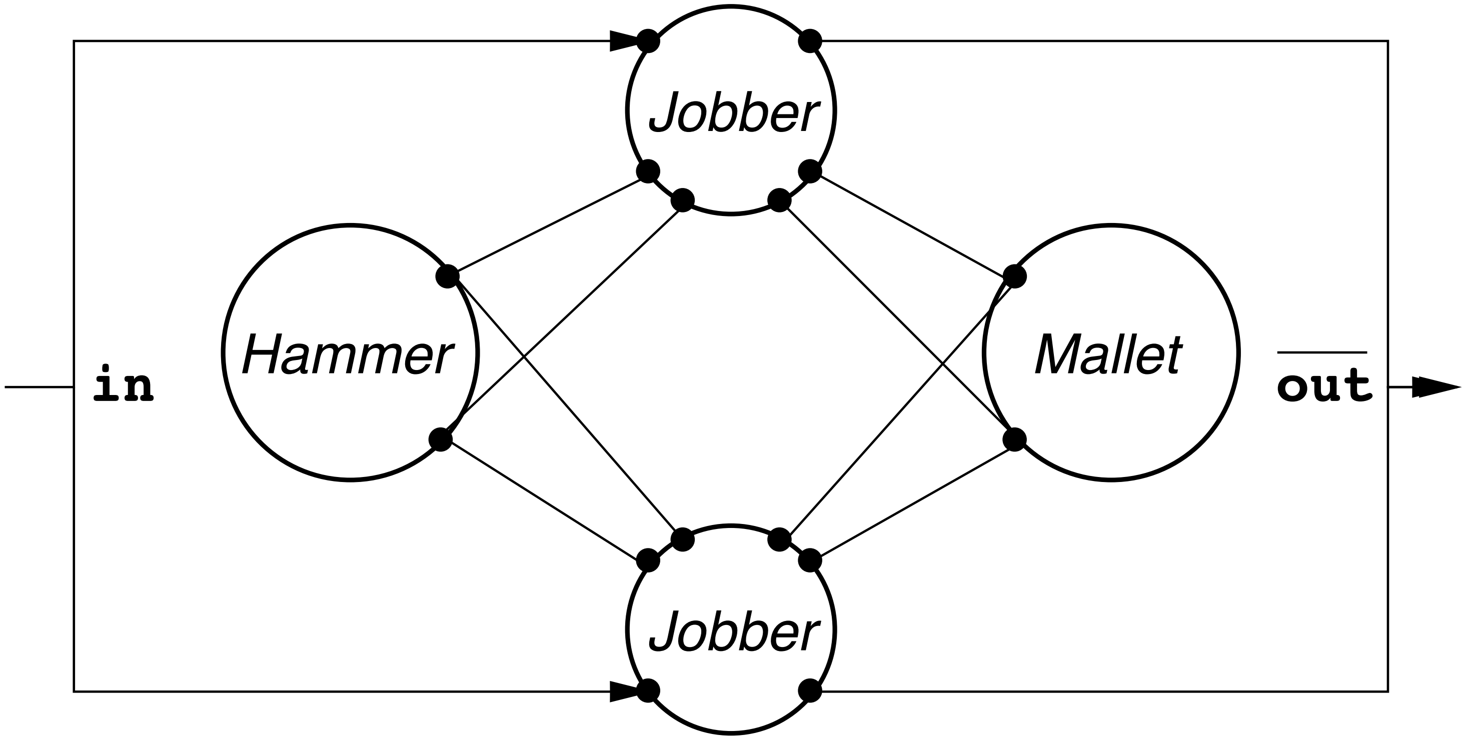


– $\mathit{Twice} := \texttt{in}(x).\overline{\texttt{out}}(2 * x).\mathit{Twice}$.

– Output actions may take expressions.

These slides were obtained by copy&paste&edit from W. Schreiner's concurrency lectures (Kepler University, Linz).

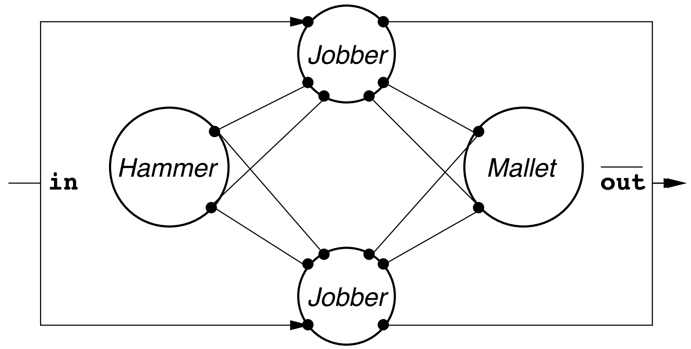
Example: The JobShop



These slides were obtained by copy&paste&edit from W. Schreiner's concurrency lectures (Kepler University, Linz).

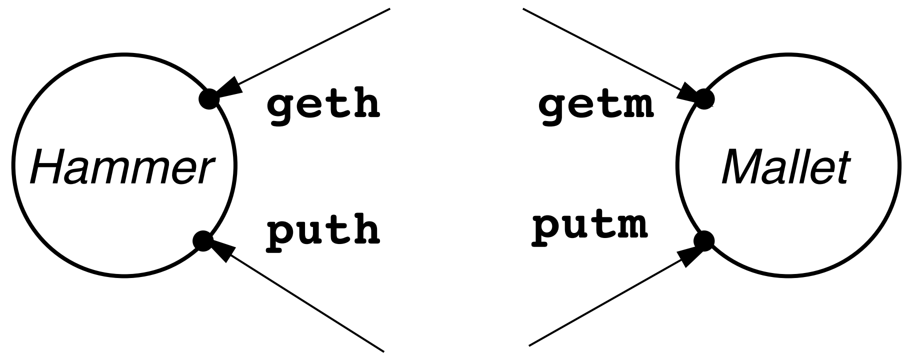
Example: The JobShop

- A simple production line:
  - Two people (the *jobbers*).
  - Two tools (hammer and mallet).
  - *Jobs* arrive sequentially on a belt to be processed.
- Ports may be linked to multiple ports.
  - 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.



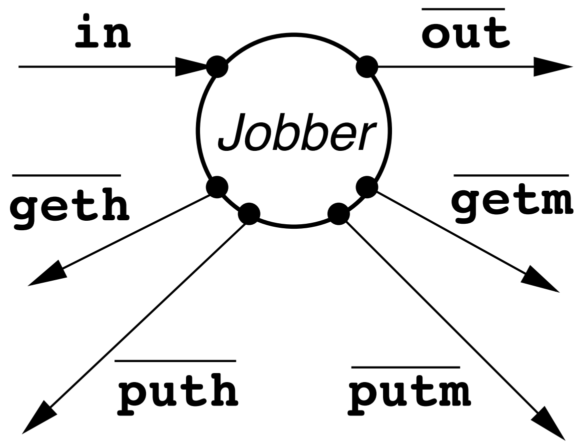
The tools of the JobShop

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



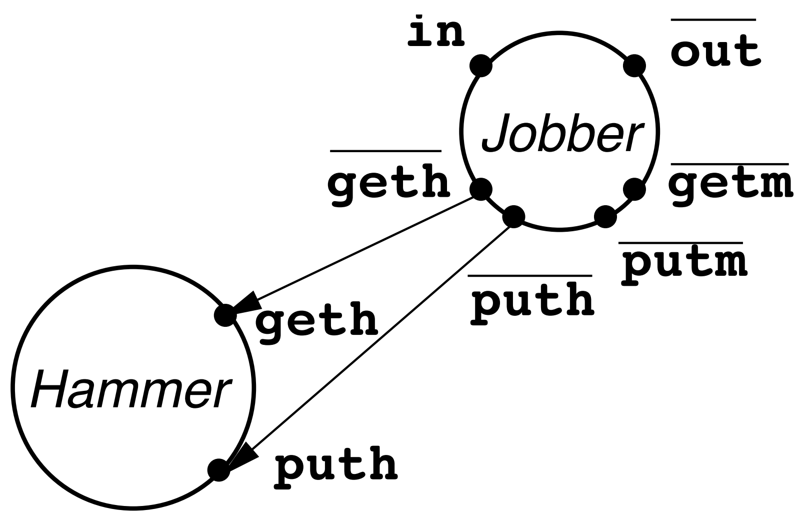
The jobbers of the JobShop

- 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 $\mathit{Usetool}(\mathit{job})$
  - $\mathit{Usetool}(\mathit{job}) := \mathit{Uhammer}(\mathit{job}) + \mathit{Umallet}(\mathit{job})$
  - $\mathit{Uhammer}(\mathit{job}) := \overline{\mathtt{geth}}.\overline{\mathtt{puth}}.\mathit{Finish}(\mathit{job})$
  - $\mathit{Umallet}(\mathit{job}) := \overline{\mathtt{getm}}.\overline{\mathtt{putm}}.\mathit{Finish}(\mathit{job})$
  - $\mathit{Finish}(\mathit{job}) := \overline{\mathtt{out}}(\mathit{done}(\mathit{job})).\mathit{Jobber}$

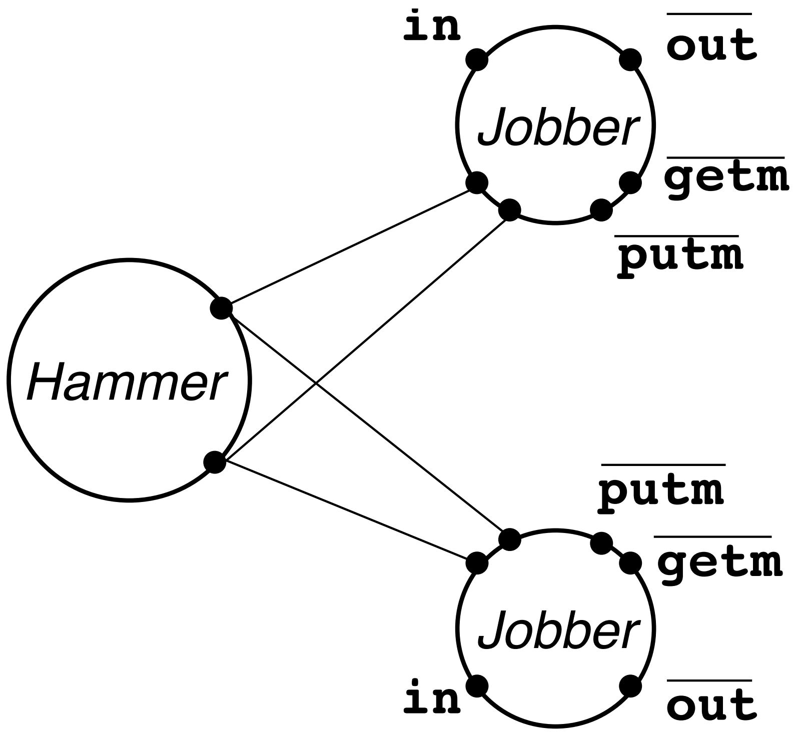


Composition of the agents

- *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 composition

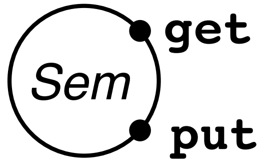


- *Internalisation* of ports:
  - No further agents may be connected to ports:
  - *Restriction* operator \
  - $\backslash L$ internalizes all ports L.
  - $(\mathit{Jobber} \mid \mathit{Jobber} \mid \mathit{Hammer})\backslash\{\texttt{geth},\texttt{puth}\}$
- Complete system:
  - $\mathit{Jobshop}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

These slides were obtained by copy&paste&edit from W. Schreiner's concurrency lectures (Kepler University, Linz).

Reformulations

- *Relabelling* Operator
  - $P[l_1'/l_1, \ldots, l_n'/l_n]$
  - $f(\bar{l}) = \overline{f(l)}$



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

In need of equality of agents

- *Strongjobber* only needs hands:
  - $\mathit{Strongjobber} :=$
    $\mathtt{in}(\mathit{job}).\overline{\mathtt{out}}(\mathit{done}(\mathit{job})).\mathit{Strongjobber}$
- Claim:
  - $\mathit{Jobshop} = \mathit{Strongjobber} \mid \mathit{Strongjobber}$
  - *Specification* of system *Jobshop*
  - Proof of equality required.

In which sense are the processes equal?

The core calculus
No value transmission: just synchronization

- Agent expressions
  - Agent constants and variables
  - *Prefix* $\alpha.E$
  - *Summation* ΣE_i
  - *Composition* $E_1 | E_2$
  - *Restriction* $E \backslash L$
  - *Relabelling* $E[f]$

Definitions of agents

Generalization of binary "+"

- Names and co-names
  - Set A of *names* ($\texttt{geth}$, $\texttt{ackin}$, $\ldots$)
  - Set $\underline{A}$ of *co-names* ($\overline{\texttt{geth}}$, $\overline{\texttt{ackin}}$, $\ldots$)
  - Set of *labels* $L = A \cup \overline{A}$
- Actions
  - *Completed (perfect)* action τ.
  - *Act* $= L \cup \{\tau\}$
- Transition $P \xrightarrow{l} Q$ with action l
  - *Hammer* $\xrightarrow{\texttt{geth}}$ *Busyhammer*

Transition rules of the core calculus

- Act $\quad \alpha.E \xrightarrow{\alpha} E$
- Sum$_j$ $\quad \dfrac{E_j \xrightarrow{\alpha} E'_j}{\sum E_i \xrightarrow{\alpha} E'_j}$
- Com$_1$ $\quad \dfrac{E \xrightarrow{\alpha} E'}{E|F \xrightarrow{\alpha} E'|F}$
- Com$_2$ $\quad \dfrac{F \xrightarrow{\alpha} F'}{E|F \xrightarrow{\alpha} E|F'}$
- Com$_3$ $\quad \dfrac{E \xrightarrow{l} E' \quad F \xrightarrow{\overline{l}} F'}{E|F \xrightarrow{\tau} E'|F'}$

This rule makes clear that no more than two agents participate in communication.

This rule rules out transitions with hidden names.

- Res $\quad \dfrac{E \xrightarrow{\alpha} E'}{E\backslash L \xrightarrow{\alpha} E'\backslash L} \quad (\alpha, \overline{\alpha}$ not in $L)$
- Rel $\quad \dfrac{E \xrightarrow{\alpha} E'}{E[f] \xrightarrow{f(\alpha)} E'[f]}$
- Con $\quad \dfrac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'} \quad (A := P)$

This is about the application of definitions for agents.

The value-passing calculus

- Values passed between agents
  - Can be reduced to basic calculus.
  - $C := \mathtt{in}(x).C'(x)$
    $C'(x) := \overline{\mathtt{out}}(x).C$
  - $C := \Sigma_v\ \mathtt{in}_v.C'_v$
    $C'_v := \overline{\mathtt{out}}_v.C\ (v \in V)$
  - *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)

Derivation trees

(Exhaustive application of the transition relation)

- *Derivation tree* of E

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

Behavioral equivalence: two agent expressions are behaviorally equivalent if they yield the same total derivation trees.

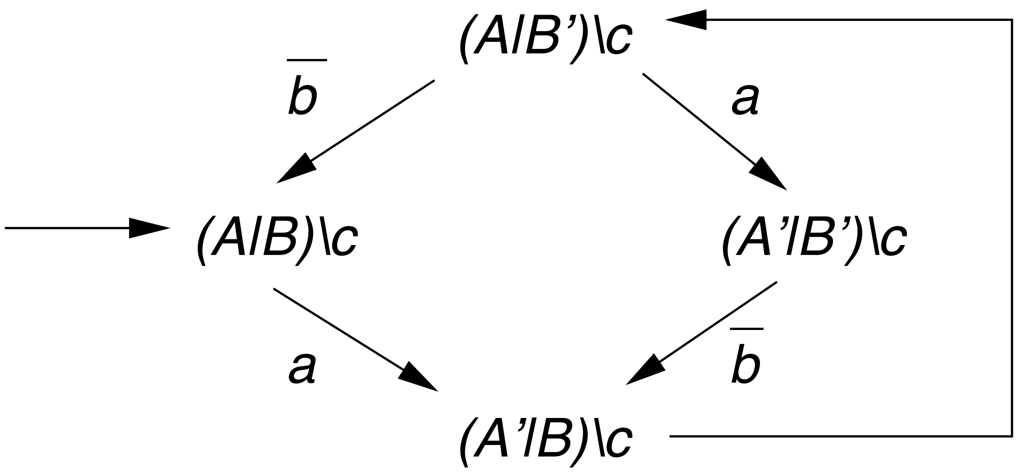
From *infinite* derivation trees ...

$$\boxed{\begin{array}{l} -\ A := a.A',\ A' := \overline{c}.A \\ -\ B := c.B',\ B' := \overline{b}.B \end{array}}$$

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

... to *finite* transition graphs

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



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

No "|"

Behavior can be defined by + and . only!

Internal versus external actions

- Action τ:
  - Simultaneous action of both agents.
  - *Internal* to composed agent.
- Internal actions should be ignored.
  - Only external actions are visible.
  - Two systems are *observationally equivalent* if they exhibit same pattern of external actions.
  - $P \xrightarrow{\tau} P_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_n$ o-equivalent to $P \xrightarrow{\tau} P_n$
  - $\alpha.\tau.P$ o-equivalent to $\alpha.P$

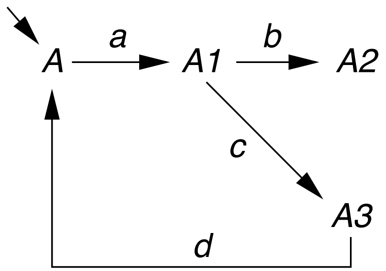
Internal actions take no "time".

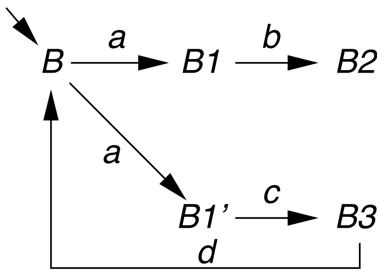
- Simpler variant of $(A|B)\backslash c$:
  - $(A|B)\backslash c$ o-equivalent to $a.D$
  - $D := a.\overline{b}.D + \overline{b}.a.D$

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

In need of bisimulation

- Example agents A and B
  - $A = a.(b.0 + c.d.A)$
  - $B = a.b.0 + a.c.d.B$





- "Language understood" by A and B
  - $(a.c.d)^*.a.b.0$
  - A and B seem equivalent.
- Ports a, b, c, d.
  - Initially only a is "unlocked".
  - Observer "presses button" a.
  - In A, b and c are "unlocked".
  - In B, sometimes b, sometimes c is "unlocked".
  - A and B can be experimentally distinguished!

Think of repeatedly "replaying" the system from the state that was obtained by pressing *a*.

Bisimulation (very informally)

- Two agent expressions P, Q are bisimular:
  - If P can do an α action towards P',
  - then Q can do an α action towards Q',
  - such that P' and Q' are again bisimular,
  - and v.v.

Intuitively two systems are bisimilar if they match each other's moves. In this sense, each of the systems cannot be distinguished from the other by an observer. [Wikipedia]

Laws

Summation laws

- $P + Q = Q + P$
- $P + (Q + R) = (P + Q) + R$
- $P + P = P$
- $P + 0 = P$

- Composition laws
  - $P|Q = Q|P$
  - $P|(Q|R) = (P|Q)|R$
  - $P|0 = P$
- Restriction laws
  - $P \backslash L = P$, if $L(P) \cap (L \cup \overline{L}) = \emptyset$.
  - $P \backslash K \backslash L = P \backslash (K \cup L)$
  - $\ldots$
- Relabelling laws
  - $P[\mathit{Id}] = P$
  - $P[f][f'] = P[f' \circ f]$
  - $\ldots$

Non-laws

- $\tau.P = P$
  - $A = a.A + \tau.b.A$
  - $A' = a.A' + b.A'$
  - A may switch to state in which only b is possible.
  - A' *always* allows a or b.
- $\alpha.(P + Q) = \alpha.P + \alpha.Q$
  - $a.(b.P + c.Q) = a.b.P + a.c.Q$
  - $b.P$ is a-derivative of right side, not capable of c action.
  - a-derivative of left side is capable of c action!
  - Action sequence a, c may yield deadlock for right side.

- **Summary**: *CCS*
  - *An algebraic approach to system modeling.*
  - *Approach amenable to formal analysis.*
  - *Equivalence is based on communication behavior.*
- **Prepping**: *Read CCS tutorial* [AcetoLI05]
- **Lab**: *Model CCS in Prolog*
- **Outlook**:
  - Ending of Prolog section
  - Beginning of Haskell section
  - Midterm