Models of Concurrency

GERARDO SCHNEIDER

UPPSALA UNIVERSITY

DEPARTMENT OF INFORMATION TECHNOLOGY

UPPSALA, SWEDEN

Thanks to **Frank Valencia**

UPPSALA UNIVERSITET

Concurrency is Everywhere

Concurrent Systems: Multiple agents (processes) that interact among each other.

Key issues :

- Message-Passing & Shared-Memory
- Synchronous & Asynchronous
- Reactive Systems
- Mobile Systems
- Secure Systems
- Timed Systems

Concurrency is Everywhere

Concurrent Systems: Multiple agents (processes) that interact among each other.

Example: *The Internet* (a complex system!). It combines many of the before-mentioned issues!

Concurrency: A Serious Challenge

- Need for **Formal Models** to describe and analyze concurrent systems.
- Models for sequential computation (functions f: Inputs$\longrightarrow$Outputs) don't apply;
  Concurrent computation is usually:
  - *Non-Terminating*
  - *Reactive* (or *Interactive*)
  - *Nondeterministic* (*Unpredictable*).
  - ...etc.

Concurrency: A Serious Challenge

- Formal models must be simple, expressive, formal and provide techniques (e.g., λ calculus)

Concurrency: A Serious Challenge

- In concurrency theory:
  - There are several (too many?) models focused in specific phenomena.
  - New models typically arise as extensions of well-established ones.
  - There is no yet a “canonical (all embracing) model” for concurrency.

Concurrency: A Serious Challenge

- In concurrency theory:
  - There are several (too many?) models focused in specific phenomena.
  - New models typically arise as extensions of well-established ones.
  - There is no yet a “canonical (all embracing) model” for concurrency.
- Why?
  - ...Probably because concurrency is a very broad (young) area.

Concurrency: a Serious Challenge

Some Well-Established Concurrency Models:

- Process Algebras (Process Calculi):
  - Milner's CCS & Hoare's CSP (Synchronous communication)
  - Milner's π-calculus (CCS Extension to Mobility)
  - Saraswat's CCP (Shared Memory Communication)
- Petri Nets: First well-established concurrency theory—extension of automata theory

UPPSALA UNIVERSITET

Agenda

- Basic concepts from Automata Theory
- CCS
  - Basic Theory
  - Process Logics
  - Applications: Concurrency Work Bench (?)
- π-calculus
- Petri Nets

Automata

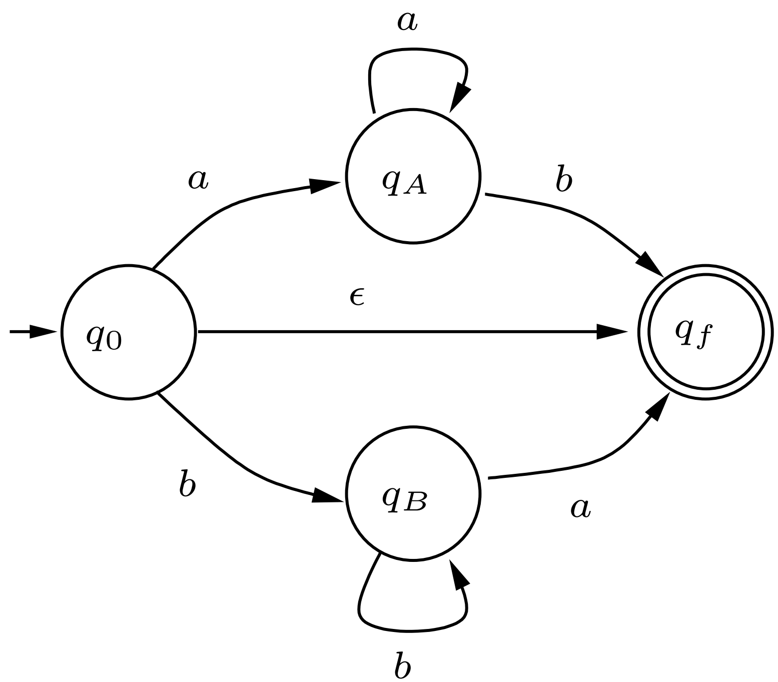
Definition: An automata A over an alphabet Act is a tuple (Q, Q_0, Q_f, T) where

- $S(A) = Q = \{q_0, q_1, \ldots\}$ is the set of states
- $S_0(A) = Q_0 \subseteq Q$ is the set of initial states
- $S_f(A) = Q_f \subseteq Q$ is the set of accepting (or final) states
- $T(A) = T \subseteq Q \times \ Act \ \times Q$ is the set of transitions

Usually $(q, a, q') \in T$ is written as $q \xrightarrow{a} q'$

UPPSALA UNIVERSITET

Automaton Example



A over $\{a, b\}$ with $S(A) = \{q_0, q_A, q_B, q_f\}$,
$S_0(A) = \{q_0\}$, $S_f(A) = \{q_f\}$,
$T(A) = \{q_0 \xrightarrow{a} q_A, \ldots\}$.

UPPSALA UNIVERSITET

Regular Sets

Definition (Acceptance, Regularity)

- A over Act **accepts** $s = a_1 \ldots a_n \in Act^*$ if there are $q_0 \xrightarrow{a_1} q_1$, $q_1 \xrightarrow{a_2} q_2, \ldots, q_{n-1} \xrightarrow{a_n} q_n$ in $T(A)$ s.t., $q_0 \in S_o(A)$ and $q_n \in S_f(A)$.
- The **language** of (or recognized by) A, $L(A)$, is the set of sequences accepted by A.

Regular Sets

Definition (Acceptance, Regularity)

- Regular sets are those recognized by *finite-state automata* (**FSA**): I.e., S is regular iff $S = L(A)$ for some FSA A.
- Regular Expressions (e.g., $a.(b+c)^*$) are "equally expressive" to FSA.

UPPSALA UNIVERSITET

Automata: Some Nice Properties

Proposition:

1. Deterministic and Non-Deterministic FSA are equally “expressive”.
2. Regular sets are closed under (a) union, (b) complement, (c) intersection.

Automata: Exercises

Exercises: (1) Prove 2.b and 2.c.

(2)$\star$ Prove that emptiness problem of a given FSA is *decidable*.

(3)$\star$ Prove that language equivalence of two given FSA is *decidable.*

(4)$\star\star\star$ Let B a FSA. Construct a FSA A such that

$s \in L(A)$ iff for *every suffix* s' of $s, s' \in L(B)$

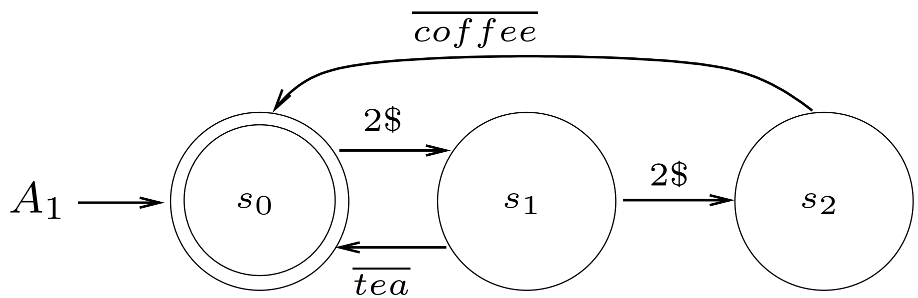
UPPSALA UNIVERSITET

Automata Theory; what is it good fo

- Classic Automata Theory is solid and foundational, and it has several applications in computer science.

Automata Theory; what is it good fo

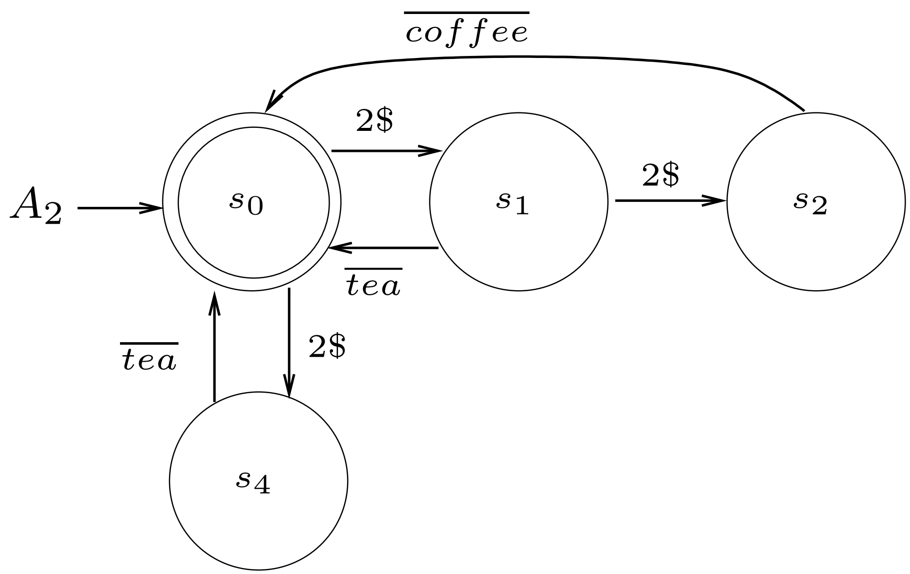
- Classic Automata Theory is solid and foundational, and it has several applications in computer science.
- **Example:** Two Vending-Machines



UPPSALA UNIVERSITET

Automata Theory; what is it good fo

- Classic Automata Theory is solid and foundational, and it has several applications in computer science.
- **Example:** Two Vending-Machines



UPPSALA UNIVERSITET

Automata Theory; what is it good fo

- Classic Automata Theory is solid and foundational, and it has several applications in computer science.
- **Example:** Two Vending-Machines
- If $L(A_1) = L(A_2)$ then language equivalence (trace equivalence) is too weak for interactive behaviour!

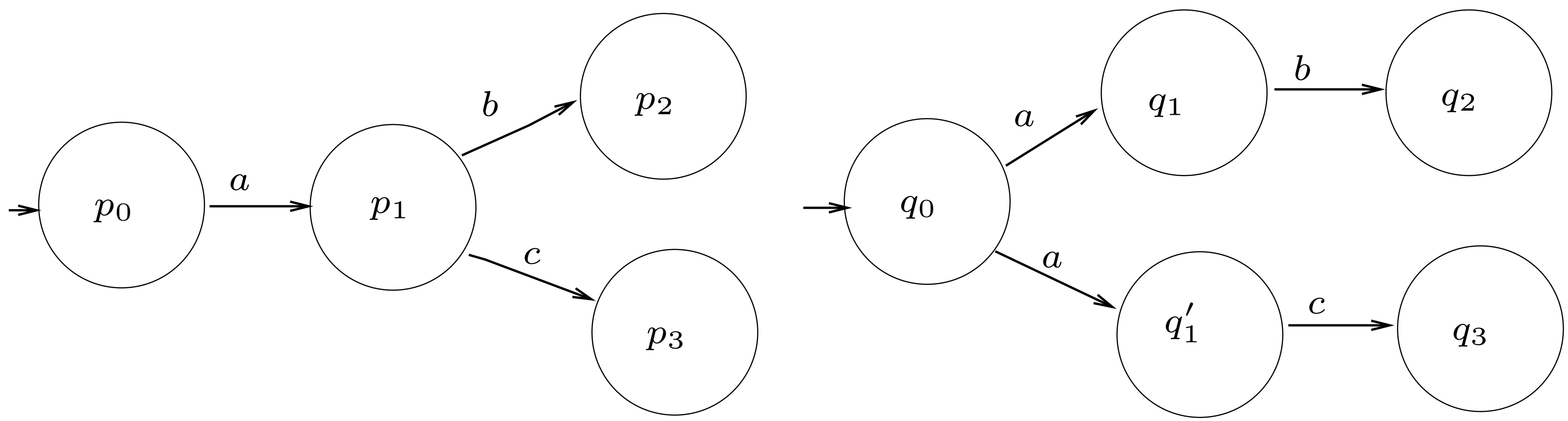
UPPSALA UNIVERSITET

Automata Theory: The Problem

- The theory allows to deduce that $a \cdot (b + c) = a \cdot b + a \cdot c$

Automata Theory: The Problem

- The theory allows to deduce that $a \cdot (b + c) = a \cdot b + a \cdot c$



Automata Theory: The Problem

- The theory allows to deduce that $a \cdot (b + c) = a \cdot b + a \cdot c$
- That is, the automata in the example are equivalent and we want to differentiate them
- We need a stronger equivalence that does not validate the above.

Simulation & Bisimulation Relations

Transition Systems are just automata in which final and initial states are irrelevant

Definition: Let T be transition system.
A relation $R \subseteq S(T) \times S(T)$ is a *simulation* iff for every $(p,q) \in R$:

If $p \xrightarrow{a} p'$ then there exists q' such that $q \xrightarrow{a} q'$ and $(p',q') \in R$

A relation R is a *bisimulation* iff R and its *converse* R^{-1} are both simulations.

UPPSALA UNIVERSITET

Bisimilarity

Definition: We say that p simulates q iff there exists a *simulation* R such that $(p, q) \in R$.
Also, p and q are *bisimilar*, written $p \sim q$, if there exists a *bisimulation* R such that $(p, q) \in R$.

Example: In the previous example p_0 simulates q_0 but q_0 cannot simulate p_0, so p_0 and q_0 are not bisimilar.

Question: If p simulates q and q simulates p; are p and q bisimilar?

Bisimilarity

Definition: We say that p simulates q iff there exists a *simulation* R such that $(p, q) \in R$. Also, p and q are *bisimilar*, written $p \sim q$, if there exists a *bisimulation* R such that $(p, q) \in R$.

Example: In the previous example p_0 simulates q_0 but q_0 cannot simulate p_0, so p_0 and q_0 are not bisimilar.

Question: If p simulates q and q simulates p; are p and q bisimilar?

Answer: **No!** $P = a.0 + a.b.0$ and $Q = a.b.0$

Road Map

- We have seen:
  Basic Classic Automata theory, and
  Transition Systems, Bisimilarity
- Next: Process Calculi, in particular CCS.
  Processes represented as transition systems and their behavioural equivalence given by bisimilarity.

Process Calculi: Key Issues

- (Syntax) Constructs that fit the intended phenomena
  E.g. Atomic actions, parallelism, nondeterminism, locality, recursion.
- (Semantics) How to give meaning to the constructs
  E.g. Operational, denotational, or algebraic semantics
- (Equivalences) How to compare processes
  E.g. Observable Behaviour, process equivalences, congruences...

Process Calculi: Key Issues

- (Specification) How to specify and prove process properties
  E.g. Logic for expressing process specifications (Hennessy-Milner Logic)
- (Expressiveness) How expressive are the constructs?

CCS: Calculus for Synchron. Communic.

Underlying sets (basic atoms)

- A set $\mathcal{N} = a, b, \ldots$ of names and $\overline{\mathcal{N}} = \{\overline{a} \mid a \in \mathcal{N}\}$ of co-names
- A set $\mathcal{L} = \mathcal{N} \cup \overline{\mathcal{N}}$ of labels (ranged over by $l, l', \ldots$)
- A set $Act = \mathcal{L} \cup \{\tau\}$ of actions (ranged over by $\mathrm{a}, \mathrm{b}, \ldots$)
- Action τ is called the *silent or unobservable action*
- Actions a and $\overline{a}$ are "complementary": $a = \overline{\overline{a}}$

UPPSALA UNIVERSITET

CCS: Process Syntax

$$P, Q, \ldots := 0 \mid \mathbf{a}.P \mid P \parallel Q \mid P{+}Q \mid P\backslash a \mid A(a_1, ..., a_n)$$

- Bound names of P, $bn(P)$: Those with a bound occurrence in P.
- Free names of P, $fn(P)$: Those with a not bound occurrence in P.
- For each (call) $A(a_1, \ldots, a_n)$ there is a unique process definition $A(b_1 \ldots b_n) = P$, with $fn(P) \subseteq \{b_1, \ldots, b_n\}$.
- The set of all processes is denoted by $\mathcal{P}$.

UPPSALA UNIVERSITET

CCS: Operational Semantics

ACT $\dfrac{}{\mathbf{a}.P \xrightarrow{\mathbf{a}} P}$

SUM$_1$ $\dfrac{P \xrightarrow{\mathbf{a}} P'}{P + Q \xrightarrow{\mathbf{a}} P'}$

SUM$_2$ $\dfrac{Q \xrightarrow{\mathbf{a}} Q'}{P + Q \xrightarrow{\mathbf{a}} Q'}$

COM$_1$ $\dfrac{P \xrightarrow{\mathbf{a}} P'}{P \parallel Q \xrightarrow{\mathbf{a}} P' \parallel Q}$

COM$_2$ $\dfrac{Q \xrightarrow{\mathbf{a}} Q'}{P \parallel Q \xrightarrow{\mathbf{a}} P \parallel Q'}$

COM$_3$ $\dfrac{P \xrightarrow{l} P' \quad Q \xrightarrow{\bar{l}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$

UPPSALA UNIVERSITET

CCS: Operational Semantics

$$\text{RES}\ \frac{P \xrightarrow{\mathbf{a}} P'}{P\backslash a \xrightarrow{\mathbf{a}} P'\backslash a}\ \text{if } \mathbf{a} \neq a \text{ and } \mathbf{a} \neq \overline{a}$$

$$\text{REC}\ \frac{P_A[b_1, \ldots, b_n/a_1, \ldots, a_n] \xrightarrow{\mathbf{a}} P'}{A(b_1, \ldots, b_n) \xrightarrow{\mathbf{a}} P'}\ \text{if } A(a_1, \ldots, a_n) \stackrel{\texttt{def}}{=} P_A$$

CCS: Operational Semantics

$$\text{RES} \quad \frac{P \xrightarrow{\mathbf{a}} P'}{P\backslash a \xrightarrow{\mathbf{a}} P'\backslash a} \quad \text{if } \mathbf{a} \neq a \text{ and } \mathbf{a} \neq \overline{a}$$

$$\text{REC} \quad \frac{P_A[b_1, \ldots, b_n/a_1, \ldots, a_n] \xrightarrow{\mathbf{a}} P'}{A(b_1, \ldots, b_n) \xrightarrow{\mathbf{a}} P'} \quad \text{if } A(a_1, \ldots, a_n) \stackrel{\text{def}}{=} P_A$$

Notation: Instead of $P\backslash a$ we will use the infix notation: $(\nu a)P$ (“new” operator).

CCS: Bisimilarity

The ***labelled transition system*** *of CCS has $\mathcal{P}$ as its states and its transitions are those given by the operational (labelled) semantics.* Hence, define $P \sim Q$ iff the states corresponding to P and Q are bisimilar.

Exercise Write a CCS expression for the vending machine (in parallel with some thirsty user:-).

CCS: Bisimilarity

Questions Do we have?

- $P \parallel Q \sim Q \parallel P$
- $P \parallel 0 \sim P$
- $(P \parallel Q) \parallel R \sim P \parallel (Q \parallel R)$
- $(\nu a)0 \sim 0$
- $P \parallel (\nu a)Q \sim (\nu a)(P \parallel Q)$
- $(\nu a)P \sim (\nu b)P[b/a]$

CCS: The expansion law

- Notice that $a.0 \parallel b.0$ is bisimilar to the summation form $a.b.0 + b.a.0$
- More generally, we have the expansion law which allows to express systems in summation form.

UPPSALA UNIVERSITET

CCS: The expansion law

- Notice that $a.0 \parallel b.0$ is bisimilar to the summation form $a.b.0 + b.a.0$
- More generally, we have the expansion law

I.e.

$$(\nu\vec{a})(P_1 \parallel \ldots \parallel P_n) \quad \sim$$

$$\Sigma\{\mathbf{a}_i.(\nu\vec{a})(P_1 \parallel \ldots \parallel P_i' \parallel \ldots \parallel P_n) \mid P_i \xrightarrow{\mathbf{a}_i} P_i',\ \overline{\mathbf{a}}, \mathbf{a}_i \notin \vec{a}\}$$

$$+$$

$$\Sigma\{\tau.(\nu\vec{a})(P_1 \parallel \ldots \parallel P_i' \parallel \ldots \parallel P_j' \ldots \parallel P_n) \mid P_i \xrightarrow{l} P_i',\ P_j \xrightarrow{\bar{l}} P_j'\}$$

UPPSALA UNIVERSITET

CCS: The expansion law

- Notice that $a.0 \parallel b.0$ is bisimilar to the summation form $a.b.0 + b.a.0$
- So, every move in $(\nu\vec{a})(P_1 \parallel \ldots \parallel P_n)$ is either one of the P_i or a communication between some P_i and P_j

UPPSALA UNIVERSITET

Congruence Issues

- Suppose that $P \sim Q$. We would like

$$P \parallel R \sim Q \parallel R$$

More generally, we would like

$$C[P] \sim C[Q]$$

where $C[.]$ is a process context

Congruence Issues

- Suppose that $P \sim Q$. We would like

  $$P \parallel R \sim Q \parallel R$$

  More generally, we would like

  $$C[P] \sim C[Q]$$

  where $C[.]$ is a process context

- I.e., we want $\sim$ to be a congruence. The notion of congruence allows us to replace "*equals with equals*"

UPPSALA UNIVERSITET

Congruence Issues

- Suppose that $P \sim Q$. We would like

$$P \parallel R \sim Q \parallel R$$

More generally, we would like

$$C[P] \sim C[Q]$$

where $C[.]$ is a process context

Question. How can we prove that $\sim$ is a congruence?

Observable Behaviour

- In principle, P and Q should be *equivalent* iff another process (the environment, an observer) cannot *observe* any difference in their behaviour
- Notice $\tau.P \not\sim P$, although τ is an unobservable action. So $\sim$ could be too *strong*
  So, we look for other notion of equivalence focused in terms of *observable* actions (i.e., actions $\xrightarrow{l}$, $l \in \mathcal{L}$)

UPPSALA UNIVERSITET

Observations

- Think of any $\xrightarrow{l}$ as an **observation**; or that an action $\xrightarrow{a}$ by P can be *observed* by an action $\xrightarrow{\bar{a}}$ by $P's$ environment
- An **experiment** e as a sequence $l_1.l_2\ldots l_n$ of observable actions

Notation: If $s = \mathbf{a}_1 \ldots \mathbf{a}_n \in Act^*$ then define

$$\overset{s}{\Longrightarrow} = (\xrightarrow{\tau})^* \xrightarrow{\mathbf{a}_1} (\xrightarrow{\tau})^* \ldots (\xrightarrow{\tau})^* \xrightarrow{\mathbf{a}_n} (\xrightarrow{\tau})^*$$

UPPSALA UNIVERSITET

Observations

- Think of any $\xrightarrow{l}$ as an **observation**; or that an action $\xrightarrow{a}$ by P can be *observed* by an action $\xrightarrow{\overline{a}}$ by $P's$ environment
- An experiment e as a sequence $l_1.l_2 \ldots l_n$ of observable actions
- Notice that $\stackrel{e}{\Longrightarrow}$ for $e = l_1.l_2 \ldots l_n \in \mathcal{L}$ denotes a sequence of observable actions inter-spread with τ actions: *The notion of experiment*.

Trace Equivalence

Definition: (Trace Equivalence) P and Q are ***trace equivalent***, written $P \sim_t Q$, iff for every experiment (here called trace) $e = l_1 \ldots l_n \in \mathcal{L}^*$

$$P \stackrel{e}{\Longrightarrow} \quad \text{iff} \quad Q \stackrel{e}{\Longrightarrow}$$

UPPSALA UNIVERSITET

Trace Equivalence

Examples.

- $\tau.P \;\; \sim_t \;\; P$ (nice!)
- $a.b.0 + a.c.0 \quad \sim_t \quad a.(b.0 + c.0)$ (not that nice!)
- $a.b.0 + a.0 \;\; \sim_t \;\; a.b.0$ (not sensitive to deadlocks)
- $a.0 + b.0 \;\; \sim_t \;\; (\nu c)(c.0 \parallel \overline{c}.a.0 \parallel \overline{c}.b.0)$

Failures Equivalence

Definition: (Failures Equivalence) A pair (e, L), where $e \in \mathcal{L}^*$ (i.e. a *trace*) and $L \subset \mathcal{L}$, is a failure for P iff

$$(1) P \overset{e}{\Longrightarrow} P' \quad (2)\ P' \overset{l}{\not\longrightarrow} \text{ for all } l \in L, \quad (3) P' \overset{\tau}{\not\longrightarrow}$$

P and Q are ***failures equivalent***, written $P \sim_f Q$, iff they have the same failures.

Fact $\sim_f \subset \sim_t$.

UPPSALA UNIVERSITET

Failures Equivalence

Examples

- $\tau.P \;\sim_f\; P$
- $a.b.0 + a.c.0 \quad \not\sim_f \quad a.(b.0 + c.0)$ **(Exercise)**
- $a.b.0 + a.0 \;\not\sim_f\; a.b.0$.
- $a.0 + b.0 \;\not\sim_f\; (\nu c)(c.0 \parallel \overline{c}.a.0 \parallel \overline{c}.b.0)$ **(Exercise)**
- $a.(b.c.0 + b.d.0) \;\sim_f\; a.b.c.0 + a.b.d.0$.
- Let $D = \tau.D$. We have $\tau.0 \not\sim_f D$.

Weak Bisimilarity

Definition: A *symmetric* binary relation R on processes is a *weak bisimulation* iff for every $(P, Q) \in R$:

If $P \stackrel{e}{\Longrightarrow} P'$ and $e \in \mathcal{L}^*$ then there exists Q' such that $Q \stackrel{e}{\Longrightarrow} Q'$ and $(P', Q') \in R$.

P and Q are *weakly bisimilar* , written $P \approx Q$ iff there exists a *weak bisimulation* containing the pair (P, Q).

UPPSALA UNIVERSITET

Weak Bisimilarity

Examples.

- $\tau.P \;\approx\; P$, $a.\tau.P \approx a.P$
- However, $a.0 + b.0 \not\approx a.0 + \tau.b.0$
- $a.(b.c.0 + b.d.0) \;\not\approx\; a.b.c.0 + a.b.d.0$ **(Exercise)**
- Let $D = \tau.D$. We have $\tau.0 \approx D$.
- $a.0 + b.0 \;\not\approx\; (\nu c)(c.0 \parallel \overline{c}.a.0 \parallel \overline{c}.b.0)$ **(Exercise)**

An alternative definition of weak bisimulation

- Verifying bisimulation using the previous definition could be hard (there are infinitely many experiments e!).
- Fortunately, we have an alternative formulation easier to work with:

  Proposition: R is a weak bisimulation iff

  If $P \stackrel{\mathbf{a}}{\Longrightarrow} P'$ and $\mathbf{a} \in Act$ then there exists Q' such that $Q \stackrel{\hat{\mathbf{a}}}{\Longrightarrow} Q'$ and $(P', Q') \in R$. Where $\hat{\mathbf{a}} = \mathbf{a}$ if $\mathbf{a} \in \mathcal{L}$ (i.e. observable), otherwise $\hat{\mathbf{a}} = \epsilon$.

UPPSALA UNIVERSITET

Road Map

- We have seen:
  Basic Classic Automata theory, and
  Transition Systems, Bisimilarity
- Also: Process Calculi, in particular CCS.
  Processes represented as transition systems
  and their behavioural equivalence given by
  bisimilarity.
- **Next:** Process Logics

Process Logic: Verification and Specification

- Process can be used to *specify and verify* the behaviour system (E.g. Vending Machines).
  E.g., $2p.\overline{tea}.0 + 2p.\overline{coffe}.0$ specify a machine which does not satisfy the behaviour specified by a $2p.(\overline{tea}.0 + \overline{coffe}.0)$
- In Computer Science we use logics for *specification* and *verification* of properties. A logic whose formulae can express, e.g.,
  - "P will never not execute a bad action", or
  - "P eventually executes a good action"

Hennessy&Milner Logic

The syntax of the logic:

$$F \;:= \texttt{true} \mid \texttt{false} \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \langle K \rangle\, F \mid [K]F$$

where K is a set of actions

- The boolean operators are interpreted as in propositional logic

Hennessy&Milner Logic

The syntax of the logic:

$$F \ := \texttt{true} \mid \texttt{false} \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \langle K \rangle \, F \mid [K]F$$

where K is a set of actions

- $\langle K \rangle \, F$ (possibility) asserts (of a given P): It is *possible* for P to do a $\mathtt{a} \in K$ and then evolve into a Q that satisfy F
- $[K]F$ (necessity) asserts (of a given P): If P can do a $\mathtt{a} \in K$ then it *must* evolve into a Q which satisfies F

Hennesy&Milner Logic: Semantics

The compliance of P with the specification F, written $P \models F$, is given by:

$$
\begin{array}{lll}
P \not\models \mathtt{false} & & \\
P \models \mathtt{true} & & \\
P \models F_1 \wedge F_2 & \text{iff} & P \models F_1 \text{ and } P \models F_2 \\
P \models F_1 \vee F_2 & \text{iff} & P \models F_1 \text{ or } P \models F_2 \\
P \models \langle K \rangle F & \text{iff} & \text{for some } Q \\
 & & P \xrightarrow{\mathbf{a}} Q, \mathbf{a} \in K \text{ and } Q \models F \\
P \models [K]F & \text{iff} & \text{if } P \xrightarrow{\mathbf{a}} Q \text{ and } \mathbf{a} \in K \text{ then } Q \models F
\end{array}
$$

Hennesy&Milner Logic: Semantics

Example. Let

$$P_1 = a.(b.0 + c.0),\ P_2 = a.b.0 + a.c.0$$

Also let

$$F = \langle\{a\}\rangle\,(\langle\{b\}\rangle\,\texttt{true} \wedge \langle\{c\}\rangle\,\texttt{true})$$

Notice that $P_1 \models F$ but $P_2 \not\models F$.

Theorem $P \sim Q$ if and only, for every F, $P \models F$ iff $Q \models F$.

UPPSALA UNIVERSITET

A Linear Temporal Logic

The syntax of the formulae is given by

$$F := \texttt{true} \mid \texttt{false} \mid L \mid F_1 \vee F_2 \mid F_1 \wedge F_2 \mid \Diamond F \mid \Box F$$

where L is a set of non-silent actions.

- Formulae assert properties of traces
- Boolean operators are interpreted as usual
- L asserts (of a given trace s) that the first action of s must be in $L \cup \{\tau\}$

A Linear Temporal Logic

The syntax of the formulae is given by

$$F := \texttt{true} \mid \texttt{false} \mid L \mid F_1 \vee F_2 \mid F_1 \wedge F_2 \mid \Diamond F \mid \Box F$$

where L is a set of non-silent actions.

- $\Diamond F$ asserts (of a given trace s) that at some point in s, F holds.
- $\Box F$ asserts (of a given trace s) that at every point in s, F holds.

Temporal Logic: Semantics

An infinite sequence of actions $s = \mathbf{a}_1.\mathbf{a}_2 \ldots$ satisfies (or is a model of) F, written $s \models F$, iff $\langle s, 1\rangle \models F$, where

$\langle s, i\rangle \models \mathtt{true}$		
$\langle s, i\rangle \not\models \mathtt{false}$		
$\langle s, i\rangle \models L$	iff	$\mathbf{a}_i \in L \cup \tau$
$\langle s, i\rangle \models F_1 \vee F_2$	iff	$\langle s, i\rangle \models F_1$ or $\langle s, i\rangle \models F_2$
$\langle s, i\rangle \models F_1 \wedge F_2$	iff	$\langle s, i\rangle \models F_1$ and $\langle s, i\rangle \models F_2$
$\langle s, i\rangle \models \Box F$	iff	for all $j \geq i$ $\langle s, j\rangle \models F$
$\langle s, i\rangle \models \Diamond F$	iff	there is a $j \geq i$ s.t. $\langle s, j\rangle \models F$

UPPSALA UNIVERSITET

Temporal Logic: Semantics

- Moreover,

$$P \models F$$

iff whenever $P \stackrel{s}{\Longrightarrow}$ then $\hat{s} \models F$, where $\hat{s} = s.\tau.\tau.\ldots$

Temporal Logic: Example

Example. Consider

$$A(a,b,c) \stackrel{\text{def}}{=} a.(b.A(a,b,c) + c.A(a,b,c))$$ and
$$B(a,b,c) \stackrel{\text{def}}{=} a.b.B(a,b,c) + a.c.B(a,b,c).$$

- Notice that the trace equivalent processes $A(a,b,c)$ and $B(a,b,c)$ satisfy $\Box\Diamond(b \vee c)$
- I.e. they always eventually do b or c

Temporal Logic: Example

Theorem If $P \sim_t Q$ then for every linear temporal formula F,

$$P \models F \text{ iff } Q \models F$$

Question. Does the other direction of the theorem hold?

UPPSALA UNIVERSITET

Temporal Logic: Exercises

Exercises: Which one of the following equivalences are true?

- $\Box(F \vee G) \equiv \Box F \vee \Box G$?
- $\Diamond(F \vee G) \equiv \Diamond F \vee \Diamond G$?
- $\Box\Diamond F \equiv \Diamond\Box F$?
- $\Box\Diamond F \equiv \Diamond F$?

UPPSALA UNIVERSITET

Temporal Logic: Exercises

Exercises: Which one of the following equivalences are true?

- $\Box(F \vee G) \not\equiv \Box F \vee \Box G$
- $\Diamond(F \vee G) \equiv \Diamond F \vee \Diamond G$
- $\Box\Diamond F \not\equiv \Diamond\Box F$
- $\Box\Diamond F \not\equiv \Diamond F$

UPPSALA UNIVERSITET

Road Map

- Basic classic automata theory and the limitation of language equivalence.
- Bisimilarity Equivalence for automata (transition systems).
- CCS
  - Behaviour $\longrightarrow$ transitions systems
  - Behaviour using: bisimilarity, trace equivalence, failures equivalence, weak bisimilarity
  - Specification of properties using HM and Temporal Logics.

Road Map

- Basic classic automata theory and the limitation of language equivalence.
- Bisimilarity Equivalence for automata (transition systems).
- CCS
- Mobility and the π calculus

Mobility

What kind of *process mobility* are we talking about?

- Processes move in the ***physical space*** of computing sites
- Processes move in the ***virtual space*** *of linked processes*
- *Links* move, in the *virtual space of linked processes*

Mobility

What kind of *process mobility* are we talking about?

- *Links* move, in the *virtual space* *(of linked processes)*

The last one is the π-calculus' choice; for economy, flexibility, and simplicity.

- The π calculus extends CCS with the ability of sending private and public *links* (*names*).

UPPSALA UNIVERSITET

$\pi-$Calculus: Syntax

$$P := P \parallel P \mid \Sigma_{i \in I}\, \alpha_i.P \mid (\nu a)P \mid !P \mid \textbf{if } a = b \textbf{ then } P$$

where $\alpha := \tau \mid \overline{a}(b) \mid a(x)$

- Names=Channels=Ports=Links.
- $\overline{a}(b).P$: "send b on channel a and then activate P"
- $a(x).P$: "receive a name on channel a (if any), and replace x with it in P"

UPPSALA UNIVERSITET

$\pi-$Calculus: Syntax

$$P := P \parallel P \mid \Sigma_{i \in I}\, \alpha_i.P \mid (\nu a)P \mid\, !P \mid \textbf{if } a = b \textbf{ then } P$$

where $\alpha := \tau \mid \overline{a}(b) \mid a(x)$

- $(\nu a).P$: "create a fresh name a private to P"
- $(\nu a)P$ and $a(x).Q$ are the only binders
- $!P$: "replicate P" i.e., $!P$ represents $P \parallel P \parallel P \parallel \ldots$

UPPSALA UNIVERSITET

Mobility: Example

Client & Printer-Server:

- The printer-server
  $Serv = (\nu p)\,(s(r).\overline{r}(p) \;\;\|\;\; p(j).Print)$
- The Client $Client = \overline{s}(c).c(plink).\overline{plink}(job)$
- The System $Client \parallel Serv$.

Mobility: Exercises

Write an agent that

- Reads sthg from port a and sends it twice along port b
- Reads two ports and sends the fi rst along the second
- Sends b and c on channel a so that ***only one*** (sequential) process receive both b and c
- Contains three agents P, Q, R such that P can communicate with both Q and R; but Q and R cannot communicate
- Generates ***infinitely many*** different names—and send them along channel a.

UPPSALA UNIVERSITET

Reaction Semantics of π

The reactive semantics of π consists of a *structural congruence* $\equiv$ and the *reactive rules*.

- The **structural congruence** describe irrelevant syntactic aspects of processes
- The **reactive rules** describe the evolutions due to synchronous communication between processes.

Structural Congruence

Definition: (Structural Congruence) The relation $\equiv$ is the smallest process equivalence satisfying:

- $P \equiv Q$ if P can be *alpha-converted* into Q.
- $P \parallel 0 \equiv P, \quad P \parallel Q \equiv Q \parallel P,$
  $(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)$
- $(\nu a)0 \equiv 0, \quad (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$
- $(\nu a)(P \parallel Q) \equiv P \parallel (\nu a)Q$ if $a \notin fn(P)$
- $!P \equiv P \parallel !P$

UPPSALA UNIVERSITET

Structural Congruence

Examples. Notice that $(\nu c)(\overline{a}(b) \parallel 0) \equiv \overline{a}(b)$.

Do we have?

1. $(\nu a)P \equiv P$ if $a \notin fn(P)$
2. $x(y).y(z) \equiv x(z).y(y)$
3. $x \parallel y \equiv x.y + y.x$
4. $x(y).x(z) \parallel y(z).z(y) \equiv y(y).y(z) \parallel x(z).x(y)$

UPPSALA UNIVERSITET

Structural Congruence

Examples. Notice that $(\nu c)(\overline{a}(b) \parallel 0) \equiv \overline{a}(b)$.

Do we have?

1. $(\nu a)P \equiv P$ if $a \notin fn(P)$ True!
2. $x(y).y(z) \equiv x(z).y(y)$
3. $x \parallel y \equiv x.y + y.x$
4. $x(y).x(z) \parallel y(z).z(y) \equiv y(y).y(z) \parallel x(z).x(y)$

UPPSALA UNIVERSITET

Structural Congruence

Examples. Notice that $(\nu c)(\overline{a}(b) \parallel 0) \equiv \overline{a}(b)$.

Do we have?

1. $(\nu a)P \equiv P$ if $a \notin fn(P)$ True!
2. $x(y).y(z) \equiv x(z).y(y)$ Not true! $(\ldots \parallel \overline{x}(d))$
3. $x \parallel y \equiv x.y + y.x$
4. $x(y).x(z) \parallel y(z).z(y) \equiv y(y).y(z) \parallel x(z).x(y)$

UPPSALA UNIVERSITET

Structural Congruence

Examples. Notice that $(\nu c)(\overline{a}(b) \parallel 0) \equiv \overline{a}(b)$.

Do we have?

1. $(\nu a)P \equiv P$ if $a \notin fn(P)$ True!
2. $x(y).y(z) \equiv x(z).y(y)$ Not true! $(\ldots \parallel \overline{x}(d))$
3. $x \parallel y \equiv x.y + y.x$ Not true! (If $y = \overline{x}$ then: $a(x,x).(x \parallel \overline{y}) \not\equiv a(x,x).(x\overline{y} + \overline{y}.x)$)
4. $x(y).x(z) \parallel y(z).z(y) \equiv y(y).y(z) \parallel x(z).x(y)$

UPPSALA UNIVERSITET

Structural Congruence

Examples. Notice that $(\nu c)(\overline{a}(b) \parallel 0) \equiv \overline{a}(b)$.

Do we have?

1. $(\nu a)P \equiv P$ if $a \notin fn(P)$ True!
2. $x(y).y(z) \equiv x(z).y(y)$ Not true! $(\ldots \parallel \overline{x}(d))$
3. $x \parallel y \equiv x.y + y.x$ Not true! (If $y = \overline{x}$ then: $a(x, x).(x \parallel \overline{y}) \not\equiv a(x, x).(x\overline{y} + \overline{y}.x)$)
4. $x(y).x(z) \parallel y(z).z(y) \equiv y(y).y(z) \parallel x(z).x(y)$ True!

UPPSALA UNIVERSITET

Reactive Rules

TAU $\dfrac{}{\tau.P + M \longrightarrow P}$

REACT $\dfrac{}{(a(x).P + M) \parallel (\overline{a}(b).Q + N) \longrightarrow P[b/x] \parallel Q}$

STRUCT $\dfrac{P \longrightarrow P'}{Q \longrightarrow Q'}$ if $P \equiv Q$ and $P' \equiv Q'$

PAR $\dfrac{P \longrightarrow P'}{P \parallel Q \longrightarrow P' \parallel Q}$ RES $\dfrac{P \longrightarrow P'}{(\nu a)P \longrightarrow (\nu a)P'}$

Reactive Rules

- How to express that a private channel can / cannot be “exported”?

Reactive Rules

- How to express that a private channel can / cannot be “exported”?
  Example:

  $$((\nu a)\overline{x}(a).0) \parallel x(y).P$$

  How can we reflect that $\overline{x}$ communicate with x?

Reactive Rules

- How to express that a private channel can / cannot be "exported"?
  Example:

  $$((\nu a)\overline{x}(a).0) \parallel x(y).P$$

  How can we reflect that $\overline{x}$ communicate with x?
- It seems that the rules do not allow to do so
- This is hidden somewhere; where?

UPPSALA UNIVERSITET

Reactive Rules

- How to express that a private channel can / cannot be "exported"?
  Example:

  $$((\nu a)\overline{x}(a).0) \parallel x(y).P$$

  How can we reflect that $\overline{x}$ communicate with x?
- It seems that the rules do not allow to do so
- This is hidden somewhere; where?
  Answer: In the STRUCT rule!

Reactive Rules

Example. Give reductions for

$$x(z).\overline{y}(z) \;\; \| \;\; !(\nu y)\overline{x}(y).Q$$

π: Some Remarks

We have **not** considered in this course:

- How to introduce recursion: recursive operator vs. replication
- Notions of process equivalence: weak, early, open, late bisimulations; barbed congruence
- Variants of semantics: symbolic, late, early semantics, etc

Road Map

- Basic classic automata theory and the limitation of language equivalence.
- Bisimilarity Equivalence for automata (transition systems).
- CCS
- Mobility and the π calculus
- Petri Nets

UPPSALA UNIVERSITET

Petri Nets

- A Petri net is a bipartite graph whose
  - Classes of nodes (*places*) represent system conditions and resources
  - Each place can contain *tokens*
  - *Transitions* represent system activities
- First model for (true) concurrency (Petri, 1962)
- Widely used for analysis and verification of concurrent systems

UPPSALA UNIVERSITET

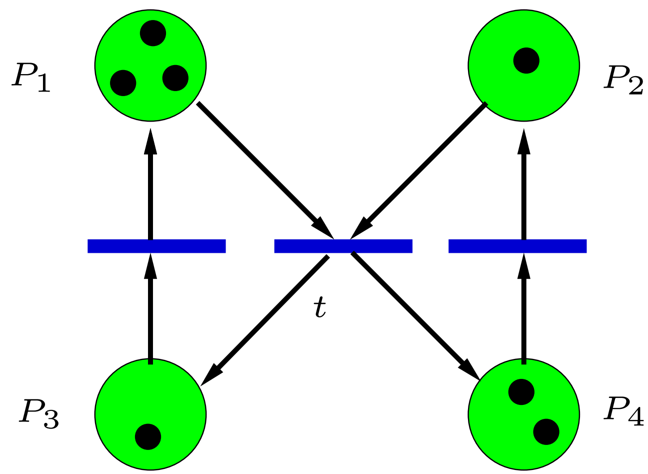
Petri Nets (more formally)

- A Petri net is a tuple $N = (P, A, T, M_0)$:
  - P is a finite set of *places*
  - A is a finite set of *actions* (or *labels*)
  - $T \subseteq \mathcal{M}(P) \times A \times \mathcal{M}(P)$ is a finite set of *transitions*
  - M_0 is the *initial marking*

where $\mathcal{M}(P)$ is a collection of multisets (bags) over P

UPPSALA UNIVERSITET

Graphical representation



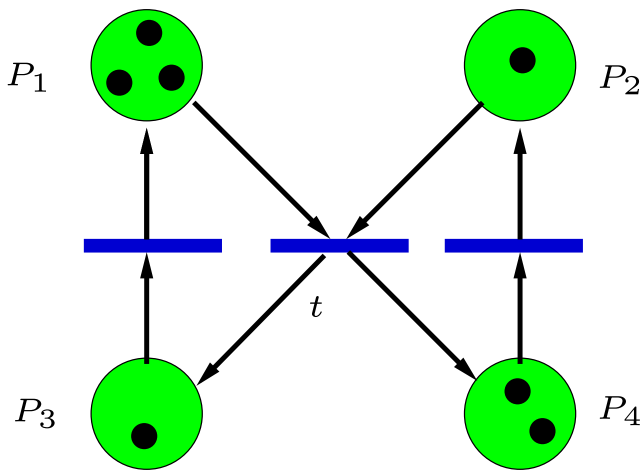
Marking

M : is a mapping from places to the set of natural numbers

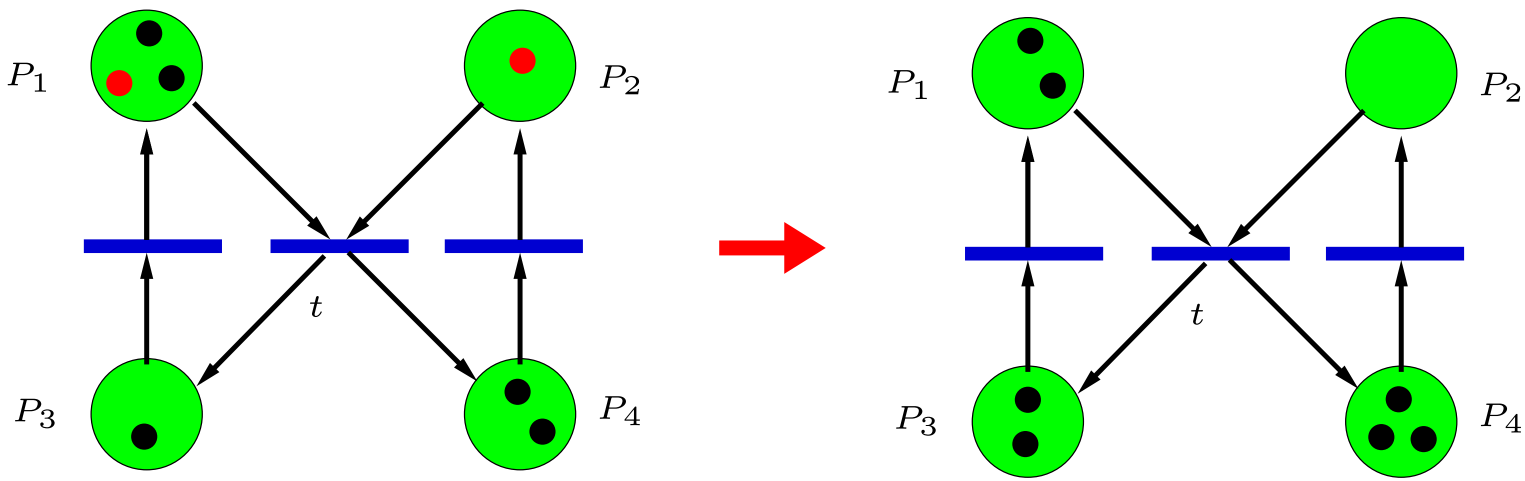
$$M(P_1) = 3 \quad M(P_2) = 1$$
$$M(P_3) = 1 \quad M(P_4) = 2$$

UPPSALA UNIVERSITET

Transition relation (fi ring)

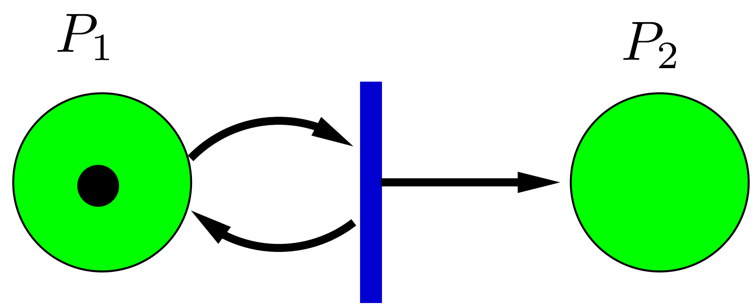


Transition relation (firing)



Petri Nets: Some remarks

Petri nets are infinite-state systems
Example:

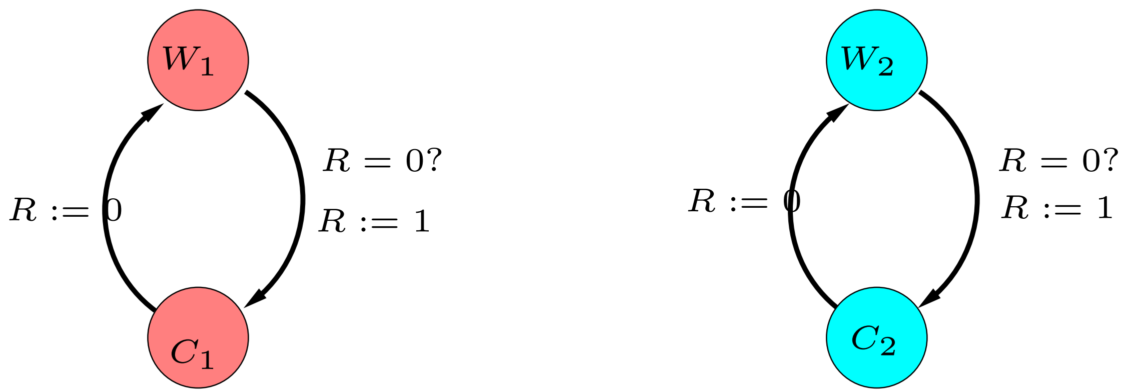


Starting with $M(P_1) = 1$ and $M(P_2) = 0$ (10)
Firing the transition successively gives:
$11, 12, 13, 14, \ldots$

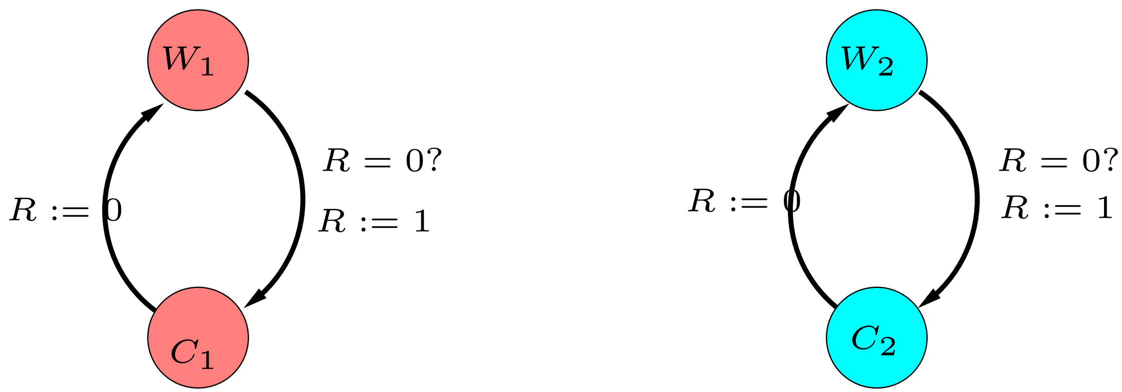
Exercise: How to generate the Natural numbers, using Petri nets?

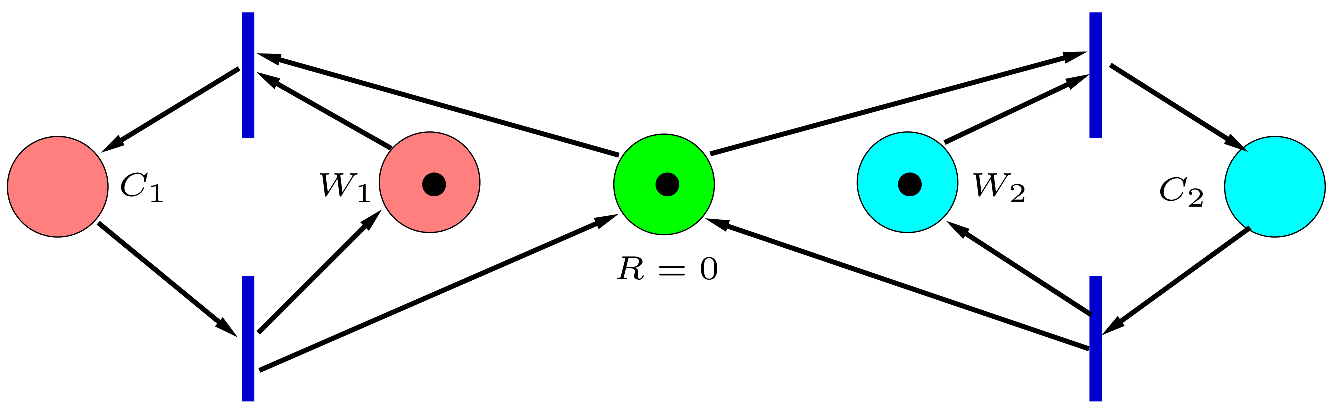
UPPSALA UNIVERSITET

Example: mutex

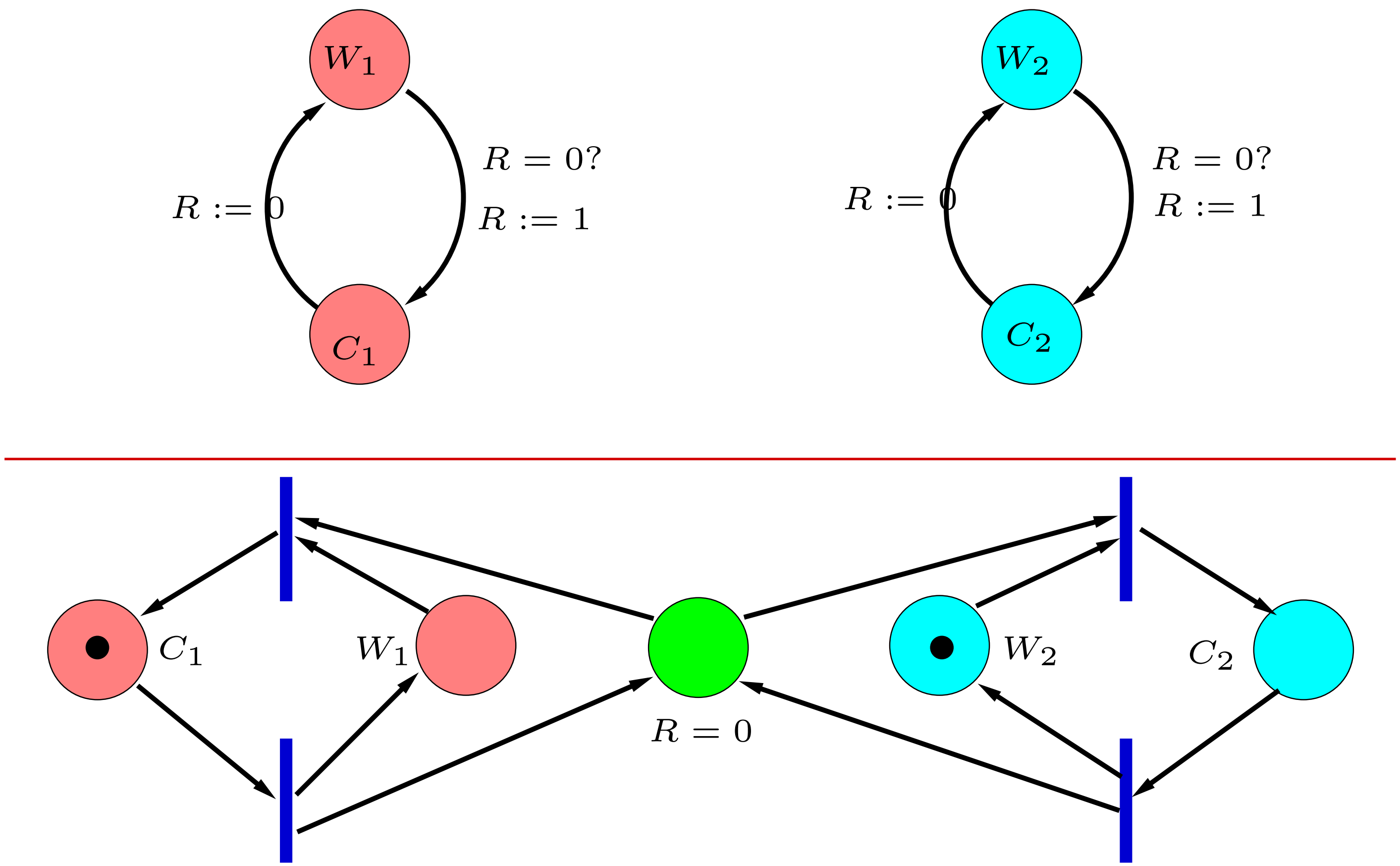


Example: mutex



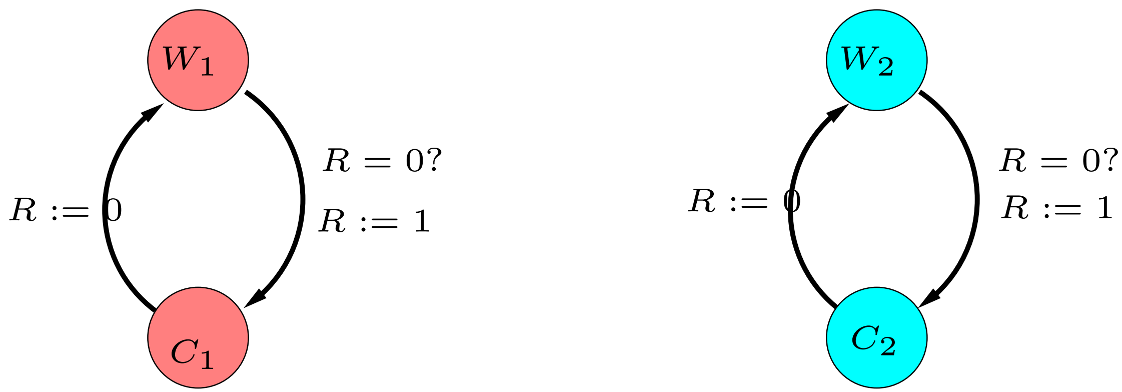


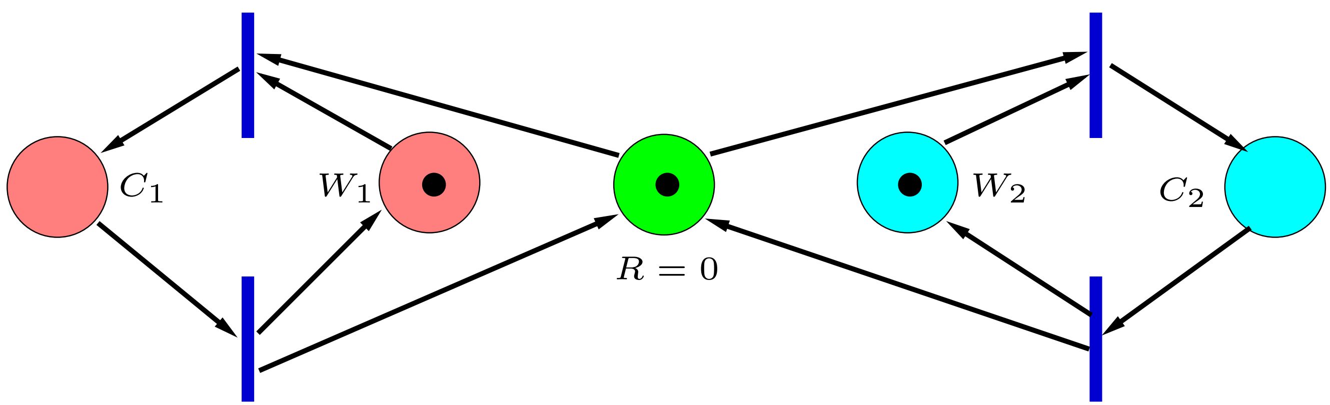
Example: mutex



UPPSALA UNIVERSITET

Example: mutex





UPPSALA UNIVERSITET

Road Map

- Basic classic automata theory and the limitation of language equivalence.
- Bisimilarity Equivalence for automata (transition systems).
- CCS
- Mobility and the π calculus
- Petri Nets