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 ($\pi$-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 $\text{in}$.
  - Output port $\text{out}$.

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

*Process behaviors are defined by (mutually recursive) equations.*
Behavior Descriptions

• Agent names can take parameters.

• Prefix $\text{in}(x)$
  
  – Handshake in which value is received at port $\text{in}$ and becomes the value of variable $x$.

• Agent expression $\text{in}(x).C'(x)$
  
  – Perform handshake and proceed as described by $C'$.

• Agent expression $\text{out}(x).C$
  
  – Output the value of $x$ at port $\text{out}$ and proceed according to the definition of $C$.

• Scope of local variables:
  
  – $\text{Input}$ prefix introduces variable whose scope is the agent expression $C$.
  
  – Formal parameter of defining equation introduces variable whose scope is the equation.
Another Example

\[\text{Buff}_n(s)\]

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

- 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 nondeterministic.

- Combining forms
  - Summation \(P + Q\) of two agents.
  - Sequencing \(\alpha.P\) of action \(\alpha\) and agent \(P\).

Process definitions may be parameterized.
Further 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
  – $Twice := \text{in}(x). \text{out}(2 \times x). Twice.$
  – 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.
  - $\text{in}$ and $\text{out}$ are external.

- Internal ports are not labelled:
  - Ports by which jobbers acquire and release tools.
The Tools

• Behaviors:
  
  – \textit{Hammer} := \text{geth}.\text{Busyhammer}
  \quad \text{Busyhammer} := \text{pth}.\text{Hammer}
  
  – \textit{Mallet} := \text{geth}.\text{Busymallet}
  \quad \text{Busymallet} := \text{pth}.\text{Mallet}

• \textit{Sort} = \text{set of labels}
  
  – \textit{P} : \text{L} \ldots \text{agent P has sort L}
  
  – \textit{Hammer}: \{\text{geth, puth}\}
  \quad \textit{Mallet}: \{\text{getm, putm}\}
  
  \quad \textit{Jobshop}: \{\text{in, 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:
  - \( \text{Jobber} := \text{in}(\text{job}).\text{Start}(\text{job}) \)
  - \( \text{Start}(\text{job}) := \text{if } \text{easy}(\text{job}) \text{ then } \text{Finish}(\text{job}) \)
  - \( \text{else if } \text{hard}(\text{job}) \text{ then } \text{Uhammer}(\text{job}) \)
  - \( \text{else } \text{Usetool}(\text{job}) \)
  - \( \text{Usetool}(\text{job}) := \text{Uhammer}(\text{job}) + \text{Umallet}(\text{job}) \)
  - \( \text{Uhammer}(\text{job}) := \text{geth}.\text{puth}.\text{Finish}(\text{job}) \)
  - \( \text{Umallet}(\text{job}) := \text{getm}.\text{putm}.\text{Finish}(\text{job}) \)
  - \( \text{Finish}(\text{job}) := \text{out}(\text{done}(\text{job})).\text{Jobber} \)
Composition of Agents

- **Jobber-Hammer** subsystem
  - Jobber | Hammer
  - Composition operator |
  - Agents may proceed independently or interact through complementary ports.
  - Join complementary ports.

- **Two jobbers sharing hammer:**
  - Jobber | Hammer | Jobber
  - Composition is commutative and associative.
Further Compositon

- \textit{Internalisation} of ports:
  - No further agents may be connected to ports:
  - \textit{Restriction} operator $\backslash$
  - $\backslash L$ internalizes all ports $L$.
  - $(\text{Jobber} | \text{Jobber} | \text{Hammer}) \backslash \{\text{geth}, \text{puth}\}$

- \textbf{Complete system}:
  - $\text{Jobshop} := (\text{Jobber} | \text{Jobber} | \text{Hammer} | \text{Mallet}) \backslash L$
  - $L := \{\text{geth}, \text{puth}, \text{getm}, \text{putm}\}$
Reformulations

• Alternative formulation:
  – \(((\text{Jobber} \mid \text{Jobber} \mid \text{Hammer})\setminus\{\text{geth, puth}\}
    \mid \text{Mallet})\setminus\{\text{getm, putm}\}\)
  – Algebra of combinators with certain laws of equivalence.

• Relabelling Operator
  – \(P[l'_1/l_1, \ldots, l'_n/l_n]\)
  – \(f(l) = \overline{f(l)}\)

• Semaphore agent
  – \(\text{Sem} := \text{get.put.Sem}\)

• Reformulation of tools
  – \(\text{Hammer} := \text{Sem}[\text{geth}/\text{get}, \text{puth}/\text{put}]\)
  – \(\text{Mallet} := \text{Sem}[\text{getm}/\text{get}, \text{putm}/\text{put}]\)
Equality of Agents

• *Strongjobber* only needs hands:
  – \( \text{Strongjobber} := \text{in}(\text{job}).\overline{\text{out}}(\text{done}(\text{job})).\text{Strongjobber} \)

• Claim:
  – \( \text{Jobshop} = \text{Strongjobber} \mid \text{Strongjobber} \)
  – Specification of system \( \text{Jobshop} \)
  – 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 \( \sum E_i \)
  - Composition \( E_1|E_2 \)
  - Restriction \( E\setminus L \)
  - Relabelling \( E[f] \)

- Names and co-names
  - Set \( A \) of names (geth, ackin, \ldots)
  - Set \( \overline{A} \) of co-names (get\texth, ack\textin, \ldots)
  - Set of labels \( L = A \cup \overline{A} \)

- Actions
  - Completed (perfect) action \( \tau \).
  - Act = \( L \cup \{\tau\} \)

- Transition \( P \xrightarrow{l} Q \) with action \( l \)
  - Hammer \( \text{geth} \rightarrow \) Busyhammer
The Transition Rules

- **Act** \( \alpha.E \xrightarrow{\alpha} E \)
- **Sum** \( \sum \frac{E_j \xrightarrow{\alpha} E'_j}{E \xrightarrow{\alpha} E'} \)
- **Com\(_1\)** \( \frac{E \xrightarrow{\alpha} E'}{E|F \xrightarrow{\alpha} E'|F} \)
- **Com\(_2\)** \( \frac{F \xrightarrow{\alpha} F'}{E|F \xrightarrow{\alpha} E|F'} \)
- **Com\(_3\)** \( \frac{E \xrightarrow{l} E' \quad F \xrightarrow{r} F'}{E|F \xrightarrow{\tau} E'|F'} \)
- **Res** \( \frac{E \xrightarrow{\alpha} E'}{E \backslash L \xrightarrow{\alpha} E' \backslash L} \) \( (\alpha, \overline{\alpha} \text{ not in } L) \)
- **Rel** \( \frac{E \xrightarrow{\alpha} E'}{E[f] \xrightarrow{f(\alpha)} E'[f]} \)
- **Con** \( \frac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'} \) \( (A := P) \)
The Value-Passing Calculus

• Values passed between agents
  – Can be reduced to basic calculus.
  – \( C' := \text{in}(x).C''(x) \)
    \( C''(x) := \text{out}(x).C \)
  – \( C' := \sum_v \text{in}_v.C'_v \)
    \( C'_v := \text{out}_v.C \ (v \in V) \)
  – Families of ports and agents.

• The full language
  – Prefixes \( a(x).E, \overline{a}(e).E, \tau.E \)
  – Conditional if \( b \) then \( E \)

• Translation
  – \( a(x).E \Rightarrow \sum_v.E\{v/x\} \)
  – \( \overline{a}(e).E \Rightarrow \overline{a}_e.E \)
  – \( \tau.E \Rightarrow \tau.E \)
  – if \( b \) then \( E \Rightarrow (E, \text{if } b \text{ and } 0, \text{otherwise}) \)
Derivatives and Derivation Trees

- **Immediate derivative of** $E$
  - Pair $(\alpha, E')$
  - $E \xrightarrow{\alpha} E'$
  - $E'$ is $\alpha$-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** $E$

```
       E_{11} \ldots
        / \       \
       / \       \
   E_1  / \  \alpha_2 \xrightarrow{\alpha_1} \xrightarrow{\alpha_12} E_{12} \ldots
        \   \   \   \    \
       \   \   \   \    \
     E  \   \   \   \    \
        \   \   \   \    \
       \xrightarrow{\alpha_1} \xrightarrow{\alpha_2} E_2 \ldots
```

Wolfgang Schreiner
Examples of Derivation Trees

• Partial derivation tree

\[
\begin{align*}
(E|F) \setminus a \\
\tau \downarrow & \\
((a.E + b.0)| \overline{a}.F) \setminus a \\
\downarrow b & \\
(0| \overline{a}.F) \setminus a
\end{align*}
\]

• \(a.X + b.Y\)

\[
\begin{align*}
X \\
\downarrow a \\
a.X + b.Y \\
\downarrow b \\
Y
\end{align*}
\]

• Behavioural equivalence
  
  Two agent expressions are \emph{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 $A|B$

$A \xrightarrow{a} A'$ allows $A|B \xrightarrow{a} A'|B$

$A' \xrightarrow{c} A$ allows $A'|B \xrightarrow{c} A|B$

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

• Restriction $(A|B)\setminus c$

$P \xrightarrow{\alpha} P'$ allows $P\setminus L \xrightarrow{\alpha} P'\setminus L$

(if $\alpha, \overline{\alpha}$ not in $L$)
Transition Trees and Graphs

- **Transition (derivation) tree**

\[
\begin{align*}
(A|B)\backslash c \\
\downarrow a \\
(A'|B)\backslash c \\
\downarrow \tau \\
(A|B')\backslash c \\
\swarrow b \\
(A|B)\backslash c \\
\downarrow a \\
(A'|B)\backslash c \\
\ldots
\end{align*}
\]

- **Transition graph**

\[
\begin{array}{c}
\stackrel{a}{\longrightarrow} (A|B)\backslash c \\
\stackrel{b}{\longrightarrow} (A'|B)\backslash c \\
\stackrel{a}{ightleftharpoons} (A'|B')\backslash c \\
\stackrel{b}{\longrightarrow} (A|B)\backslash c
\end{array}
\]

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

*Behavior can be defined by + and \cdot only!*
Internal versus External Actions

- **Action \( \tau \):**
  - 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 \( a.D \)
  - \( D := a.b.D + \overline{b}.a.D \)
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 $\sim$:
  – $\tau$ is treated like any other (observable) action.
  – Too strong to be considered as equality.

• **Weak** (observation) equivalence $\approx$:
  – $\tau$ 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 $B$
  
  \[- A = a.(b.0 + c.d.A) \]
  \[- B = a.b.0 + a.c.d.B \]

  ![Diagram]

• “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 $P \xrightarrow{\alpha} P'$, then $Q \xrightarrow{\alpha} Q'$ with $(P', Q') \in S$ and vice versa.
  - For every action $\alpha$, every $\alpha$-derivative of $P$ is equivalent to some $\alpha$-derivative of $Q$.

- **Example**

\[
\begin{array}{c}
\text{a} \rightarrow A \rightarrow \overline{c} \rightarrow c \rightarrow B \rightarrow \overline{b} \\
\end{array}
\]

- Claim: $((|B)c, C_1)

- True if $S$ is a strong bisimulation:
  
  $S = \{ (|B)c, C_1), (|B)c, C_3), (|B)c, C_0), (|B)c, C_2) \}$

- 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 $S$.
  - $\sim = \bigcup\{S: S$ is a strong bisimulation\}.

- **Corollaries:**
  - $\sim$ is the largest strong bisimulation.
  - $\sim$ is an equivalence relation.

- **Proposition:**
  - $P \sim Q$ iff, for all $\alpha$,
    - 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** $P_1 \sim P_2$
  - $\alpha.P_1 \sim \alpha.P_2$
  - $P_1 + Q \sim P_2 + Q$
  - $P_1|Q \sim P_2|Q$
  - $P_1\setminus L \sim P_2\setminus L$
  - $P_1[f] \sim P_2[f]$
Observation Equivalence

• (Observation) equivalence:
  – $\tau$ action may be matched by zero or more $\tau$ actions.

• Auxiliary definitions:
  – $\hat{t}$ is the action sequence gained by deleting all occurrences of $\tau$ from $t$.
  – $E \xrightarrow{t} E'$, if $t = \alpha_1 \ldots \alpha_n$ and $E \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} E'$.
  – $E \xrightarrow{\hat{t}} E'$ if $t = \alpha_1 \ldots \alpha_n$ and $E(\tau \rightarrow)^\ast \xrightarrow{\alpha_1} \ldots (\tau \rightarrow)^\ast \xrightarrow{\alpha_n} \ldots (\tau \rightarrow)^\ast E'$.
  – $E'$ is a $t$-descendant of $E$ iff $E \xrightarrow{\hat{t}} E'$.

• Relationship
  – $P \xrightarrow{t} P'$ implies $P \xrightarrow{\hat{t}} P'$ implies $P \xrightarrow{\hat{t}} P'$

• (Weak) bisimulation
  – Binary relation $S$ such that $(P, Q) \in S$ implies
  – if $P \xrightarrow{\alpha} P'$, then $Q \xrightarrow{\hat{\alpha}} Q'$ 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 $S$.
  – $\approx = \cup \{ S : S$ 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 $B_1 \xrightarrow{b} B_1$!)
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 $P \xrightarrow{\tau} P'$. 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 $P \xrightarrow{\alpha} P'$, then $Q \xrightarrow{\alpha} Q'$ with $P' \approx Q'$ (and vice versa).
  – Preserved under all process operators.

• Relationship to observation equivalence:
  – $P$ is stable if $P$ has no $\tau$-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
  - $P\setminus L = P$, if $L(P) \cap (L \cup \overline{L}) = \emptyset$.
  - $P\setminus K\setminus L = P\setminus (K \cup L)$
  - ...

• Relabelling laws
  - $P[Id] = P$
  - $P[f][f'] = P[f' \circ f]$
  - ...

Dynamic Laws

• Monoid laws
  - $P + Q = Q + P$
  - $P + (Q + R) = (P + Q) + R$
  - $P + P = P$
  - $P + 0 = P$

• $\tau$ 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 \( P \equiv (P_1[f_1]|\ldots|P_n[f_n]) \setminus L \)
  - \( P = \sum \{ f_1(\alpha). (P_1[f_1]|\ldots|P'_i[f_i]|\ldots|P_n[f_n]) \setminus L : P_i \xrightarrow{\alpha} P'_i, f_i(\alpha) \) not in \( L \cup L' \}
    + \sum \{ \tau. (P_1[f_1]|\ldots|P'_i[f_i]|\ldots|P'_j[f_j]|\ldots|P_n[f_n]) \setminus L : P_i \xrightarrow{l_1} P'_i, P_j \xrightarrow{l_2} P'_j, f_i(l_1) = f_i(l_2), i < j \}

- Corollary
  - Let \( P \equiv (P_1|\ldots|P_n) \setminus L \)
  - \( P = \sum \{ \alpha. (P_1|\ldots|P'_i|\ldots|P_n) \setminus L : P_i \xrightarrow{\alpha} P'_i, \alpha \) not in \( L \cup L' \}
    + \sum \{ \tau. (P_1|\ldots|P'_i|\ldots|P'_j|\ldots P_n) \setminus L : P_i \xrightarrow{l_1} P'_i, P_j \xrightarrow{l_2} P'_j, i < j \}

- Example
  - \( P_1 = a. P_1' + b. P_1'' \)
  - \( P_2 = \overline{a}. P_2' + c. P_2'' \)
  - \( (P_1|P_2) \setminus a = b. (P_1''|P_2) \setminus a + c. (P_1|P_2'') \setminus a + \tau. (P_1'|P_2') \setminus a \)
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.