Technische Universität Darmstadt

Telecooperation Lab
Prof. Dr. Max Mühlhäuser

TK1: Distributed Systems - Programming & Algorithms

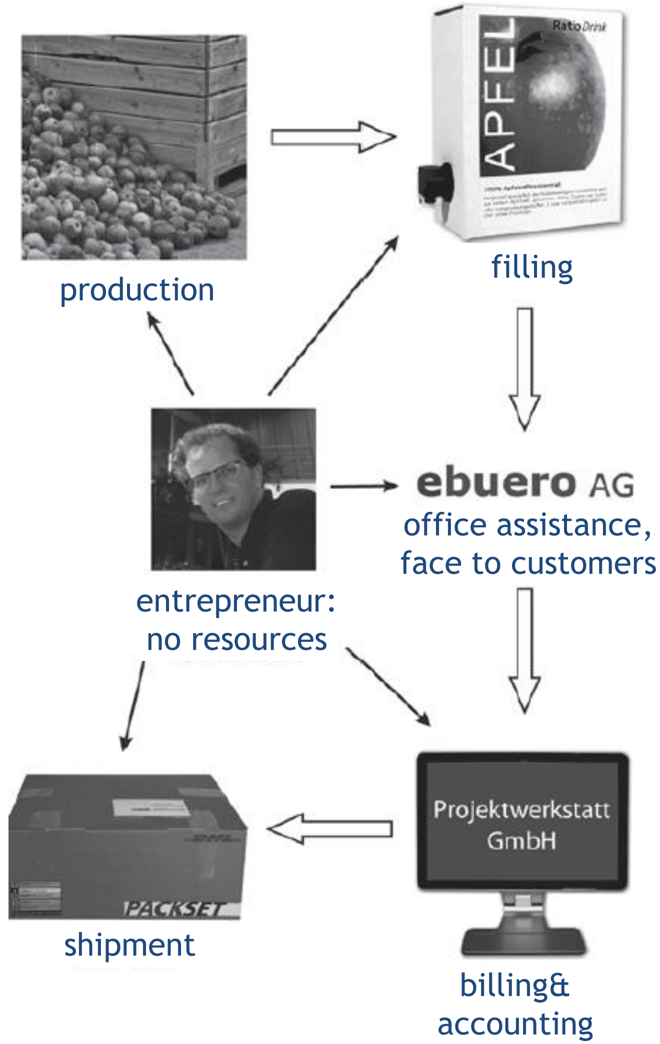
Chapter 2: Distributed Programming

Section 4: Formal Approaches, Process Calculi

Lecturer: Prof. Dr. Max Mühlhäuser

Copyrighted material – for TUD student use only

IoS example: Service Composition of Business Services



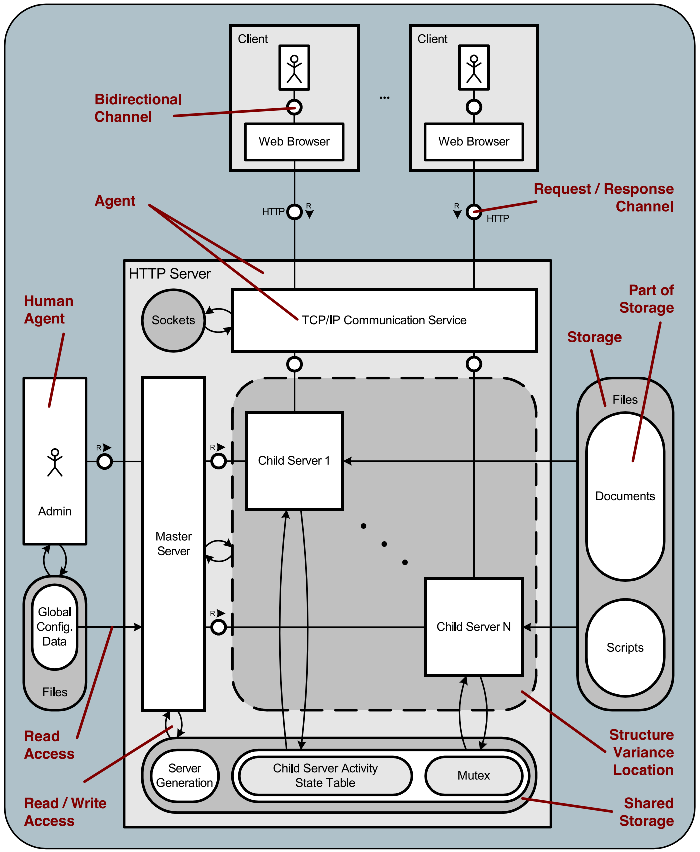
Internet-of-Services example:

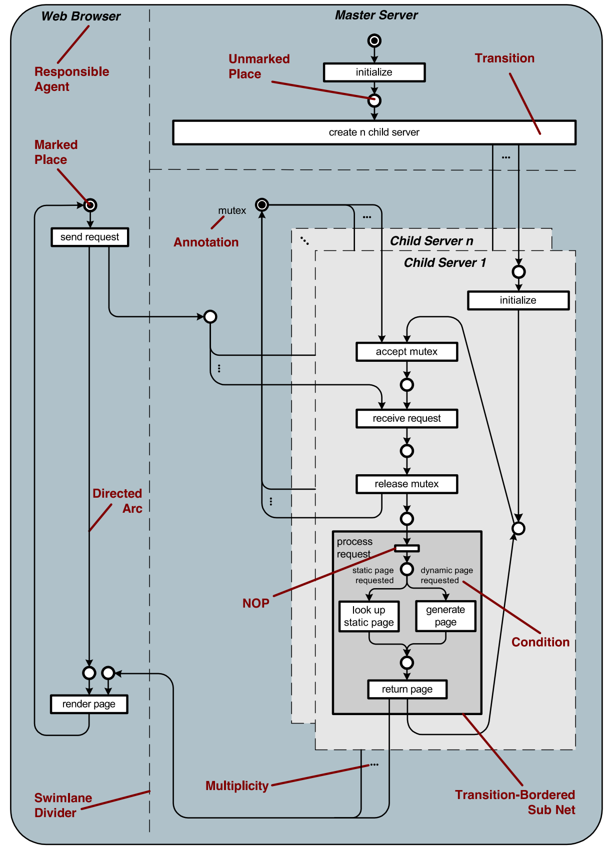
Business idea: Deliver fruit juice concentrate directly to end users. ➔ less expensive than buying it in bottles

A small business process composed of 5 (external!) business services:

1. Manufacturer of fruit juice concentrate
2. Filling station for fruit juice concentrate
3. Virtual office
4. External accounting service
5. Delivery service

IoS example: Systems Modeled via *Fundamental Modeling Concepts Notation*

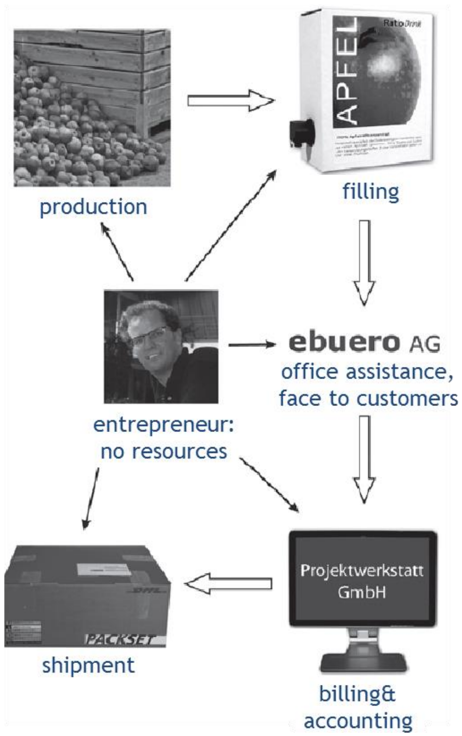




- common ways to model „entire distributed sytems“:
  - business process models / notations (coarse grained)
  - control/data flow models, e.g., FMCN (see above)
  - formal models → check properties, „equivalence“: this is (briefly) discussed in this subchapter

Formal Approaches.

- Why using formal approaches for distributed systems ?
  - Get the ability to specify unambiguous models of distributed systems. If many different teams are supposed to interpret the model, the result should always be the same.
  - Formal descriptions enable automated verification and therefore, developers can be supported finding and fixing errors, even for highly complex systems.



Labeled Transition Systems

Definition 1 (Labeled Transition System) *Let* Act *be a fixed set of actions. A labeled transition system* $T = (S, \mathsf{Act}, \rightarrow)$ *over* Act *consists of*

- *A set* S *of states and*
- *a set* $\rightarrow \subseteq S \times \mathsf{Act} \times S$ *of transition between states.*
- A transition system is called finite if the state set as well as the set of transitions is finite.
- $s \xrightarrow{a} s'$ is written instead of $(s, a, s') \in \rightarrow$.
- If $|\mathsf{Act}| = 1$, then T is equivalent to an unlabeled transition system.

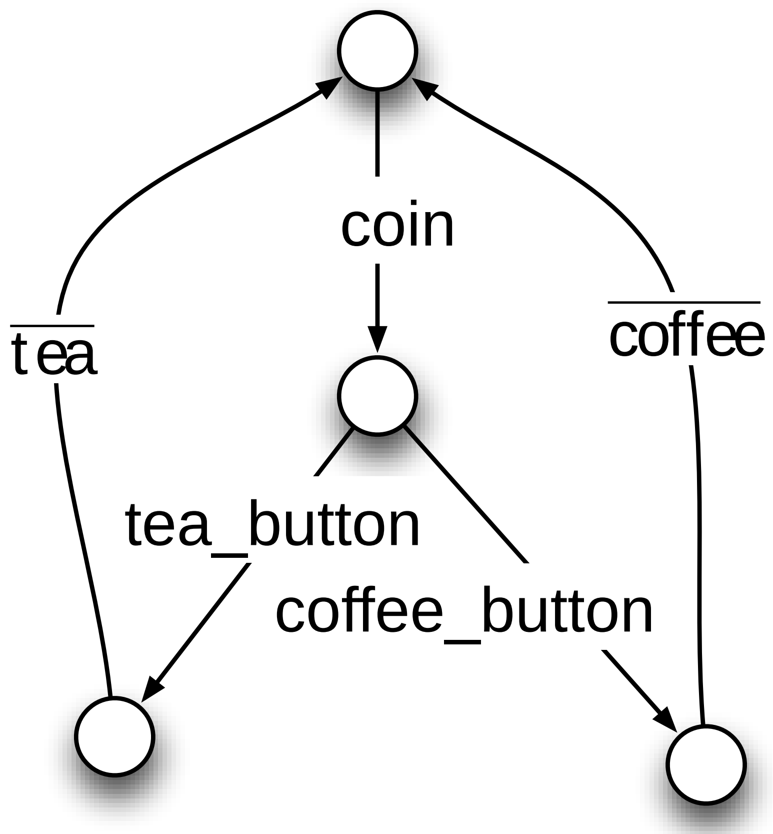
Hint: A LTS is very similar to a finite state automata. Two differences exist:

- In a LTS the set of states and transitions are not necessarily finite, or even countable.
- A finite-state automaton distinguishes a special "start" state and a set of special "final" states.

LTS Example: Simple vending machine

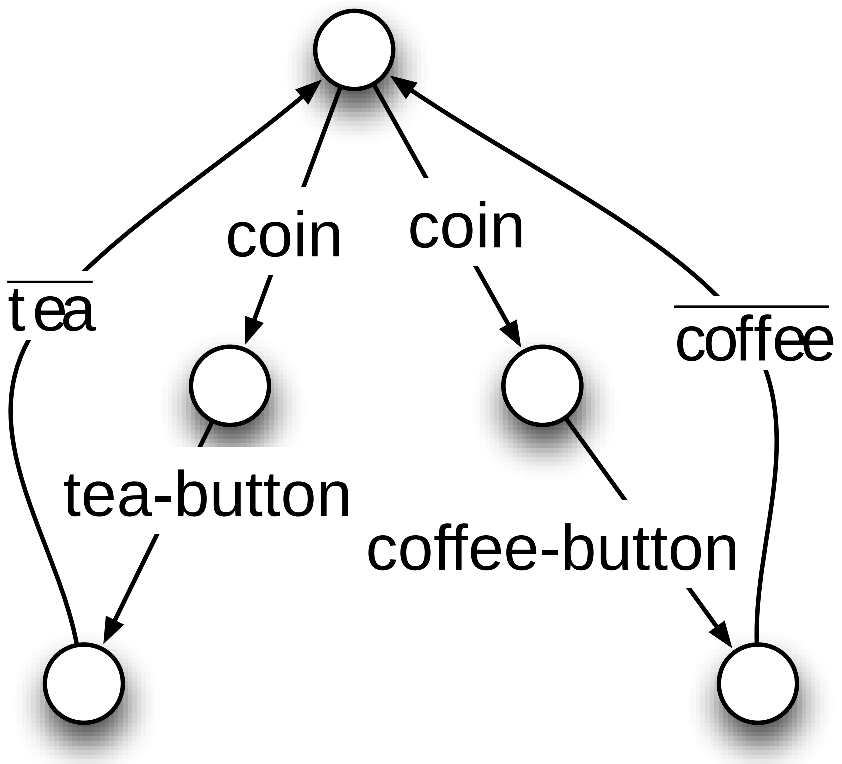
- A classical example: the vending machine
  - Aim is to model a very simple machine that
    - takes a coin
    - Outputs tea or coffee after pressing a button
    - May potentially behave non-deterministically
    - May output only one coffee or one tea

Labeled Transition System Example



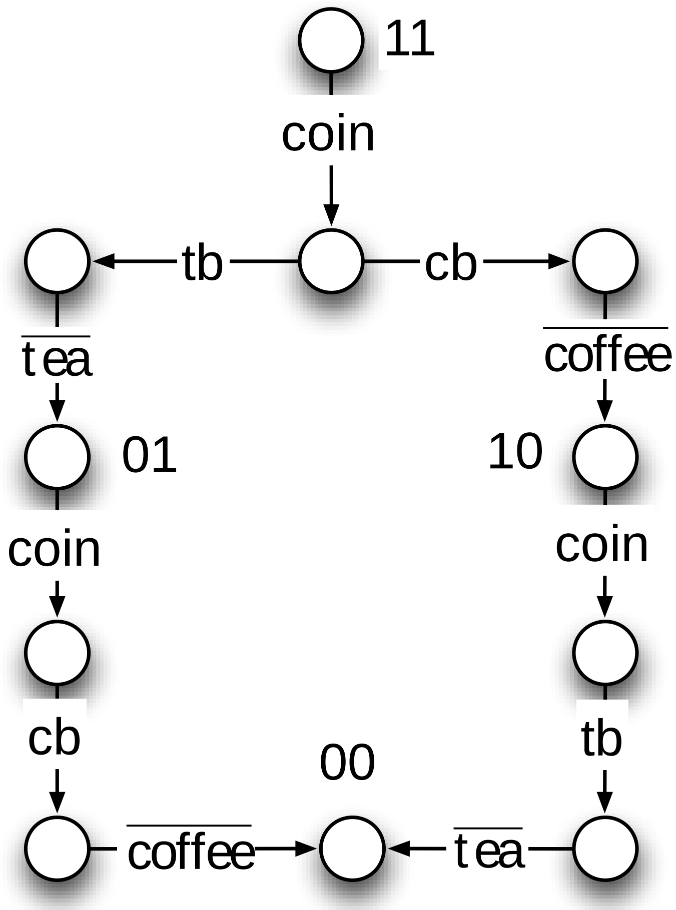
- A simple vending machine that outputs tea or coffee
- Overlined actions are outputs
- Other actions are inputs

Labeled Transition System. Example



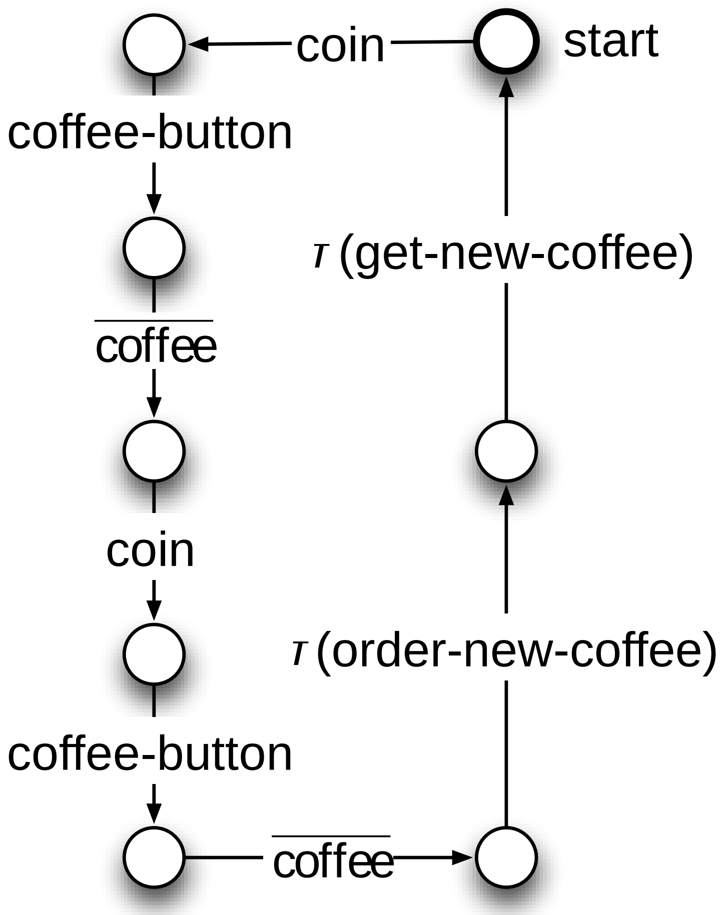
- Non-deterministic
  - Vending machine that makes a choice of beverages for the user

Labeled Transition System. Example



- A vending machine that outputs exactly one coffee and exactly one tea
- It already has 11 states

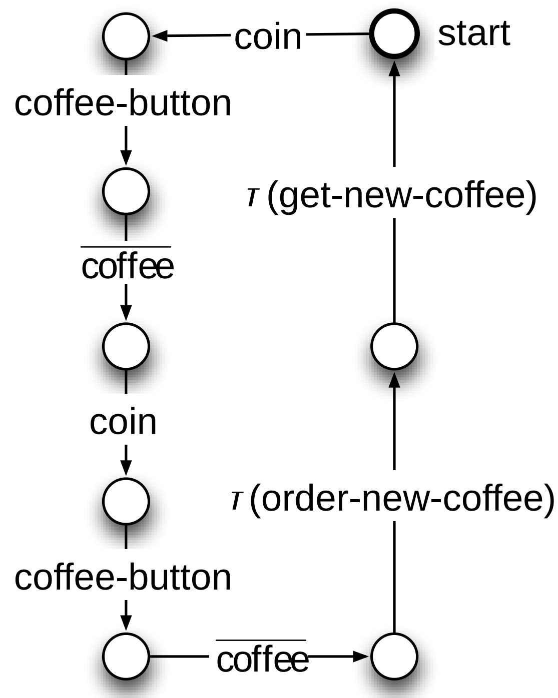
Labeled Transition System. Example



- Machine that outputs two coffees
- Then it orders new supplies
- After refilling, two more coffees available
- Ordering and supplying is not observable (τ)

Summary: Labeled Transition Systems

- Transition systems represent states and transitions between states
- Parallelism is not directly represented
- Strong similarity to automata
- Hard to model complex systems manually
- Good for algorithmic analysis like:
  - Does action a take place before action b?
  - Are there deadlocks in the system ? (deadlocks: nodes without outgoing edges)

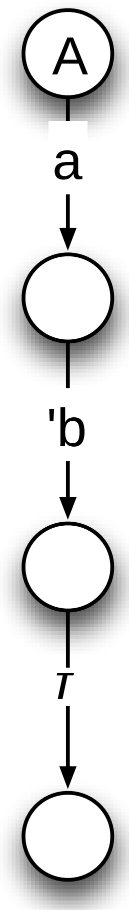


Process Algebras

- Process algebras are formalisms to model distributed systems
  - A process algebra consists of a set of actions and a set of operations defined on these actions
  - A process is a system with a specified behaviour
  - A typical process algebra
    - Consists of three types of actions: Send, Receive and Hidden
    - Operations like: sequence, exclusive or, parallel, restriction , …
    - Does not consists of data types, function

- Popular process algebras are
  - **CCS** (Calculus of communicating systems)
  - CSP (Communicating sequential processes)
  - Pi-Calculus
  - mCRL

Calculus of Communicating Systems (CCS). Informal Introduction.



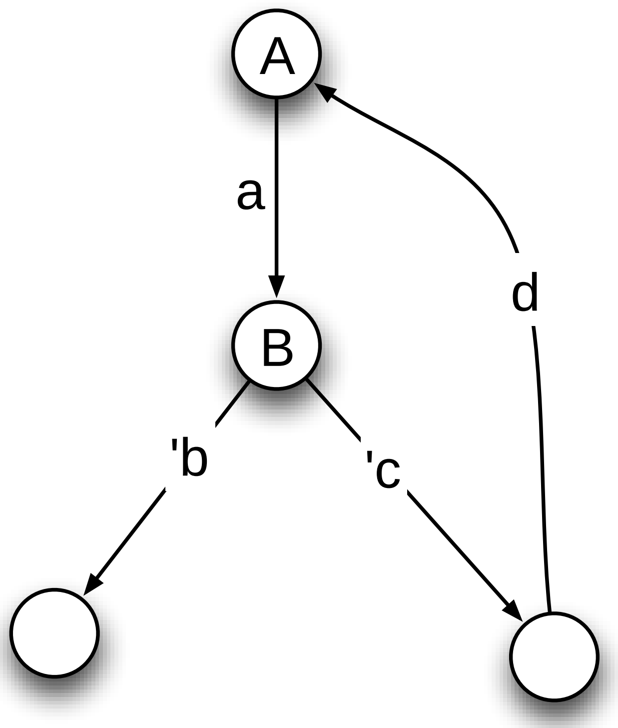
- Approach: Write transition systems in textual form.

A := a.'b.$\boldsymbol{\tau}$.**0**

Where

- A is called a process
- a is a receive action
- 'b is a send action
- . defines a sequence of actions
- $\boldsymbol{\tau}$ is a hidden action
- **0** is the nil operator

Calculus of Communicating Systems (CCS). Informal Introduction.



A := a.B

B := ‘b.**0** + ‘c.d.A

Where

- A, B are process identifiers
- + is an exclusive or operator

Calculus of Communicating Systems (CCS). Informal Introduction

$$\begin{tikzcd} & a.\mathbf{0} \mid b.\mathbf{0} \arrow[dl, "a"'] \arrow[dr, "b"] & \\ \mathbf{0} \mid b.\mathbf{0} \arrow[dr, "b"'] & & a.\mathbf{0} \mid \mathbf{0} \arrow[dl, "a"] \\ & \mathbf{0} \mid \mathbf{0} & \end{tikzcd}$$

$A := a.\mathbf{0} \mid b.\mathbf{0}$

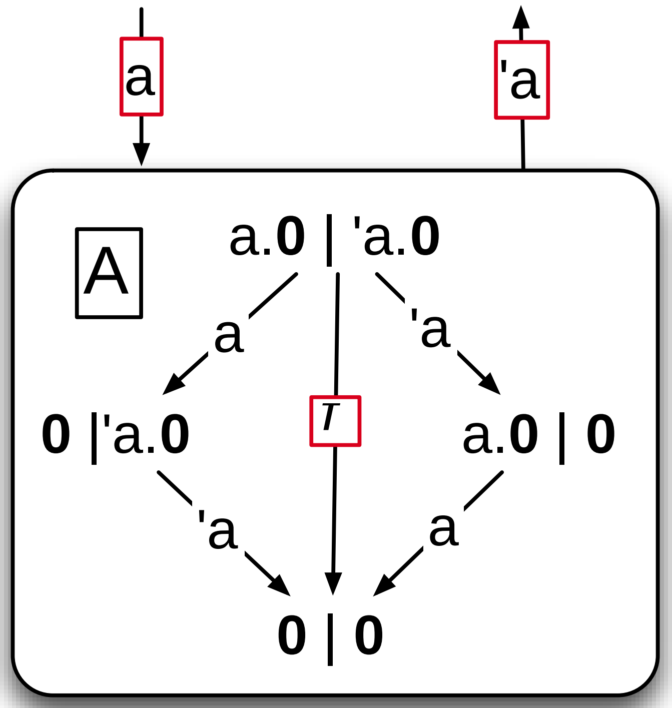
Where

- | is is the parallel operator

A can receive messages a **and** b

Remark: Simplified graphical LTS notation used.

Calculus of Communicating Systems (CCS). Informal Introduction

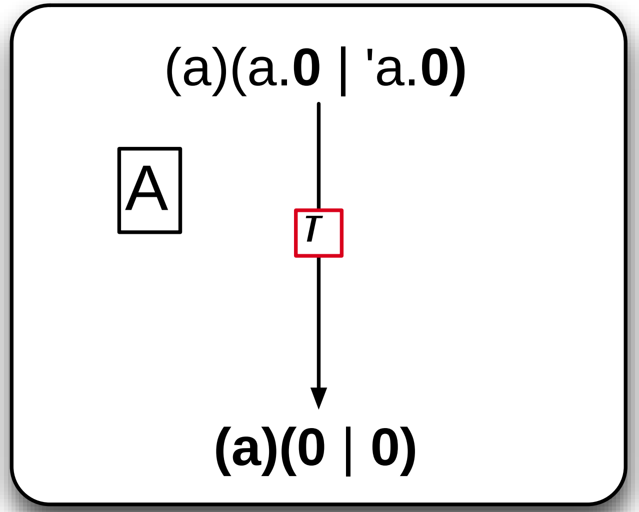


A := a.**0** | ‘a.**0**

The system A can receive a message a **and** send a message a **and** can synchronize itself by performing a τ action:

The left subsystem receives a message “a” send by the right subsystem.

Calculus of Communicating Systems (CCS) Informal Introduction



A := (a)(a.**0** | ‘a.**0**)

Where

- (a) is the restriction operator

The system A can only synchronize itself by performing a τ action because the action a is restricted to internal communications.

Calculus of Communicating Systems (CCS). Informal Introduction

$$
\begin{tikzcd}
(a.b.\mathbf{0})[a/c, b/d] \arrow[d, "c"] \\
(b.\mathbf{0})[a/c, b/d] \arrow[d, "d"] \\
(\mathbf{0})[a/c, b/d]
\end{tikzcd}
$$

- A := (a.b.**0**)[a/c, b/d]

Where

- [...] is the renaming operator

Actions ≠ the τ action can be renamed before performing them

Calculus of Communicating Systems (CCS). Informal Introduction

Remarks concerning the **renaming operator**

- Renamings of the form [a/b, b/c] are done independently. Specifically, all a are renamed to b and not to c.
- Renamings of the form [a/b][b/c] are done sequentially from left to right. ➔ All a are renamed to c.

A process identifier (K := P) is used to define an arbitrary behavior of a process.

- Example

$$\mathrm{Counter}_0 := \mathrm{up.Counter}_1$$
$$\mathrm{Counter}_n := \mathrm{up.Counter}_{n+1} + \mathrm{down.Counter}_{n-1} \quad (n > 0)$$

Calculus of Communicating Systems (CCS) Syntax

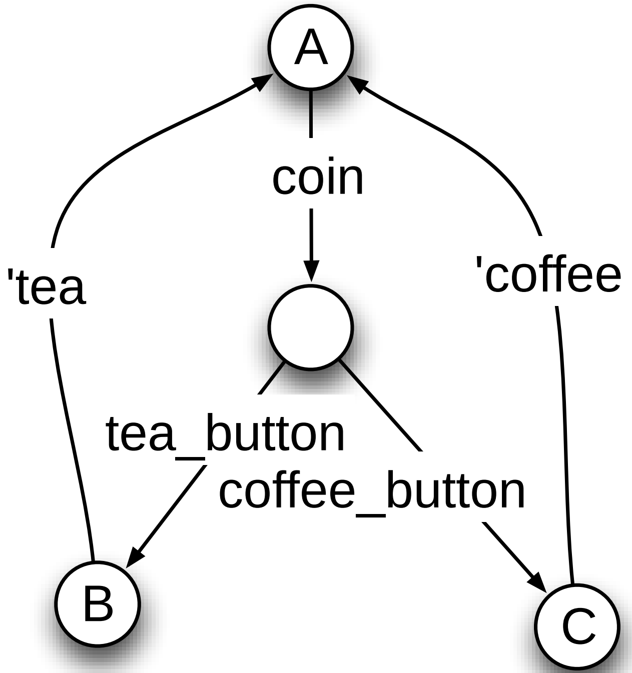
Definition 2 (CCS Syntax) *Let $\mathcal{L}$ be a set of labels and let $Act = \{\tau\} \cup \mathcal{L} \cup \{'a | a \in \mathcal{L}\}$ be the set of all actions. Then the syntax of process P is given by:*

Processes	P, Q	$::=$	$\mathbf{0}$	Inactive Process (Nil)
		$\mid$	$a.P$	Action (Act),
		$\mid$	$\sum_{i \in I} P_i$	Non-deterministic choice (Cho)
		$\mid$	$P \mid Q$	Parallel (Par)
		$\mid$	$(a_1, a_2, \ldots, a_n)P$	Restriction (Res), $a_i \in \mathcal{L}$
		$\mid$	K	Identifier (Ide)
		$\mid$	$P[f]$	Renaming (Ren) $f : \mathcal{L} \rightarrow \mathcal{L}$

Table 1: Syntax of the Calculus of Communicating Systems (CCS)

with $a, a_i \in Act$, $f : \mathcal{L} \rightarrow \mathcal{L}$, and $P, P_i \in \mathcal{P}$ where $\mathcal{P}$ is the set of processes.

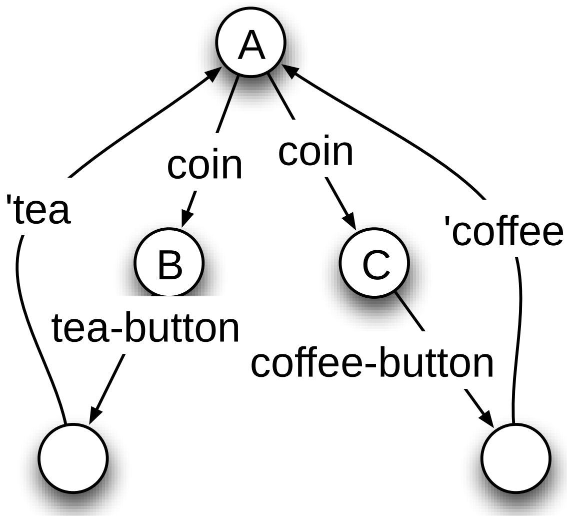
Calculus of Communicating Systems (CCS) Example



- A := coin.(tea_button.B + coffee_button.C)
- B := ‘tea.A
- C := ‘coffee.A

Calculus of Communicating Systems (CCS) Example

- Non deterministic behavior

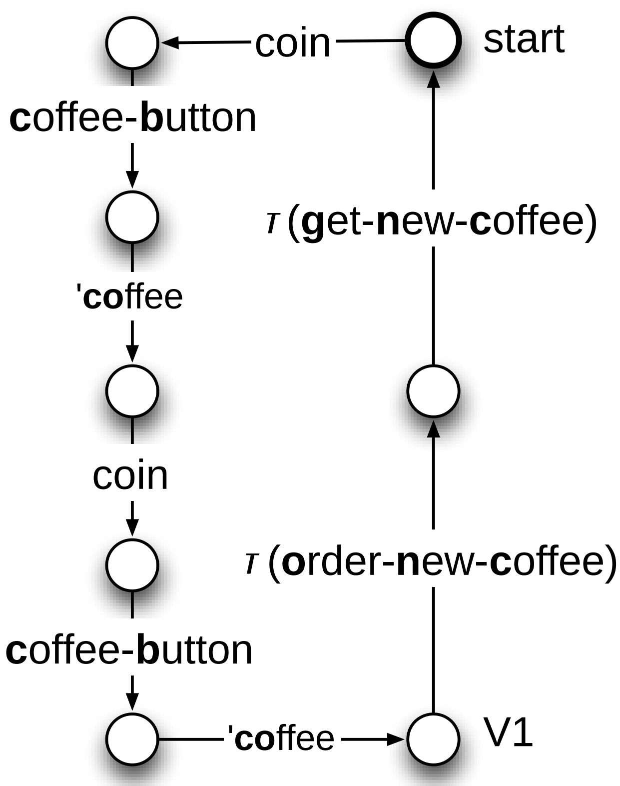


A := coin.B + coin.C

B := tea-button.‘tea.A

C := coffee-button.‘coffee.A

Calculus of Communicating Systems (CCS) Example



Vendor:= coin.cb.'co.coin.cb.'co.V1

V1 := 'onc.gnc.**Vendor**

Supplier := onc.'gnc.**Supplier**

System := (gnc, onc)(**Vendor**|**Supplier**)

- The System is the composition of Vendor and Supplier
- (gnc, onc) keeps the message exchange inside of System (restriction)
- External observers see restricted messages as τ operations

Structural, Operational Semantics

Definition 3 (SOS rules) ***S****tructural* ***O****perational* ***S****emantics (SOS) rules are rules of the type:*

$$\frac{Pre_i, \ldots, Pre_n}{Imp} \quad Cond_j, \ldots, Cond_k$$

The Pre_i are the premises and the $Cond_i$ are the conditions that are met if the implication Imp is satisfied. If $n = 0$ and $k = 0$ then there are no premises and conditions and Imp holds always. Rules without premises are called ***axioms****.*

Calculus of Communicating Systems (CCS) Semantics

Definition 4 (CCS Semantics) *The semantics of CCS is specified by the following rules:*

Name	Sym.	Structural Operational Semantic
Action (Act)	.	$\dfrac{}{a.P \xrightarrow{a} P}$ (1)
Process Identifier (Ide)	:=	$\dfrac{P \xrightarrow{a} P'}{A \xrightarrow{a} P'}\ if\ A := P$ (2)
Choice (Cho)	$\sum$	$\dfrac{P_j \xrightarrow{a} P'_j}{\sum_{i \in I} P_i \xrightarrow{a} P'_j}\ j \in I$ (3)
Parallel Composition (Par)	$\mid$	$\dfrac{P \xrightarrow{a} P'}{P \mid Q \xrightarrow{a} P' \mid Q}$ (4) $\dfrac{Q \xrightarrow{a} Q'}{P \mid Q \xrightarrow{a} P \mid Q'}$ (5) $\dfrac{P \xrightarrow{a} P',\ Q \xrightarrow{\bar{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$ (6)
Restriction (Res)	(a_j)	$\dfrac{P \xrightarrow{a_i} P'}{(a_j)P \xrightarrow{a_i} (a_j)P'}$ if $a_i \neq a_j$ (7)
Renaming (Ren)	f	$\dfrac{P \xrightarrow{a} P'}{P[f] \xrightarrow{f(a)} P'[f]}\ where f(\tau) = \tau, f(\bar{a}) = \overline{f(a)}$ (8)

Table 2: Semantics of CCS

Calculus of Communicating Systems (CCS). Semantics [1].

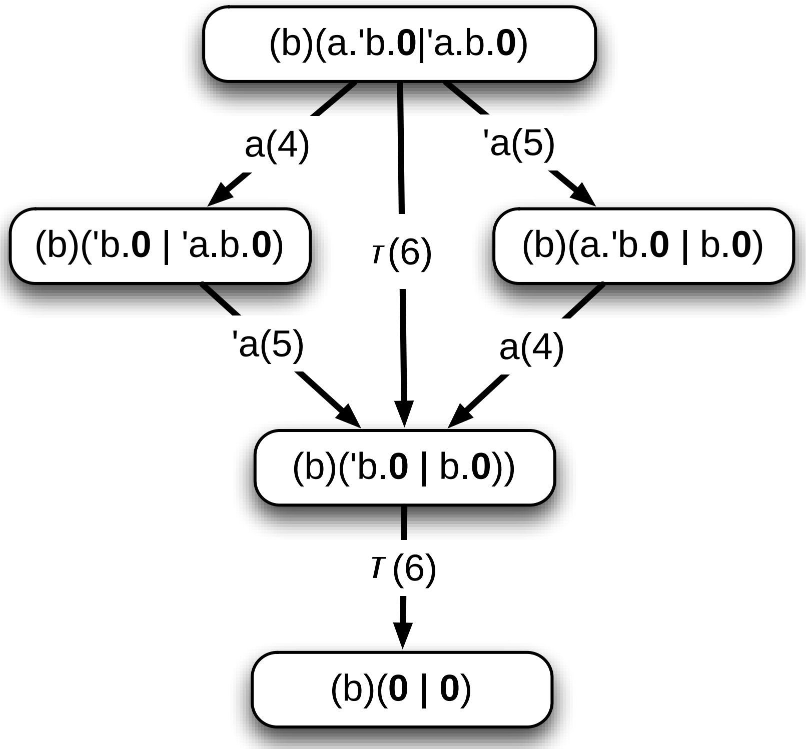
A CCS process P can always be associated with its transition system $LTS(P)$.

Definition 5 (Labeled Transition System of a Process [2]) *Let P be a CCS process. The transition system of P consists of:*

- *the set S of states which contains P itself and all processes which are reachable from P via transitions,* and
- *the transition relation $\longrightarrow$ between processes in S, which is specified by derivation rules given above.*

Remark: the transition system of a process may be infinite. An example is the process $A := a.(b.\mathbf{0}|A)$.

Inferring LTS from CCS Terms by Applying SOS Rules. Example



A := (b)(a.’b.**0** | ‘a.b.**0**)

The **l**abeled **t**ransition **s**ystem of the process **A** is inferred by applying the **s**tructural **o**perational **s**emantics rules. The “number” of each rule given above are written next to the action labels.

Calculus of Communicating Systems (CCS). Summary

- CCS is a simple process algebra with a formal SOS semantics.
- It consists of three different types of action: Send, Receive, Hidden /Internal Action.
- CCS consists of 6 operators defined on the actions: Sequence, non-deterministic choice, parallel operator, restriction, renaming, process identifier.
- Because of the actions and the operations it is suitable for modeling distributed systems.
- The semantics is defined by Structural Operational Semantics (SOS) rules.
- A Labeled Transition System (LTS) can be inferred out of CCS terms and can be used for system analysis.
- Although CCS is simple it has a high expressiveness. Nevertheless it is not expressive enough for most practical problems, it is suitable for educational purposes.

Behavioral Equivalences. Motivation.

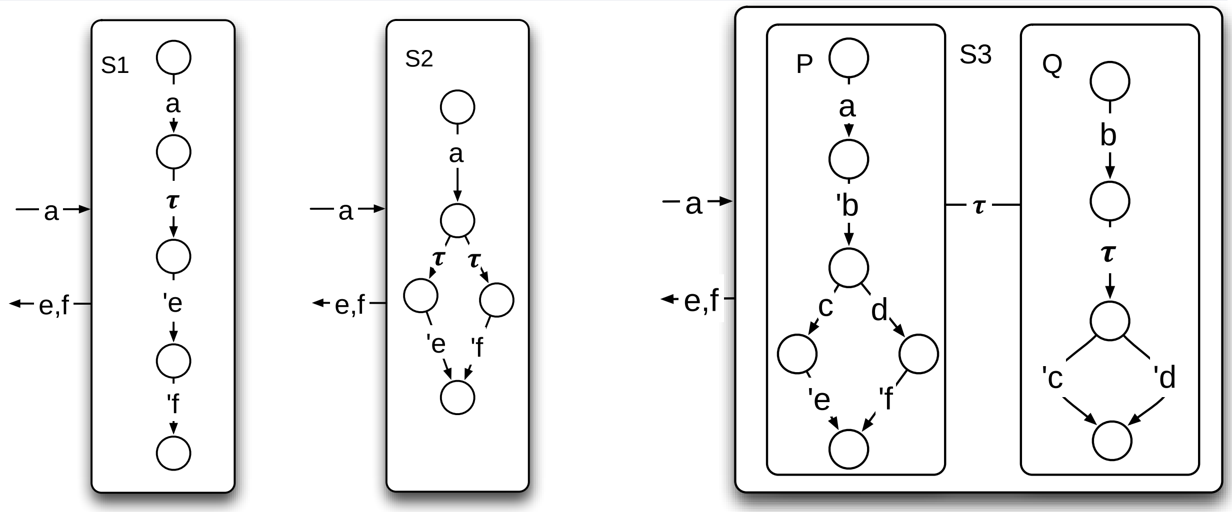
$$S1 = a.\tau.'\mathrm{e}.'\mathrm{f}.\mathbf{0}$$

$$S2 = a.(\tau.'\mathrm{e}.\mathbf{0} + \tau.'\mathrm{f}.\mathbf{0})$$

$$P = a.'\mathrm{b}.(\mathrm{c}.'\mathrm{e}.\mathbf{0} + \mathrm{d}.'\mathrm{f}.\mathbf{0})$$

$$Q = b.\tau.('\mathrm{c}.\mathbf{0} + '\mathrm{d}.\mathbf{0})$$

$$S3 = (b, c, d)\left(P \middle| Q\right)$$



- The services {S1, S2, S3} have the same static interface: receive a, send e,f
- But do they have the same dynamic interface (i.e. the internal behavior, or just behavior) hence one can be substituted by another ?
- Remark: Service-compositions are also services. Therefore arbitrary complex services can be build by service compositions.

  ➔ Finding services with the same behavior manually is not trivial.

Behavioral Equivalences

- Let ***P*** *be* a set of processes P with |***P***| > 1.
  - How to figure out if two or more processes have an equivalent behavior although they might be syntactically different ?
  - How to define a behavioral equivalence ?

Definition 6 (Strong Bisimulation) *A binary relation* R *over the set of states of an* LTS *is a bisimulation iff whenever* $(s_1, s_2) \in \mathrm{R}$ *and* $\alpha \in$ ***Act*** *is an action:*

- *if* $s_1 \xrightarrow{\alpha} s_1'$, *then there is a transition* $s_2 \xrightarrow{\alpha} s_2'$ *such that* $(s_1', s_2') \in \mathrm{R}$;
- *if* $s_2 \xrightarrow{\alpha} s_2'$, *then there is a transition* $s_1 \xrightarrow{\alpha} s_1'$ *such that* $(s_1', s_2') \in \mathrm{R}$.

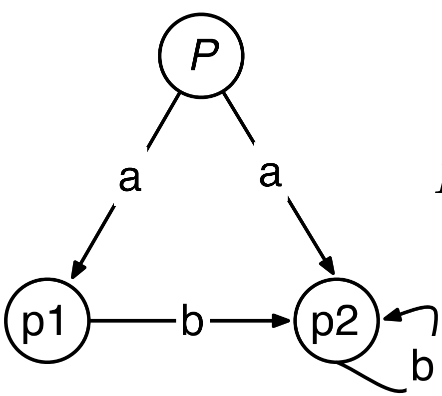
Two states s *and* s' *are bisimilar, written* $s \sim s'$, *iff there is a bisimulation that relates them. Henceforth the relation* $\sim$ *will be referred to as strong bisimulation equivalence or strong bisimulation.*

Behavioral Equivalences

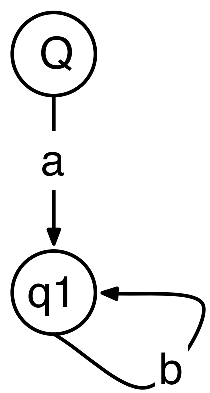
Definition 7 (Strong Bisimilarity of Processes) *Two CCS processes P, Q are called* *strongly bisimilar* *(in symbols: $P \sim Q$) whenever their states in the transition systems of P and Q are* *strongly bisimilar.*

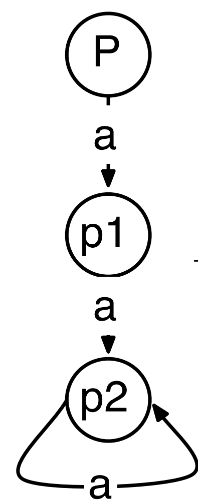
To prove that two processes are related by $\sim$ it suffices to exhibit a strong bisimulation that relates them.

Examples

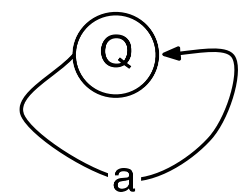


$P \sim Q$



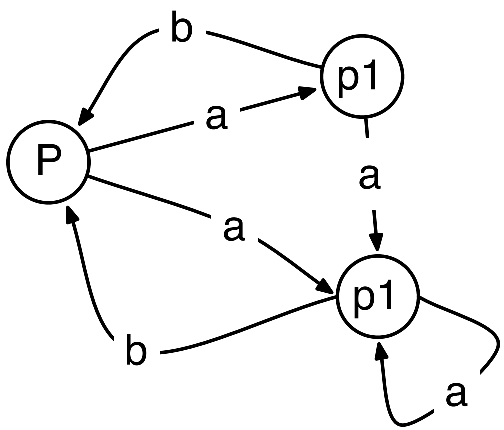


$P \sim Q$

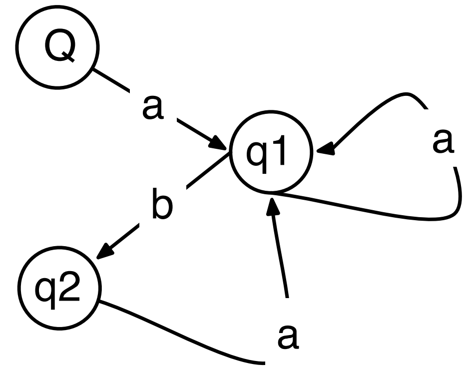


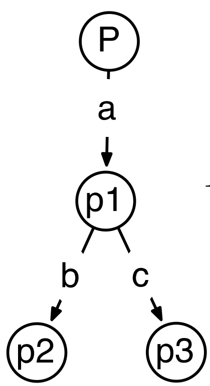
Behavioral Equivalences

- More examples for strong bisimilarity

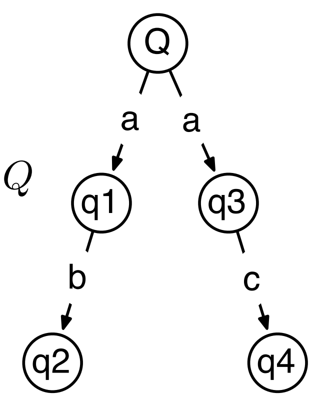


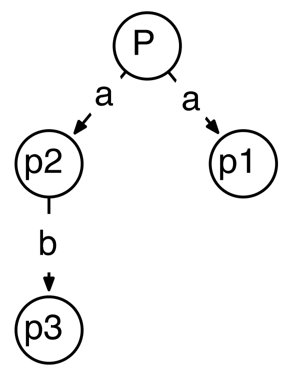
$P \sim Q$



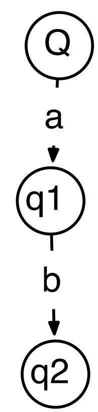


$P \not\sim Q$





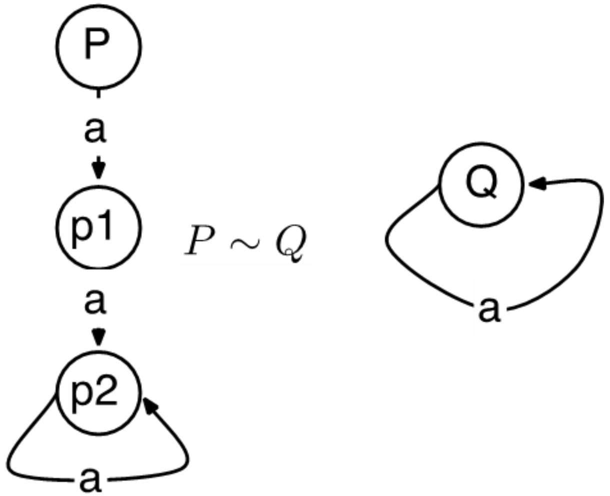
$P \not\sim Q$



Behavioral Equivalences

- Strong bisimilarity takes every τ into consideration.
- Therefore it holds for example: $a.\mathbf{0} \nsim a.\tau.\mathbf{0} \nsim a.\tau.\tau.\mathbf{0}$
- τ is an unobservable behavior and should not always be taken into account for behavioral equivalences checks

➔ A coarser equivalence is needed, i.e. more states are related to each other



Behavioral Equivalences

Definition 8 (Weak transition relation) *For a set* $\texttt{Act}$ *of actions we define the following relations:*

- $s_1 \stackrel{\epsilon}{\Longrightarrow} s_2$ *if and only if* $s_1(\stackrel{\tau}{\longrightarrow})^* s_2$, *i.e.*, $s_1 \stackrel{\tau}{\longrightarrow} \ldots \stackrel{\tau}{\longrightarrow} s_2$.
- *For an* $\alpha \in \texttt{Act}$ *we have* $s_1 \stackrel{\alpha}{\Longrightarrow} s_2$ *iff* $s_1 \stackrel{\epsilon}{\Longrightarrow}\stackrel{\alpha}{\longrightarrow}\stackrel{\epsilon}{\Longrightarrow} s_2$.

Standard transitions will in the following also be called strong transitions in order to distinguish them from weak transitions.

Definition 9 (Weak Bisimulation) *A binary relation* R *over the set of states of an* LTS *is a weak bisimulation iff whenever* $(s_1, s_2) \in \mathrm{R}$ *and* $\alpha \in \texttt{Act}$ *is an action:*

- *if* $s_1 \stackrel{\alpha}{\longrightarrow} s_1'$, *then there is a transition* $s_2 \stackrel{\hat{\alpha}}{\Longrightarrow} s_2'$ *such that* $(s_1', s_2') \in \mathrm{R}$;
- *if* $s_2 \stackrel{\alpha}{\longrightarrow} s_2'$, *then there is a transition* $s_1 \stackrel{\hat{\alpha}}{\Longrightarrow} s_1'$ *such that* $(s_1', s_2') \in \mathrm{R}$.

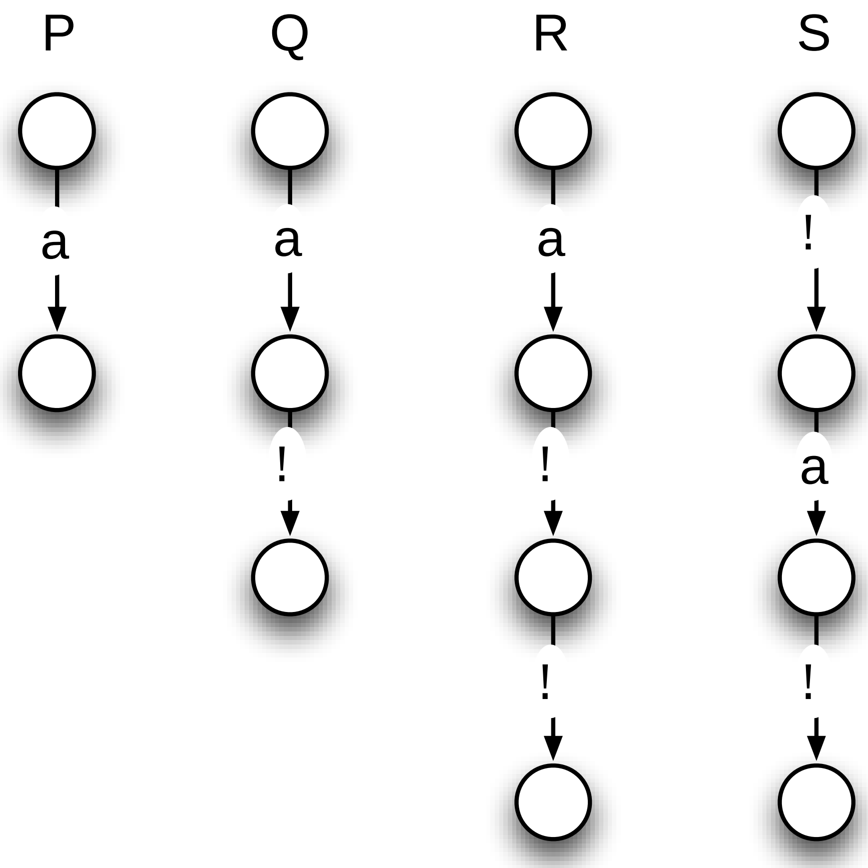
Where $\hat{\alpha} = \alpha$, *whenever* $\alpha \in \texttt{Act}\backslash\{\tau\}$, *and* $\hat{\tau} = \epsilon$.

Two states s *and* s' *are weakly bisimilar, written* $s \approx s'$, *iff there is a weak bisimulation that relates them. Henceforth the relation* $\approx$ *will be referred to as weak bisimilarity.*

Behavioral Equivalences

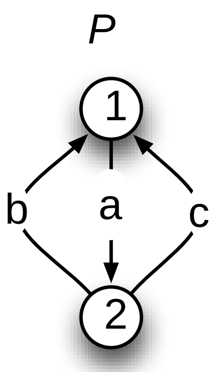
Definition 10 (Weak Bisimilarity of Processes) *Two CCS processes P, Q are called weakly bisimilar (in symbols: $P \approx Q$) whenever their states in the transition systems of P and Q are weakly bisimilar.*

- Example: Every process of P, Q, R, S is pairwise weakly equivalent!



Behavioral Equivalences

- Example 2 weak bisimulation

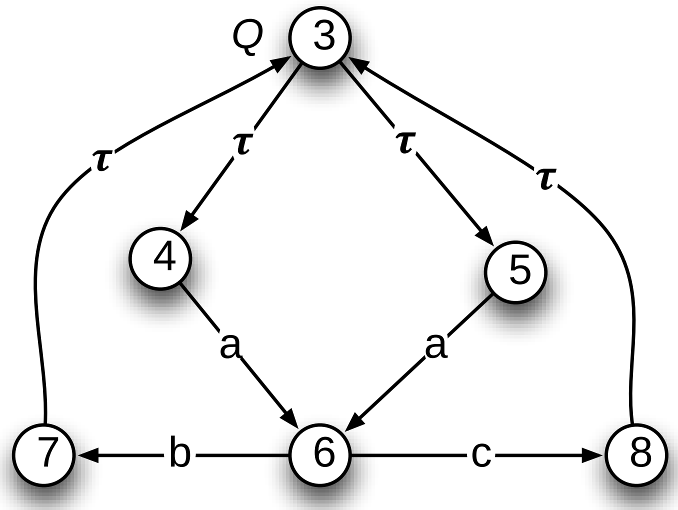


$$P = a.(b.P + c.P)$$

$$Q = \tau.a.Q_1 + \tau.a.Q_1$$

$$Q_1 = b.\tau.Q + c.\tau.Q$$

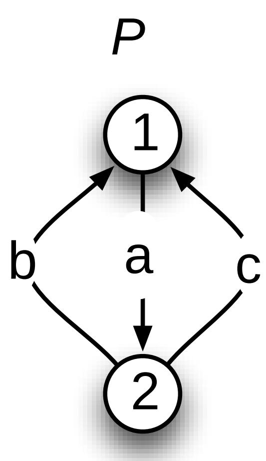
$$P \approx Q$$



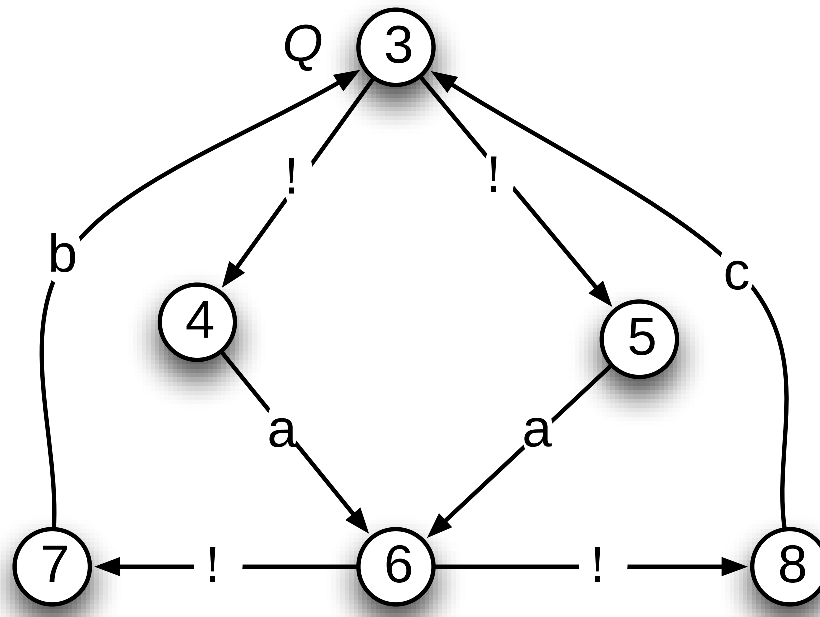
$$R = \{(1,3), (1,4), (1,5), (2,6), (1,7), (1,8)\}$$

Behavioral Equivalences

- Example 3: not a weak bisimulation



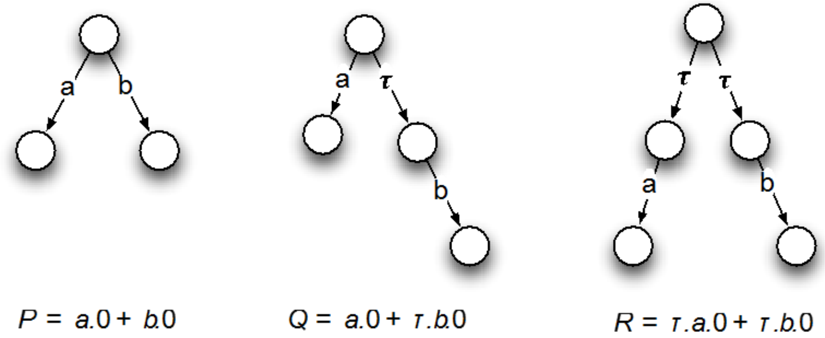
$$P = a.(b.P + c.P)$$
$$Q = \tau.a.Q_1 + \tau.a.Q_1$$
$$Q_1 = \tau.b.Q + \tau.c.Q$$
$$P \not\approx Q$$



System P can perform a strong transition $1 \xrightarrow{a} 2$ that can be simulated by system Q with $3 \overset{a}{\Longrightarrow} 7$ (not only!). Then in system P a transition can take place $2 \xrightarrow{c} 1$ and state 7 does not allow any further weak c-transition.

Behavioral Equivalences

- Weak bisimulation example 4:
- Why not just ignore all τ 's ?



- No process of P, Q, R is pairwise weakly equivalent!

Behavioral Equivalences

- It can be shown that strong bisimilarity is preserved by the CCS operators, i.e., it is a congruence.

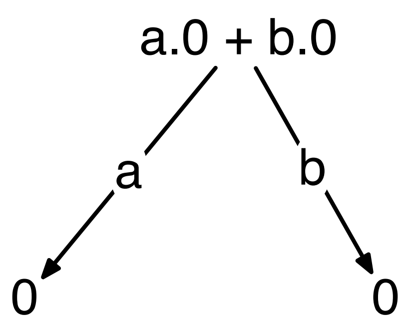
Proposition 1 ($\sim$ is a congruence)
Assume that $P_1 \sim P_2$. This implies:

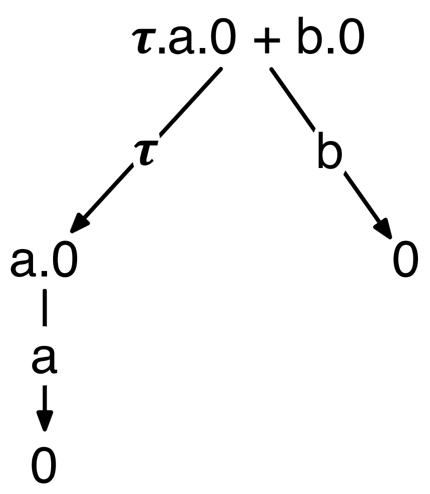
$$a.P_1 \sim a.P_2$$
$$P_1 + Q \sim P_2 + Q$$
$$P_1|Q \sim P_2|Q$$
$$(L)P_1 \sim (L)P_2$$
$$P_1[f] \sim P_2[f]$$

Behavioral Equivalences

- Weak bisimulation is not preserved by non-deterministic choice.

$$\text{It holds} \qquad a.\mathbf{0} \approx \tau.a.\mathbf{0}$$
$$\text{but} \qquad a.\mathbf{0} + b.\mathbf{0} \not\approx \tau.a.\mathbf{0} + b.\mathbf{0}$$





Behavioral Equivalences

- Problem: The behavior of the overall system can be modified by substituting one subsystem (process) by another one.
- Solution: Refine weak bisimilarity so that it becomes a congruence, i.e., it is preserved by CCS operators.
  - The resulting equivalence is called observational congruence

Behavioral Equivalences

Definition 11 (Observational Congruence) .

- *Let* $P \stackrel{\tau}{\Longrightarrow} Q$ *iff* $P \stackrel{\tau}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^* Q$

Two CCS processes P, Q *are called observationally congruent (in symbols:* $P \approx^c Q$*) whenever* $\alpha \in \mathtt{Act} \cup \{\tau\}$ *is an action and:*

- *if* $P \stackrel{\alpha}{\longrightarrow} P'$*, then there is a transition* $Q \stackrel{\alpha}{\Longrightarrow} Q'$ *and* $P' \approx Q'$*;*
- *if* $Q \stackrel{\alpha}{\longrightarrow} Q'$*, then there is a transition* $P \stackrel{\alpha}{\Longrightarrow} P'$ *and* $Q' \approx P'$*.*

Hint: Only the first τ of one process has to be simulated by an τ of the other process. The remaining subprocesses only have to satisfy the weak bisimulation ($P' \approx Q'$).

- Do not mix it up with observational equivalence which is a common synonym for weak bisimulation.

Synchronous vs. asynchronous communication.

- Synchronous vs. asynchronous communication
- Communication in CCS is synchronous. The communication partners wait for each other and send and receive take place at once

Synchronous communication

$$P := \text{'a}.b.P'$$
$$Q := a.Q'$$
$$(a)\big(P \mid Q\big) \xrightarrow{\tau} (a)\big(b.P' \mid Q'\big)$$

- Asynchronous communication can be carried out in several ways:

1) Asynchronous communication by a parallel split of the send operation. Capacity is infinite.

$$P := \big(\text{'a}.\mathbf{0} \mid b.P'\big)$$
$$Q := a.Q'$$
$$(a)\big(P \mid Q\big) \xrightarrow{b} (a)\big((\text{'a}.\mathbf{0}|P') \mid a.Q\big)$$

Summary and Context

- There are currently no wide-spread programming languages that support holistic specification of a distributed system
  - such languages were more wide spread, may become wide spread again
  - on the other hand: push paradigm / decoupling is a counter trend!
- „holistic models“ and „formal checking“are relevant
  - distributed systems difficult to understand (parallel activities vs. brain!)
  - business processes → Internet-of-Services: „inside“ of foreign service?
  - required: formal description of the „semantics“ of processes
- LTLs good for checking properties, but difficult to read/use
- calculi have been around „forever“, are still relevant
  - CCS, CSP, alpha, pi, … (we just looked at CCS)
  - supported: checking of properties (via LTL), checking of „equivalence“ (strong / weak bisimulation, observational congruence)