ETH Zürich
Chair of Software Engineering

Concepts of Concurrent Computation

Bertrand Meyer
Sebastian Nanz

Lecture 8: CCS

Process Calculi

- **Question**: Why do we need a theoretical model of concurrent computation?
- Turing machines or the λ-calculus have proved to be useful models of sequential systems
- Abstracting away from implementation details yields general insights into programming and computation
- Process calculi help to focus on the essence of concurrent systems: interaction

The Calculus of Communicating Systems (CCS)

- We study the Calculus of Communicating Systems (CCS)
- Introduced by [Milner 1980]
- Milner's general model:
  - A concurrent system is a collection of processes
  - A process is an independent agent that may perform internal activities in isolation or may interact with the environment to perform shared activities
- Milner's insight: Concurrent processes have an algebraic structure

$$\boxed{P_1}\ op\ \boxed{P_2} \Rightarrow \boxed{P_1\ op\ P_2}$$

- This is why a process calculus is sometime called a process algebra

Introductory Example: A Simple Process

- A coffee and tea machine may take an order for either tea or coffee, accept the appropriate payment, pour the ordered drink, and terminate:

$$tea.coin.\overline{cup\_of\_tea}.0 + coffee.coin.coin.\overline{cup\_of\_coffee}.0$$

- We have the following elements of syntax:
  - Actions: *tea*, $\overline{cup\_of\_tea}$, etc.
  - Sequential composition: the dot "." (first do action *tea*, then *coin*, ...)
  - Non-deterministic choice: the plus "+" (either do *tea* or *coffee*)
  - Terminated process: 0

Introductory Example: Execution of a Simple Process

- When a process executes it performs some action, and becomes a new process
- The execution of an action a is symbolized by a transition $\xrightarrow{a}$

$$
\begin{aligned}
& tea.coin.\overline{cup\_of\_tea}.0 + coffee.coin.coin.\overline{cup\_of\_coffee}.0 \\
\xrightarrow{tea} \quad & coin.\overline{cup\_of\_tea}.0 \\
\xrightarrow{coin} \quad & \overline{cup\_of\_tea}.0 \\
\xrightarrow{\overline{cup\_of\_tea}} \quad & 0
\end{aligned}
$$

Syntax of CCS

Syntax of CCS

- **Goal**: In the following we introduce the syntax of CCS step-by-step

Basic principle

1. Define atomic processes that model the simplest possible behavior
2. Define composition operators that build more complex behavior from simpler ones

The Terminal Process

The simplest possible behavior is no behavior

Terminal process

We write 0 (pronounced “nil”) for the terminal or inactive process

- 0 models a system that is either deadlocked or has terminated
- 0 is the only atomic process of CCS

Names and Actions

- We assume an infinite set $\mathcal{A}$ of port names, and a set $\bar{\mathcal{A}} = \{\bar{a} \mid a \in \mathcal{A}\}$ of complementary port names

Input actions

When modeling we use a name a to denote an input action, i.e. the receiving of input from the associated port a

Output actions

We use a co-name $\bar{a}$ to denote an output action, i.e. the sending of output to the associated port a

Internal actions

We use τ to denote the distinguished internal action

- The set of actions Act is given by $Act = \mathcal{A} \cup \bar{\mathcal{A}} \cup \{\tau\}$

Action Prefixing

The simplest actual behavior is sequential behavior

Action prefixing

If P is a process we write

$$\alpha.P$$

to denote the prefixing of P with the action α

- $\alpha.P$ models a system that is ready to perform the action, α, and then behaves as P, i.e.

$$\alpha.P \xrightarrow{\alpha} P$$

Example: Action Prefixing

A process that starts a timer, performs some internal computation, and then stops the timer:

$$\overline{go}.\tau.\overline{stop}.0 \xrightarrow{\overline{go}} \tau.\overline{stop}.0 \xrightarrow{\tau} \overline{stop}.0 \xrightarrow{\overline{stop}} 0$$

Process Interfaces

Interfaces

The set of input and output actions that a process P may perform in isolation constitutes the interface of P

- The interface enumerates the ports that P may use to interact with the environment

Example: The interface of the coffee and tea machine is:

$$tea, coffee, coin, \overline{cup\_of\_tea}, \overline{cup\_of\_coffee}$$

Non-deterministic Choice

A more advanced sequential behavior is that of alternative behaviors

Non-deterministic choice

If P and Q are processes then we write

$$P + Q$$

to denote the non-deterministic choice between P and Q

- $P + Q$ models a process that can either behave as P (discarding Q) or as Q (discarding P)

Example: Non-deterministic Choice

$$tea.coin.\overline{cup\_of\_tea}.0 + coffee.coin.coin.\overline{cup\_of\_coffee}.0$$
$$\xrightarrow{tea} \quad coin.\overline{cup\_of\_tea}.$$

Note that:

- prefixing binds harder than plus and
- the choice is made by the initial *coffee*/*tea* button press

Process Constants and Recursion

The most advanced sequential behavior is the recursive behavior

Process constants

A process may be the invocation of a process constant, $\mathrm{K} \in \mathcal{K}$

This is only meaningful if K is defined beforehand

Recursive definition

If K is a process constant and P is a process we write

$$\mathrm{K} \stackrel{\text{def}}{=} P$$

to give a recursive definition of the behavior of K
(recursive if P invokes K)

Example: Recursion (1)

A system clock, SC, sends out regular clock signals forever:

$$\mathrm{SC} \stackrel{\text{def}}{=} \overline{\mathit{tick}}.\mathrm{SC}$$

The system SC may behave as:

$$\overline{\mathit{tick}}.\mathrm{SC} \xrightarrow{\overline{\mathit{tick}}} \mathrm{SC} \xrightarrow{\overline{\mathit{tick}}} \dots$$

Example: Recursion (2)

A fully automatic coffee and tea machine CTM works as follows:

$$\mathrm{CTM} \stackrel{\text{def}}{=} \textit{tea}.\textit{coin}.\overline{\textit{cup\_of\_tea}}.\mathrm{CTM} + \textit{coffee}.\textit{coin}.\textit{coin}.\overline{\textit{cup\_of\_coffee}}.\mathrm{CTM}$$

The system CTM may e.g. do:

$$\begin{array}{ll} & \textit{tea}.\textit{coin}.\overline{\textit{cup\_of\_tea}}.\mathrm{CTM} + \textit{coffee}.\textit{coin}.\textit{coin}.\overline{\textit{cup\_of\_coffee}}.\mathrm{CTM} \\ \xrightarrow{\textit{tea}} & \textit{coin}.\overline{\textit{cup\_of\_tea}}.\mathrm{CTM} \\ \xrightarrow{\textit{coin}} & \overline{\textit{cup\_of\_tea}}.\mathrm{CTM} \\ \xrightarrow{\overline{\textit{cup\_of\_tea}}} & \mathrm{CTM} \\ \xrightarrow{\alpha} & \ldots \end{array}$$

This will serve drinks ad infinitum

Parallel Composition

Finally: concurrent behavior

Parallel composition

If P and Q are processes we write

$$P \mid Q$$

to denote the parallel composition of P and Q

- $P \mid Q$ models a process that behaves like P and Q in parallel:
  - Each may proceed independently
  - If P is ready to perform an action a and Q is ready to perform the complementary action $\overline{a}$, they may interact

Example: Parallel Composition

Recall the coffee and tea machine:

$$\mathrm{CTM} \stackrel{\text{def}}{=} \mathit{tea}.\mathit{coin}.\overline{\mathit{cup\_of\_tea}}.\mathrm{CTM} + \mathit{coffee}.\mathit{coin}.\mathit{coin}.\overline{\mathit{cup\_of\_coffee}}.\mathrm{CTM}$$

Now consider the regular customer – the Computer Scientist, CS:

$$\begin{aligned} \mathrm{CS} \stackrel{\text{def}}{=} & \quad \overline{\mathit{tea}}.\overline{\mathit{coin}}.\mathit{cup\_of\_tea}.\overline{\mathit{teach}}.\mathrm{CS} \\ + & \quad \overline{\mathit{coffee}}.\overline{\mathit{coin}}.\overline{\mathit{coin}}.\mathit{cup\_of\_coffee}.\overline{\mathit{publish}}.\mathrm{CS} \end{aligned}$$

Example: Parallel Composition

Recall the coffee and tea machine:

$$\mathrm{CTM} \stackrel{\text{def}}{=} \mathit{tea}.\mathit{coin}.\overline{\mathit{cup\_of\_tea}}.\mathrm{CTM} + \mathit{coffee}.\mathit{coin}.\mathit{coin}.\overline{\mathit{cup\_of\_coffee}}.\mathrm{CTM}$$

Now consider the regular customer – the Computer Scientist, CS:

$$\begin{aligned}\mathrm{CS} \stackrel{\text{def}}{=} & \quad \overline{\mathit{tea}}.\overline{\mathit{coin}}.\mathit{cup\_of\_tea}.\overline{\mathit{teach}}.\mathrm{CS} \\ + & \quad \overline{\mathit{coffee}}.\overline{\mathit{coin}}.\overline{\mathit{coin}}.\mathit{cup\_of\_coffee}.\overline{\mathit{publish}}.\mathrm{CS}\end{aligned}$$

- CS must drink coffee to publish
- CS can only teach on tea

Example: Parallel Composition

On an average Tuesday morning the system

$$\mathrm{CTM} \,|\, \mathrm{CS}$$

is likely to behave as follows:

$$
\begin{array}{ll}
 & (\textit{tea}.\textit{coin}.\overline{\textit{cup\_of\_tea}}.\mathrm{CTM} + \textit{coffee}.\textit{coin}.\textit{coin}.\overline{\textit{cup\_of\_coffee}}.\mathrm{CTM}) \\
| & (\overline{\textit{tea}}.\overline{\textit{coin}}.\textit{cup\_of\_tea}.\overline{\textit{teach}}.\mathrm{CS} + \overline{\textit{coffee}}.\overline{\textit{coin}}.\overline{\textit{coin}}.\textit{cup\_of\_coffee}.\overline{\textit{publish}}.\mathrm{CS}) \\
\xrightarrow{\tau} & (\textit{coin}.\overline{\textit{cup\_of\_tea}}.\mathrm{CTM}) \,|\, (\overline{\textit{coin}}.\textit{cup\_of\_tea}.\overline{\textit{teach}}.\mathrm{CS}) \\
\xrightarrow{\tau} & (\overline{\textit{cup\_of\_tea}}.\mathrm{CTM}) \,|\, (\textit{cup\_of\_tea}.\overline{\textit{teach}}.\mathrm{CS}) \\
\xrightarrow{\tau} & \mathrm{CTM} \,|\, (\overline{\textit{teach}}.\mathrm{CS}) \\
\xrightarrow{\overline{\textit{teach}}} & \mathrm{CTM} \,|\, \mathrm{CS}
\end{array}
$$

- Note that the synchronisation of actions such as $\textit{tea}/\overline{\textit{tea}}$ is expressed by a τ-action (i.e. regarded as an internal step)

Restriction

We control unwanted interactions with the environment by restricting the scope of port names

Restriction

if P is a process and A is a set of port names we write

$$P \setminus A$$

for the restriction of the scope of each name in A to P

- Removes each name $a \in A$ and the corresponding co-name $\overline{a}$ from the interface of P
- Makes each name $a \in A$ and the corresponding co-name $\overline{a}$ inaccessible to the environment

Example: Restriction

- Recall the coffee and tea machine and the computer scientist:

$$\mathrm{CTM} \mid \mathrm{CS}$$

- Restricting the coffee and tea machine on *coffee* makes the *coffee*-button inaccessible to the computer scientist:

$$(\mathrm{CTM} \setminus \{\mathit{coffee}\}) \mid \mathrm{CS}$$

- As a consequence CS can only teach, and never publish

Summary: Syntax of CCS

$$
\begin{array}{llll}
P ::= & K & | & \text{process constants } (K \in \mathcal{K}) \\
 & \alpha.P & | & \text{prefixing } (\alpha \in Act) \\
 & \sum_{i \in I} P_i & | & \text{summation } (I \text{ is an arbitrary index set}) \\
 & P_1 | P_2 & | & \text{parallel composition} \\
 & P \setminus L & & \text{restriction } (L \subseteq \mathcal{A})
\end{array}
$$

The set of all terms generated by the abstract syntax is called
CCS process expressions

Notation

$$P_1 + P_2 = \sum_{i \in \{1,2\}} P_i \qquad Nil = 0 = \sum_{i \in \emptyset} P_i$$

CCS Program

CCS program

A collection of defining equations of the form

$$K \stackrel{\text{def}}{=} P$$

where $K \in \mathcal{K}$ is a process constant and $P \in \mathcal{P}$ is a CCS process expression

- Only one defining equation per process constant
- Recursion is allowed: e.g. $A \stackrel{\text{def}}{=} \overline{a}.A \mid A$
- Note that the program itself gives only the definitions of process constants: we can only execute processes (which can however mention the process constants defined in the program)

Exercise: Syntax of CCS

Which of the following expressions are correctly built CCS expressions? Assume that A, B are process constants and that a, b are port names.

- $a.b.\mathrm{A} + \mathrm{B}$
- $(a.0 + \overline{a}.\mathrm{A}) \setminus \{a, b\}$
- $(a.0 \,|\, \overline{a}.\mathrm{A}) \setminus \{a, \tau\}$
- $\tau.\tau.\mathrm{B} + 0$
- $(a.b.\mathrm{A} + \overline{a}.0) \,|\, \mathrm{B}$
- $(a.b.\mathrm{A} + \overline{a}.0).\mathrm{B}$

Exercise: Syntax of CCS

Which of the following expressions are correctly built CCS expressions? Assume that A, B are process constants and that a, b are port names.

- $a.b.\mathrm{A} + \mathrm{B}$ ✓
- $(a.0 + \overline{a}.\mathrm{A}) \setminus \{a, b\}$
- $(a.0 \,|\, \overline{a}.\mathrm{A}) \setminus \{a, \tau\}$
- $\tau.\tau.\mathrm{B} + 0$
- $(a.b.\mathrm{A} + \overline{a}.0) \,|\, \mathrm{B}$
- $(a.b.\mathrm{A} + \overline{a}.0).\mathrm{B}$

Exercise: Syntax of CCS

Which of the following expressions are correctly built CCS expressions? Assume that A, B are process constants and that *a*, *b* are port names.

- $a.b.\mathrm{A} + \mathrm{B}$ ✓
- $(a.0 + \overline{a}.\mathrm{A}) \setminus \{a, b\}$ ✓
- $(a.0 \,|\, \overline{a}.\mathrm{A}) \setminus \{a, \tau\}$
- $\tau.\tau.\mathrm{B} + 0$
- $(a.b.\mathrm{A} + \overline{a}.0) \,|\, \mathrm{B}$
- $(a.b.\mathrm{A} + \overline{a}.0).\mathrm{B}$

Exercise: Syntax of CCS

Which of the following expressions are correctly built CCS expressions? Assume that A, B are process constants and that *a*, *b* are port names.

- $a.b.\mathrm{A} + \mathrm{B}$ ✓
- $(a.0 + \overline{a}.\mathrm{A}) \setminus \{a, b\}$ ✓
- $(a.0 \mid \overline{a}.\mathrm{A}) \setminus \{a, \tau\}$ ×
- $\tau.\tau.\mathrm{B} + 0$
- $(a.b.\mathrm{A} + \overline{a}.0) \mid \mathrm{B}$
- $(a.b.\mathrm{A} + \overline{a}.0).\mathrm{B}$

Exercise: Syntax of CCS

Which of the following expressions are correctly built CCS expressions? Assume that A, B are process constants and that a, b are port names.

- $a.b.\mathrm{A} + \mathrm{B}$ ✓
- $(a.0 + \overline{a}.\mathrm{A}) \setminus \{a, b\}$ ✓
- $(a.0 \,|\, \overline{a}.\mathrm{A}) \setminus \{a, \tau\}$ ✗
- $\tau.\tau.\mathrm{B} + 0$ ✓
- $(a.b.\mathrm{A} + \overline{a}.0) \,|\, \mathrm{B}$
- $(a.b.\mathrm{A} + \overline{a}.0).\mathrm{B}$

Exercise: Syntax of CCS

Which of the following expressions are correctly built CCS expressions? Assume that A, B are process constants and that a, b are port names.

- $a.b.\mathrm{A} + \mathrm{B}$ ✓
- $(a.0 + \overline{a}.\mathrm{A}) \setminus \{a, b\}$ ✓
- $(a.0 \,|\, \overline{a}.\mathrm{A}) \setminus \{a, \tau\}$ ✗
- $\tau.\tau.\mathrm{B} + 0$ ✓
- $(a.b.\mathrm{A} + \overline{a}.0) \,|\, \mathrm{B}$ ✓
- $(a.b.\mathrm{A} + \overline{a}.0).\mathrm{B}$

Exercise: Syntax of CCS

Which of the following expressions are correctly built CCS expressions? Assume that A, B are process constants and that a, b are port names.

- $a.b.\mathrm{A} + \mathrm{B}$ ✓
- $(a.0 + \overline{a}.\mathrm{A}) \setminus \{a, b\}$ ✓
- $(a.0 \mid \overline{a}.\mathrm{A}) \setminus \{a, \tau\}$ x
- $\tau.\tau.\mathrm{B} + 0$ ✓
- $(a.b.\mathrm{A} + \overline{a}.0) \mid \mathrm{B}$ ✓
- $(a.b.\mathrm{A} + \overline{a}.0).\mathrm{B}$ x

Operational Semantics of CCS

Operational Semantics

- **Goal**: Formalize the execution of a CCS process

Syntax		Semantics
CCS	$\longrightarrow$	LTS
(process term + equations)		(labelled transition systems)

Labelled Transition System

Definition

A labelled transition system (LTS) is a triple $(Proc, Act, \{\xrightarrow{\alpha} \mid \alpha \in Act\})$ where

- $Proc$ is a set of processes (the states),
- Act is a set of actions (the labels), and
- for every $\alpha \in Act$, $\xrightarrow{\alpha} \subseteq Proc \times Proc$ is a binary relation on processes called the transition relation

We use the infix notation $P \xrightarrow{\alpha} P'$ to say that $(P, P') \in \xrightarrow{\alpha}$

It is customary to distinguish the initial process (the start state)

Labelled Transition Systems

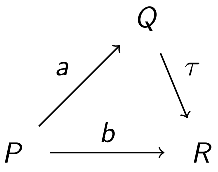
Conceptually it is often beneficial to think of a (finite) LTS as something that can be drawn as a directed (process) graph

- Processes are the nodes
- Transitions are the edges

Example: The LTS

$$\{\{P, Q, R\}, \{a, b, \tau\}, \{P \xrightarrow{a} Q, P \xrightarrow{b} R, Q \xrightarrow{\tau} R\}\}$$

corresponds to the graph



- **Question**: How can we produce an LTS (semantics) of a process term (syntax)?

Informal Translation

- Terminal process: 0

  behavior: $0 \not\rightarrow$

- Action prefixing: $\alpha.P$

  behavior: $\alpha.P \xrightarrow{\alpha} P$

- Non-deterministic choice: $\alpha.P + \beta.Q$

  behavior: $P \xleftarrow{\alpha} \alpha.P + \beta.Q \xrightarrow{\beta} Q$

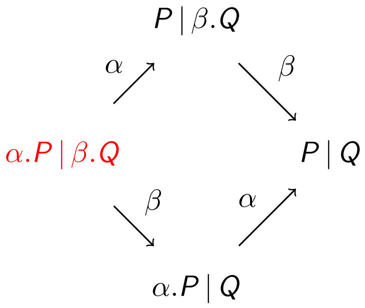
- Recursion: $\mathrm{X} \stackrel{\text{def}}{=} \cdots .\alpha.\mathrm{X}$

  behavior: $\mathrm{X} \xrightarrow{\cdots} \alpha.\mathrm{X}$, $\alpha.\mathrm{X} \xrightarrow{\alpha} \mathrm{X}$

Informal Translation

- Parallel composition: $\alpha.P \mid \beta.Q$
  Combines sequential composition and choice to obtain interleaving

behavior:

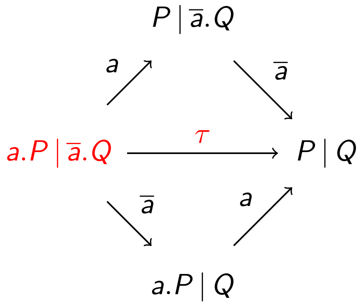


- What about interaction?

Process Interaction

- Concurrent processes, i.e. P and Q in $P \mid Q$, may interact where their interfaces are compatible
- A synchronizing interaction between two processes (sub-systems), P and Q, is an activity that is internal to $P \mid Q$
- Parallel composition: $\alpha.P \mid \beta.Q$
  Allows interaction if $\beta = \overline{\alpha}$

behavior:



Structural Operational Semantics for CCS

Structural Operational Semantics (SOS) [Plotkin 1981]

Small-step operational semantics where the behavior of a system is inferred using syntax driven rules

Given a collection of CCS defining equations, we define the following LTS $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$:

- *Proc* is the set of all CCS process expressions
- *Act* is the set of all CCS actions including τ
- the transition relation is given by SOS rules of the form:

$$\text{RULE} \quad \frac{\textit{premises}}{\textit{conclusion}} \quad \textit{conditions}$$

SOS rules for CCS

$$\text{ACT} \quad \frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \text{SUM}_j \quad \frac{P_j \xrightarrow{\alpha} P'_j}{\sum_{i\in I} P_i \xrightarrow{\alpha} P'_j} \; j \in I$$

$$\text{COM1} \quad \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \text{COM2} \quad \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

$$\text{COM3} \quad \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

$$\text{RES} \quad \frac{P \xrightarrow{\alpha} P'}{P \setminus L \xrightarrow{\alpha} P' \setminus L} \; \alpha, \overline{\alpha} \notin L \qquad \text{CON} \quad \frac{P \xrightarrow{\alpha} P'}{K \xrightarrow{\alpha} P'} \; K \overset{\text{def}}{=} P$$

Exercise: Derivations

Let $A \stackrel{\text{def}}{=} a.A$. Show that

$$\big((A \mid \overline{a}.0) \mid b.0\big) \xrightarrow{a} \big((A \mid \overline{a}.0) \mid b.0\big).$$

$$\frac{}{(A \mid \overline{a}.0) \mid b.0 \xrightarrow{a} (A \mid \overline{a}.0) \mid b.0}$$

Exercise: Derivations

Let $A \stackrel{\text{def}}{=} a.A$. Show that

$$((A \,|\, \overline{a}.0) \,|\, b.0) \xrightarrow{a} ((A \,|\, \overline{a}.0) \,|\, b.0).$$

$$\text{COM1}\ \dfrac{\overline{A \,|\, \overline{a}.0 \xrightarrow{a} A \,|\, \overline{a}.0}}{(A \,|\, \overline{a}.0) \,|\, b.0 \xrightarrow{a} (A \,|\, \overline{a}.0) \,|\, b.0}$$

Exercise: Derivations

Let $A \stackrel{\text{def}}{=} a.A$. Show that

$$\big((A \,|\, \overline{a}.0) \,|\, b.0\big) \xrightarrow{a} \big((A \,|\, \overline{a}.0) \,|\, b.0\big).$$

$$\text{COM1}\ \dfrac{\text{COM1}\ \dfrac{\dfrac{}{A \xrightarrow{a} A}\, A \stackrel{\text{def}}{=} a.A}{A \,|\, \overline{a}.0 \xrightarrow{a} A \,|\, \overline{a}.0}}{(A \,|\, \overline{a}.0) \,|\, b.0 \xrightarrow{a} (A \,|\, \overline{a}.0) \,|\, b.0}$$

Exercise: Derivations

Let $A \stackrel{\mathrm{def}}{=} a.A$. Show that

$$\big((A \mid \overline{a}.0) \mid b.0\big) \xrightarrow{a} \big((A \mid \overline{a}.0) \mid b.0\big).$$

$$\text{COM1}\ \dfrac{\text{COM1}\ \dfrac{\text{CON}\ \dfrac{\overline{a.A \xrightarrow{a} A}}{A \xrightarrow{a} A}\ A \stackrel{\mathrm{def}}{=} a.A}{A \mid \overline{a}.0 \xrightarrow{a} A \mid \overline{a}.0}}{(A \mid \overline{a}.0) \mid b.0 \xrightarrow{a} (A \mid \overline{a}.0) \mid b.0}$$

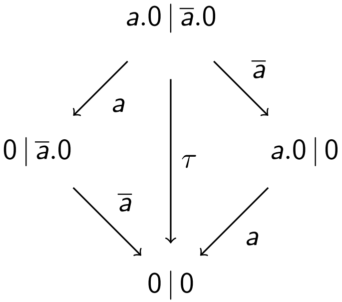
Exercise: Derivations

Let $A \stackrel{\text{def}}{=} a.A$. Show that

$$\big((A \,|\, \overline{a}.0) \,|\, b.0\big) \xrightarrow{a} \big((A \,|\, \overline{a}.0) \,|\, b.0\big).$$

$$\text{COM1}\ \dfrac{\text{COM1}\ \dfrac{\text{CON}\ \dfrac{\text{ACT}\ \dfrac{}{a.A \xrightarrow{a} A}}{A \xrightarrow{a} A}\ A \stackrel{\text{def}}{=} a.A}{A \,|\, \overline{a}.0 \xrightarrow{a} A \,|\, \overline{a}.0}}{(A \,|\, \overline{a}.0) \,|\, b.0 \xrightarrow{a} (A \,|\, \overline{a}.0) \,|\, b.0}$$

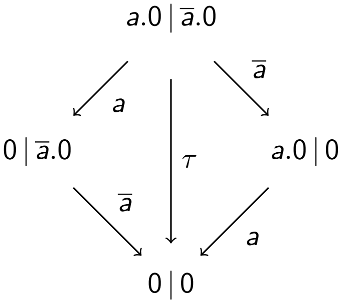
Restriction and Interaction



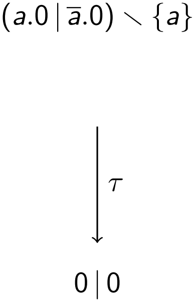
LTS of $a.0 \mid \overline{a}.0$

LTS of $(a.0 \mid \overline{a}.0) \setminus \{a\}$

Restriction and Interaction



LTS of $a.0 \mid \overline{a}.0$



LTS of $(a.0 \mid \overline{a}.0) \setminus \{a\}$

- Restriction can be used to produce closed systems, i.e. their actions can only be taken internally (visible as τ-actions)

Behavioral Equivalence

Behavioral Equivalence

- **Goal**: Express the notion that two concurrent systems "behave in the same way"
- We are not interested in syntactical equivalence, but only in the fact that the processes have the same behavior
- Main idea: two processes are behaviorally equivalent if and only if an external observer cannot tell them apart
- Bisimulation [Park 1980]: Two processes are equivalent if they have the same traces and the states that they reach are also equivalent

Strong Bisimilarity

Let $(Proc, Act, \{ \xrightarrow{\alpha} \mid \alpha \in Act\})$ be an LTS

Strong Bisimulation

A binary relation $R \subseteq Proc \times Proc$ is a strong bisimulation iff whenever $(P, Q) \in R$ then for each $\alpha \in Act$:

- if $P \xrightarrow{\alpha} P'$ then $Q \xrightarrow{\alpha} Q'$ for some Q' such that $(P', Q') \in R$
- if $Q \xrightarrow{\alpha} Q'$ then $P \xrightarrow{\alpha} P'$ for some P' such that $(P', Q') \in R$

Strong Bisimilarity

Two processes $P_1, P_2 \in Proc$ are strongly bisimilar ($P_1 \sim P_2$) if and only if there exists a strong bisimulation R such that $(P_1, P_2) \in R$

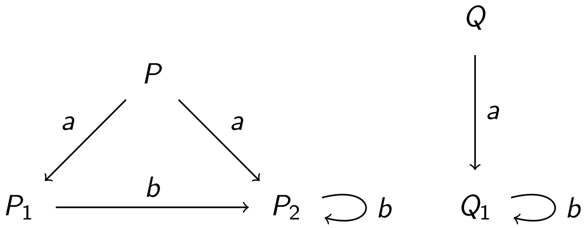
$$\sim \; = \; \cup\{R \mid R \text{ is a strong bisimulation}\}$$

Strong Bisimilarity of CCS Processes

- The concept of strong bisimilarity is defined for LTS
- The semantics of CCS is given in terms of LTS, whose states are CCS processes
- Thus, the definition also applies to CCS processes
  - Two processes are bisimilar if there is a concrete strong bisimulation relation that relates them
  - To show that two processes are bisimilar it suffices to exhibit such a concrete relation

Example: Strong Bisimulation

Consider the processes P and Q with the following behavior:



We claim that they are bisimilar

Example: Strong Bisimulation

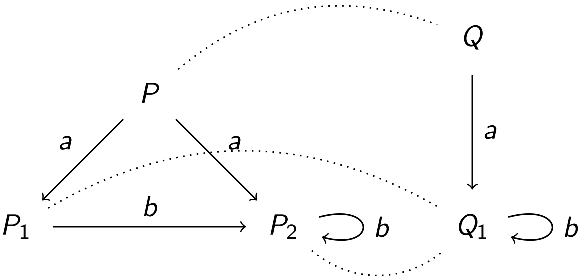
To show our claim we exhibit the following strong bisimulation relation:

$$\mathcal{R} = \{(P, Q), (P_1, Q_1), (P_2, Q_1)\}$$

- (P, Q) is in $\mathcal{R}$
- $\mathcal{R}$ is a bisimulation:
  - For each pair of states in $\mathcal{R}$, all possible transitions from the first can be matched by corresponding transitions from the second
  - For each pair of states in $\mathcal{R}$, all possible transitions from the second can be matched by corresponding transitions from the first

Example: Strong Bisimulation

Graphically, we show $\mathcal{R}$ with dotted lines:



Now it is easy to see that:

- For each pair of states in $\mathcal{R}$, all possible transitions from the first can be matched by corresponding transitions from the second
- For each pair of states in $\mathcal{R}$, all possible transitions from the second can be matched by corresponding transitions from the first

Exercise: Strong Bisimulation

Consider the processes

$$\mathrm{P} \stackrel{\text{def}}{=} a.(b.0 + c.0)$$
$$\mathrm{Q} \stackrel{\text{def}}{=} a.b.0 + a.c.0$$

and show that $P \not\sim Q$