The Calculus of Communicating Systems

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, A-4040 Linz, Austria

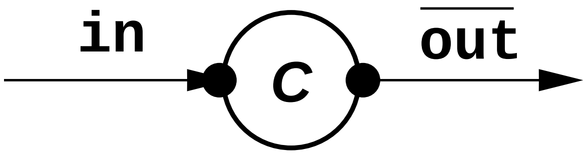
Wolfgang.Schreiner@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/schreine

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.

A Simple Example



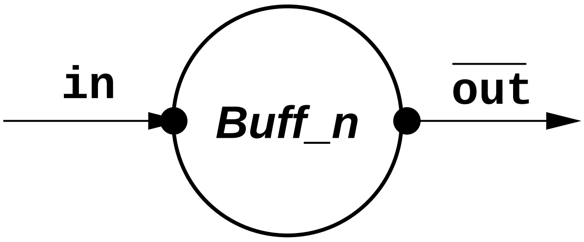
- *Agent* C
  - Dynamic system is network of *agents*.
  - Each agent has own identity persisting over time.
  - Agent performs *actions* (external communications or internal actions).
  - *Behavior* of a system is its (observable) capability of communication.
- Agent has labeled *ports*.
  - Input port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

Process behaviors are defined by (mutually recursive) equations.

Behavior Descriptions

- 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.

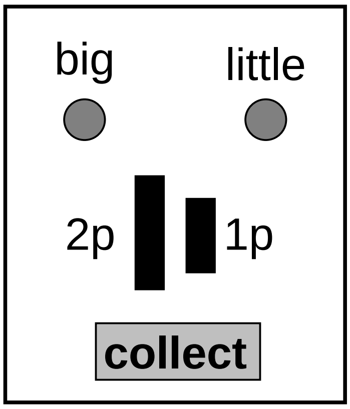
Another Example



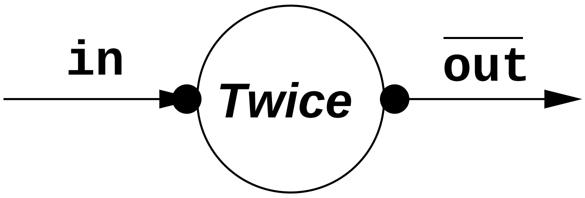
- Bounded buffer $\mathit{Buff}_n(s)$
  - $\mathit{Buff}_n \langle\ \rangle := \mathtt{in}(x).\mathit{Buff}_n \langle x \rangle$
  - $\mathit{Buff}_n \langle v_1, \ldots, v_n \rangle :=$
    $\overline{\mathtt{out}}(v_n).\mathit{Buff}_n \langle v_1, \ldots, v_{n-1} \rangle$
  - $\mathit{Buff}_n \langle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

Further Examples

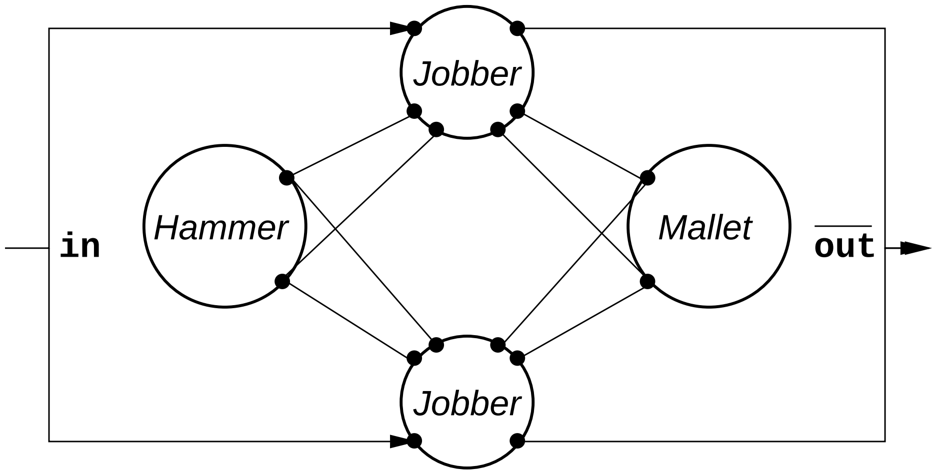


- A vending machine:
  - Big chocolade costs 2p, small one costs 1p.
  - V := $\texttt{2p.big.collect.}V$
    $+ \texttt{1p.little.collect.}V$



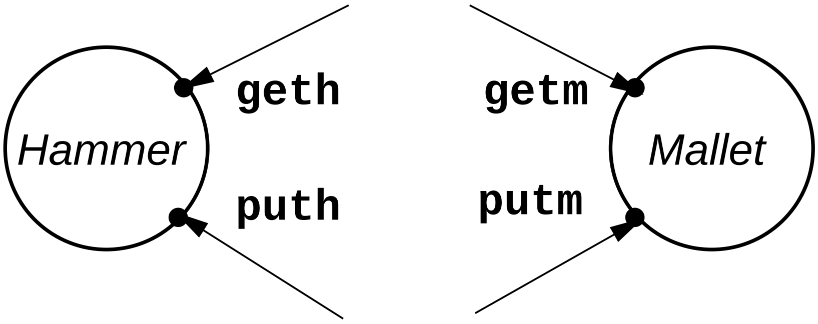
- A multiplier
  - *Twice* := $\texttt{in}(x).\overline{\texttt{out}}(2 * x).$*Twice*.
  - Output actions may take expressions.

A Larger Example: The Jobshop



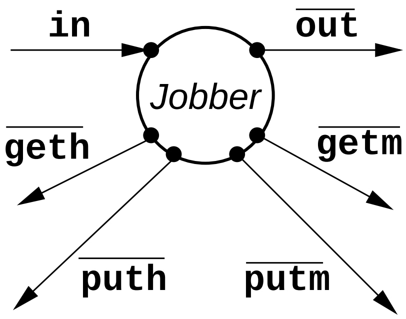
- A simple production line:
  - Two people (the *jobbers*).
  - Two tools (hammer and mallet).
  - *Jobs* arrive sequentially on a belt to be processed.
- Ports may be linked to multiple ports.
  - Jobbers compete for use of hammer.
  - Jobbers compete for use of job.
  - Source of non-determinism.
- 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



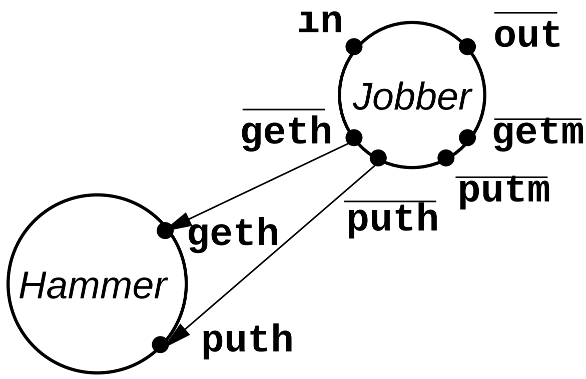
- Behaviors:
  - *Hammer* := $\texttt{geth}$.*Busyhammer*
    Busyhammer := $\texttt{puth}$.*Hammer*
  - *Mallet* := $\texttt{geth}$.*Busymallet*
    Busymallet := $\texttt{puth}$.*Mallet*
- *Sort* = set of labels
  - $P : L$ . . . agent P has sort L
  - *Hammer*: $\{\texttt{geth}, \texttt{puth}\}$
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

The Jobbers



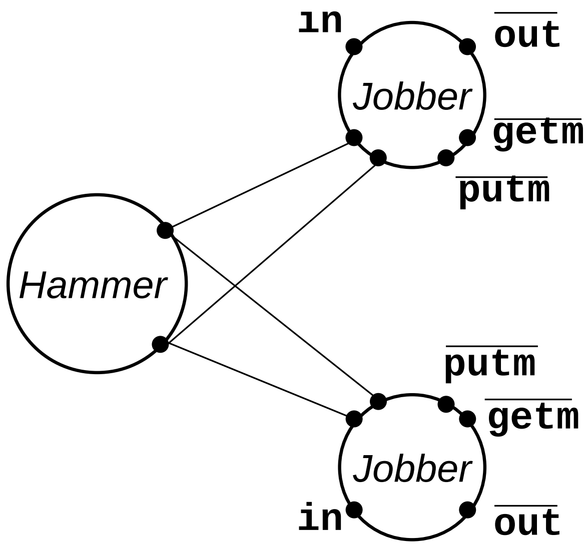
- Different kinds of jobs:
  - Easy jobs done with hands.
  - Hard jobs done with hammer.
  - Other jobs done with hammer or mallet.
- Behavior:
  - $\mathit{Jobber} := \texttt{in}(\mathit{job}).\mathit{Start}(\mathit{job})$
  - $\mathit{Start}(\mathit{job}) :=$ **if** $\mathit{easy}(\mathit{job})$ **then**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

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.

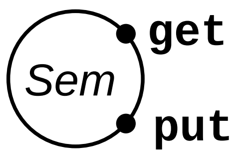
Further Compositon



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

Reformulations

- Alternative formulation:
  - ((*Jobber* | *Jobber* | *Hammer*)\{geth, puth}
    | *Mallet*)\{getm, 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
  - *Sem* := get.put.*Sem*
- Reformulation of tools
  - *Hammer* := *Sem*[geth/get, puth/put]
  - *Mallet* := *Sem*[getm/get, putm/put]

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 between agents
  - Just synchronization.
- Agent expressions
  - Agent constants and variables
  - *Prefix* $\alpha.E$
  - *Summation* ΣE_i
  - *Composition* $E_1|E_2$
  - *Restriction* $E\backslash L$
  - *Relabelling* $E[f]$
- Names and co-names
  - Set A of *names* (`geth`, `ackin`, ...)
  - Set $\underline{A}$ of *co-names* ($\overline{\texttt{geth}}$, $\overline{\texttt{ackin}}$, ...)
  - 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* $\stackrel{\texttt{geth}}{\rightarrow}$ *Busyhammer*

The Transition Rules

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

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)

Derivatives and Derivation Trees

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

$$((a.E + b.0)|\ \overline{a}.F)\backslash a \;\overset{\tau}{\nearrow}\; (E|F)\backslash a$$

$$((a.E + b.0)|\ \overline{a}.F)\backslash a \;\overset{b}{\searrow}\; (0|\ \overline{a}.F)\backslash a$$

- $a.X + b.Y$

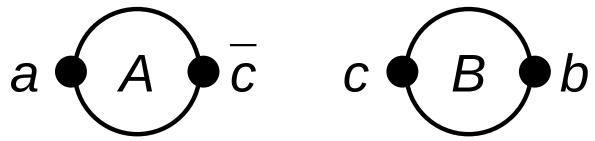
$$a.X + b.Y \;\overset{a}{\nearrow}\; X$$

$$a.X + b.Y \;\overset{b}{\searrow}\; Y$$

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

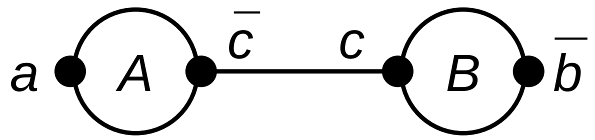
Transitions

- Agents A and B



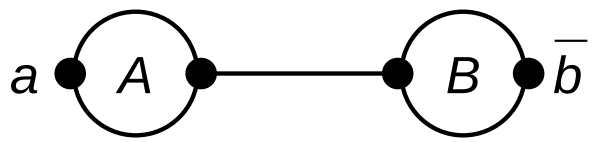
  - $A := a.A'$, $A' := \overline{c}.A$
  - $B := c.B'$, $B' := \overline{b}.B$

- 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 $A'|B \stackrel{\tau}{\to} A|B'$

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



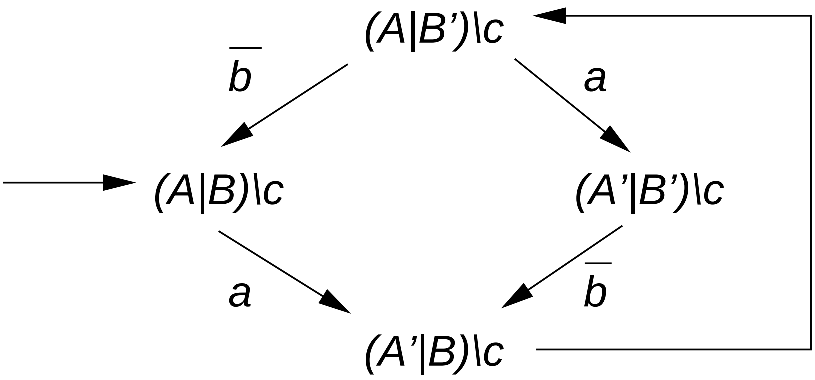
  - $P \stackrel{\alpha}{\to} P'$ allows $P\backslash L \stackrel{\alpha}{\to} P'\backslash L$
    (if α, $\overline{\alpha}$ not in L)

Transition Trees and Graphs

- *Transition (derivation) tree*

$$(A|B)\backslash c$$
$$\downarrow a$$
$$(A'|B)\backslash c$$
$$\downarrow \tau$$
$$(A|B')\backslash c$$
$$\swarrow \overline{b} \qquad\qquad \searrow a$$
$$(A|B)\backslash c \qquad\qquad (A'|B')\backslash c$$
$$\downarrow a \qquad\qquad \downarrow \overline{b}$$
$$(A'|B)\backslash c \qquad\qquad (A'|B)\backslash c$$
$$\ldots \qquad\qquad \ldots$$

- *Transition graph*



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

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

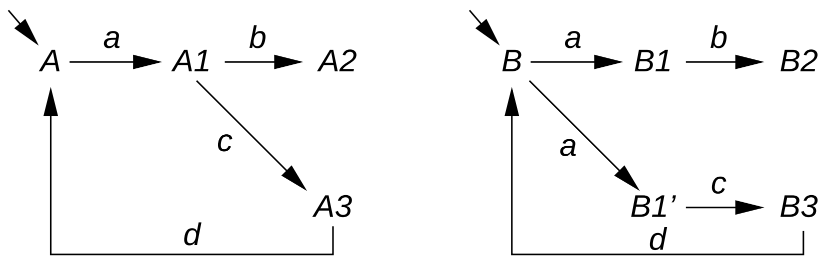
Equality of Agents

- Equality:
  - Two agents P and Q should be considered equal if and only if no distinction can be detected by external agent interacting with them.
- *Strong* (behavioral) equivalence $\sim$:
  - τ is treated like any other (observable) action.
  - Too strong to be considered as equality.
- *Weak* (observation) equivalence $\approx$:
  - τ cannot be observed by external agent.
  - Not a congruence relation, thus not suitable as equality.
- Observation *congruence* $=$:
  - *Congruence relation*, i.e. preserved by all contexts.
  - Suitable notion for process equality.
- Relations:
  - $P{\sim}Q$ implies $P{=}Q$ implies $P{\approx}Q$

Observation congruence is the equality of the process algebra.

Languages of Agents

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

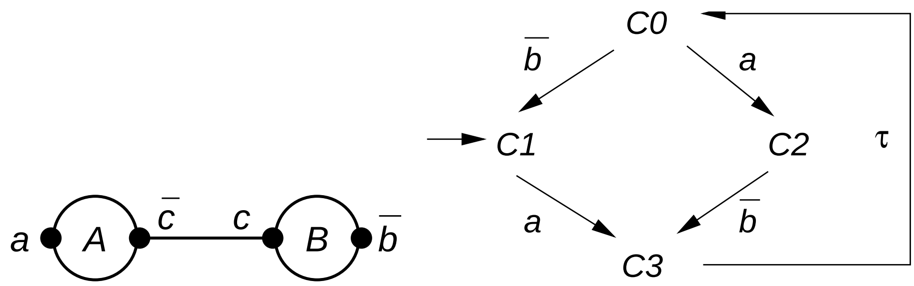


- "Language understood" by A and B
  - $(a.c.d)^*.a.b.0$
  - A and B seem equivalent.
- Ports a, b, c, d.
  - Initially only a is "unlocked".
  - Observer "presses button" a.
  - In A, b and c are "unlocked".
  - In B, sometimes b, sometimes c is "unlocked".
  - A and B can be experimentally distinguished!

Even agents with the same language can be experimentally distinguished.

Strong Bisimulation

- *Strong bisimulation*
  - Binary relation S over agents such that $(P, Q) \in S$ implies
  - If $P \stackrel{\alpha}{\rightarrow} P'$, then $Q \stackrel{\alpha}{\rightarrow} Q'$ with $(P', Q') \in S$ and vice versa.
  - For every action α, every α-derivative of P is equivalent to some α-derivative of Q.
- Example



  - Claim: $(A|B)\backslash c = C_1$
  - True if S is a strong bisimulation:
    $S = \{$ $((A|B)\backslash c,\ C_1)$, $((A'|B)\backslash c,\ C_3)$,
    $((A|B')\backslash c,\ C_0)$, $((A'|B')\backslash c,\ C_2)$ $\}$
  - Check derivatives of each of the eight agents.

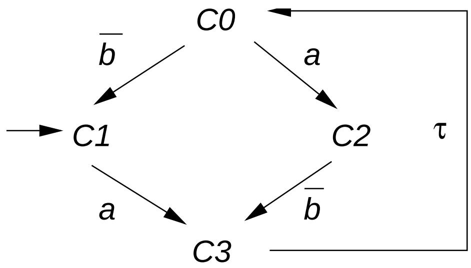
Strong Equivalence

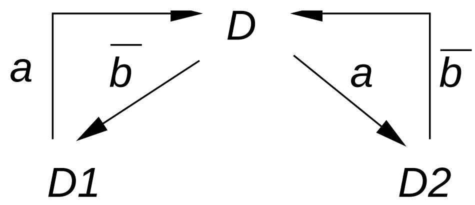
- *Strong equivalence* $P{\sim}Q$
  - $P{\sim}Q$, if $(P,Q) \in S$ for some strong bisimulation S.
  - $\sim = \cup\{S$: S is a strong bisimulation$\}$.
- Corollaries:
  - $\sim$ is the largest strong bisimulation.
  - $\sim$ is an equivalence relation.
- Proposition:
  - $P{\sim}Q$ iff, for all α,
  - If $P \xrightarrow{\alpha} P'$, then $Q \xrightarrow{\alpha} Q'$ with $(P',Q') \in S$ and vice versa.
- Strong equivalence is a congruence.
  - Substitutive under all combinators and recursive definitions.
- Let $P_1 \sim P_2$
  - $\alpha.P_1 \sim \alpha.P_2$
  - $P_1 + Q \sim P_2 + Q$
  - $P_1|Q \sim P_2|Q$
  - $P_1 \backslash L \sim P_2 \backslash L$
  - $P_1[f] \sim P_2[f]$

Observation Equivalence

- (Observation) equivalence:
  - τ action may be matched by zero or more τ actions.
- Auxiliary definitions:
  - $\hat{t}$ is the action sequence gained by deleting all occurences of τ from t.
  - $E \stackrel{t}{\rightarrow} E'$, if $t = \alpha_1 \ldots \alpha_n$ and $E \stackrel{\alpha_1}{\rightarrow} \ldots \stackrel{\alpha_n}{\rightarrow} E'$.
  - $E \stackrel{t}{\Rightarrow} E'$ if $t = \alpha_1 \ldots \alpha_n$ and $E (\stackrel{\tau}{\rightarrow})^* \stackrel{\alpha_1}{\rightarrow} (\stackrel{\tau}{\rightarrow})^* \ldots (\stackrel{\tau}{\rightarrow})^* \stackrel{\alpha_n}{\rightarrow} (\stackrel{\tau}{\rightarrow})^* E'$.
  - E' is a t-descendant of E iff $E \stackrel{\hat{t}}{\Rightarrow} E'$.
- Relationship
  - $P \stackrel{t}{\rightarrow} P'$ implies $P \stackrel{t}{\Rightarrow} P'$ implies $P \stackrel{\hat{t}}{\Rightarrow} P'$
- *(Weak) bisimulation*
  - Binary relation S such that $(P, Q) \in S$ implies
  - if $P \stackrel{\alpha}{\rightarrow} P'$, then $Q \stackrel{\widehat{\alpha}}{\Rightarrow} Q'$ with $(P', Q') \in S$ (and vice versa).
- *Observation equivalence* $P \approx Q$
  - $P \approx Q$ if $(P, Q) \in S$ for some weak bisimulation S.
  - $\approx = \cup \{ S : S \text{ is a weak bisimulation} \}$

Examples





- Agents C_0 and D
  - Bisimulation $S = \{(C_0, D), (C_1, D_1), (C_2, D_2), (C_3, D)\}$
  - No *strong* bisimulation containing (C_3, D) since $C_3 \xrightarrow{\tau} C_0$ but there is no $D \xrightarrow{\tau} D'$.
- Agents A and B
  - $A_0 = a.A_0 + b.A_1 + \tau.A_1$
    $A_1 = a.A_1 + \tau.A_2$
    $A_2 = b.A_0$
  - $B_1 = a.B_1 + \tau.B_2$
    $B_2 = b.B_1$
  - Bisimulation $S = \{ (A_0, B_1), (A_1, B_1), (A_2, B_2) \}$ (note that $B_1 \overset{b}{\Rightarrow} B_1$!)

Properties of Bisimulation

- Propositions:
  - $\approx$ is the largest bisimulation.
  - $\approx$ is an equivalence relation.
  - $P \approx \tau.P$
- $\approx$ is *not* a congruence:
  - $\approx$ not preserved by summation.
  - $a.0 + b.0 \approx a.0{+}\tau.b.0$ does *not* hold!
  - Proof: if (P,Q) were in a bisimulation S, then, since $Q \xrightarrow{\tau} b.0$, we need $(P', b.0)$ in S with $P \overset{\epsilon}{\Rightarrow} P'$. But the only P' is P itself but $(P, b.0)$ can be not in S, since $P \xrightarrow{a}$ 0, while $b.0$ has no a-descendant.

Equality not yet fully captured.

Observation Congruence

- $P = Q$ (*observation congruence*)
  - If $P \stackrel{\alpha}{\rightarrow} P'$, then $Q \stackrel{\alpha}{\Rightarrow} Q'$ with $P' \approx Q'$ (and vice versa).
  - Preserved under all process operators.
- Relationship to observation equivalence:
  - P is *stable* if P has no τ-derivative.
  - If $P \approx Q$ and both are stable, then $P = Q$.
  - If $P \approx Q$ then $\alpha.P = \alpha.Q$

Observation congruence is the equality of the process algebra.

Equational Laws

- Static laws
  - Static combinators: composition, restriction, labelling.
  - Action rules do not change graph structure.
  - Algebra of flow graphs.
- Dynamic laws
  - Dynamic combinators: prefix, summation, constants.
  - Action rules change graph structure.
  - Algebra of transition graphs.
- Expansion law
  - Relating static laws to dynamic laws.

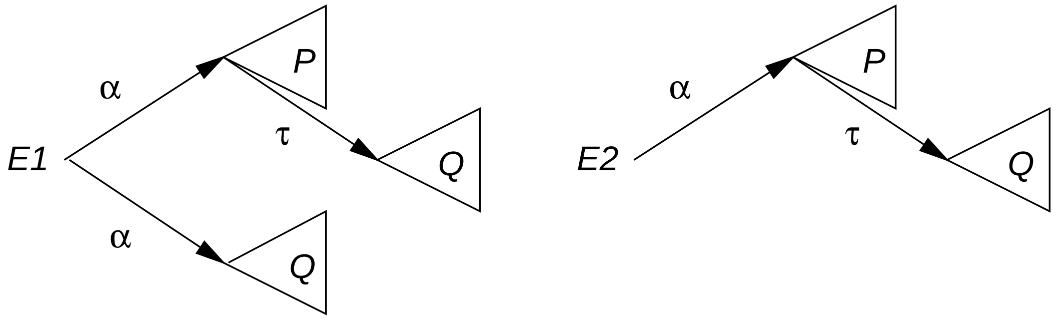
Laws for equality reasoning on processes.

Static Laws

- 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
  - $P[\mathit{Id}] = P$
  - $P[f][f'] = P[f' \circ f]$
  - $\ldots$

Dynamic Laws

- Monoid laws
  - $P + Q = Q + P$
  - $P + (Q + R) = (P + Q) + R$
  - $P + P = P$
  - $P + 0 = P$
- τ laws
  - $\alpha.\tau.P = \alpha.P$
  - $P + \tau.P = \tau.P$
  - $\alpha.(P + \tau.Q) + \alpha.Q = \alpha.(P + \tau.Q)$



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.

The Expansion Law

- The Expansion Law
  - Let $P \equiv\ (P_1[f_1]|\ldots|P_n[f_n])\backslash L$
  - $P = \Sigma\{f_1(\alpha).(P_1[f_1]|\ldots|P_i'[f_i]|\ldots|P_n[f_n])\backslash L$: $P_i \xrightarrow{\alpha} P_i'$, $f_i(\alpha)$ not in $L \cup \overline{L}\}$ $+\Sigma\{\tau.(P_1[f_1]|\ldots|P_i'[f_i]|\ldots|P_j'[f_j]|\ldots|P_n[f_n])\backslash L$: $P_i \xrightarrow{l_1} P_i'$, $P_j \xrightarrow{l_2} P_j'$, $f_i(l_1) = \overline{f_i(l_2)}, i < j\}$
- Corollary
  - Let $P \equiv\ (P_1|\ldots|P_n)\backslash L$
  - $P = \Sigma\{\alpha.(P_1|\ldots|P_i'|\ldots|P_n)\backslash L$: $P_i \xrightarrow{\alpha} P_i'$, α not in $L \cup L'\}$ $+\Sigma\{\tau.(P_1|\ldots|P_i'|\ldots|P_j'|\ldots P_n)\backslash L$: $P_i \xrightarrow{l} P_i'$, $P_j \xrightarrow{\overline{l}} P_j'$, $i < j\}$
- Example
  - $P_1 = a.P_1' + b.P_1''$
  - $P_2 = \overline{a}.P_2' + c.P_2''$
  - $(P_1|P_2)\backslash a = b.(P_1''|P_2)\backslash a + c.(P_1|P_2'')\backslash a + \tau.(P_1'|P_2')\backslash a$

Summary

- Algebraic approach to system modeling.
  - Main interest: how do processes interact with each other?
  - Processes/specifications are described by terms.
  - Calculus describes process reactions by term manipulation.
- Central notions:
  - Strong bisimilarity: equivalence even for internal actions.
  - Observation equivalence: equivalence only for observable actions.
  - Observation congruence: observation equivalence preserved under all substitutions.

An implementation must "equal" (be observationally congruent to) its specification.