Modelling and Analysing Concurrent Systems

RIO 2023 Summer School of Informatics
Rio Cuarto, Argentina; February 13–17, 2023

Lecture 2: Behavioural Equivalences

Thomas Noll
Software Modelling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-22-23/rio/

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

The Exam

Homework

- Modelling and implementation of a simple communication protocol using the CAAL tool
- Working in teams of $\leq$ 2 people
- Details made available soon via
  `https://moves.rwth-aachen.de/teaching/ws-22-23/rio/`
- Report to be submitted via e-mail no later than March 31, 2023 (but preferably earlier)

Outline of Lecture 2

Syntax of CCS I

Definition (Syntax of CCS)

- Let A be a set of (action) names.
- $\overline{A} := \{\overline{a} \mid a \in A\}$ denotes the set of co-names.
- $Act := A \cup \overline{A} \cup \{\tau\}$ is the set of actions with the silent (or: unobservable) action τ.
- Let Pid be a set of process identifiers.
- The set Prc of process expressions is defined by the following syntax:

$$
\begin{array}{rll}
P ::= & \mathrm{nil} & \text{(inaction)} \\
\mid & \alpha.P & \text{(prefixing)} \\
\mid & P_1 + P_2 & \text{(choice)} \\
\mid & P_1 \parallel P_2 & \text{(parallel composition)} \\
\mid & P \setminus L & \text{(restriction)} \\
\mid & P[f] & \text{(relabelling)} \\
\mid & C & \text{(process call)}
\end{array}
$$

where $\alpha \in Act$, $\emptyset \neq L \subseteq A$, $C \in Pid$, and $f : Act \to Act$ such that $f(\tau) = \tau$ and $f(\overline{a}) = \overline{f(a)}$ for each $a \in A$.

Syntax of CCS II

Definition (continued)

- A (recursive) process definition is an equation system of the form

$$(C_i = P_i \mid 1 \leq i \leq k)$$

where $k \geq 1$, $C_i \in \mathit{Pid}$ (pairwise distinct), and $P_i \in \mathit{Prc}$ (with identifiers from $\{C_1, \ldots, C_k\}$).

Semantics of CCS I

Reminder: $P ::= \text{nil} \mid \alpha.P \mid P_1 + P_2 \mid P_1 \parallel P_2 \mid P \setminus L \mid P[f] \mid C$

Definition (Semantics of CCS)

A process definition $(C_i = P_i \mid 1 \leq i \leq k)$ determines the LTS $(Prc, Act, \longrightarrow)$ whose transitions can be inferred from the following rules ($P, P', Q, Q' \in Prc$, $\alpha \in Act$, $\lambda \in A \cup \overline{A}$, $a \in A$):

$$\text{(Act)}\ \frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \text{(Sum}_1\text{)}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \text{(Sum}_2\text{)}\ \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\text{(Par}_1\text{)}\ \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \qquad \text{(Par}_2\text{)}\ \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'} \qquad \text{(Com)}\ \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$\text{(Res)}\ \frac{P \xrightarrow{\alpha} P' \quad (\alpha, \overline{\alpha} \notin L)}{P \setminus L \xrightarrow{\alpha} P' \setminus L} \qquad \text{(Rel)}\ \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]} \qquad \text{(Call)}\ \frac{P \xrightarrow{\alpha} P' \quad (C = P)}{C \xrightarrow{\alpha} P'}$$

Example (continued)

(3) Parallel two-place buffer:

$$B_{\|} = (B[f] \parallel B[g]) \setminus com$$
$$B = in.\overline{out}.B$$

($f := [out \mapsto com]$, $g := [in \mapsto com]$)

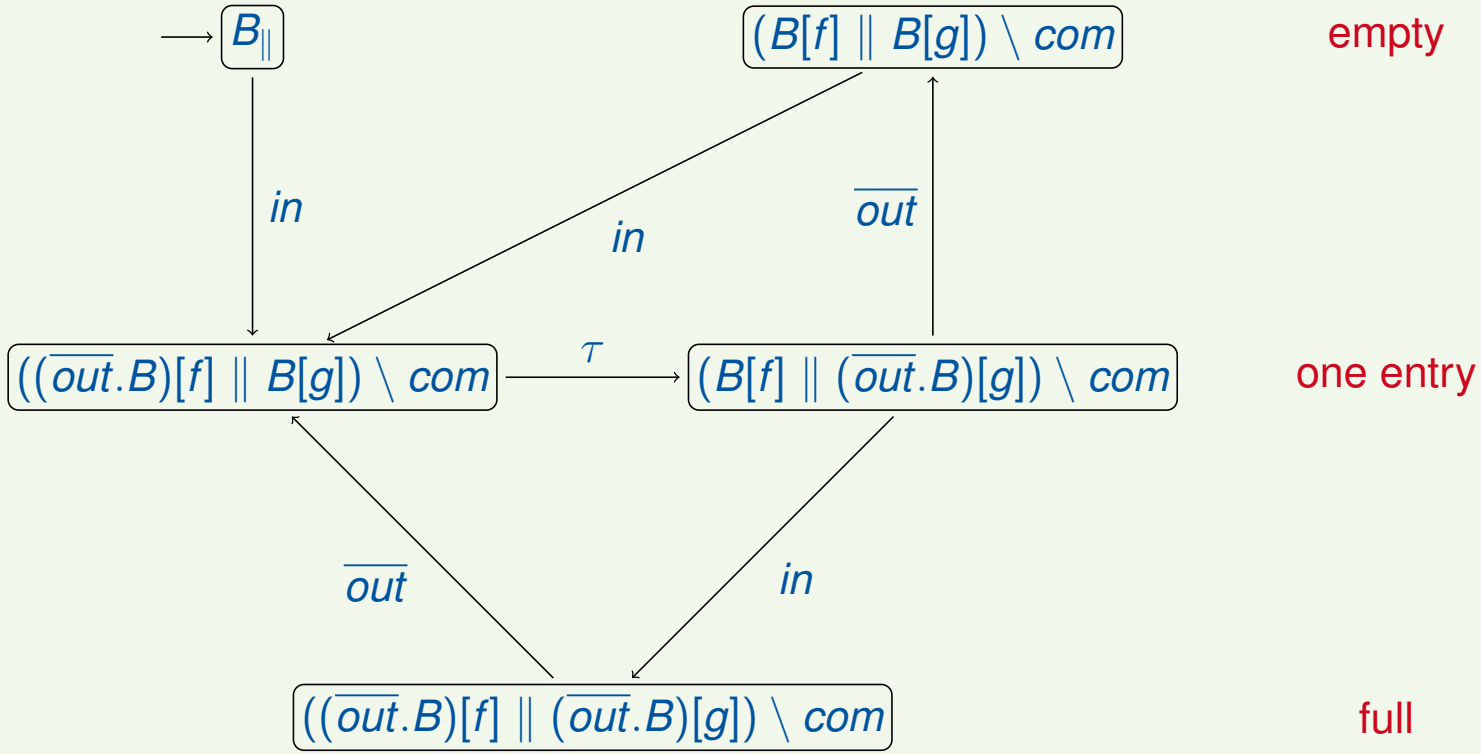
First step:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{in.\overline{out}.B \xrightarrow{in} \overline{out}.B}\,(\text{Act})}{B \xrightarrow{in} \overline{out}.B}\,(\text{Call})}{B[f] \xrightarrow{in} (\overline{out}.B)[f]}\,(\text{Rel})}{B[f] \parallel B[g] \xrightarrow{in} (\overline{out}.B)[f] \parallel B[g]}\,(\text{Par}_1)}{(B[f] \parallel B[g]) \setminus com \xrightarrow{in} ((\overline{out}.B)[f] \parallel B[g]) \setminus com}\,(\text{Res})}{B_{\|} \xrightarrow{in} ((\overline{out}.B)[f] \parallel B[g]) \setminus com}\,(\text{Call})$$

Example (continued)

(3) Parallel two-place buffer: $B_{\|} = (B[f] \parallel B[g]) \setminus com$ $\quad (f := [out \mapsto com], g := [in \mapsto com])$
$B = in.\overline{out}.B$

Complete LTS:



Outline of Lecture 2

Preliminaries

- When using process algebras like CCS, an important approach is to model both the specification and implementation as CCS processes, say *Spec* and *Impl*.
  - two-place buffer (Example 1.3): sequential “specification” vs. parallel implementation
  - mutual exclusion (later)

Preliminaries

- When using process algebras like CCS, an important approach is to model both the specification and implementation as CCS processes, say *Spec* and *Impl*.
  - two-place buffer (Example 1.3): sequential “specification” vs. parallel implementation
  - mutual exclusion (later)
- This gives rise to the natural question: when are two CCS processes behaving the same?

Preliminaries

- When using process algebras like CCS, an important approach is to model both the specification and implementation as CCS processes, say *Spec* and *Impl*.
  - two-place buffer (Example 1.3): sequential “specification” vs. parallel implementation
  - mutual exclusion (later)
- This gives rise to the natural question: when are two CCS processes behaving the same?
- As there are many different interpretations of “behaving the same”, different behavioural equivalences have emerged.

Behavioural Equivalence

Implementation

$$
\begin{aligned}
CM &= coin.\overline{coffee}.CM \\
CS &= \overline{pub}.\overline{coin}.coffee.CS \\
Uni &= (CM \parallel CS) \setminus \{coin, coffee\}
\end{aligned}
$$

Behavioural Equivalence

Implementation

$$CM = coin.\overline{coffee}.CM$$

$$CS = \overline{pub}.\overline{coin}.coffee.CS$$

$$Uni = (CM \parallel CS) \setminus \{coin, coffee\}$$

Specification

$$Spec = \overline{pub}.Spec$$

Behavioural Equivalence

Implementation

$$\begin{aligned} CM &= coin.\overline{coffee}.CM \\ CS &= \overline{pub}.\overline{coin}.coffee.CS \\ Uni &= (CM \parallel CS) \setminus \{coin, coffee\} \end{aligned}$$

Specification

$$Spec = \overline{pub}.Spec$$

Question

Are the specification *Spec* and implementation *Uni* behaviourally equivalent:

$$Spec \stackrel{?}{\equiv} Uni$$

Equivalence Relations

Some reasonable required properties

- Reflexivity: $P \equiv P$ for every process P
- Symmetry: $P \equiv Q$ if and only if $Q \equiv P$
- Transitivity: $Spec_0 \equiv \ldots \equiv Spec_n \equiv Impl$ implies that $Spec_0 \equiv Impl$

Equivalence Relations

Some reasonable required properties

- Reflexivity: $P \equiv P$ for every process P
- Symmetry: $P \equiv Q$ if and only if $Q \equiv P$
- Transitivity: $Spec_0 \equiv \ldots \equiv Spec_n \equiv Impl$ implies that $Spec_0 \equiv Impl$

Definition 2.3 (Equivalence relation)

A binary relation $\equiv \subseteq S \times S$ over a set S is an equivalence if

- it is reflexive: $s \equiv s$ for every $s \in S$,
- it is symmetric: $s \equiv t$ implies $t \equiv s$ for every $s, t \in S$,
- it is transitive: $s \equiv t$ and $t \equiv u$ implies $s \equiv u$ for every $s, t, u \in S$.

Equivalence Relations

Some reasonable required properties

- Reflexivity: $P \equiv P$ for every process P
- Symmetry: $P \equiv Q$ if and only if $Q \equiv P$
- Transitivity: $Spec_0 \equiv \ldots \equiv Spec_n \equiv Impl$ implies that $Spec_0 \equiv Impl$

Definition 2.3 (Equivalence relation)

A binary relation $\equiv \subseteq S \times S$ over a set S is an equivalence if

- it is reflexive: $s \equiv s$ for every $s \in S$,
- it is symmetric: $s \equiv t$ implies $t \equiv s$ for every $s, t \in S$,
- it is transitive: $s \equiv t$ and $t \equiv u$ implies $s \equiv u$ for every $s, t, u \in S$.

Remark: equivalences induce quotient structures with equivalence classes as elements:

$$S/\equiv \;:=\; \{[s]_\equiv \mid s \in S\} \subseteq 2^S \quad \text{where} \quad [s]_\equiv \;:=\; \{s' \in S \mid s' \equiv s\} \subseteq S$$

Isomorphism: An Example Behavioural Equivalence

Definition 2.4 (LTS isomorphism)

Two LTSs $T_1 = (S_1, Act_1, \longrightarrow_1)$ and $T_2 = (S_2, Act_2, \longrightarrow_2)$ are isomorphic, denoted $T_1 \equiv_{iso} T_2$, if there exists a bijection $f : S_1 \to S_2$ such that

$$s \xrightarrow{\alpha}_1 t \quad \text{if and only if} \quad f(s) \xrightarrow{\alpha}_2 f(t).$$

Isomorphism: An Example Behavioural Equivalence

Definition 2.4 (LTS isomorphism)

Two LTSs $T_1 = (S_1, Act_1, \longrightarrow_1)$ and $T_2 = (S_2, Act_2, \longrightarrow_2)$ are isomorphic, denoted $T_1 \equiv_{iso} T_2$, if there exists a bijection $f : S_1 \to S_2$ such that

$$s \xrightarrow{\alpha}_1 t \quad \text{if and only if} \quad f(s) \xrightarrow{\alpha}_2 f(t).$$

It follows immediately that $\equiv_{iso}$ is an equivalence.

Isomorphism: An Example Behavioural Equivalence

Definition 2.4 (LTS isomorphism)

Two LTSs $T_1 = (S_1, Act_1, \longrightarrow_1)$ and $T_2 = (S_2, Act_2, \longrightarrow_2)$ are isomorphic, denoted $T_1 \equiv_{iso} T_2$, if there exists a bijection $f : S_1 \to S_2$ such that

$$s \xrightarrow{\alpha}_1 t \quad \text{if and only if} \quad f(s) \xrightarrow{\alpha}_2 f(t).$$

It follows immediately that $\equiv_{iso}$ is an equivalence.

Assumption

From now on, we will consider processes modulo isomorphism, i.e., we do not distinguish CCS processes with isomorphic LTSs.

Isomorphism: An Example Behavioural Equivalence

Definition 2.4 (LTS isomorphism)

Two LTSs $T_1 = (S_1, Act_1, \longrightarrow_1)$ and $T_2 = (S_2, Act_2, \longrightarrow_2)$ are isomorphic, denoted $T_1 \equiv_{iso} T_2$, if there exists a bijection $f : S_1 \to S_2$ such that

$$s \xrightarrow{\alpha}_1 t \quad \text{if and only if} \quad f(s) \xrightarrow{\alpha}_2 f(t).$$

It follows immediately that $\equiv_{iso}$ is an equivalence.

Assumption

From now on, we will consider processes modulo isomorphism, i.e., we do not distinguish CCS processes with isomorphic LTSs.

Caveat

But: isomorphism is very distinctive. For instance,

$$X = a.X \quad \text{and} \quad Y = a.a.Y$$

are distinguished although both can (only) execute infinitely many a-actions and should thus be considered equivalent.

Outline of Lecture 2

The Wish List for Behavioural Equivalences

(1) Less distinctive than isomorphism: an equivalence should distinguish less processes than LTS isomorphism does, i.e., $\equiv$ should be coarser than LTS isomorphism:

$$LTS(P) \equiv_{iso} LTS(Q) \Rightarrow P \equiv Q.$$

The Wish List for Behavioural Equivalences

(1) Less distinctive than isomorphism: an equivalence should distinguish less processes than LTS isomorphism does, i.e., $\equiv$ should be coarser than LTS isomorphism:

$$LTS(P) \equiv_{iso} LTS(Q) \Rightarrow P \equiv Q.$$

(2) More distinctive than trace equivalence: an equivalence should distinguish more processes than trace equivalence does, i.e., $\equiv$ should be finer than trace equivalence:

$$P \equiv Q \Rightarrow Tr(P) = Tr(Q).$$

The Wish List for Behavioural Equivalences

(1) Less distinctive than isomorphism: an equivalence should distinguish less processes than LTS isomorphism does, i.e., $\equiv$ should be coarser than LTS isomorphism:

$$LTS(P) \equiv_{iso} LTS(Q) \Rightarrow P \equiv Q.$$

(2) More distinctive than trace equivalence: an equivalence should distinguish more processes than trace equivalence does, i.e., $\equiv$ should be finer than trace equivalence:

$$P \equiv Q \Rightarrow Tr(P) = Tr(Q).$$

(3) Congruence property: the equivalence must be substitutive with respect to all CCS operators (see next slide).

The Wish List for Behavioural Equivalences

(1) Less distinctive than isomorphism: an equivalence should distinguish less processes than LTS isomorphism does, i.e., $\equiv$ should be coarser than LTS isomorphism:

$$LTS(P) \equiv_{iso} LTS(Q) \Rightarrow P \equiv Q.$$

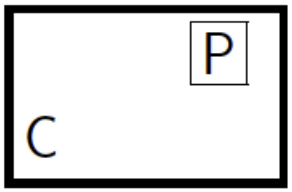
(2) More distinctive than trace equivalence: an equivalence should distinguish more processes than trace equivalence does, i.e., $\equiv$ should be finer than trace equivalence:

$$P \equiv Q \Rightarrow Tr(P) = Tr(Q).$$

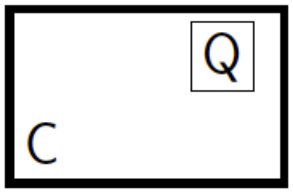
(3) Congruence property: the equivalence must be substitutive with respect to all CCS operators (see next slide).

(4) Deadlock preservation: equivalent processes should have the same deadlock behaviour, i.e., they can either both deadlock, or both cannot.

What is a Congruence?

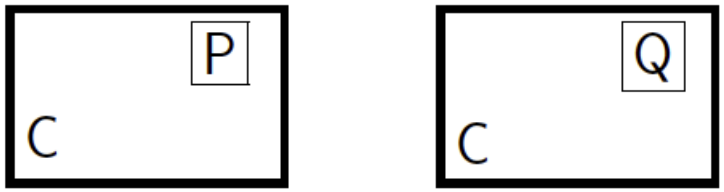


$C(P)$



$C(Q)$

What is a Congruence?



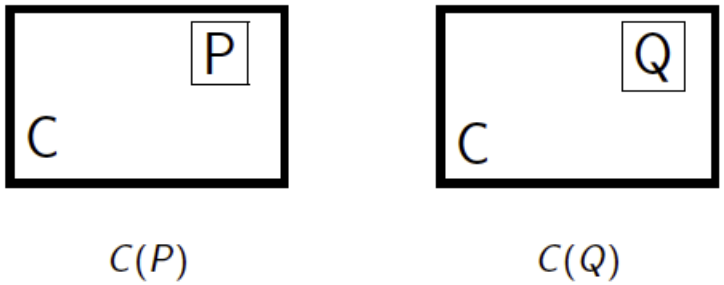
$C(P)$ $C(Q)$

CCS contexts informally

A CCS context is a CCS process fragment $C(\circ)$ with a "hole" in it, for example:

- $\circ$ (empty context)
- $a.\mathrm{nil} + \circ$
- $(\circ[a \mapsto b] \parallel B) \setminus b$

What is a Congruence?



$C(P)$ $C(Q)$

CCS contexts informally

A CCS context is a CCS process fragment $C(\circ)$ with a "hole" in it, for example:

- $\circ$ (empty context)
- $a.\mathrm{nil} + \circ$
- $(\circ[a \mapsto b] \parallel B) \setminus b$

CCS congruences informally

Relation $\equiv$ is a CCS congruence whenever $P \equiv Q$ implies $C(P) \equiv C(Q)$ for every CCS context C.

The Importance of Congruences

CCS congruences informally

Relation $\equiv$ is a CCS congruence whenever $P \equiv Q$ implies $C(P) \equiv C(Q)$ for every CCS context C.

The Importance of Congruences

CCS congruences informally

Relation $\equiv$ is a CCS congruence whenever $P \equiv Q$ implies $C(P) \equiv C(Q)$ for every CCS context C.

Important motivations for requiring $\equiv$ to be a congruence on processes:

(1) Model-based development through refinement: replacing (part of) an abstract model *Spec* by a more detailed model *Impl*.

(2) Optimisation: replacing (part of) an implementation *Impl* by a more efficient implementation *Impl*$'$.

CCS Congruences Formally

Definition 2.5 (CCS congruence)

An equivalence relation $\equiv \subseteq \mathit{Prc} \times \mathit{Prc}$ is a CCS congruence if it is preserved by all CCS constructs, i.e., if $P, Q \in \mathit{Prc}$ with $P \equiv Q$ then:

$$
\begin{aligned}
\alpha.P &\equiv \alpha.Q && \text{for every } \alpha \in \mathit{Act} \\
P + R &\equiv Q + R && \text{for every } R \in \mathit{Prc} \\
P \parallel R &\equiv Q \parallel R && \text{for every } R \in \mathit{Prc} \\
P \setminus L &\equiv Q \setminus L && \text{for every } L \subseteq A \\
P[f] &\equiv Q[f] && \text{for every } f : A \to A
\end{aligned}
$$

CCS Congruences Formally

Definition 2.5 (CCS congruence)

An equivalence relation $\equiv \subseteq \mathit{Prc} \times \mathit{Prc}$ is a CCS congruence if it is preserved by all CCS constructs, i.e., if $P, Q \in \mathit{Prc}$ with $P \equiv Q$ then:

$$
\begin{aligned}
\alpha.P &\equiv \alpha.Q && \text{for every } \alpha \in \mathit{Act} \\
P + R &\equiv Q + R && \text{for every } R \in \mathit{Prc} \\
P \parallel R &\equiv Q \parallel R && \text{for every } R \in \mathit{Prc} \\
P \setminus L &\equiv Q \setminus L && \text{for every } L \subseteq A \\
P[f] &\equiv Q[f] && \text{for every } f : A \to A
\end{aligned}
$$

Thus, a CCS congruence is substitutive for all possible CCS contexts.

Deadlocks

Definition 2.6 (Deadlock)

Let $P, Q \in Prc$ and $w \in Act^*$ such that

$$P \xrightarrow{w} Q \quad \text{and} \quad Q \not\longrightarrow$$

(where $\xrightarrow{w} := \xrightarrow{\alpha_1} \circ \ldots \circ \xrightarrow{\alpha_n}$ for $w = a_1 \ldots a_n$). Then Q is called a w-deadlock of P.

Deadlocks

Definition 2.6 (Deadlock)

Let $P, Q \in Prc$ and $w \in Act^*$ such that

$$P \xrightarrow{w} Q \quad \text{and} \quad Q \not\longrightarrow$$

(where $\xrightarrow{w} := \xrightarrow{\alpha_1} \circ \ldots \circ \xrightarrow{\alpha_n}$ for $w = a_1 \ldots a_n$). Then Q is called a w-deadlock of P.

Example 2.7

$P = a.b.\text{nil} + a.\text{nil}$ has an a-deadlock, whereas $Q = a.b.\text{nil}$ has not.
Such properties are important as it can be crucial that a certain action is eventually enabled.

Deadlocks

Definition 2.6 (Deadlock)

Let $P, Q \in Prc$ and $w \in Act^*$ such that

$$P \xrightarrow{w} Q \quad \text{and} \quad Q \not\longrightarrow$$

(where $\xrightarrow{w} := \xrightarrow{\alpha_1} \circ \ldots \circ \xrightarrow{\alpha_n}$ for $w = a_1 \ldots a_n$). Then Q is called a w-deadlock of P.

Example 2.7

$P = a.b.\mathrm{nil} + a.\mathrm{nil}$ has an a-deadlock, whereas $Q = a.b.\mathrm{nil}$ has not.
Such properties are important as it can be crucial that a certain action is eventually enabled.

Definition 2.8 (Deadlock sensitivity)

Relation $\equiv \subseteq Prc \times Prc$ is deadlock sensitive whenever:

$$P \equiv Q \quad \text{implies} \quad \left(\forall w \in Act^* : P \text{ has a } w\text{-deadlock iff } Q \text{ has a } w\text{-deadlock}\right).$$

Outline of Lecture 2

Process Traces I

Goal: reduce processes to the action sequences they can perform.

Definition 2.9 (Trace language)

For every $P \in Prc$, let

$$Tr(P) := \{w \in Act^* \mid \text{ex. } P' \in Prc \text{ such that } P \xrightarrow{w} P'\}$$

be the trace language of P. $P, Q \in Prc$ are called trace equivalent if $Tr(P) = Tr(Q)$.

Process Traces I

Goal: reduce processes to the action sequences they can perform.

Definition 2.9 (Trace language)

For every $P \in Prc$, let

$$Tr(P) := \{w \in Act^* \mid \text{ex. } P' \in Prc \text{ such that } P \xrightarrow{w} P'\}$$

be the trace language of P. $P, Q \in Prc$ are called trace equivalent if $Tr(P) = Tr(Q)$.

Example 2.10 (One-place buffer)

$B = in.\overline{out}.B$

$\Rightarrow Tr(B) = (in \cdot \overline{out})^* \cdot (in + \varepsilon)$

Process Traces II

Remarks:

- The trace language of $P \in Prc$ is accepted by the LTS of P, interpreted as a (finite or infinite) automaton with initial state P and where every state is final.

Process Traces II

Remarks:

- The trace language of $P \in Prc$ is accepted by the LTS of P, interpreted as a (finite or infinite) automaton with initial state P and where every state is final.
- Trace equivalence is obviously an equivalence relation (i.e., reflexive, symmetric, and transitive).

Process Traces II

Remarks:

- The trace language of $P \in Prc$ is accepted by the LTS of P, interpreted as a (finite or infinite) automaton with initial state P and where every state is final.
- Trace equivalence is obviously an equivalence relation (i.e., reflexive, symmetric, and transitive).
- Trace equivalence is less distinctive than LTS isomorphism: the trace language of a process consists of the (finite) paths in the LTS. Thus:

$$LTS(P) = LTS(Q) \Rightarrow Tr(P) = Tr(Q)$$

Process Traces II

Remarks:

- The trace language of $P \in Prc$ is accepted by the LTS of P, interpreted as a (finite or infinite) automaton with initial state P and where every state is final.
- Trace equivalence is obviously an equivalence relation (i.e., reflexive, symmetric, and transitive).
- Trace equivalence is less distinctive than LTS isomorphism: the trace language of a process consists of the (finite) paths in the LTS. Thus:

$$LTS(P) = LTS(Q) \Rightarrow Tr(P) = Tr(Q)$$

- Later we will see: trace equivalence is too coarse, i.e., identifies too many processes $\Rightarrow$ bisimulation

Outline of Lecture 2

Trace Equivalence is a Congruence

Theorem 2.11

Trace equivalence is a CCS congruence.

Trace Equivalence is a Congruence

Theorem 2.11

Trace equivalence is a CCS congruence.

Proof.

By structural induction over the syntax of CCS processes.

Trace Equivalence is a Congruence

Theorem 2.11

Trace equivalence is a CCS congruence.

Proof.

By structural induction over the syntax of CCS processes.
For $+$ this proceeds as follows:

Trace Equivalence is a Congruence

Theorem 2.11

Trace equivalence is a CCS congruence.

Proof.

By structural induction over the syntax of CCS processes.
For $+$ this proceeds as follows:

- Let $P, Q \in Prc$ with $Tr(P) = Tr(Q)$.

Trace Equivalence is a Congruence

Theorem 2.11

Trace equivalence is a CCS congruence.

Proof.

By structural induction over the syntax of CCS processes.
For $+$ this proceeds as follows:

- Let $P, Q \in Prc$ with $Tr(P) = Tr(Q)$.
- Then for $R \in Prc$ it holds:

$$Tr(P + R) = Tr(P) \cup Tr(R) = Tr(Q) \cup Tr(R) = Tr(Q + R).$$

Trace Equivalence is a Congruence

Theorem 2.11

Trace equivalence is a CCS congruence.

Proof.

By structural induction over the syntax of CCS processes.
For $+$ this proceeds as follows:

- Let $P, Q \in Prc$ with $Tr(P) = Tr(Q)$.
- Then for $R \in Prc$ it holds:

$$Tr(P + R) = Tr(P) \cup Tr(R) = Tr(Q) \cup Tr(R) = Tr(Q + R).$$

- Thus, $P + R$ and $Q + R$ are trace equivalent.

Trace Equivalence is a Congruence

Theorem 2.11

Trace equivalence is a CCS congruence.

Proof.

By structural induction over the syntax of CCS processes.
For $+$ this proceeds as follows:

- Let $P, Q \in Prc$ with $Tr(P) = Tr(Q)$.
- Then for $R \in Prc$ it holds:

$$Tr(P + R) = Tr(P) \cup Tr(R) = Tr(Q) \cup Tr(R) = Tr(Q + R).$$

- Thus, $P + R$ and $Q + R$ are trace equivalent.

For the other CCS constructs, the proof goes along similar lines. □

Two Coffee/Tea Machines

Example 2.12

Consider the coffee/tea machines CTM and its variant CTM':

$$\begin{aligned} CTM &= coin.\left(\overline{coffee}.CTM + \overline{tea}.CTM\right) \\ CTM' &= coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM'. \end{aligned}$$

Two Coffee/Tea Machines

Example 2.12

Consider the coffee/tea machines CTM and its variant CTM':

$$\begin{aligned} CTM &= coin.\left(\overline{coffee}.CTM + \overline{tea}.CTM\right) \\ CTM' &= coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM'. \end{aligned}$$

Note the difference between the two processes. Nevertheless:

$$Tr(CTM) = Tr(CTM').$$

Two Coffee/Tea Machines

Example 2.12

Consider the coffee/tea machines CTM and its variant CTM':

$$\begin{aligned} CTM &= coin.\left(\overline{coffee}.CTM + \overline{tea}.CTM\right) \\ CTM' &= coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM'. \end{aligned}$$

Note the difference between the two processes. Nevertheless:

$$Tr(CTM) = Tr(CTM').$$

Are we satisfied?

Two Coffee/Tea Machines

Example 2.12

Consider the coffee/tea machines CTM and its variant CTM':

$$\begin{aligned} CTM &= coin.\left(\overline{coffee}.CTM + \overline{tea}.CTM\right) \\ CTM' &= coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM'. \end{aligned}$$

Note the difference between the two processes. Nevertheless:

$$Tr(CTM) = Tr(CTM').$$

Are we satisfied?

No, as CTM and CTM' differ in the context:

$$C(\circ) = (\underbrace{\circ}_{\text{hole}} \parallel CA) \setminus \{coin, coffee, tea\} \quad \text{with} \quad CA = \overline{coin}.coffee.CA.$$

Two Coffee/Tea Machines

Example 2.12

Consider the coffee/tea machines CTM and its variant CTM':

$$\begin{aligned} CTM &= coin.\left(\overline{coffee}.CTM + \overline{tea}.CTM\right) \\ CTM' &= coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM'. \end{aligned}$$

Note the difference between the two processes. Nevertheless:

$$Tr(CTM) = Tr(CTM').$$

Are we satisfied?

No, as CTM and CTM' differ in the context:

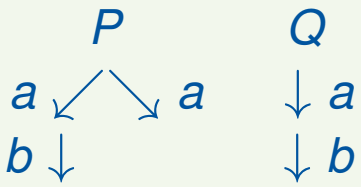
$$C(\circ) = (\underbrace{\circ}_{\text{hole}} \parallel CA) \setminus \{coin, coffee, tea\} \quad \text{with} \quad CA = \overline{coin}.coffee.CA.$$

Why? $C(CTM')$ may yield a deadlock, but $C(CTM)$ does not.

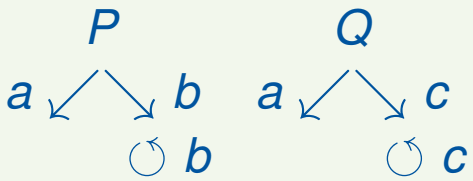
Traces and Deadlocks

Example 2.13 (Traces and deadlocks)

Traces and deadlocks are independent in the following sense:



same traces
different deadlocks

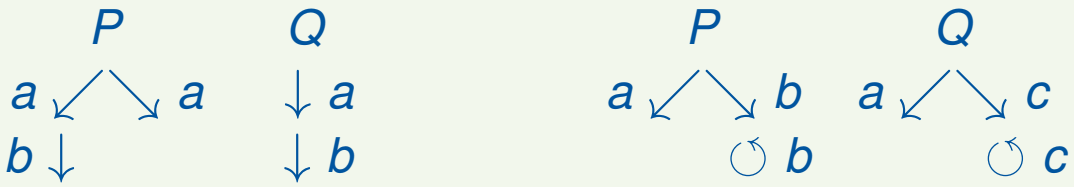


different traces
same deadlocks

Traces and Deadlocks

Example 2.13 (Traces and deadlocks)

Traces and deadlocks are independent in the following sense:



same traces
different deadlocks

different traces
same deadlocks

But: processes with finite trace sets and identical deadlocks are trace equivalent (since every trace is a prefix of some deadlock).

Summary: Trace Equivalence

(1) Trace equivalence equates processes that have the same traces, i.e., can execute the same action sequences.

(2) Isomorphism implies trace equivalence.

(3) Trace equivalence trivially implies trace equivalence.

(4) Trace equivalence is a CCS congruence

(5) Trace equivalence is not deadlock sensitive.

Outline of Lecture 2

Rationale

Observation

In order for a behavioural equivalence to be deadlock sensitive, it has to take the branching structure of processes into account.

Rationale

Observation

In order for a behavioural equivalence to be deadlock sensitive, it has to take the branching structure of processes into account.

This is achieved by an equivalence that is defined according to the following scheme:

Bisimulation scheme

$P, Q \in Prc$ are equivalent iff, for every action α, every α-successor of P is equivalent to some α-successor of Q, and vice versa.

Rationale

Observation

In order for a behavioural equivalence to be deadlock sensitive, it has to take the branching structure of processes into account.

This is achieved by an equivalence that is defined according to the following scheme:

Bisimulation scheme

$P, Q \in Prc$ are equivalent iff, for every action α, every α-successor of P is equivalent to some α-successor of Q, and vice versa.

Two variants will be considered in this course:

(1) Strong bisimulation: ignore the special role of τ-actions

(2) Weak bisimulation: treat τ-actions as invisible

Strong Bisimulation I

Definition 2.14 (Strong bisimulation) (Park 1981, Milner 1989)

A binary relation $\rho \subseteq \mathit{Prc} \times \mathit{Prc}$ is a strong bisimulation whenever for every $(P, Q) \in \rho$ and $\alpha \in \mathit{Act}$:

(1) if $P \xrightarrow{\alpha} P'$, then there exists $Q' \in \mathit{Prc}$ such that $Q \xrightarrow{\alpha} Q'$ and $P' \,\rho\, Q'$, and

(2) if $Q \xrightarrow{\alpha} Q'$, then there exists $P' \in \mathit{Prc}$ such that $P \xrightarrow{\alpha} P'$ and $P' \,\rho\, Q'$.

Strong Bisimulation I

Definition 2.14 (Strong bisimulation) (Park 1981, Milner 1989)

A binary relation $\rho \subseteq \mathit{Prc} \times \mathit{Prc}$ is a strong bisimulation whenever for every $(P, Q) \in \rho$ and $\alpha \in \mathit{Act}$:

(1) if $P \xrightarrow{\alpha} P'$, then there exists $Q' \in \mathit{Prc}$ such that $Q \xrightarrow{\alpha} Q'$ and $P' \,\rho\, Q'$, and

(2) if $Q \xrightarrow{\alpha} Q'$, then there exists $P' \in \mathit{Prc}$ such that $P \xrightarrow{\alpha} P'$ and $P' \,\rho\, Q'$.

Note: strong bisimulations are not necessarily equivalences (e.g., $\rho = \emptyset$).

Strong Bisimulation I

Definition 2.14 (Strong bisimulation) (Park 1981, Milner 1989)

A binary relation $\rho \subseteq \mathit{Prc} \times \mathit{Prc}$ is a strong bisimulation whenever for every $(P, Q) \in \rho$ and $\alpha \in \mathit{Act}$:

(1) if $P \xrightarrow{\alpha} P'$, then there exists $Q' \in \mathit{Prc}$ such that $Q \xrightarrow{\alpha} Q'$ and $P' \,\rho\, Q'$, and

(2) if $Q \xrightarrow{\alpha} Q'$, then there exists $P' \in \mathit{Prc}$ such that $P \xrightarrow{\alpha} P'$ and $P' \,\rho\, Q'$.

Note: strong bisimulations are not necessarily equivalences (e.g., $\rho = \emptyset$).

Definition 2.15 (Strong bisimilarity)

Processes $P, Q \in \mathit{Prc}$ are strongly bisimilar, denoted $P \sim Q$, iff there is a strong bisimulation ρ with $P \,\rho\, Q$.

Strong Bisimulation I

Definition 2.14 (Strong bisimulation) (Park 1981, Milner 1989)

A binary relation $\rho \subseteq \mathit{Prc} \times \mathit{Prc}$ is a strong bisimulation whenever for every $(P, Q) \in \rho$ and $\alpha \in \mathit{Act}$:

(1) if $P \xrightarrow{\alpha} P'$, then there exists $Q' \in \mathit{Prc}$ such that $Q \xrightarrow{\alpha} Q'$ and $P' \,\rho\, Q'$, and

(2) if $Q \xrightarrow{\alpha} Q'$, then there exists $P' \in \mathit{Prc}$ such that $P \xrightarrow{\alpha} P'$ and $P' \,\rho\, Q'$.

Note: strong bisimulations are not necessarily equivalences (e.g., $\rho = \emptyset$).

Definition 2.15 (Strong bisimilarity)

Processes $P, Q \in \mathit{Prc}$ are strongly bisimilar, denoted $P \sim Q$, iff there is a strong bisimulation ρ with $P \,\rho\, Q$. Thus,

$$\sim \;=\; \bigcup \{\rho \subseteq \mathit{Prc} \times \mathit{Prc} \mid \rho \text{ is a strong bisimulation}\}.$$

Strong Bisimulation I

Definition 2.14 (Strong bisimulation) (Park 1981, Milner 1989)

A binary relation $\rho \subseteq \mathit{Prc} \times \mathit{Prc}$ is a strong bisimulation whenever for every $(P, Q) \in \rho$ and $\alpha \in \mathit{Act}$:

(1) if $P \xrightarrow{\alpha} P'$, then there exists $Q' \in \mathit{Prc}$ such that $Q \xrightarrow{\alpha} Q'$ and $P' \,\rho\, Q'$, and

(2) if $Q \xrightarrow{\alpha} Q'$, then there exists $P' \in \mathit{Prc}$ such that $P \xrightarrow{\alpha} P'$ and $P' \,\rho\, Q'$.

Note: strong bisimulations are not necessarily equivalences (e.g., $\rho = \emptyset$).

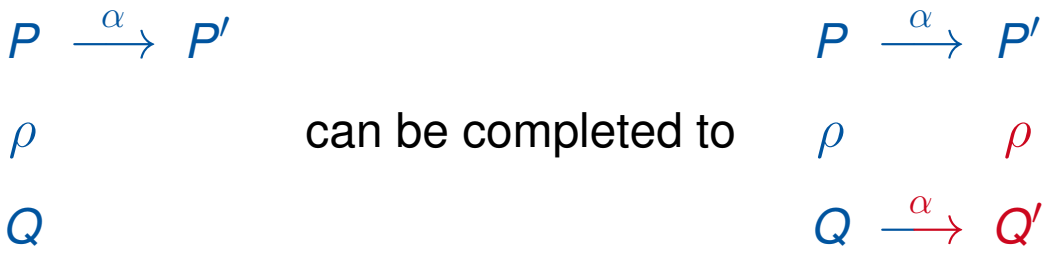
Definition 2.15 (Strong bisimilarity)

Processes $P, Q \in \mathit{Prc}$ are strongly bisimilar, denoted $P \sim Q$, iff there is a strong bisimulation ρ with $P \,\rho\, Q$. Thus,

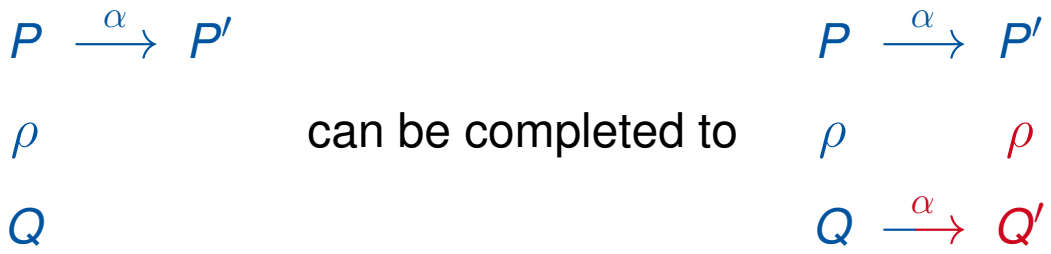
$$\sim \;=\; \bigcup \{\rho \subseteq \mathit{Prc} \times \mathit{Prc} \mid \rho \text{ is a strong bisimulation}\}.$$

Relation $\sim$ is called strong bisimilarity.

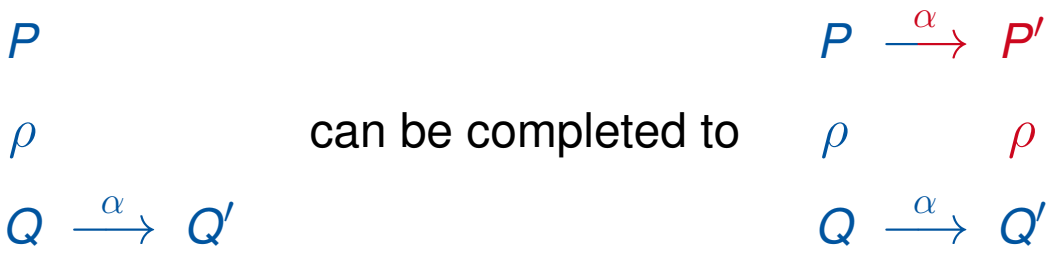
Strong Bisimulation II



Strong Bisimulation II



and



Examples

Definition (Strong bisimulation) (Park 1981, Milner 1989)

A binary relation $\rho \subseteq \mathit{Prc} \times \mathit{Prc}$ is a strong bisimulation whenever for every $(P, Q) \in \rho$ and $\alpha \in \mathit{Act}$:

(1) if $P \xrightarrow{\alpha} P'$, then there exists $Q' \in \mathit{Prc}$ such that $Q \xrightarrow{\alpha} Q'$ and $P' \,\rho\, Q'$, and

(2) if $Q \xrightarrow{\alpha} Q'$, then there exists $P' \in \mathit{Prc}$ such that $P \xrightarrow{\alpha} P'$ and $P' \,\rho\, Q'$.

Example 2.16 (A first example)

Claim: $P \sim Q$ where $P = a.P_1 + a.P_2$ $\quad Q = a.Q_1$

$P_1 = b.P_2$ $\quad Q_1 = b.Q_1$

$P_2 = b.P_2$

Examples

Definition (Strong bisimulation) (Park 1981, Milner 1989)

A binary relation $\rho \subseteq \mathit{Prc} \times \mathit{Prc}$ is a strong bisimulation whenever for every $(P, Q) \in \rho$ and $\alpha \in \mathit{Act}$:

(1) if $P \xrightarrow{\alpha} P'$, then there exists $Q' \in \mathit{Prc}$ such that $Q \xrightarrow{\alpha} Q'$ and $P' \,\rho\, Q'$, and

(2) if $Q \xrightarrow{\alpha} Q'$, then there exists $P' \in \mathit{Prc}$ such that $P \xrightarrow{\alpha} P'$ and $P' \,\rho\, Q'$.

Example 2.16 (A first example)

Claim: $P \sim Q$ where $P = a.P_1 + a.P_2$, $P_1 = b.P_2$, $P_2 = b.P_2$, $Q = a.Q_1$, $Q_1 = b.Q_1$

Proof: $\rho = \{(P, Q), (P_1, Q_1), (P_2, Q_1)\}$ is a strong bisimulation

Examples

Definition (Strong bisimulation) (Park 1981, Milner 1989)

A binary relation $\rho \subseteq Prc \times Prc$ is a strong bisimulation whenever for every $(P, Q) \in \rho$ and $\alpha \in Act$:

(1) if $P \xrightarrow{\alpha} P'$, then there exists $Q' \in Prc$ such that $Q \xrightarrow{\alpha} Q'$ and $P' \,\rho\, Q'$, and

(2) if $Q \xrightarrow{\alpha} Q'$, then there exists $P' \in Prc$ such that $P \xrightarrow{\alpha} P'$ and $P' \,\rho\, Q'$.

Example 2.16 (A first example)

Claim: $P \sim Q$ where $P = a.P_1 + a.P_2$, $P_1 = b.P_2$, $P_2 = b.P_2$ and $Q = a.Q_1$, $Q_1 = b.Q_1$

Proof: $\rho = \{(P, Q), (P_1, Q_1), (P_2, Q_1)\}$ is a strong bisimulation

Example 2.17 (Relating a finite to an infinite-state process)

Claim: $P_0 \sim Q$ where $P_i = a.P_{i+1}$ for $i \in \mathbb{N}$ and $Q = a.Q$.

Examples

Definition (Strong bisimulation) (Park 1981, Milner 1989)

A binary relation $\rho \subseteq Prc \times Prc$ is a strong bisimulation whenever for every $(P, Q) \in \rho$ and $\alpha \in Act$:

(1) if $P \xrightarrow{\alpha} P'$, then there exists $Q' \in Prc$ such that $Q \xrightarrow{\alpha} Q'$ and $P' \,\rho\, Q'$, and

(2) if $Q \xrightarrow{\alpha} Q'$, then there exists $P' \in Prc$ such that $P \xrightarrow{\alpha} P'$ and $P' \,\rho\, Q'$.

Example 2.16 (A first example)

Claim: $P \sim Q$ where $P = a.P_1 + a.P_2$, $P_1 = b.P_2$, $P_2 = b.P_2$, $Q = a.Q_1$, $Q_1 = b.Q_1$

Proof: $\rho = \{(P, Q), (P_1, Q_1), (P_2, Q_1)\}$ is a strong bisimulation

Example 2.17 (Relating a finite to an infinite-state process)

Claim: $P_0 \sim Q$ where $P_i = a.P_{i+1}$ for $i \in \mathbb{N}$ and $Q = a.Q$.

Proof: $\rho = \{(P_i, Q) \mid i \in \mathbb{N}\}$ is a strong bisimulation.

A Counterexample

Example 2.18 (Vending machines; cf. Example 2.12)

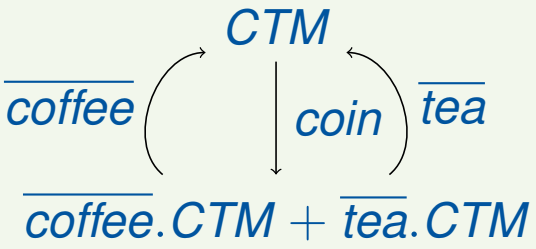
Show $CTM \not\sim CTM'$ where $\begin{aligned} CTM &= coin.\left(\overline{coffee}.CTM + \overline{tea}.CTM\right) \\ CTM' &= coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM'. \end{aligned}$

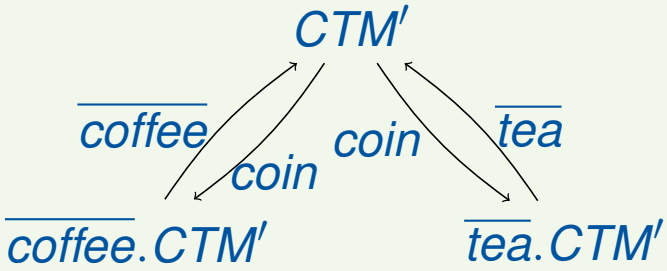
A Counterexample

Example 2.18 (Vending machines; cf. Example 2.12)

Show $CTM \not\sim CTM'$ where $CTM = coin.\left(\overline{coffee}.CTM + \overline{tea}.CTM\right)$
$CTM' = coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM'.$

Corresponding LTSs:



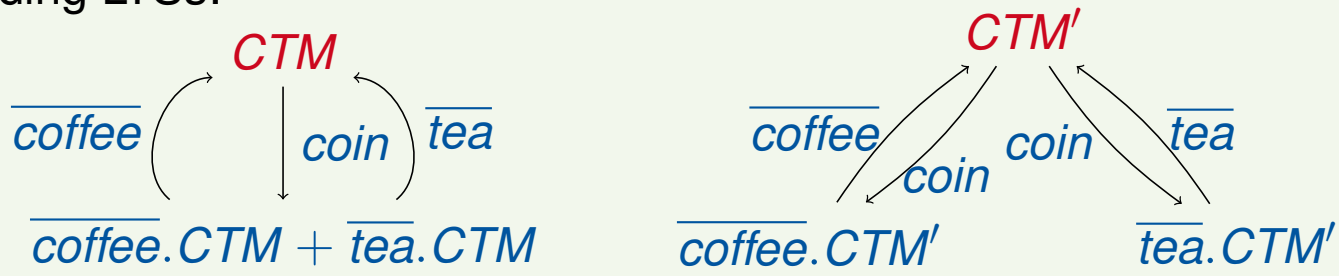


A Counterexample

Example 2.18 (Vending machines; cf. Example 2.12)

Show $CTM \not\sim CTM'$ where $CTM = coin.\left(\overline{coffee}.CTM + \overline{tea}.CTM\right)$
$CTM' = coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM'.$

Corresponding LTSs:



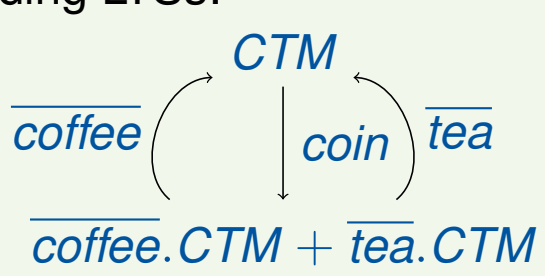
Assumption: there exists bisimulation ρ such that $CTM\ \rho\ CTM'$.

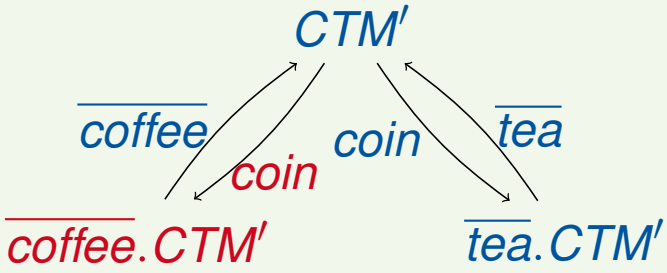
A Counterexample

Example 2.18 (Vending machines; cf. Example 2.12)

Show $CTM \not\sim CTM'$ where $CTM = coin.\left(\overline{coffee}.CTM + \overline{tea}.CTM\right)$
$CTM' = coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM'.$

Corresponding LTSs:





Assumption: there exists bisimulation ρ such that $CTM \; \rho \; CTM'$.

- First CTM' chooses the left *coin*-transition.

A Counterexample

Example 2.18 (Vending machines; cf. Example 2.12)

Show $CTM \not\sim CTM'$ where $CTM = coin.\left(\overline{coffee}.CTM + \overline{tea}.CTM\right)$
$CTM' = coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM'.$

Corresponding LTSs:

Assumption: there exists bisimulation ρ such that $CTM \; \rho \; CTM'$.

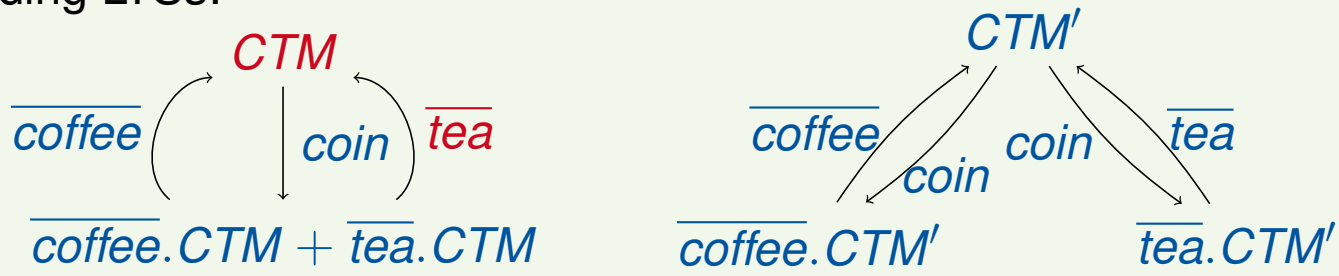
- First CTM' chooses the left *coin*-transition.
- The only possible reaction by CTM is its *coin*-transition; thus $(\overline{coffee}.CTM + \overline{tea}.CTM) \; \rho \; \overline{coffee}.CTM'$ must hold.

A Counterexample

Example 2.18 (Vending machines; cf. Example 2.12)

Show $CTM \not\sim CTM'$ where $CTM = coin.\left(\overline{coffee}.CTM + \overline{tea}.CTM\right)$
$CTM' = coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM'.$

Corresponding LTSs:



Assumption: there exists bisimulation ρ such that $CTM \; \rho \; CTM'$.

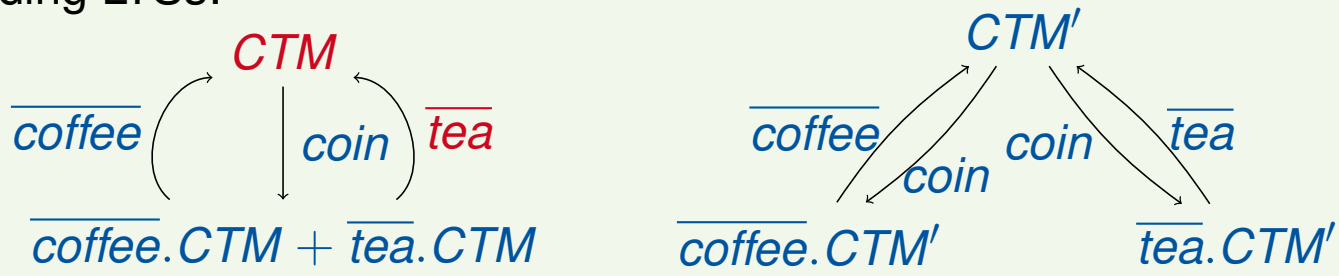
- First CTM' chooses the left *coin*-transition.
- The only possible reaction by CTM is its *coin*-transition; thus $(\overline{coffee}.CTM + \overline{tea}.CTM) \; \rho \; \overline{coffee}.CTM'$ must hold.
- CTM proceeds by selecting the $\overline{tea}$-transition.

A Counterexample

Example 2.18 (Vending machines; cf. Example 2.12)

Show $CTM \not\sim CTM'$ where $CTM = coin.\left(\overline{coffee}.CTM + \overline{tea}.CTM\right)$
$CTM' = coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM'.$

Corresponding LTSs:



Assumption: there exists bisimulation ρ such that $CTM\ \rho\ CTM'$.

- First CTM' chooses the left *coin*-transition.
- The only possible reaction by CTM is its *coin*-transition; thus $(\overline{coffee}.CTM + \overline{tea}.CTM)\ \rho\ \overline{coffee}.CTM'$ must hold.
- CTM proceeds by selecting the $\overline{tea}$-transition.
- But CTM' cannot react to this step. ↯

Outline of Lecture 2

Equivalence property of Strong Bisimilarity

Lemma 2.19 (Equivalence property of $\sim$)

$\sim$ *is an equivalence relation (i.e., reflexive, symmetric, and transitive).*

Proof.

omitted □

Bisimulation on Paths

Lemma 2.20 (Bisimulation on paths)

Whenever we have:

$$P_0 \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} P_2 \xrightarrow{\alpha_3} P_3 \xrightarrow{\alpha_4} P_4 \ldots\ldots$$

$$\rho$$

$$Q_0$$

Bisimulation on Paths

Lemma 2.20 (Bisimulation on paths)

Whenever we have:

$$P_0 \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} P_2 \xrightarrow{\alpha_3} P_3 \xrightarrow{\alpha_4} P_4 \ldots\ldots$$

$$\rho$$

$$Q_0$$

this can be completed to

$$\begin{array}{ccccccccccc} P_0 & \xrightarrow{\alpha_1} & P_1 & \xrightarrow{\alpha_2} & P_2 & \xrightarrow{\alpha_3} & P_3 & \xrightarrow{\alpha_4} & P_4 \ldots\ldots \\ \rho & & \rho & & \rho & & \rho & & \rho & & \\ Q_0 & \xrightarrow{\alpha_1} & Q_1 & \xrightarrow{\alpha_2} & Q_2 & \xrightarrow{\alpha_3} & Q_3 & \xrightarrow{\alpha_4} & Q_4 \ldots\ldots \end{array}$$

Bisimulation on Paths

Lemma 2.20 (Bisimulation on paths)

Whenever we have:

$$\begin{array}{ccccccccccc} P_0 & \xrightarrow{\alpha_1} & P_1 & \xrightarrow{\alpha_2} & P_2 & \xrightarrow{\alpha_3} & P_3 & \xrightarrow{\alpha_4} & P_4 & \ldots\ldots \\ \rho & & & & & & & & & \\ Q_0 & & & & & & & & & \end{array}$$

this can be completed to

$$\begin{array}{ccccccccccc} P_0 & \xrightarrow{\alpha_1} & P_1 & \xrightarrow{\alpha_2} & P_2 & \xrightarrow{\alpha_3} & P_3 & \xrightarrow{\alpha_4} & P_4 & \ldots\ldots \\ \rho & & \rho & & \rho & & \rho & & \rho & \\ Q_0 & \xrightarrow{\alpha_1} & Q_1 & \xrightarrow{\alpha_2} & Q_2 & \xrightarrow{\alpha_3} & Q_3 & \xrightarrow{\alpha_4} & Q_4 & \ldots\ldots \end{array}$$

Proof.

by induction on the length of the path □

Strong Bisimulation vs. Trace Equivalence

Theorem 2.21

$P \sim Q$ implies that P and Q are trace equivalent. The reverse does generally not hold.

Strong Bisimulation vs. Trace Equivalence

Theorem 2.21

$P \sim Q$ implies that P and Q are trace equivalent. The reverse does generally not hold.

Proof.

The implication from left to right follows from Lemma 2.20.

Strong Bisimulation vs. Trace Equivalence

Theorem 2.21

$P \sim Q$ implies that P and Q are trace equivalent. The reverse does generally not hold.

Proof.

The implication from left to right follows from Lemma 2.20.

Consider the other direction:

- Take $P = a.P_1$ with $P_1 = b.\text{nil} + c.\text{nil}$ and $Q = a.b.nil + a.c.\text{nil}$.

Strong Bisimulation vs. Trace Equivalence

Theorem 2.21

$P \sim Q$ implies that P and Q are trace equivalent. The reverse does generally not hold.

Proof.

The implication from left to right follows from Lemma 2.20.

Consider the other direction:

- Take $P = a.P_1$ with $P_1 = b.\text{nil} + c.\text{nil}$ and $Q = a.b.nil + a.c.\text{nil}$.
- Then: $Tr(P) = \{\varepsilon, a, ab, ac\} = Tr(Q)$.
- Thus, P and Q are trace equivalent.

Strong Bisimulation vs. Trace Equivalence

Theorem 2.21

$P \sim Q$ implies that P and Q are trace equivalent. The reverse does generally not hold.

Proof.

The implication from left to right follows from Lemma 2.20.

Consider the other direction:

- Take $P = a.P_1$ with $P_1 = b.\text{nil} + c.\text{nil}$ and $Q = a.b.nil + a.c.\text{nil}$.
- Then: $Tr(P) = \{\varepsilon, a, ab, ac\} = Tr(Q)$.
- Thus, P and Q are trace equivalent.
- But: $P \not\sim Q$, as there is no state in the LTS of Q that is bisimilar to P_1 (cf. Example 2.18).
- Why? Since no state in Q can perform both b and c.

□

Deterministic Transition Systems

Definition 2.22 (Determinism)

$P \in Prc$ is deterministic whenever for every of its reachable states R it holds:

$$\left(R \xrightarrow{\alpha} R' \text{ and } R \xrightarrow{\alpha} R''\right) \quad \text{implies} \quad R' = R''.$$

Deterministic Transition Systems

Definition 2.22 (Determinism)

$P \in Prc$ is deterministic whenever for every of its reachable states R it holds:

$$\left(R \xrightarrow{\alpha} R' \text{ and } R \xrightarrow{\alpha} R''\right) \quad \text{implies} \quad R' = R''.$$

Theorem 2.23 (Determinism implies coincidence of $\sim$ and trace equivalence) (Park)

For deterministic P and Q: $P \sim Q$ iff $Tr(P) = Tr(Q)$.

Proof.

omitted □

Bisimulation is a Congruence

Theorem 2.24 (CCS congruence property of $\sim$)

Strong bisimilarity $\sim$ is a CCS congruence, that is, whenever $P, Q \in Prc$ such that $P \sim Q$,

$$
\begin{array}{rcll}
\alpha.P & \sim & \alpha.Q & \text{for every action } \alpha \\
P + R & \sim & Q + R & \text{for every process } R \\
P \parallel R & \sim & Q \parallel R & \text{for every process } R \\
P \setminus L & \sim & Q \setminus L & \text{for every set } L \subseteq A \\
P[f] & \sim & Q[f] & \text{for every relabelling } f : A \to A
\end{array}
$$

Proof.

omitted □

Deadlock Sensitivity of Strong Bisimulation

Definition (Deadlock; cf. Definition 2.6)

Let $P, Q \in Prc$ and $w \in Act^*$ such that $P \xrightarrow{w} Q$ and $Q \not\longrightarrow$. Then Q is called a w-deadlock of P.

Definition (Deadlock sensitivity; cf. Definition 2.8)

Relation $\equiv \subseteq Prc \times Prc$ is deadlock sensitive whenever:

$$P \equiv Q \text{ implies } \left(\forall w \in Act^* : P \text{ has a } w\text{-deadlock iff } Q \text{ has a } w\text{-deadlock}\right).$$

Deadlock Sensitivity of Strong Bisimulation

Definition (Deadlock sensitivity; cf. Definition 2.8)

Relation $\equiv \subseteq \mathit{Prc} \times \mathit{Prc}$ is deadlock sensitive whenever:

$$P \equiv Q \text{ implies } \left(\forall w \in \mathit{Act}^* : P \text{ has a } w\text{-deadlock iff } Q \text{ has a } w\text{-deadlock}\right).$$

Theorem 2.25

$\sim$ *is deadlock sensitive.*

Deadlock Sensitivity of Strong Bisimulation

Definition (Deadlock sensitivity; cf. Definition 2.8)

Relation $\equiv \subseteq \mathit{Prc} \times \mathit{Prc}$ is deadlock sensitive whenever:

$$P \equiv Q \text{ implies } \left(\forall w \in \mathit{Act}^* : P \text{ has a } w\text{-deadlock iff } Q \text{ has a } w\text{-deadlock}\right).$$

Theorem 2.25

$\sim$ *is deadlock sensitive.*

Proof.

Let $P \sim Q$.

- We assume that, for some $w \in \mathit{Act}^*$, P has a w-deadlock but Q does not.

Deadlock Sensitivity of Strong Bisimulation

Definition (Deadlock sensitivity; cf. Definition 2.8)

Relation $\equiv \subseteq \mathit{Prc} \times \mathit{Prc}$ is deadlock sensitive whenever:

$$P \equiv Q \text{ implies } \left(\forall w \in \mathit{Act}^* : P \text{ has a } w\text{-deadlock iff } Q \text{ has a } w\text{-deadlock}\right).$$

Theorem 2.25

$\sim$ is deadlock sensitive.

Proof.

Let $P \sim Q$.

- We assume that, for some $w \in \mathit{Act}^*$, P has a w-deadlock but Q does not.
- Thus, there exists $P' \in \mathit{Prc}$ such that $P \xrightarrow{w} P'$ and $P' \not\longrightarrow$.
- Moreover, for all $Q' \in \mathit{Prc}$ with $Q \xrightarrow{w} Q'$ there exist $\alpha \in \mathit{Act}$ and $Q'' \in \mathit{Prc}$ such that $Q' \xrightarrow{\alpha} Q''$.

Deadlock Sensitivity of Strong Bisimulation

Definition (Deadlock sensitivity; cf. Definition 2.8)

Relation $\equiv \subseteq \mathit{Prc} \times \mathit{Prc}$ is deadlock sensitive whenever:

$$P \equiv Q \text{ implies } \left(\forall w \in \mathit{Act}^* : P \text{ has a } w\text{-deadlock iff } Q \text{ has a } w\text{-deadlock}\right).$$

Theorem 2.25

$\sim$ *is deadlock sensitive.*

Proof.

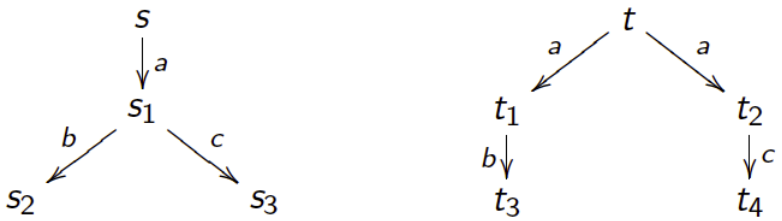
Let $P \sim Q$.

- We assume that, for some $w \in \mathit{Act}^*$, P has a w-deadlock but Q does not.
- Thus, there exists $P' \in \mathit{Prc}$ such that $P \xrightarrow{w} P'$ and $P' \not\longrightarrow$.
- Moreover, for all $Q' \in \mathit{Prc}$ with $Q \xrightarrow{w} Q'$ there exist $\alpha \in \mathit{Act}$ and $Q'' \in \mathit{Prc}$ such that $Q' \xrightarrow{\alpha} Q''$.
- For $P \xrightarrow{w} P'$, Lemma 2.20 (bisimulation on paths) yields Q' with $Q \xrightarrow{w} Q'$ and $P' \sim Q'$.
- Thus $P' \not\longrightarrow$ and $Q' \xrightarrow{\alpha} Q''$ cannot hold at the same time. ↯

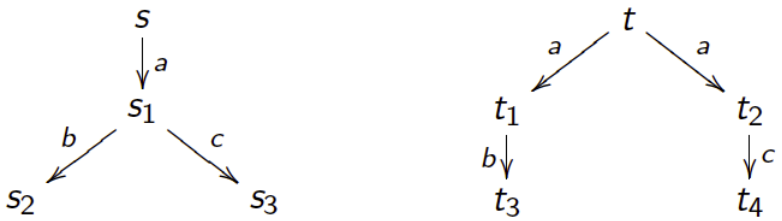
□

Outline of Lecture 2

How to Show Non-Bisimilarity?



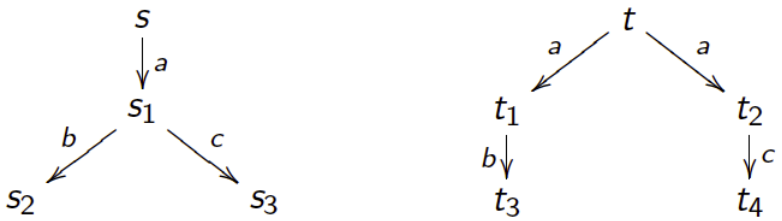
How to Show Non-Bisimilarity?



Alternatives to prove that $s \not\sim t$

- Enumerate all binary relations and show that none of those containing (s, t) is a strong bisimulation.

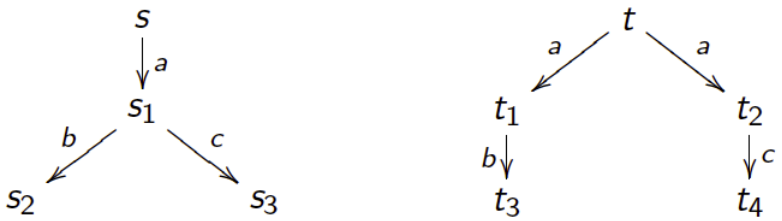
How to Show Non-Bisimilarity?



Alternatives to prove that $s \not\sim t$

- Enumerate all binary relations and show that none of those containing (s, t) is a strong bisimulation.
  - This is expensive, as there are 2^{k^2} binary relations on a set S with $|S| = k$.

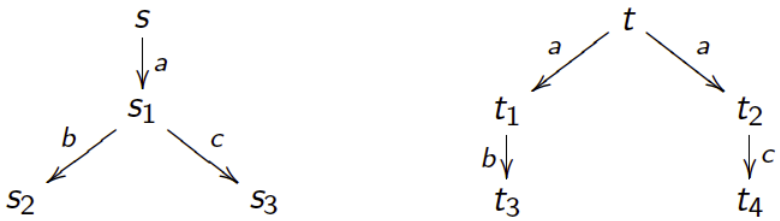
How to Show Non-Bisimilarity?



Alternatives to prove that $s \not\sim t$

- Enumerate all binary relations and show that none of those containing (s, t) is a strong bisimulation.
  - This is expensive, as there are 2^{k^2} binary relations on a set S with $|S| = k$.
- Make certain observations which will enable to disqualify many bisimulation candidates in one step.

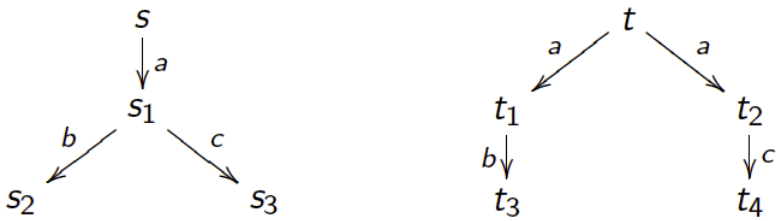
How to Show Non-Bisimilarity?



Alternatives to prove that $s \not\sim t$

- Enumerate all binary relations and show that none of those containing (s, t) is a strong bisimulation.
  - This is expensive, as there are 2^{k^2} binary relations on a set S with $|S| = k$.
- Make certain observations which will enable to disqualify many bisimulation candidates in one step.
  - Yields heuristics – how about completeness?

How to Show Non-Bisimilarity?



Alternatives to prove that $s \not\sim t$

- Enumerate all binary relations and show that none of those containing (s, t) is a strong bisimulation.
  - This is expensive, as there are 2^{k^2} binary relations on a set S with $|S| = k$.
- Make certain observations which will enable to disqualify many bisimulation candidates in one step.
  - Yields heuristics – how about completeness?
- Use game characterisation of strong bisimilarity.

The Strong Bisimulation Game

Let $(S, Act, \longrightarrow)$ be an LTS and $s, t \in S$. Question: does $s \sim t$ hold?

The Strong Bisimulation Game

Let $(S, Act, \longrightarrow)$ be an LTS and $s, t \in S$. Question: does $s \sim t$ hold?

We define a game with two players: an “attacker” and a “defender”.

The Strong Bisimulation Game

Let $(S, Act, \longrightarrow)$ be an LTS and $s, t \in S$. Question: does $s \sim t$ hold?

We define a game with two players: an "attacker" and a "defender".

- The game is played in rounds, and configurations of the game are pairs of states from $S \times S$.

The Strong Bisimulation Game

Let $(S, Act, \longrightarrow)$ be an LTS and $s, t \in S$. Question: does $s \sim t$ hold?

We define a game with two players: an "attacker" and a "defender".

- The game is played in rounds, and configurations of the game are pairs of states from $S \times S$.
- In each round, the game is in a current configuration.

The Strong Bisimulation Game

Let $(S, Act, \longrightarrow)$ be an LTS and $s, t \in S$. Question: does $s \sim t$ hold?

We define a game with two players: an "attacker" and a "defender".

- The game is played in rounds, and configurations of the game are pairs of states from $S \times S$.
- In each round, the game is in a current configuration.
- Initially, the configuration (s, t) is chosen as the current one.

The Strong Bisimulation Game

Let $(S, Act, \longrightarrow)$ be an LTS and $s, t \in S$. Question: does $s \sim t$ hold?

We define a game with two players: an "attacker" and a "defender".

- The game is played in rounds, and configurations of the game are pairs of states from $S \times S$.
- In each round, the game is in a current configuration.
- Initially, the configuration (s, t) is chosen as the current one.

Intuition

The defender wants to show that $s \sim t$ while the attacker aims to prove the opposite.

Rules of the Bisimulation Game

Rules

In each round, the current configuration (s, t) is changed as follows:

(1) the attacker chooses one of the two processes in the current configuration, say t, and makes an $\xrightarrow{\alpha}$-move for some $\alpha \in \mathit{Act}$ to t', say,

Rules of the Bisimulation Game

Rules

In each round, the current configuration (s, t) is changed as follows:

(1) the attacker chooses one of the two processes in the current configuration, say t, and makes an $\xrightarrow{\alpha}$-move for some $\alpha \in \mathit{Act}$ to t', say, and

(2) the defender must respond by making an $\xrightarrow{\alpha}$-move in the other process s of the current configuration under the same action α, yielding $s \xrightarrow{\alpha} s'$.

Rules of the Bisimulation Game

Rules

In each round, the current configuration (s, t) is changed as follows:

(1) the attacker chooses one of the two processes in the current configuration, say t, and makes an $\xrightarrow{\alpha}$-move for some $\alpha \in \mathit{Act}$ to t', say, and

(2) the defender must respond by making an $\xrightarrow{\alpha}$-move in the other process s of the current configuration under the same action α, yielding $s \xrightarrow{\alpha} s'$.

The pair of processes (s', t') becomes the new current configuration.

The game continues with another round.

Rules of the Bisimulation Game

Rules

In each round, the current configuration (s, t) is changed as follows:

(1) the attacker chooses one of the two processes in the current configuration, say t, and makes an $\xrightarrow{\alpha}$-move for some $\alpha \in \mathit{Act}$ to t', say, and

(2) the defender must respond by making an $\xrightarrow{\alpha}$-move in the other process s of the current configuration under the same action α, yielding $s \xrightarrow{\alpha} s'$.

The pair of processes (s', t') becomes the new current configuration.
The game continues with another round.

Results

(1) If one player cannot move, the other player wins:
- attacker cannot move if $s \not\to$ and $t \not\to$
- defender cannot move if no matching transition available

Rules of the Bisimulation Game

Rules

In each round, the current configuration (s, t) is changed as follows:

(1) the attacker chooses one of the two processes in the current configuration, say t, and makes an $\xrightarrow{\alpha}$-move for some $\alpha \in \mathit{Act}$ to t', say, and

(2) the defender must respond by making an $\xrightarrow{\alpha}$-move in the other process s of the current configuration under the same action α, yielding $s \xrightarrow{\alpha} s'$.

The pair of processes (s', t') becomes the new current configuration.
The game continues with another round.

Results

(1) If one player cannot move, the other player wins:
- attacker cannot move if $s \not\to$ and $t \not\to$
- defender cannot move if no matching transition available

(2) If the game is played *ad infinitum*, the defender wins.

Examples

Example 2.26 (Bisimulation games)

(1) Use the CAAL games feature to show $P \sim Q$ where

$$\begin{aligned} P &= a.P_1 + a.P_2 & \qquad Q &= a.Q_1 \\ P_1 &= b.P_2 & \qquad Q_1 &= b.Q_1 \\ P_2 &= b.P_2 \end{aligned}$$

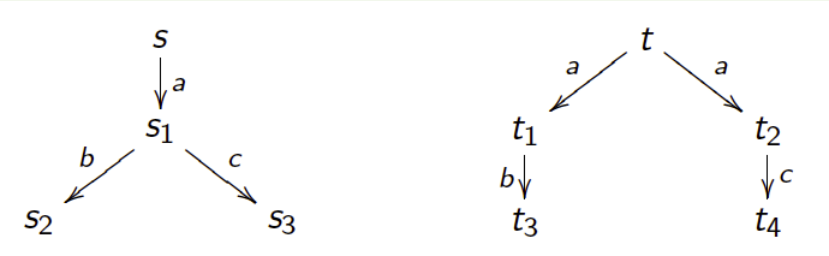
Examples

Example 2.26 (Bisimulation games)

(1) Use the CAAL games feature to show $P \sim Q$ where

$$\begin{aligned} P &= a.P_1 + a.P_2 & \qquad Q &= a.Q_1 \\ P_1 &= b.P_2 & \qquad Q_1 &= b.Q_1 \\ P_2 &= b.P_2 \end{aligned}$$

(2) Use the CAAL games feature to show that $s \not\sim t$ where:



Two winning strategies for attacker in configuration (s, t):

- start with $s \xrightarrow{a} s_1$
- start with $t \xrightarrow{a} t_1$

Game Characterisation of Bisimulation

Theorem 2.27 (Game characterisation of bisimulation) (Stirling 1995, Thomas 1993)

(1) $s \sim t$ *iff the defender has a universal winning strategy from configuration* (s, t).

(2) $s \not\sim t$ *iff the attacker has a universal winning strategy from configuration* (s, t).

(By means of a universal winning strategy, a player can always win, regardless of how the other player selects their moves.)

Game Characterisation of Bisimulation

Theorem 2.27 (Game characterisation of bisimulation) (Stirling 1995, Thomas 1993)

(1) $s \sim t$ *iff the defender has a universal winning strategy from configuration* (s, t).

(2) $s \not\sim t$ *iff the attacker has a universal winning strategy from configuration* (s, t).

(By means of a universal winning strategy, a player can always win, regardless of how the other player selects their moves.)

Proof.

by relating winning strategy of defender/attacker to existence/non-existence of strong bisimulation relation □

Game Characterisation of Bisimulation

Theorem 2.27 (Game characterisation of bisimulation) (Stirling 1995, Thomas 1993)

(1) $s \sim t$ *iff the defender has a universal winning strategy from configuration* (s, t).

(2) $s \not\sim t$ *iff the attacker has a universal winning strategy from configuration* (s, t).

(By means of a universal winning strategy, a player can always win, regardless of how the other player selects their moves.)

Proof.

by relating winning strategy of defender/attacker to existence/non-existence of strong bisimulation relation □

Thus, a bisimulation game can be used to prove bisimilarity as well as non-bisimilarity. It often provides elegant arguments for $s \not\sim t$.

Summary: Strong Bisimulation

- Strong bisimulation of processes is based on mutually mimicking each other.
- Strong bisimilarity $\sim$:
  (1) is an equivalence relation,
  (2) is strictly coarser than LTS isomorphism,
  (3) is strictly finer than trace equivalence,
  (4) is a CCS congruence,
  (5) is deadlock sensitive, and
  (6) can be checked using a two-player game.

Outline of Lecture 2

Inadequacy of Strong Bisimilarity

Example 2.28 (Two-place buffers; cf. Example 1.6)

(1) Sequential two-place buffer:

$$\begin{aligned} B_0 &= in.B_1 \\ B_1 &= \overline{out}.B_0 + in.B_2 \\ B_2 &= \overline{out}.B_1 \end{aligned}$$

(2) Parallel two-place buffer:

$$\begin{aligned} B_{\|} &= (B[f] \parallel B[g]) \setminus com \\ B &= in.\overline{out}.B \end{aligned}$$

$(f := [out \mapsto com], g := [in \mapsto com])$

Inadequacy of Strong Bisimilarity

Example 2.28 (Two-place buffers; cf. Example 1.6)

(1) Sequential two-place buffer:

$$B_0 = in.B_1$$
$$B_1 = \overline{out}.B_0 + in.B_2$$
$$B_2 = \overline{out}.B_1$$

(2) Parallel two-place buffer:

$$B_{\|} = (B[f] \parallel B[g]) \setminus com$$
$$B = in.\overline{out}.B$$

$(f := [out \mapsto com], g := [in \mapsto com])$

Observation:

(verify by playing CAAL game)

Inadequacy of Strong Bisimilarity

Example 2.28 (Two-place buffers; cf. Example 1.6)

(1) Sequential two-place buffer:

$$B_0 = in.B_1$$
$$B_1 = \overline{out}.B_0 + in.B_2$$
$$B_2 = \overline{out}.B_1$$

(2) Parallel two-place buffer:

$$B_{\|} = (B[f] \parallel B[g]) \setminus com$$
$$B = in.\overline{out}.B$$

$(f := [out \mapsto com], g := [in \mapsto com])$

Observation:

(verify by playing CAAL game)

Conclusion

- The requirement in $\sim$ to exactly match all actions is often too strong.
- This suggests to weaken this and not insist on exact matching of τ-actions.

The Rationales for Abstracting from τ-Actions

- τ-actions are internal and thus unobservable.

The Rationales for Abstracting from τ-Actions

- τ-actions are internal and thus unobservable.
- This is natural in parallel communication resulting in τ:
  - synchronization in CCS is binary handshaking
  - observation means communication with the process
  - thus the result of any communication is unobservable

The Rationales for Abstracting from τ-Actions

- τ-actions are internal and thus unobservable.
- This is natural in parallel communication resulting in τ:
  - synchronization in CCS is binary handshaking
  - observation means communication with the process
  - thus the result of any communication is unobservable
- Strong bisimilarity treats τ-actions as any other action.

The Rationales for Abstracting from τ-Actions

- τ-actions are internal and thus unobservable.
- This is natural in parallel communication resulting in τ:
  - synchronization in CCS is binary handshaking
  - observation means communication with the process
  - thus the result of any communication is unobservable
- Strong bisimilarity treats τ-actions as any other action.
- Can we retain the nice properties of $\sim$ while "abstracting" from τ-actions? Yes!

Outline of Lecture 2

Weak Transition Relation

Definition 2.29 (Weak transition relation)

For $\alpha \in \mathit{Act}$, $\overset{\alpha}{\Longrightarrow} \subseteq \mathit{Prc} \times \mathit{Prc}$ is given by

$$\overset{\alpha}{\Longrightarrow} := \begin{cases} \left(\overset{\tau}{\longrightarrow}\right)^* \circ \overset{\alpha}{\longrightarrow} \circ \left(\overset{\tau}{\longrightarrow}\right)^* & \text{if } \alpha \neq \tau \\ \left(\overset{\tau}{\longrightarrow}\right)^* & \text{if } \alpha = \tau. \end{cases}$$

where $\left(\overset{\tau}{\longrightarrow}\right)^*$ denotes the reflexive and transitive closure of relation $\overset{\tau}{\longrightarrow}$.

Informal meaning

- If $\alpha \neq \tau$, then $s \overset{\alpha}{\Longrightarrow} t$ means that from s we can get to t by doing zero or more τ actions, followed by the action α, followed by zero or more τ actions.
- If $\alpha = \tau$, then $s \overset{\alpha}{\Longrightarrow} t$ means that from s we can reach t by doing zero or more τ actions.

Weak Bisimulation I

Definition 2.30 (Weak bisimulation) (Milner 1989)

A binary relation $\rho \subseteq \mathit{Prc} \times \mathit{Prc}$ is a weak bisimulation whenever for every $(P, Q) \in \rho$ and $\alpha \in \mathit{Act}$ (including $\alpha = \tau$):

(1) if $P \xrightarrow{\alpha} P'$, then there exists $Q' \in \mathit{Prc}$ such that $Q \stackrel{\alpha}{\Longrightarrow} Q'$ and $P' \,\rho\, Q'$, and

(2) if $Q \xrightarrow{\alpha} Q'$, then there exists $P' \in \mathit{Prc}$ such that $P \stackrel{\alpha}{\Longrightarrow} P'$ and $P' \,\rho\, Q'$.

Weak Bisimulation I

Definition 2.30 (Weak bisimulation) (Milner 1989)

A binary relation $\rho \subseteq \mathit{Prc} \times \mathit{Prc}$ is a weak bisimulation whenever for every $(P, Q) \in \rho$ and $\alpha \in \mathit{Act}$ (including $\alpha = \tau$):

(1) if $P \xrightarrow{\alpha} P'$, then there exists $Q' \in \mathit{Prc}$ such that $Q \stackrel{\alpha}{\Longrightarrow} Q'$ and $P' \,\rho\, Q'$, and

(2) if $Q \xrightarrow{\alpha} Q'$, then there exists $P' \in \mathit{Prc}$ such that $P \stackrel{\alpha}{\Longrightarrow} P'$ and $P' \,\rho\, Q'$.

Remark

Each clause in the definition of weak bisimulation subsumes two cases:

- $P \xrightarrow{\alpha} P'$ where $\alpha \neq \tau$:
  implies ex. $Q' \in \mathit{Prc}$ such that $Q \,(\xrightarrow{\tau})^* \xrightarrow{\alpha} (\xrightarrow{\tau})^* \, Q'$ and $P' \,\rho\, Q'$
- $P \xrightarrow{\tau} P'$:
  implies ex. $Q' \in \mathit{Prc}$ such that $Q \,(\xrightarrow{\tau})^* \, Q'$ and $P' \,\rho\, Q'$
  (where $Q' = Q$ is admissible)

Weak Bisimulation II

Definition (Weak bisimulation) (Milner 1989)

A binary relation $\rho \subseteq \mathit{Prc} \times \mathit{Prc}$ is a weak bisimulation whenever for every $(P, Q) \in \rho$ and $\alpha \in \mathit{Act}$ (including $\alpha = \tau$):

(1) if $P \xrightarrow{\alpha} P'$, then there exists $Q' \in \mathit{Prc}$ such that $Q \stackrel{\alpha}{\Longrightarrow} Q'$ and $P' \,\rho\, Q'$, and

(2) if $Q \xrightarrow{\alpha} Q'$, then there exists $P' \in \mathit{Prc}$ such that $P \stackrel{\alpha}{\Longrightarrow} P'$ and $P' \,\rho\, Q'$.

Definition 2.31 (Weak bisimilarity)

Processes P and Q are weakly bisimilar, denoted $P \approx Q$, iff there is a weak bisimulation ρ with $P \,\rho\, Q$.

Weak Bisimulation II

Definition (Weak bisimulation) (Milner 1989)

A binary relation $\rho \subseteq \mathit{Prc} \times \mathit{Prc}$ is a weak bisimulation whenever for every $(P, Q) \in \rho$ and $\alpha \in \mathit{Act}$ (including $\alpha = \tau$):

(1) if $P \xrightarrow{\alpha} P'$, then there exists $Q' \in \mathit{Prc}$ such that $Q \stackrel{\alpha}{\Longrightarrow} Q'$ and $P' \,\rho\, Q'$, and

(2) if $Q \xrightarrow{\alpha} Q'$, then there exists $P' \in \mathit{Prc}$ such that $P \stackrel{\alpha}{\Longrightarrow} P'$ and $P' \,\rho\, Q'$.

Definition 2.31 (Weak bisimilarity)

Processes P and Q are weakly bisimilar, denoted $P \approx Q$, iff there is a weak bisimulation ρ with $P \,\rho\, Q$. Thus,

$$\approx = \bigcup \{\rho \subseteq \mathit{Prc} \times \mathit{Prc} \mid \rho \text{ is a weak bisimulation}\}.$$

Weak Bisimulation II

Definition (Weak bisimulation) (Milner 1989)

A binary relation $\rho \subseteq \mathit{Prc} \times \mathit{Prc}$ is a weak bisimulation whenever for every $(P, Q) \in \rho$ and $\alpha \in \mathit{Act}$ (including $\alpha = \tau$):

(1) if $P \xrightarrow{\alpha} P'$, then there exists $Q' \in \mathit{Prc}$ such that $Q \stackrel{\alpha}{\Longrightarrow} Q'$ and $P' \,\rho\, Q'$, and

(2) if $Q \xrightarrow{\alpha} Q'$, then there exists $P' \in \mathit{Prc}$ such that $P \stackrel{\alpha}{\Longrightarrow} P'$ and $P' \,\rho\, Q'$.

Definition 2.31 (Weak bisimilarity)

Processes P and Q are weakly bisimilar, denoted $P \approx Q$, iff there is a weak bisimulation ρ with $P \,\rho\, Q$. Thus,

$$\approx \;=\; \bigcup \{\rho \subseteq \mathit{Prc} \times \mathit{Prc} \mid \rho \text{ is a weak bisimulation}\}.$$

Relation $\approx$ is called weak bisimilarity or observational equivalence.

Examples

Example 2.32

(1) Let $P = \tau.Q$ with $Q = a.\mathrm{nil}$.

- obviously $P \not\sim Q$; claim: $P \approx Q$
- proof: $\rho = \{(P, Q), (Q, Q), (\mathrm{nil}, \mathrm{nil})\}$ is a weak bisimulation with $P \,\rho\, Q$

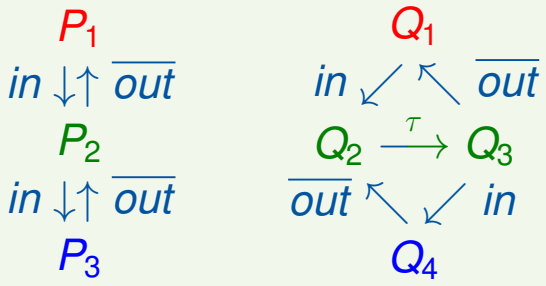
Examples

Example 2.32

(1) Let $P = \tau.Q$ with $Q = a.\mathrm{nil}$.

- obviously $P \not\sim Q$; claim: $P \approx Q$
- proof: $\rho = \{(P, Q), (Q, Q), (\mathrm{nil}, \mathrm{nil})\}$ is a weak bisimulation with $P \,\rho\, Q$

(2) Sequential and parallel two-place buffer are weakly bisimilar:



$$\rho = \{(P_1, Q_1), (P_2, Q_2), (P_2, Q_3), (P_3, Q_4)\}$$

(verify by playing CAAL game)

Outline of Lecture 2

Properties of Weak Bisimilarity

Lemma 2.33 (Properties of $\approx$)

(1) *$P \sim Q$ implies $P \approx Q$.*

(2) *$\approx$ is an equivalence relation (reflexive, symmetric, transitive).*

(3) *$\approx$ is (non-τ) deadlock sensitive.*[1]

Proof.

omitted □

[1] Where w-deadlocks are considered on traces with all τ-actions removed.

Congruence

Lemma 2.34 (Partial CCS congruence property of $\approx$)

Whenever $P, Q \in Prc$ such that $P \approx Q$,

$$
\begin{array}{rcll}
\alpha.P & \approx & \alpha.Q & \text{for every action } \alpha \\
P \parallel R & \approx & Q \parallel R & \text{for every process } R \\
P \setminus L & \approx & Q \setminus L & \text{for every set } L \subseteq A \\
P[f] & \approx & Q[f] & \text{for every relabelling } f : A \to A
\end{array}
$$

Proof.

omitted □

Congruence

Lemma 2.34 (Partial CCS congruence property of $\approx$)

Whenever $P, Q \in Prc$ such that $P \approx Q$,

$$\begin{array}{rcll} \alpha.P & \approx & \alpha.Q & \text{for every action } \alpha \\ P \parallel R & \approx & Q \parallel R & \text{for every process } R \\ P \setminus L & \approx & Q \setminus L & \text{for every set } L \subseteq A \\ P[f] & \approx & Q[f] & \text{for every relabelling } f : A \to A \end{array}$$

Proof.

omitted □

What about non-deterministic choice?

- $\tau.a.\mathrm{nil} \approx a.\mathrm{nil}$ (cf. Ex. 2.32(1)) and $b.\mathrm{nil} \approx b.\mathrm{nil}$ (reflexivity).

Congruence

Lemma 2.34 (Partial CCS congruence property of $\approx$)

Whenever $P, Q \in Prc$ such that $P \approx Q$,

$$\begin{array}{rcll} \alpha.P & \approx & \alpha.Q & \text{for every action } \alpha \\ P \parallel R & \approx & Q \parallel R & \text{for every process } R \\ P \setminus L & \approx & Q \setminus L & \text{for every set } L \subseteq A \\ P[f] & \approx & Q[f] & \text{for every relabelling } f : A \to A \end{array}$$

Proof.

omitted □

What about non-deterministic choice?

- $\tau.a.\mathrm{nil} \approx a.\mathrm{nil}$ (cf. Ex. 2.32(1)) and $b.\mathrm{nil} \approx b.\mathrm{nil}$ (reflexivity).
- But $\tau.a.\mathrm{nil} + b.\mathrm{nil} \not\approx a.\mathrm{nil} + b.\mathrm{nil}$ (why?).

Congruence

Lemma 2.34 (Partial CCS congruence property of $\approx$)

Whenever $P, Q \in Prc$ such that $P \approx Q$,

$$\begin{array}{rcll} \alpha.P & \approx & \alpha.Q & \text{for every action } \alpha \\ P \parallel R & \approx & Q \parallel R & \text{for every process } R \\ P \setminus L & \approx & Q \setminus L & \text{for every set } L \subseteq A \\ P[f] & \approx & Q[f] & \text{for every relabelling } f : A \to A \end{array}$$

Proof.

omitted □

What about non-deterministic choice?

- $\tau.a.\mathrm{nil} \approx a.\mathrm{nil}$ (cf. Ex. 2.32(1)) and $b.\mathrm{nil} \approx b.\mathrm{nil}$ (reflexivity).
- But $\tau.a.\mathrm{nil} + b.\mathrm{nil} \not\approx a.\mathrm{nil} + b.\mathrm{nil}$ (why?).
- Thus, weak bisimilarity is not a CCS congruence, which motivates a slight adaptation of $\approx$.

Outline of Lecture 2

Observation Congruence

Definition 2.35 (Observation congruence) (Milner 1989)

$P, Q \in Prc$ are observationally congruent, denoted $P \approx^c Q$, if for every $\alpha \in Act$ (including $\alpha = \tau$):

(1) if $P \xrightarrow{\alpha} P'$, then there is a sequence of transitions $Q \stackrel{\tau}{\Longrightarrow} \circ \xrightarrow{\alpha} \circ \stackrel{\tau}{\Longrightarrow} Q'$ such that $P' \approx Q'$ and

(2) if $Q \xrightarrow{\alpha} Q'$, then there is a sequence of transitions $P \stackrel{\tau}{\Longrightarrow} \circ \xrightarrow{\alpha} \circ \stackrel{\tau}{\Longrightarrow} P'$ such that $P' \approx Q'$.

Observation Congruence

Definition 2.35 (Observation congruence) (Milner 1989)

$P, Q \in Prc$ are observationally congruent, denoted $P \approx^c Q$, if for every $\alpha \in Act$ (including $\alpha = \tau$):

(1) if $P \xrightarrow{\alpha} P'$, then there is a sequence of transitions $Q \stackrel{\tau}{\Longrightarrow} \circ \xrightarrow{\alpha} \circ \stackrel{\tau}{\Longrightarrow} Q'$ such that $P' \approx Q'$ and

(2) if $Q \xrightarrow{\alpha} Q'$, then there is a sequence of transitions $P \stackrel{\tau}{\Longrightarrow} \circ \xrightarrow{\alpha} \circ \stackrel{\tau}{\Longrightarrow} P'$ such that $P' \approx Q'$.

Remark

- $\approx^c$ differs from $\approx$ only in that $\approx^c$ requires τ-moves by P or Q to be mimicked by at least one τ-move in the other process.
- Moreover, this only applies to the first step; the successors just have to satisfy $P' \approx Q'$ (and not necessarily $P' \approx^c Q'$).

Observation Congruence

Definition 2.35 (Observation congruence) (Milner 1989)

$P, Q \in Prc$ are observationally congruent, denoted $P \approx^c Q$, if for every $\alpha \in Act$ (including $\alpha = \tau$):

(1) if $P \xrightarrow{\alpha} P'$, then there is a sequence of transitions $Q \stackrel{\tau}{\Longrightarrow} \circ \xrightarrow{\alpha} \circ \stackrel{\tau}{\Longrightarrow} Q'$ such that $P' \approx Q'$ and

(2) if $Q \xrightarrow{\alpha} Q'$, then there is a sequence of transitions $P \stackrel{\tau}{\Longrightarrow} \circ \xrightarrow{\alpha} \circ \stackrel{\tau}{\Longrightarrow} P'$ such that $P' \approx Q'$.

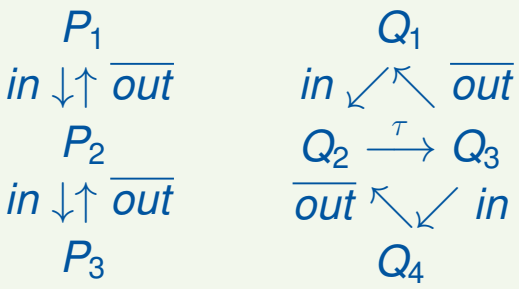
Remark

- $\approx^c$ differs from $\approx$ only in that $\approx^c$ requires τ-moves by P or Q to be mimicked by at least one τ-move in the other process.
- Moreover, this only applies to the first step; the successors just have to satisfy $P' \approx Q'$ (and not necessarily $P' \approx^c Q'$).
- Thus: if $P \not\xrightarrow{\tau}$ and $Q \not\xrightarrow{\tau}$, then $P \approx^c Q$ iff $P \approx Q$.

Examples

Example 2.36

(1) Sequential and parallel two-place buffer:

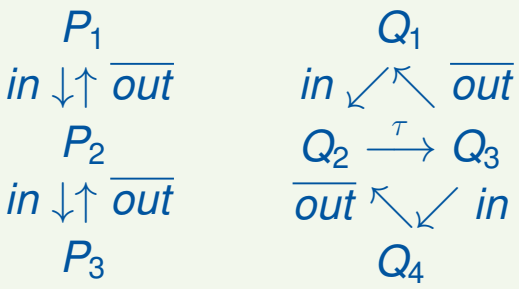


$P_1 \approx^c Q_1$ since $P_1 \approx Q_1$ (cf. Example 2.32(2)) and neither P_1 nor Q_1 has initial τ-steps.

Examples

Example 2.36

(1) Sequential and parallel two-place buffer:



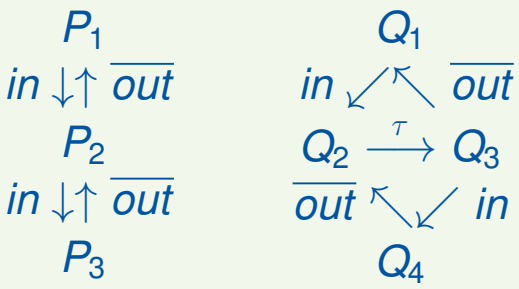
$P_1 \approx^c Q_1$ since $P_1 \approx Q_1$ (cf. Example 2.32(2)) and neither P_1 nor Q_1 has initial τ-steps.

(2) $\tau.a.\mathrm{nil} \not\approx^c a.\mathrm{nil}$ (since $\tau.a.\mathrm{nil} \xrightarrow{\tau}$ but $a.\mathrm{nil} \not\xrightarrow{\tau}$);
thus the counterexample to congruence of $\approx$ for $+$ does not apply.

Examples

Example 2.36

(1) Sequential and parallel two-place buffer:



$P_1 \approx^c Q_1$ since $P_1 \approx Q_1$ (cf. Example 2.32(2)) and neither P_1 nor Q_1 has initial τ-steps.

(2) $\tau.a.\mathrm{nil} \not\approx^c a.\mathrm{nil}$ (since $\tau.a.\mathrm{nil} \xrightarrow{\tau}$ but $a.\mathrm{nil} \not\xrightarrow{\tau}$);
thus the counterexample to congruence of $\approx$ for $+$ does not apply.

(3) $a.\tau.\mathrm{nil} \approx^c a.\mathrm{nil}$ (since $\tau.\mathrm{nil} \approx \mathrm{nil}$).

Properties of Observation Congruence

Lemma 2.37

For every $P, Q \in Prc$,

(1) *$P \sim Q$ implies $P \approx^c Q$, and $P \approx^c Q$ implies $P \approx Q$*

(2) *$\approx^c$ is an equivalence relation*

(3) *$\approx^c$ is a CCS congruence*

(4) *$\approx^c$ is (non-τ) deadlock-sensitive*

(5) *$P \approx^c Q$ if and only if $P + R \approx Q + R$ for every $R \in Prc$*

Proof.

omitted □

Properties of Observation Congruence

Lemma 2.37

For every $P, Q \in Prc$,

(1) $P \sim Q$ *implies* $P \approx^c Q$, *and* $P \approx^c Q$ *implies* $P \approx Q$

(2) $\approx^c$ *is an equivalence relation*

(3) $\approx^c$ *is a CCS congruence*

(4) $\approx^c$ *is (non-*τ*) deadlock-sensitive*

(5) $P \approx^c Q$ *if and only if* $P + R \approx Q + R$ *for every* $R \in Prc$

Proof.

omitted □

Note: (5) states that $\approx^c$ is the “minimal fix” to establish congruence of $\approx$.