Theory of concurrency

Lecture 6

Processes

Processes: **Specifications**

- A *specification* of a product is a description of the way it is intended to behave.
- For processes, the most relevant observation of behaviour is
  - *the trace of events* that occur up to a given moment in time.
- The special variable tr is used for an arbitrary trace of a process.

Processes: **Specifications**

X1. The *owner* of a vending machine does not wish to make a loss by installing it.

- The number of chocolates dispensed must never exceed the number of coins inserted

$$NOLOSS = (\#(tr \restriction \{choc\}) \leq \#(tr \restriction \{coin\}))$$

- The abbreviation for the number of occurrences of the symbol *c* in *tr*:
  - $tr \downarrow c = \#(tr \restriction \{c\})$

X2. The *customer* of a vending machine wants to ensure that it will not absorb further coins until it has dispensed the chocolate already paid for

$$FAIR1 = ((tr \downarrow coin) \leq (tr \downarrow choc) + 1)$$

X3. The *manufacturer* of a simple vending machine must meet the requirements both of its owner and its customer

$$VMSPEC = NOLOSS \wedge FAIR1 = (0 \leq ((tr \downarrow coin) - (tr \downarrow choc)) \leq 1)$$

Processes: **Specifications**

X4. The complex vending machine is forbidden to accept three pennies in a row:

$$VMCFIX = (\neg \langle in1p \rangle^3 \text{ in } tr)$$

X5. The specification of a mended machine

$$MENDVMC = (tr \in traces(VMC) \wedge VMCFIX)$$

X6. The specification of *VMS2*

$$0 \leq ((tr \downarrow coin) - (tr \downarrow choc) \leq 2$$

$$\begin{aligned} VMC = (&in2p \rightarrow (large \rightarrow VMC \mid small \rightarrow out1p \rightarrow VMC) \\ &\mid in1p \rightarrow (small \rightarrow VMC \mid in1p \rightarrow (large \rightarrow VMC \mid in1p \rightarrow STOP))) \end{aligned}$$

$$VMS2 = (coin \rightarrow VMCRED)$$

$$VMCRED = \mu X \bullet (coin \rightarrow choc \rightarrow X \mid choc \rightarrow coin \rightarrow X)$$

Processes: Specifications: **Satisfaction**

- P is a product which meets a specification S
  - P *satisfies* S
    - P sat S
  - Every possible observation of the behaviour of P is described by S
    - $\forall tr \bullet tr \in traces(P) \Rightarrow S$.

Processes: Specifications: **Satisfaction**

- The specification *true*
  - places no constraints whatever on observations of a product
  - is satisfied by all products (even a broken product):
- **L1.** P sat *true*
- If a product satisfies two different specification, it also satisfies their conjunction:
- **L2A. if** P sat S **and** P sat T **then** P sat $(S \wedge T)$
- Generalisation to infinite conjunctions:
- **L2.** **if** $\forall n \bullet (P \text{ sat } S(n))$ **then** P sat $(\forall n \bullet S(n))$ **if** P does not depend on n.
  - $S(n)$ is a predicate containing the variable n
- The law for a weaker specification:
- **L3. If** P sat S **and** $S \Rightarrow T$ **then** P sat T
- P sat $(S \Rightarrow T)$ is an abbreviation for P sat S, $S \Rightarrow T$, P sat T
  - if $S \Rightarrow T$

Processes: Specifications: **Proofs**

- Laws for mathematical reasoning to
  - ensure that a process *P* meets its specification *S*.
- Notation:
  - The specification *S(tr)* contains *tr* as a free variable.
- Any observation of the process *STOP* is an empty trace:

L4A. $STOP$ sat $(tr = \langle\rangle)$

- Unfolding:

L4B. If P sat $S(tr)$ **then** $(c \rightarrow P)$ sat $(tr = \langle\rangle \vee (tr_0 = c \wedge S(tr')))$

- For double prefixing:

L4C. If P sat $S(tr)$ **then** $(c \rightarrow d \rightarrow P)$ sat $(tr \leq \langle c, d\rangle \vee (tr \geq \langle c, d\rangle \wedge S(tr'')))$

Processes: Specifications: **Proofs**

- For binary choice:

L4D. If P sat $S(tr)$ **and** Q sat $T(tr)$ **then**

$$(c \rightarrow P \mid d \rightarrow Q) \text{ sat } (tr = \langle\rangle \lor (tr_0 = c \land S(tr')) \lor (tr_0 = d \land T(tr')))$$

- The law for general choice generalises:

L4. If $\forall x : B \bullet (P(x) \text{ sat } S(tr, x))$ **then** $(x : B \rightarrow P(x)) \text{ sat } (tr = \langle\rangle \lor (tr_0 \in B \land S(tr', tr_0)))$

- The law for the after operator:

L5. If P sat $S(tr)$ **and** $s \in traces(P)$ **then** P / s sat $S(s\frown tr)$

  - If tr is a trace of (P / s), $s\frown tr$ is a trace of P, and therefore must be described by any specification which P satisfies

Processes: Specifications: **Proofs**

- The law for recursively defined process:

L6. If $F(x)$ is guarded **and** $STOP$ sat S **and** $((X \text{ sat } S) \Rightarrow (F(X) \text{ sat } S))$

then $(\mu X \bullet F(X))$ sat S

- Consequence:
  - $F^n(STOP)$ sat S for all n.
- The proof by induction.
- $F^n(STOP)$ fully describes at least the first n steps of the behaviour of $\mu X \bullet F(X)$
  - since F is guarded.
- So each trace of $\mu X \bullet F(X)$ is a trace of $F^n(STOP)$ for some n.
- This trace must therefore satisfy the same specification as $F^n(STOP)$, which is S.

$VMSPEC = NOLOSS \wedge FAIR1 = (0 \leq ((tr \downarrow coin) - (tr \downarrow choc)) \leq 1)$
$VMS = (coin \rightarrow (choc \rightarrow VMS))$

Processes: Specifications: **Proofs**

X1. Prove that *VMS* sat *VMSPEC*

1. *STOP* sat $tr = \langle\rangle$ **[L4A]**
   $\Rightarrow 0 \leq (tr \downarrow coin = tr \downarrow choc) \leq 1$ **[since $(\langle\rangle \downarrow coin) = (\langle\rangle \downarrow choc) = 0$]**

- The conclusion follows by an (implicit) appeal to L3.

2. Assume X sat $(0 \leq ((tr \downarrow coin) - (tr \downarrow choc)) \leq 1)$, then

$(coin \rightarrow choc \rightarrow X)$ sat **[L4C]**
$(tr \leq \langle coin, choc\rangle) \vee$
$(tr \geq \langle coin, choc\rangle \wedge 0 \leq ((tr \downarrow coin) - (tr \downarrow choc)) \leq 1))$
$\Rightarrow 0 \leq ((tr \downarrow coin) - (tr \downarrow choc)) \leq 1$
since
$\langle\rangle \downarrow coin = \langle\rangle \downarrow choc = \langle coin\rangle \downarrow choc = 0$ **and**
$\langle coin\rangle \downarrow coin = (\langle coin, choc\rangle \downarrow coin) = \langle coin, choc\rangle \downarrow choc = 1$ **and**
$tr \geq \langle coin, choc\rangle \Rightarrow (tr \downarrow coin = tr'' \downarrow coin + 1 \wedge tr \downarrow choc = tr'' \downarrow choc + 1)$

- The conclusion follows by appeal to L3 and L6.

L3. Weak spec
L6. Recursion

L3. If P sat S **and** $S \Rightarrow T$ **then** P sat T
L4A. $STOP$ sat $(tr = \langle\rangle)$

Processes: Specifications: **Proofs**

- Specification satisfiability does not necessarily mean correctness:
  - $STOP$ sat $0 \leq (tr \downarrow coin - tr \downarrow choc) \leq 1$
    - $tr = \langle\rangle \Rightarrow 0 \leq (tr \downarrow coin - tr \downarrow choc) \leq 1$
    - by L3 and L4A
  - $STOP$ will not serve as an adequate vending machine.
- $STOP$ does not act wrong just because it does nothing at all.
- *STOP satisfies every specification* which is satisfiable by any process.
- Any process defined *solely* by prefixing, choice, and guarded recursions will never stop.
  - VMS will never stop.
- The only way to write a process that can stop is to include
  - explicitly the process $STOP$, or
  - the process $(x : B \rightarrow P(x))$ where B is the empty set.

$VMS = (coin \rightarrow (choc \rightarrow VMS))$

Concurrency

Concurrency: **Introduction**

- A process is defined by describing the whole range of its potential behaviour.
  - *a choice* between several different actions
    - actually *can be controlled* by the environment.
      - the customer of the vending machine may select which coin to insert.
- *The environment of a process may be described as a process*.
- The behaviour of a complete system composed
  - from the process together with its environment,
    - acting and interacting with each other as they evolve concurrently.
- The complete system is a process too:
  - its behaviour is defined in terms of the behaviour of its component processes.
- Forget the distinction between processes, environments, and systems
  - they are all processes with behaviour described and analysed in a homogeneous mode.

Concurrency: **Interaction**

- Process interactions may be regarded as
  - *events that require simultaneous participation* of both the processes involved.
  - For the time being, we will ignore all other types of events.
- The alphabets of the two processes are *the same*.
  - Each event must be a possible event in the *independent* behaviour of each process.
    - A chocolate can be extracted from a vending machine only when
      - its customer wants it and
      - the vending machine is prepared to give it.
- The process which behaves like the system composed of
  - processes P and Q interacting in lock-step synchronisation
    - $P \parallel Q$
  - P and Q are processes with the same alphabet.

Concurrency: **Interaction**

X1. A greedy customer of a vending machine.

$$\begin{aligned} GRCUST = (&toffee \rightarrow GRCUST \\ &\mid choc \rightarrow GRCUST \\ &\mid coin \rightarrow choc \rightarrow GRCUST) \end{aligned}$$

- This customer and the machine *VMCT*
  - the greed is frustrated
    - the vending machine does not allow goods to be extracted before payment
  - *VMCT* never gives a toffee
    - the customer never wants one after he has paid

$$(GRCUST \parallel VMCT) = \mu X \bullet (coin \rightarrow choc \rightarrow X)$$

- A process which is a composition of two subprocesses
  - may be described as a simple single process without ∥.

$$VMCL = in2p \rightarrow large \rightarrow VMC \mid in1p \rightarrow in1p \rightarrow (large \rightarrow VMC \mid in1p \rightarrow STOP)$$

Concurrency: **Interaction**

X2. A foolish customer wants a large biscuit no matter the coin inserted:

$$\begin{aligned} FOOLCUST = (&in2p \rightarrow large \rightarrow FOOLCUST \\ \mid\ &in1p \rightarrow large \rightarrow FOOLCUST) \end{aligned}$$

- The vending machine is not prepared to yield a large biscuit for only a small coin

$$(FOOLCUST \parallel VMC) = \mu X \bullet (in2p \rightarrow large \rightarrow X \mid in1p \rightarrow STOP)$$

- The *STOP* after the first *in1p* is known as *deadlock*.
  - Each component process is prepared to engage in some further action
    - but these actions are different.
  - Nothing further can happen
    - because the processes cannot agree on what the next action shall be.

Concurrency: Interaction: **Laws**

- The logical symmetry between a process and its environment:

L1. $P \parallel Q = Q \parallel P$

- It does not matter in which order three processes are put together:

L2. $P \parallel (Q \parallel R) = (P \parallel Q) \parallel R$

- A deadlocked process infects the whole system with deadlock:

L3A. $P \parallel STOP_{\alpha P} = STOP_{\alpha P}$

- Composition with $RUN_{\alpha P}$ makes no difference:

L3B. $P \parallel RUN_{\alpha P} = P$

$RUN_A = (x : A \rightarrow RUN_A)$

Concurrency: Interaction: **Laws**

- A pair of processes engage simultaneously in the same action:

L4A. $(c \rightarrow P) \parallel (c \rightarrow Q) = (c \rightarrow (P \parallel Q))$

- Deadlock if process disagree on what the first action should be:

L4B. $(c \rightarrow P) \parallel (d \rightarrow Q) = STOP \qquad$ **if** $c \neq d$

- A system defined in terms of concurrency can be described without concurrency:

L4. $(x : A \rightarrow P(x)) \parallel (y : B \rightarrow Q(y)) = (z : (A \cap B) \rightarrow (P(z) \parallel Q(z)))$

Concurrency: Interaction: **Laws**

X1. Let $P = (a \rightarrow b \rightarrow P \mid b \rightarrow P)$ and $Q = (a \rightarrow (b \rightarrow Q \mid a \rightarrow Q))$

- Then

$$
\begin{aligned}
(P \parallel Q) = & \\
&= a \rightarrow ((b \rightarrow P) \parallel (b \rightarrow Q \mid a \rightarrow Q)) && \text{[by L4A]} \\
&= a \rightarrow (b \rightarrow (P \parallel Q)) && \text{[by L4]} \\
&= \mu\, X \bullet (a \rightarrow b \rightarrow X) && \text{[since the recursion is guarded]}
\end{aligned}
$$

Concurrency: Interaction: **Traces**

- Each sequence of actions must be possible for both operands:

L1. $traces(P \parallel Q) = traces(P) \cap traces(Q)$

- $/\ s$ distributes through $\parallel$:

L2. $(P \parallel Q)\ /\ s = (P\ /\ s) \parallel (Q\ /\ s)$

Concurrency: **Concurrency**

- Concurrency for processes with different alphabets
  - $\alpha P \neq \alpha Q$
  - Events in the alphabet of P but not in the alphabet of Q are of no concern to Q.
    - Such events may occur independently of Q whenever P engages in them.
    - Similarly for Q.
- The set of all events that are logically possible for the system:
  - $\alpha(P \parallel Q) = \alpha P \cup \alpha Q$

Concurrency: **Concurrency**

X1. $\alpha NOISYVM = \{coin, choc, clink, clunk\}$,

- *clink* is the sound of a coin dropping into the moneybox
- *clunk* is the sound made on completion of a transaction.
- The noisy vending machine has run out of toffee

$$NOISYVM = (coin \rightarrow clink \rightarrow choc \rightarrow clunk \rightarrow NOISYVM)$$

- The customer of this machine definitely prefers toffee.
  - the *curse* is what he utters when he fails to get it.
- $\alpha CUST = \{coin, choc, curse, toffee\}$

$$CUST = (coin \rightarrow (toffee \rightarrow CUST \mid curse \rightarrow choc \rightarrow CUST))$$

- The result of the concurrent activity:

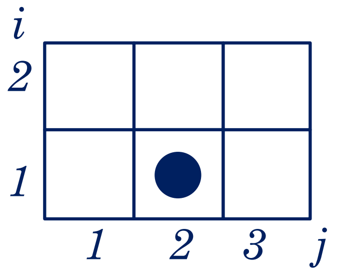
$$\begin{aligned}
&(NOISYVM \parallel CUST) = \\
&\mu X \bullet (coin \rightarrow (clink \rightarrow curse \rightarrow choc \rightarrow clunk \rightarrow X \\
&\qquad\qquad\quad \mid curse \rightarrow clink \rightarrow choc \rightarrow clunk \rightarrow X))
\end{aligned}$$

Concurrency: **Concurrency**

X2. A counter starts at the middle bottom square of the board, and may move within the board either *up*, *down*, *left* or *right*.

- $aP = \{up, down\}$ and $aQ = \{left, right\}$

$$P = (up \rightarrow down \rightarrow P) \qquad Q = (right \rightarrow left \rightarrow Q \mid left \rightarrow right \rightarrow Q)$$

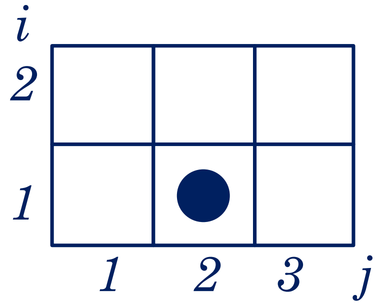


- The behaviour of this counter is $P \parallel Q$.
  - $aP \cap aQ = \varnothing$
    - The movements of the counter are an arbitrary *interleaving* of actions from the process P with actions from the process Q.
- Such interleavings are very laborious to describe without concurrency.

Concurrency: **Concurrency**

- R_{ij} is the behaviour of the counter when situated in row i and column j of the board
  - $i \in \{1, 2\}, j \in \{1, 2, 3\}$.
- $(P \parallel Q) = R_{12}$

$$
\begin{aligned}
R_{21} &= (down \rightarrow R_{11} \mid right \rightarrow R_{22}) \\
R_{11} &= (up \rightarrow R_{21} \mid right \rightarrow R_{12}) \\
R_{22} &= (down \rightarrow R_{12} \mid left \rightarrow R_{21} \mid right \rightarrow R_{23}) \\
R_{12} &= (up \rightarrow R_{22} \mid left \rightarrow R_{11} \mid right \rightarrow R_{13}) \\
R_{23} &= (down \rightarrow R_{13} \mid left \rightarrow R_{22}) \\
R_{13} &= (up \rightarrow R_{23} \mid left \rightarrow R_{12})
\end{aligned}
$$



Concurrency: Concurrency: **Laws**

- The first three laws are similar to those for interaction:

L1,2. $\|$ is symmetric and associative

L3A. $P \,\|\, STOP_{\alpha P} = STOP_{\alpha P}$

L3B. $P \,\|\, RUN_{\alpha P} = P$

- $a \in (\alpha P - \alpha Q)$, $b \in (\alpha Q - \alpha P)$ and $\{c, d\} \subseteq (\alpha P \cap \alpha Q)$.
- c and d require simultaneous participation of both P and Q:

L4A. $(c \rightarrow P) \,\|\, (c \rightarrow Q) = c \rightarrow (P \,\|\, Q)$

L4B. $(c \rightarrow P) \,\|\, (d \rightarrow Q) = STOP$ if $c \neq d$

- P engages alone in a, Q engages alone in b:

L5A. $(a \rightarrow P) \,\|\, (c \rightarrow Q) = a \rightarrow (P \,\|\, (c \rightarrow Q))$

L5B. $(c \rightarrow P) \,\|\, (b \rightarrow Q) = b \rightarrow ((c \rightarrow P) \,\|\, Q)$

L6. $(a \rightarrow P) \,\|\, (b \rightarrow Q) = a \rightarrow (P \,\|\, (b \rightarrow Q)) \mid b \rightarrow ((a \rightarrow P) \,\|\, Q))$

Concurrency: Concurrency: **Laws**

- Generalisation with the general choice operator:

L7. Let $P = (x : A \rightarrow P(x))$ and $Q = (y : B \rightarrow Q(y))$

Then $(P \parallel Q) = (z : C \rightarrow P' \parallel Q')$, where

$C = (A \cap B) \cup (A - \alpha Q) \cup (B - \alpha P)$,

$P' = P(z)$ if $z \in A$,

$P' = P$ otherwise,

$Q' = Q(z)$ if $z \in B$,

$Q' = Q$ otherwise.

- A process defined with concurrency can be redefined without concurrency.

Concurrency: Concurrency: **Laws**

X1. Let $\alpha P = \{a, c\}$ and $\alpha Q = \{b, c\}$

- $P = (a \to c \to P)$ and $Q = (c \to b \to Q)$

$$
\begin{aligned}
P \parallel Q &= (a \to c \to P) \parallel (c \to b \to Q) && \text{[by definition]} \\
&= a \to ((c \to P) \parallel (c \to b \to Q)) && \text{[by L5A]} \\
&= a \to c \to (P \parallel (b \to Q)) && \text{[by L4A} \ldots \ddagger\text{]}
\end{aligned}
$$

Also

$$
\begin{aligned}
P \parallel (b \to Q) &= (a \to (c \to P) \parallel (b \to Q) \mid b \to (P \parallel Q)) && \text{[by L6]} \\
&= (a \to b \to ((c \to P) \parallel Q) \mid b \to (P \parallel Q)) && \text{[by L5B]} \\
&= (a \to b \to c \to (P \parallel (b \to Q)) \mid b \to a \to c \to (P \parallel (b \to Q))) && \text{[by } \ddagger \text{ above]} \\
&= \mu X \bullet (a \to b \to c \to X \mid b \to a \to c \to X) && \text{[since this is guarded]}
\end{aligned}
$$

Therefore

$(P \parallel Q) = (a \to c \to \mu X \bullet (a \to b \to c \to X \mid b \to a \to c \to X))$ by $\ddagger$ above

Concurrency: Concurrency: Traces

- t is a trace of $(P \parallel Q)$.
  - $(t \upharpoonright \alpha P)$ is a trace of P
    - every event in t which belongs to αP is an event in the life of P
    - every event in t which does not belong to αP occurs without P.
  - $(t \upharpoonright \alpha Q)$ is a trace of Q
    - by a similar argument.
  - Every event in t must be in either αP or αQ.

L1. $traces(P \parallel Q) = \{\, t \mid (t \upharpoonright \alpha P) \in traces(P) \wedge (t \upharpoonright \alpha Q) \in traces(Q) \wedge t \in (\alpha P \cup \alpha Q)^* \,\}$

- Distribution of $/\ s$ operator through concurrency:

L2. $(P \parallel Q) \,/\, s = (P \,/\, (s \upharpoonright \alpha P)) \parallel (Q \,/\, (s \upharpoonright \alpha Q))$

- When $\alpha P = \alpha Q$ then $s \upharpoonright \alpha P = s \upharpoonright \alpha Q = s$
  - the laws as for interaction:

L1. $traces(P \parallel Q) = traces(P) \cap traces(Q)$
L2. $(P \parallel Q) \,/\, s = (P \,/\, s) \parallel (Q \,/\, s)$

$NOISYVM = (coin \rightarrow clink \rightarrow choc \rightarrow clunk \rightarrow NOISYVM)$
$CUST = (coin \rightarrow (toffee \rightarrow CUST \mid curse \rightarrow choc \rightarrow CUST))$

Concurrency: Concurrency: Traces

$(NOISYVM \parallel CUST) =$
$\mu X \bullet (coin \rightarrow (clink \rightarrow curse \rightarrow choc \rightarrow clunk \rightarrow X$
$\mid curse \rightarrow clink \rightarrow choc \rightarrow clunk \rightarrow X))$

X1. (See 2.3 X1.)

- $t1 = <coin, click, curse>$

  $t1 \upharpoonright \alpha NOISYVM = <coin, click>$ [is in *traces(NOISYVM)*]
  $t1 \upharpoonright \alpha CUST = <coin, curse>$ [is in *traces(CUST)*]

- Therefore $t1 \in traces(NOISYVM \parallel CUST)$
- $<coin, curse, clink> \in traces(NOISYVM \parallel CUST)$
  - similar reasoning
- The *curse* and the *clink* may be recorded in either order.
  - They may even occur simultaneously
    - no way of recording this.

Concurrency: Concurrency: **Traces**

- A trace of $(P \parallel Q)$ is a kind of interleaving of a trace of P with a trace of Q.
  - $aP \cap aQ = \{\}$
    - interleaving
  - $aP = aQ$
    - interaction

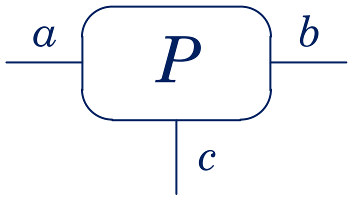
L3A. If $aP \cap aQ = \{\}$, then

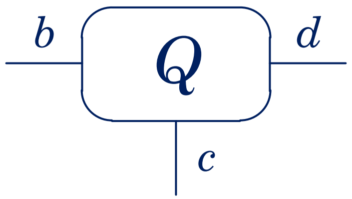
$$traces(P \parallel Q) = \{s \mid \exists t : traces(P);\ u : traces(Q) \bullet s \text{ interleaves } (t, u)\}$$

L3B. If $aP = aQ$ then $traces(P \parallel Q) = traces(P) \cap traces(Q)$

Concurrency: **Pictures**

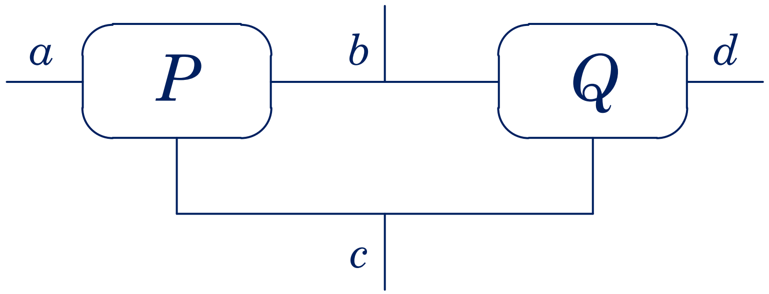
- A process P with alphabet $\{a, b, c\}$
  - lines labelled with a different events.





- A process Q with its alphabet $\{b, c, d\}$

- Concurrent process $(P \parallel Q)$ is a network in which
  - similarly labelled lines are connected
  - lines labelled by events in the alphabet of only one process are left free.



Concurrency: **Pictures**

- A system constructed from three processes is still only a single process
  - therefore be pictured as a single box
  - information is lost