# T-79.231 Parallel and Distributed Digital Systems

# Process Algebra

Marko Mäkelä

August 5, 2003

# Background

Parallel and distributed system tend to consist of relatively independent processes that communicate every now and then by exchanging messages or accessing shared memory.

With Petri nets, it is easy to model shared memory (as shared places of transitions), but in their basic form, they do not distinguish processes.* It is difficult to tell which process a given place or transition belongs to, or if they have something to do with communication. Partial order reduction methods attempt to reconstruct this information afterwards, which does not always work efficiently.

Process algebra focuses on the modelling of transitions and *compositional modelling*, where processes are modelled separately and connected to each other. Process algebras are based on transition systems, which can be compared with each other or reduced with respect to some *equivalence relation*.

*Modular nets allow the modelling of processes.

# CCS and CSP

The two best known process algebras are CCS (Calculus of Communicating Systems) by Robin Milner and CSP (Communicating Sequential Processes) by Tony Hoare.

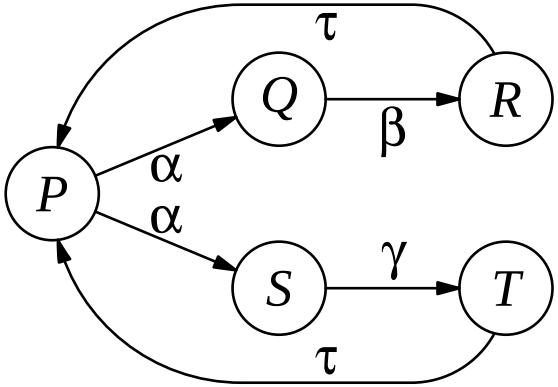
CCS distinguishes three kinds of actions: internal actions of a prosess or an *agent* ($\tau$), reception ($\alpha$) and transmission ($\overline{\alpha}$). Communication takes place between two agents.

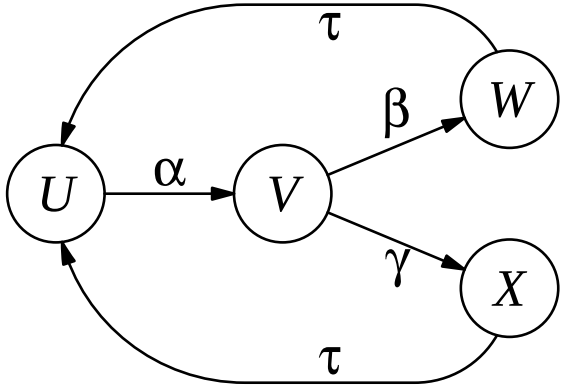
Originally (1978), Hoare defined CSP by extending Dijkstra's guarded command language with transmission and reception statements, but the current form of CSP (1985) does not tell apart different kinds of synchronisation: any number of agents can participate.

The semantics of synchronisation is defined by the parallel composition rule. One could define that an agent can only execute an action in parallel with all agents that contain an equally named action. Alternatively, it could be defined that not all agents need to synchronise. In that case, the size of the parallel composition would grow easily.

This lecture concentrates on the process algebra CCS.

# Prosess Algebra and Transition Systems




$$P ::= \alpha \,.\, Q + \alpha \,.\, S$$
$$Q ::= \beta \,.\, R$$
$$R ::= \tau \,.\, P$$
$$S ::= \gamma \,.\, T$$
$$T ::= \tau \,.\, P$$

$$P ::= \alpha \,.\, \beta \,.\, \tau \,.\, P + \alpha \,.\, \gamma \,.\, \tau \,.\, P$$

$$U ::= \alpha \,.\, V$$
$$V ::= \beta \,.\, W + \gamma \,.\, X$$
$$W ::= \tau \,.\, U$$
$$X ::= \tau \,.\, U$$

$$U ::= \alpha \,.\, (\beta \,.\, \tau \,.\, U + \gamma \,.\, \tau \,.\, U)$$

# The Basics of Process Algebras

- Every agent has a name. In mathematical notation, agents are usually denoted with capital letters $P, Q, R, \ldots$; tools may use some other notation, such as numbers $0, 1, 2, \ldots$.

- The *invisible action* $\tau$ denotes internal operations of an agent. These actions cannot be observed or affected by other agents.

- *Visible actions* are often denoted with Greek letters $\alpha, \beta, \gamma, \ldots$. The parallel composition rule defines how agents can synchronise by executing visible actions simultaneously.

- Process algebra concentrates on actions. The notation $P \xrightarrow{\alpha} Q$ means that agent $P$ behaves identically with agent $Q$ once it has executed the action $\alpha$.

# Special Characteristics of CCS

- For every visible action $\alpha \neq \tau$, there is a *complement action* $\overline{\alpha}$. The complement of a complement action is the action itself: $\overline{\overline{\alpha}} = \alpha$.
- The action and its complement can be thought as the reception and transmission of the same message.
- The parallel composition rule deals with two agents at a time: if $Q$ is capable of receiving the message $\gamma$ and $R$ is capable of sending it or vice versa ($Q \xrightarrow{\gamma} Q', R \xrightarrow{\bar{\gamma}} R'$), then $(Q||R) \xrightarrow{\tau} (Q'||R')$. The agents can also proceed independently: if $Q \xrightarrow{\beta} Q''$, then $(Q||R) \xrightarrow{\beta} (Q''||R)$.* This parallel composition is symmetric: $(Q||R) = (R||Q)$.

*This holds, even if $R$ could proceed with $\overline{\beta}$.

# The Syntax and Semantics of CCS (1/2)

Next, we shall present the syntax and semantics of the process algebra CCS.

**The empty agent ($Nil$)** is incapable of executing actions: $\nexists \alpha, P : Nil \xrightarrow{\alpha} P$.

**Action Prefixing** Let $Q$ be an agent and $\alpha$ an (invisible or visible) action. Then $P ::= \alpha\,.\,(Q)$ is an agent and $P \xrightarrow{\alpha} Q$. *The agent $P$ can execute $\alpha$ and then behave like the agent $Q$.*

**Choice** Let there be agents $Q$ and $R$ and actions $\alpha$ and $\beta$. $P ::= (Q + R)$ is an agent. If $Q \xrightarrow{\alpha} Q'$, then $P \xrightarrow{\alpha} Q'$. If $R \xrightarrow{\beta} R'$, then $P \xrightarrow{\beta} R'$. *In its "initial state," agent $P$ can choose whether it behaves like $Q$ or like $R$.*

**Restriction** Let $Q$ be an agent and $\Sigma$ be a set of visible actions, $\tau \notin \Sigma$. Then $P ::= (Q) \setminus \Sigma$ is an agent. If $Q \xrightarrow{\alpha} Q'$ and $\alpha \notin \Sigma, \overline{\alpha} \notin \Sigma$, then $((Q) \setminus \Sigma) \xrightarrow{\alpha} ((Q') \setminus \Sigma)$. *Agent $P$ is like $Q$, but it cannot execute actions in the set $\Sigma$ or their complement actions.*

# The Syntax and Semantics of CCS (2/2)

**Relabeling** Let $Q$ be an agent and $\Sigma$ the set of its visible actions, $\tau \notin \Sigma$. Let $\Sigma'$ be a set of actions and let $m : \Sigma \cup \{\tau\} \to \Sigma' \cup \{\tau\}$ such that $m(\tau) = \tau$ and $\forall \alpha \neq \tau : m(\overline{\alpha}) = \overline{m(\alpha)}$. Then $P ::= Q[m]$ is an agent. If $Q \xrightarrow{\alpha} Q'$, then $Q[m] \xrightarrow{m(\alpha)} Q'[m]$. *Agent $P$ is like $Q$, but its actions have been obtained by mapping the actions $Q$ through $m$.* Special case: hiding ($m(\alpha) = m(\overline{\alpha}) = \tau$).

**Parallel Composition** Let $Q$ and $R$ be agents. $(Q||R)$ is an agent.

1. If $Q \xrightarrow{\gamma} Q'$ and $R \xrightarrow{\overline{\gamma}} R'$, then $(Q||R) \xrightarrow{\tau} (Q'||R')$.
2. If $Q \xrightarrow{\gamma} Q''$, then $(Q||R) \xrightarrow{\gamma} (Q''||R)$.
3. If $R \xrightarrow{\gamma} R''$, then $(Q||R) \xrightarrow{\gamma} (Q||R'')$.

# On the Syntax and Semantics of Process Algebras

Different process algebras define slightly different structures and computation rules. It can take some time to learn them, because they are often defined in a compact formal notation, as in the previous slides.

For the sake of readability, parentheses are often omitted. With parentheses, one would write like this: $U ::= \alpha \,.\, ((\beta \,.\, (\tau \,.\, (U)) + \gamma \,.\, (\tau \,.\, (U))))$.

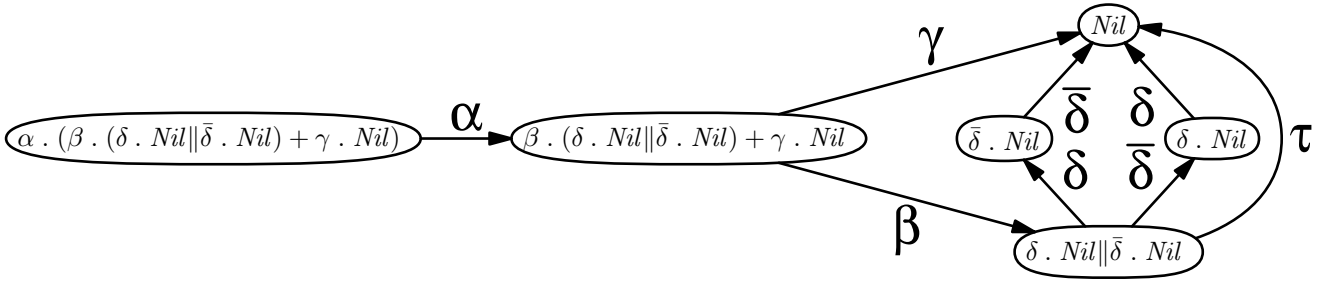
Question: is the expression $U ::= \alpha \,.\, (\beta \,.\, \tau + \gamma \,.\, \tau) \,.\, U$ in accordance with the CCS syntax we presented, if $\alpha, \beta, \gamma, \tau$ are names of actions? What about $U ::= \alpha \,.\, (\beta + \gamma) \,.\, \tau \,.\, U$?

There are various short-hand notations and conventions for process algebra. For instance, $.\, Nil$ can be omitted. Are the expressions above valid CCS, if they make use of this convention?

Marko Mäkelä

# An Example of Parallel Composition

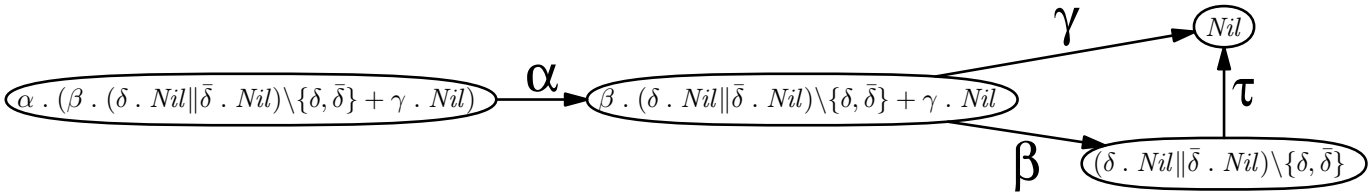
The behaviour of the agent $\alpha\,.\,(\beta\,.\,(\delta\,.\,Nil||\overline{\delta}\,.\,Nil)+\gamma\,.\,Nil)$ is depicted below:




The transition system was constructed by applying the previously presented semantic rules for CCS. The states have been labelled with the corresponding agent.

The behaviour of the agent $\delta\,.\,Nil||\overline{\delta}\,.\,Nil$ includes the interleaved executions of $\delta$ and $\overline{\delta}$.

The parallel composition can be defined to allow only concurrent synchronisation:

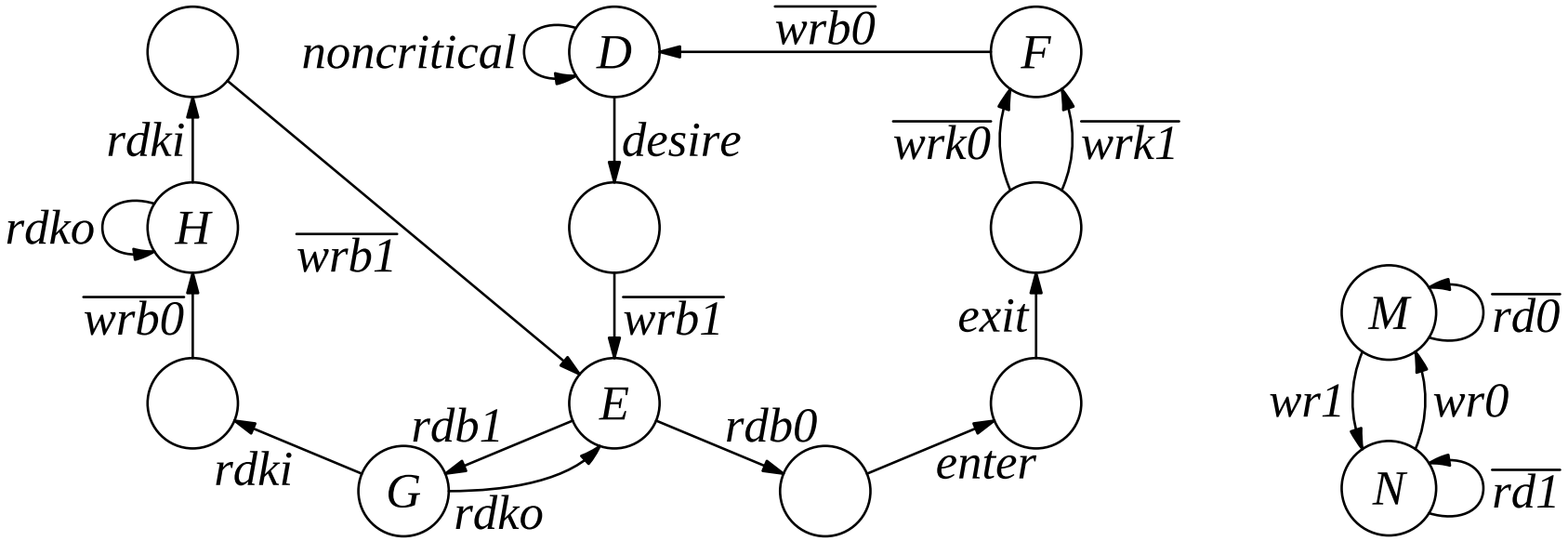



Let $P|Q$ be such a restricted parallel composition. Exercise: define it formally.

# Example: Dekker's Algorithm (1/6)

Let us represent Dekker's mutual exclusion algorithm for two processes with CCS.*

Since there is no concept of memory in CCS, storage has to be modelled explicitly. Let us first define the basic processes: the program code and an one-bit memory:




*At http://www.tcs.hut.fi/Software/maria/tools/tvt/tvt.html, you can find a more detailed example using CSP-style parallel composition, which evaluates the fairness of the algorithm.

# Example: Dekker's Algorithm (2/6)

Textually, the basic processes can be written, for instance, as follows:

$$
\begin{aligned}
M &::= \overline{rd0} \,.\, M + wr1 \,.\, N \\
N &::= \overline{rd1} \,.\, N + wr0 \,.\, M \\
D &::= noncritical \,.\, D + desire \,.\, \overline{wrb1} \,.\, E \\
E &::= rdb0 \,.\, enter \,.\, exit \,.\, (\overline{wrk0} \,.\, F + \overline{wrk1} \,.\, F) + rdb1 \,.\, G \\
F &::= \overline{wrb0} \,.\, D \\
G &::= rdko \,.\, E + rdki \,.\, \overline{wrb0} \,.\, H \\
H &::= rdko \,.\, H + rdki \,.\, \overline{wrb1} \,.\, E
\end{aligned}
$$

# Example: Dekker's Algorithm (3/6)

Dekker's algorithm comprises two processes and three shared memory places. Let us construct them from the basic processes by relabeling. If we used CSP-style parallel composition, the two-port memory $K$ could be defined exactly like the one-port memories $B_1$ and $B_2$, but in CCS it has to be defined in a different way. Exercise: Why? How?

$$
\begin{aligned}
B_1 &::= M[c(\{rd0 \mapsto b^1_\bot, rd1 \mapsto b^1_\top, wr0 \mapsto b^1_{\top\to\bot}, wr1 \mapsto b^1_{\bot\to\top}\})] \\
B_2 &::= M[c(\{rd0 \mapsto b^2_\bot, rd1 \mapsto b^2_\top, wr0 \mapsto b^2_{\top\to\bot}, wr1 \mapsto b^2_{\bot\to\top}\})] \\
P_1 &::= D[c(\{wrb1 \mapsto b^1_{\bot\to\top}, wrb0 \mapsto b^1_{\top\to\bot}, wrk0 \mapsto k^1_{2\to1}, wrk1 \mapsto k^1_{1\to2}, \\
&\qquad rdb0 \mapsto b^2_\bot, rdb1 \mapsto b^2_\top, rdko \mapsto k^1_2, rdki \mapsto k^1_1\})] \\
P_2 &::= D[c(\{wrb1 \mapsto b^2_{\bot\to\top}, wrb0 \mapsto b^2_{\top\to\bot}, wrk0 \mapsto k^2_{2\to1}, wrk1 \mapsto k^2_{1\to2}, \\
&\qquad rdb0 \mapsto b^1_\bot, rdb1 \mapsto b^1_\top, rdko \mapsto k^2_1, rdki \mapsto k^2_2\})]
\end{aligned}
$$

Above, the mapping $c(\Sigma) = \bigcup_{\alpha\in\Sigma}\{\alpha, \overline{\alpha}\}$ shortens the relabeling, because the complement actions need not be enumerated.

Marko Mäkelä

# Example: Dekker's Algorithm (4/6)

The full behaviour is $Dekker ::= K|B_1|B_2|P_1|P_2$. Because of the structure of these agents and the symmetry of the CCS parallel composition, the parallel compositions can be computed in any order. Let us compute $BP_1 ::= B_1|P_1$, from which $BP_2 ::= B_2|P_2$ can be obtained by relabeling. Then $Dekker = K|BP_1|BP_2$. First, let us find out $B_1$ and $P_1$:

$$B_1 ::= \overline{b^1_\bot} \,.\, B_1 + b^1_{\bot\to\top} \,.\, B'_1 \qquad\qquad B'_1 ::= \overline{b^1_\top} \,.\, B'_1 + b^1_{\top\to\bot} \,.\, B_1$$

$$\begin{aligned}
P_1 &::= noncritical \,.\, P_1 + desire \,.\, \overline{b^1_{\bot\to\top}} \,.\, P'_1 \\
P'_1 &::= b^2_\bot \,.\, enter \,.\, exit \,.\, (\overline{k^1_{2\to1}} \,.\, P''_1 + \overline{k^1_{1\to2}} \,.\, P''_1) + b^2_\top \,.\, P'''_1 \\
P''_1 &::= \overline{b^1_{\top\to\bot}} \,.\, P_1 \\
P'''_1 &::= k^1_2 \,.\, P'_1 + k^1_1 \,.\, \overline{b^1_{\top\to\bot}} \,.\, P''''_1 \\
P''''_1 &::= k^1_2 \,.\, P''''_1 + k^1_1 \,.\, \overline{b^1_{\bot\to\top}} \,.\, P'_1
\end{aligned}$$

# Example: Dekker's Algorithm (5/6)

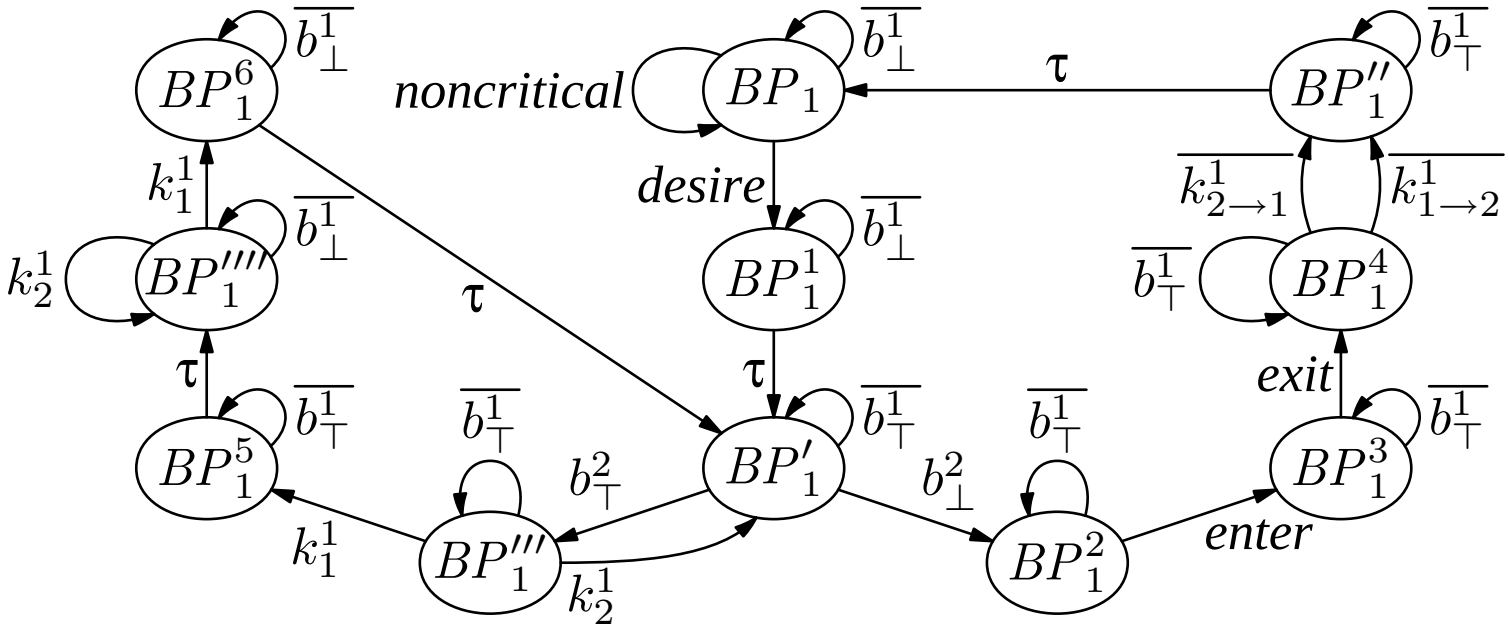
Let us compute $BP_1 ::= B_1 | P_1$. Now we cannot prefix multiple actions per line, because there are multiple actions leading to each agent:

$$
\begin{array}{lcl}
BP_1 & ::= & \overline{b^1_\bot}.BP_1 + noncritical\ .\ BP_1 \\
 & & + desire\ .\ BP^1_1 \\
BP^1_1 & ::= & \overline{b^1_\bot}.BP^1_1 + \tau\ .\ BP'_1 \\
BP'_1 & ::= & \overline{b^1_\top}.BP'_1 \\
 & & + b^2_\bot\ .\ BP^2_1 + b^2_\top\ .\ BP'''_1 \\
BP^2_1 & ::= & \overline{b^1_\top}.BP^2_1 + enter\ .\ BP^3_1 \\
BP^3_1 & ::= & \overline{b^1_\top}.BP^3_1 + exit\ .\ BP^4_1
\end{array}
\qquad
\begin{array}{lcl}
BP^4_1 & ::= & \overline{b^1_\top}.BP^4_1 \\
 & & + \overline{k^1_{2\to1}}\ .\ BP''_1 + \overline{k^1_{1\to2}}\ .\ BP''_1 \\
BP''_1 & ::= & \overline{b^1_\top}.BP''_1 + \tau\ .\ BP_1 \\
BP'''_1 & ::= & \overline{b^1_\top}.BP'''_1 + k^1_2\ .\ BP'_1 + k^1_1\ .\ BP^5_1 \\
BP^5_1 & ::= & \overline{b^1_\top}.BP^5_1 + \tau\ .\ BP''''_1 \\
BP''''_1 & ::= & \overline{b^1_\bot}.BP''''_1 + k^1_2\ .\ BP''''_1 + k^1_1\ .\ BP^6_1 \\
BP^6_1 & ::= & \overline{b^1_\bot}.BP^6_1 + \tau\ .\ BP'_1
\end{array}
$$

$BP_1$ is very similar to $P_1$, because $B_1$ can participate in every synchronisation $P_1$ "wants." The biggest difference are the self-loops $\overline{b^1_\bot}$ or $\overline{b^1_\top}$ in every state.

# Example: Dekker's Algorithm (6/6)

The transition system corresponding to $BP_1$ is a little easier to read:




$K|BP_1$ comprises already 22 states and 92 transitions, and $BP_1|BP_2$ comprises 105 and 289, and *Dekker* consists of 193 states and 428 transitions. Could the agents be simplified by removing $\tau$ actions (and hiding uninteresting actions, such as $noncritical \mapsto \tau$)?

# The Equivalences of Transition Systems

The concept of equivalence (similarity) makes it possible to compare agents with each other, and to reduce agents. Several different equivalences have been defined. We shall present two of them:

**Trace Equivalence** If the finite traces (visible sequences of transitions) of the agents $P$ and $Q$ are identical, the agents are trace equivalent, $P \approx_{\mathrm{tr}} Q$. *Examples:* $\alpha \,.\, \tau \,.\, R \approx_{\mathrm{tr}} \alpha \,.\, R$ *and* $\alpha \,.\, (\beta \,.\, R + \gamma \,.\, R) \approx_{\mathrm{tr}} \alpha \,.\, \beta \,.\, R + \alpha \,.\, \gamma \,.\, R$. *Let* $S ::= \tau \,.\, S$. *A divergence (livelock) cannot be told apart from a deadlock:* $S \approx_{\mathrm{tr}} Nil$.

**Failure Equivalence** Let there be an agent $R$, a visible action $\alpha$ and a set of visible actions $\Sigma$. If $R \xrightarrow{\alpha} R'$ and $\forall \beta \in \Sigma : \nexists R'' : R' \xrightarrow{\beta} R''$, then $\langle \alpha, \Sigma \rangle$ is the *failure* of $R$. The agents $P$ and $Q$ are failure equivalent, $P \approx_{\mathrm{f}} Q$, if their failures are identical and for each $\alpha \neq \tau, P', Q'$ s.t. $P \xrightarrow{\tau} \cdots \xrightarrow{\alpha} \xrightarrow{\tau} \cdots P'$ and $Q \xrightarrow{\tau} \cdots \xrightarrow{\alpha} \xrightarrow{\tau} \cdots Q'$, $P' \approx_{\mathrm{f}} Q'$ holds. *Example:* $\alpha \,.\, (\beta \,.\, R + \gamma \,.\, R) \not\approx_{\mathrm{f}} \alpha \,.\, \beta \,.\, R + \alpha \,.\, \gamma \,.\, R$.

# Simulation, Bisimulation and Choosing an Equivalence

Let $S$ be the set of all agents and $P \in S, Q \in S$. The agent $P$ simulates $Q$ if there exists some simulation relation $R \subseteq S \times S$ such that

1. $\langle P, Q \rangle \in R$ and
2. if $\langle P', Q' \rangle \in R$ and $P' \xrightarrow{\alpha} P''$, then there is $Q''$ such that $Q' \xrightarrow{\alpha} Q''$ and $\langle P'', Q'' \rangle \in R$.

Simulation is not an equivalence, since $P$ can simulate $Q$ even if $Q$ does not simulate $P$,* but bisimulation ($P$ simulates $Q$ and $Q$ simulates $P$) is. The TVT tool can compare and reduce transition systems with respect to strong bisimilarity relation.

The equivalence should be selected according to the property being verified. For instance, trace equivalence is often too strong, because it does not preserve the truth values of LTL formulae. If the property has been specified as an LTL formula that does not contain $\bigcirc$ connectives, the CFFD and NDFD equivalences are efficient.

*An equivalence is reflexive, symmetric and transitive.