CS780 Discrete-State Models

Instructor: Peter Kemper

R 006, phone 221-3462, email:kemper@cs.wm.edu

Office hours: Mon,Wed 3-5 pm

Today:

Milner‵s Calculus of Communicating Systems

Strong & Weak Bisimulation

Observational Congruence

Quick Reference:

Robin Milner, A Calculus of Communicating Systems, Springer, LNCS 92, 1980.

Robin Miner, Communication and Concurrency, Prentice Hall, 1989.

Outline

- Origin of Process Algebras:
  Calculus of Communicating Systems (CCS)
- Trace Equivalence
- Bisimulation
  - Strong
  - Weak
- Observational Congruence

- Credits:
  - Slides from Noll, Katoen, RWTH Aachen, Germany, 2007/08

Definition 1.2 (Syntax of CCS)

- Let N be a set of (action) names.
- $\overline{N} := \{\overline{a} \mid a \in N\}$ denotes the set of co-names.
- $Act := N \cup \overline{N} \cup \{\tau\}$ is the set of actions where τ denotes the silent (or: unobservable) action.
- Let Pid be a set of process identifiers.
- The set Prc of process expressions is defined by the following syntax:
$$\begin{array}{rcll} P & ::= & \mathsf{nil} & \text{(inaction)} \\ & \mid & \alpha.P & \text{(prefixing)} \\ & \mid & P_1 + P_2 & \text{(choice)} \\ & \mid & P_1 \parallel P_2 & \text{(parallel composition)} \\ & \mid & \mathsf{new}\, a\, P & \text{(restriction)} \\ & \mid & A(a_1, \ldots, a_n) & \text{(process call)} \end{array}$$
where $\alpha \in Act$, $a, a_i \in N$, and $A \in Pid$.

Definition 1.2 (continued)

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

$$(A_i(a_{i1}, \dots, a_{in_i}) = P_i \mid 1 \leq i \leq k)$$

where $k \geq 1$, $A_i \in Pid$ (pairwise different), $a_{ij} \in N$, and $P_i \in Prc$ (with process identifiers from $\{A_1, \dots, A_k\}$).

Meaning of CCS Operators

- nil is an inactive process that can do nothing.
- $\alpha.P$ can execute α and then behaves as P.
- An action $a \in N$ ($\overline{a} \in \overline{N}$) is interpreted as an input (output, resp.) operation. Both are complementary: if executed in parallel (i.e., in $P_1 \parallel P_2$), they are merged into a τ-action.
- $P_1 + P_2$ represents the non-deterministic choice between P_1 and P_2.
- $P_1 \parallel P_2$ denotes the concurrent execution of P_1 and P_2, involving interleaving or communication.
- The restriction $\mathsf{new}\, a\, P$ declares a as a local name which is only known in P.
- The behavior of a process call $A(a_1, \ldots, a_n)$ is defined by the right-hand side of the corresponding equation where $a_1, \ldots, a_n$ replace the formal name parameters.

Notational Conventions

- $\overline{\overline{a}}$ means a
- $P_1 + \ldots + P_n$ $(n \in \mathbb{N})$ sometimes written as $\sum_{i=1}^{n} P_i$ where $\sum_{i=1}^{0} P_i :=$ nil
- ".nil" can be omitted: $a.b$ means $a.b.\mathsf{nil}$
- new $a, b\, P$ means new a new $b\, P$
- $A(a_1, \ldots, a_n)$ sometimes written as $A(\vec{a})$, $A()$ as A
- prefixing and restriction binds stronger than composition, composition binds stronger than choice:

$$\mathsf{new}\, a\, P + b.Q \parallel R \quad \text{means} \quad (\mathsf{new}\, a\, P) + ((b.Q) \parallel R)$$

Labelled Transition System

Goal: represent behavior of system by (infinite) graph

- nodes = system states
- edges = transitions between states

Definition 2.1 (Labeled transition system)

A (Act-)labeled transition system (LTS) is a triple $(S, Act, \longrightarrow)$ consisting of

- a set S of states
- a set Act of (action) labels
- a transition relation $\longrightarrow \subseteq S \times Act \times S$

If $(s, \alpha, s') \in \longrightarrow$ we write $s \xrightarrow{\alpha} s'$. An LTS is called finite if S is so.

Remarks:

- sometimes an initial state $s_0 \in S$ is distinguished
- (finite) LTSs correspond to (finite) automata without final states

Semantics of CCS

We define the assignment

$$\begin{array}{rcl} \text{syntax} & \longrightarrow & \text{semantics} \\ \text{process definition} & \longmapsto & \text{LTS} \end{array}$$

by induction over the syntactic structure of process expressions. Here we employ derivation rules of the form

$$\frac{\text{premise(s)}}{\text{conclusion}}\,\text{rule name}$$

which can be composed to complete derivation trees.

Semantics of CCS

Definition 2.2 (Semantics of CCS)

A process definition $(A_i(a_{i1}, \ldots, a_{in_i}) = P_i \mid 1 \leq i \leq k)$ determines the LTS $(Prc, Act, \longrightarrow)$ whose transitions can be inferred from the following rules ($P, P', Q, Q' \in Prc$, $\alpha \in Act$, $\lambda \in N \cup \overline{N}$, $a \in N$):

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}\,(\mathsf{Act}) \qquad \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}\,(\mathsf{Com})$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}\,(\mathsf{Sum}_1) \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}\,(\mathsf{Sum}_2)$$

$$\frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}\,(\mathsf{Par}_1) \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'}\,(\mathsf{Par}_2)$$

$$\frac{P \xrightarrow{\alpha} P' \quad \alpha \notin \{a, \overline{a}\}}{\mathsf{new}\, a\, P \xrightarrow{\alpha} \mathsf{new}\, a\, P'}\,(\mathsf{New}) \qquad \frac{A(\vec{a}) = P \quad P[\vec{a} \mapsto \vec{b}] \xrightarrow{\alpha} P'}{A(\vec{b}) \xrightarrow{\alpha} P'}\,(\mathsf{Call})$$

(Here $P[\vec{a} \mapsto \vec{b}]$ denotes the replacement of every a_i by b_i in P.)

Semantics of CCS

Example 2.3

1. One-place buffer:

$$B(in, out) = in.\overline{out}.B(in, out)$$

2. Sequential two-place buffer:

$$\begin{array}{rcl} B_0(in, out) & = & in.B_1(in, out) \\ B_1(in, out) & = & \overline{out}.B_0(in, out) + in.B_2(in, out) \\ B_2(in, out) & = & \overline{out}.B_1(in, out) \end{array}$$

3. Parallel two-place buffer:

$$\begin{array}{rcl} B_{\|}(in, out) & = & \mathsf{new}\ com\,(B(in, com) \parallel B(com, out)) \\ B(in, out) & = & in.\overline{out}.B(in, out) \end{array}$$

Semantics of CCS

Example 2.3 (continued)

Complete LTS of parallel two-place buffer:

$$\begin{tikzcd}
B_{\|}(in, out) \arrow[d, "in"] & & \mathsf{new}\ com\,(B(in, com) \parallel B(com, out)) \arrow[dll, "in"'] \\
\mathsf{new}\ com\,(\overline{com}.B(in, com) \parallel B(com, out)) \arrow[rr, "\tau"] & & \mathsf{new}\ com\,(B(in, com) \parallel \overline{out}.B(com, out)) \arrow[u, "\overline{out}"'] \arrow[dl, "in"] \\
& \mathsf{new}\ com\,(\overline{com}.B(in, com) \parallel \overline{out}.B(com, out)) \arrow[ul, "\overline{out}"'] &
\end{tikzcd}$$

Recursion

Here: recursive processes defined using equations such as

$$B(in, out) = in.\overline{out}.B(in, out)$$

(simultaneous recursion)

Alternative: explicit fixpoint operator

- syntax: $P ::= \mathsf{nil} \mid \ldots \mid \mathsf{fix}\, A\, P \in Prc \quad$ (where $A \in Pid$)
- semantics: $\dfrac{P[A \mapsto P] \xrightarrow{\alpha} P'}{\mathsf{fix}\, A\, P \xrightarrow{\alpha} \mathsf{fix}\, A\, P'}\,(\mathsf{Fix})$
- example: $\dfrac{\dfrac{}{in.\overline{out}.in.\overline{out}.B \xrightarrow{in} \overline{out}.in.\overline{out}.B}\,(\mathsf{Act})}{\mathsf{fix}\, B\, in.\overline{out}.B \xrightarrow{in} \mathsf{fix}\, B\, \overline{out}.in.\overline{out}.B}\,(\mathsf{Fix})$

(nested scalar recursion)

Advantage: only process term level required (no equations)

$\Longrightarrow$ simplification of theory

Disadvantage: bad readability of process definitions

Equivalence

Goal: identify process expressions which have the same "meaning" but differ in their syntax

Definition 3.1 (Equivalence relation)

Let $\cong \subseteq S \times S$ be a binary relation over some set S. Then $\cong$ is called an equivalence relation if it is

- reflexive, i.e., $s \cong s$ for every $s \in S$,
- symmetric, i.e., $s \cong t$ implies $t \cong s$ for every $s, t \in S$, and
- transitive, i.e., $s \cong t$ and $t \cong u$ implies $s \cong u$ for every $s, t, u \in S$.

Equivalence of CCS Processes

- **Generally:** two syntactic objects are equivalent if they have the same “meaning”
- **Here:** two processes are equivalent if they have the same “behavior” (i.e., communication potential)
- Communication potential described by LTS
- **Idea:** choose

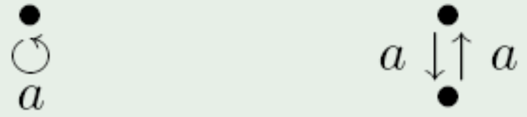
$$\text{meaning of a process } P := LTS(P)$$

- **But:** yields too many distinctions:

Example 3.2

$$X(a) = a.X(a) \quad Y(a) = a.a.Y(a)$$

LTS:



although both processes can (only) execute infinitely many a-actions, and should be considered equivalent therefore

Desired Properties of Equivalence

Wanted: a "feasible" (i.e., efficiently decidable) semantic equivalence between CCS processes which

1. identifies processes whose LTSs coincide,
2. implies trace equivalence, i.e., considers two processes equivalent only if both can execute the same actions sequences (formal definition later), and
3. is a congruence, i.e., allows to replace a subprocess by an equivalent counterpart without changing the overall semantics of the system (formal definition later).

Formally: we are looking for a congruence relation $\cong \subseteq Prc \times Prc$ such that

$$LTS(P) = LTS(Q) \implies P \cong Q \implies Tr(P) = Tr(Q)$$

Congruence

Goal: replacing a subcomponent of a system by an equivalent process should yield an equivalent systems

$\Longrightarrow$ modular system development

Definition 3.3 (CCS congruence)

An equivalence relation $\cong \subseteq Prc \times Prc$ is said to be a CCS congruence if it is preserved by the CCS constructs; that is, if $P, Q, R \in Prc$ such that $P \cong Q$ then

$$\begin{aligned}
\alpha.P &\cong \alpha.Q \\
P + R &\cong Q + R \\
R + P &\cong R + Q \\
P \parallel R &\cong Q \parallel R \\
R \parallel P &\cong R \parallel Q \\
\mathsf{new}\, a\, P &\cong \mathsf{new}\, a\, Q
\end{aligned}$$

for every $\alpha \in Act$ and $a \in N$.

Trace Equivalence

Definition 3.4 (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 3.5 (One-place buffer)

$B(in, out) = in.\overline{out}.B(in, out)$

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

Trace Equivalence

Remarks:

- The trace language of $P \in Prc$ is accepted by the LTS of P, interpreted as an automaton where every state is final.
- Trace equivalence is obviously an equivalence relation (i.e., reflexive, symmetric, and transitive).
- Trace equivalence possesses the postulated properties of a process equivalence:
  1. it identifies processes with identical LTSs: the trace language of a process consists of the (finite) paths in the LTS. Hence processes with identical LTSs are trace equivalent.
  2. it implies trace equivalence: trivial
  3. it is a congruence:

Congruence

Goal: replacing a subcomponent of a system by an equivalent process should yield an equivalent systems

$\Longrightarrow$ modular system development

Definition (CCS congruence)

An equivalence relation $\cong \subseteq Prc \times Prc$ is said to be a CCS congruence if it is preserved by the CCS constructs; that is, if $P \cong Q$ then

$$\begin{array}{rcl} \alpha.P & \cong & \alpha.Q \\ P + R & \cong & Q + R \\ R + P & \cong & R + Q \\ P \parallel R & \cong & Q \parallel R \\ R \parallel P & \cong & R \parallel Q \\ \mathsf{new}\, a\, P & \cong & \mathsf{new}\, a\, Q \end{array}$$

for every $\alpha \in Act$, $R \in Prc$, and $a \in N$.

Trace Equivalence

Theorem 3.6

Trace equivalence is a congruence.

Proof.

(only for $+$; remaining operators analogously)
Clearly we have:

$$Tr(P_1 + P_2) = Tr(P_1) \cup Tr(P_2)$$

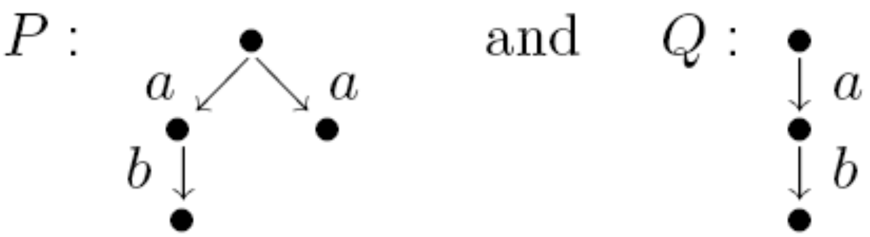
Now let $P, Q, R \in Prc$ with $Tr(P) = Tr(Q)$. Then:

$$\begin{array}{ll} Tr(P+R) & \\ \quad = & Tr(P) \cup Tr(R) \\ \quad = & Tr(Q) \cup Tr(R) \\ \quad = & Tr(Q+R) \\ \Longrightarrow & P+R, Q+R \text{ trace equiv.} \end{array} \qquad \begin{array}{ll} Tr(R+P) & \\ \quad = & Tr(R) \cup Tr(P) \\ \quad = & Tr(R) \cup Tr(Q) \\ \quad = & Tr(R+Q) \\ \Longrightarrow & R+P, R+Q \text{ trace equiv.} \end{array}$$

□

Trace Equivalence

- We have found a process equivalence with the three required properties.
- Are we satisfied? No!



are trace equivalent ($Tr(P) = Tr(Q) = \{\varepsilon, a, ab\}$)

- But P and Q are distinguishable:
  - both can execute ab
  - but P can deny b
  - while Q always has to offer b after a

⟹ take into account such deadlock properties

Deadlock

Definition 3.7 (Deadlock)

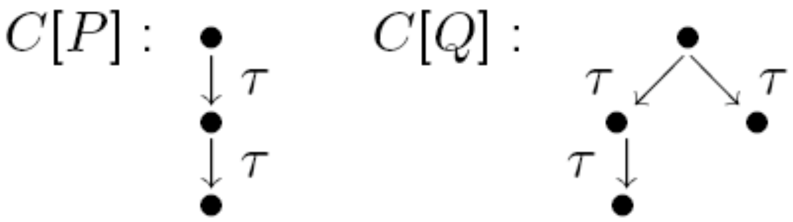
Let $P, Q \in Prc$ and $w \in Act^*$ such that $P \xrightarrow{w}{}^{*} Q$ and $Q \not\longrightarrow$.
Then Q is called a w-deadlock of P.

- Thus $P := a.b.\mathsf{nil} + a.\mathsf{nil}$ has an a-deadlock, in contrast to $Q := a.b.\mathsf{nil}$.
- Such properties are important since it can be crucial that a certain communication is eventually possible.
- We therefore extend our set of postulates: our semantic equivalence $\cong$ should
  1. identify processes with identical LTSs;
  2. imply trace equivalence;
  3. be a congruence; and
  4. be deadlock sensitive, i.e., if $P \cong Q$ and if P has a w-deadlock, then Q has a w-deadlock (and vice versa, by equivalence).

Deadlock

The combination of congruence and deadlock sensitivity also excludes the following equivalence:

If $P \cong Q$, by congruence this equivalence should hold in every context. But $C[\cdot] := \mathsf{new}\, a, b, c\, (\overline{a}.\overline{b}.\mathsf{nil} \parallel \cdot)$ yields the following conflict:



no τ-deadlock $\quad$ τ-deadlock

(Note: P and Q are obviously trace equivalent)

Desired Properties of Equivalence

Wanted: a "feasible" (i.e., efficiently decidable) semantic equivalence between CCS processes which

1. identifies processes whose LTSs coincide,
2. implies trace equivalence, i.e., considers two processes equivalent only if both can execute the same actions sequences (formal definition later), and
3. is a congruence, i.e., allows to replace a subprocess by an equivalent counterpart without changing the overall semantics of the system (formal definition later).
4. is deadlock sensitive, i.e., if $P \cong Q$ and if P has a w-deadlock, then Q has a w-deadlock (and vice versa, by equivalence).

Formally: we are looking for a deadlock-sensitive congruence relation $\cong \subseteq Prc \times Prc$ such that

$$LTS(P) = LTS(Q) \implies P \cong Q \implies Tr(P) = Tr(Q)$$

Strong Bisimulation

Observation: equivalence should be deadlock sensitive
$\Longrightarrow$ needs to take branching structure of processes into account

This is guaranteed by a definition according to the following scheme:

Bisimulation scheme

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

In the first version we will ignore the special function of the silent action τ ($\Longrightarrow$ *weak bisimulation*)

Strong Bisimulation

Definition 4.1 (Strong bisimulation)

A relation $\rho \subseteq Prc \times Prc$ is called a strong bisimulation if $P \rho Q$ implies, for every $\alpha \in Act$,

1. $P \xrightarrow{\alpha} P' \implies$ ex. $Q' \in Prc$ such that $Q \xrightarrow{\alpha} Q'$ and $P' \rho Q'$
2. $Q \xrightarrow{\alpha} Q' \implies$ ex. $P' \in Prc$ such that $P \xrightarrow{\alpha} P'$ and $P' \rho Q'$

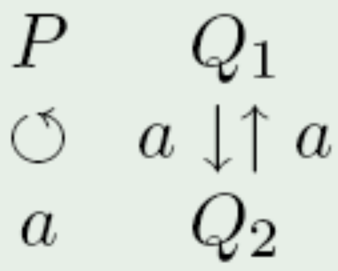
$P, Q \in Prc$ are called strongly bisimilar (notation: $P \sim Q$) if there exists a strong bisimulation ρ such that $P \rho Q$.

Theorem 4.2

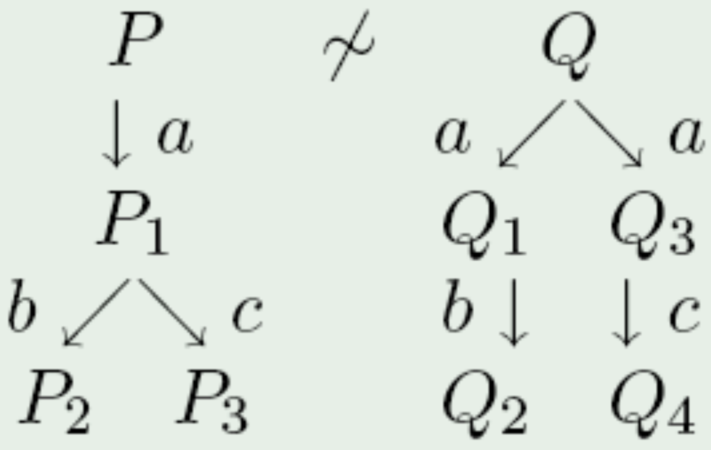
$\sim$ is an equivalence relation.

Strong Bisimulation

❶



❷



(remember: $Tr(P) = Tr(Q)$)

Strong Bisimulation

Example 4.4

Binary semaphore

(controls exclusive access to two instances of a resource)

Sequential definition:

$$\begin{array}{rcl} Sem_0(get, put) & = & get.Sem_1(get, put) \\ Sem_1(get, put) & = & get.Sem_2(get, put) + put.Sem_0(get, put) \\ Sem_2(get, put) & = & put.Sem_1(get, put) \end{array}$$

Parallel definition:

$$\begin{array}{rcl} S(get, put) & = & S_0(get, put) \parallel S_0(get, put) \\ S_0(get, put) & = & get.S_1(get, put) \\ S_1(get, put) & = & put.S_0(get, put) \end{array}$$

Proposition: $Sem_0(get, put) \sim S(get, put)$ **How to prove this?**

Strong Bisimulation

Example 4.5

Two-place buffer

Sequential definition:

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

Parallel definition:

$$\begin{aligned} B_{\|}(in, out) &= \text{new}\ com\ (B(in, com) \parallel B(com, out)) \\ B(in, out) &= in.\overline{out}.B(in, out) \end{aligned}$$

Proposition: $B_0(in, out) \not\sim B_{\|}(in, out)$ **How to prove this?**

Properties of Strong Bisimulation

It remains to show that strong bisimulation has the required properties of a process equivalence:

1. Identification of processes with identical LTSs:
   since the definition of strong bisimulation directly relies on the transition relation, processes with identical transition trees are clearly strongly bisimilar
2. Implication of trace equivalence: following slides
3. CCS congruence: following slides
4. Deadlock sensitivity: following slides

Strong Bisimulation => Trace Equivalence

Definition (Trace language; repetition)

The trace language of $P \in \mathit{Prc}$ is given by

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

Theorem 5.1

For every $P, Q \in \mathit{Prc}$, $P \sim Q$ implies $\mathit{Tr}(P) = \mathit{Tr}(Q)$.

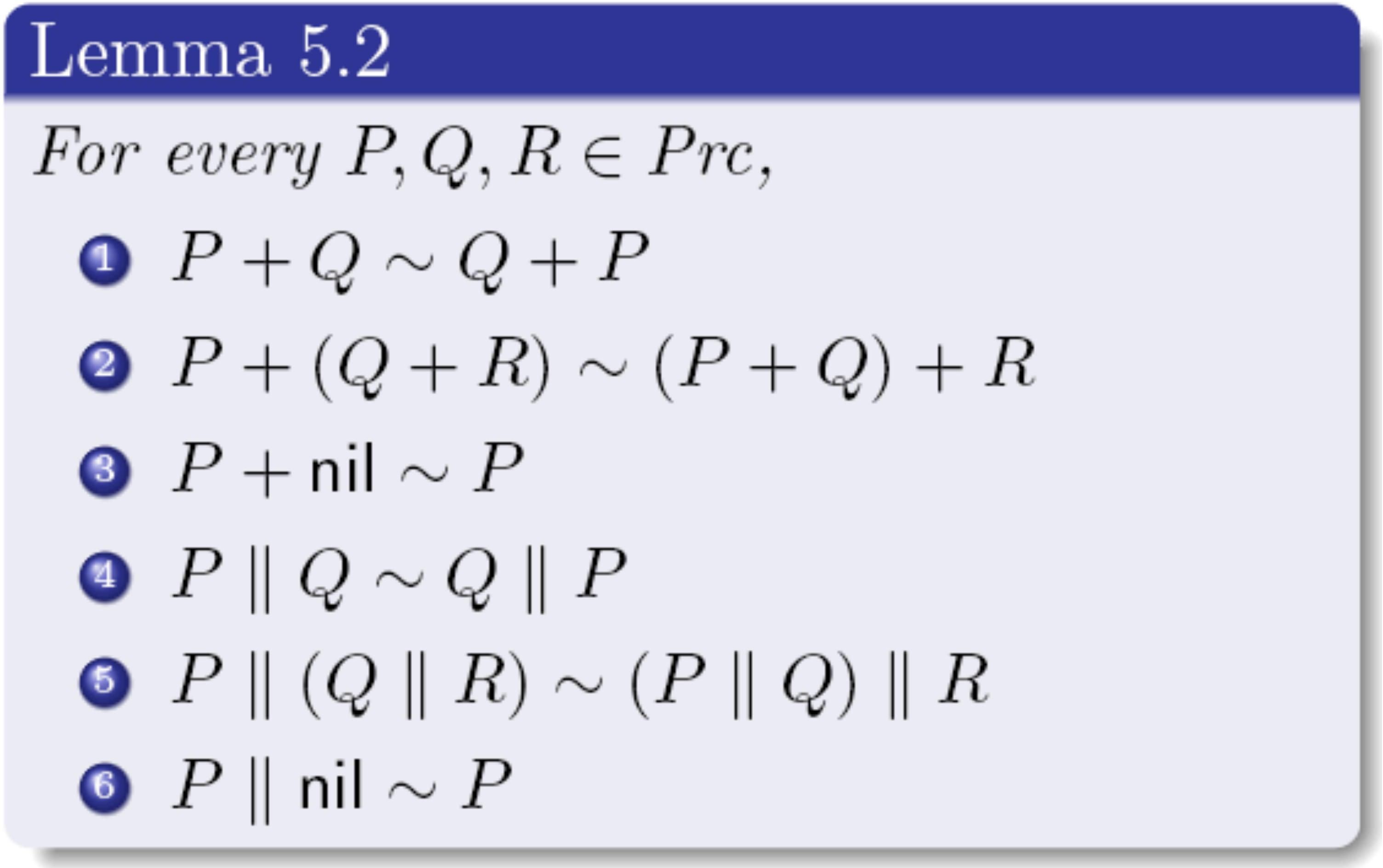
Proof.

- Assume that $P \sim Q$ but $w \in \mathit{Tr}(P) \setminus \mathit{Tr}(Q)$.
- Let $v \in \mathit{Act}^*$ be the longest prefix of w such that $v \in \mathit{Tr}(Q)$ (i.e., $w = v\alpha u$ for some $\alpha \in \mathit{Act}$ and $u \in \mathit{Act}^*$).
- Let $P', P'' \in \mathit{Prc}$ such that $P \xrightarrow{v}{}^{*} P' \xrightarrow{\alpha} P''$.
- Since $P \sim Q$ there exists $Q' \in \mathit{Prc}$ such that $Q \xrightarrow{v}{}^{*} Q'$ and $P' \sim Q'$ (by induction on $|v|$).
- But we have that $P' \xrightarrow{\alpha} P''$ whereas $Q' \not\xrightarrow{\alpha}$ $\implies$ contradiction

□

Congruence Property

- Makes use of following Lemma



Lemma 5.2

For every $P, Q, R \in Prc$,

1. $P + Q \sim Q + P$
2. $P + (Q + R) \sim (P + Q) + R$
3. $P + \mathsf{nil} \sim P$
4. $P \parallel Q \sim Q \parallel P$
5. $P \parallel (Q \parallel R) \sim (P \parallel Q) \parallel R$
6. $P \parallel \mathsf{nil} \sim P$

Congruence

Definition (CCS congruence; repetition)

An equivalence relation $\cong \subseteq Prc \times Prc$ is said to be a CCS congruence if it is preserved by the CCS constructs; that is, if $P \cong Q$ then

$$\begin{aligned} \alpha.P &\cong \alpha.Q \\ P + R &\cong Q + R \\ R + P &\cong R + Q \\ P \parallel R &\cong Q \parallel R \\ R \parallel P &\cong R \parallel Q \\ \mathsf{new}\, a\, P &\cong \mathsf{new}\, a\, Q \end{aligned}$$

for every $\alpha \in Act$, $R \in Prc$, and $a \in N$.

Theorem 5.3

$\sim$ is a CCS congruence.

Deadlock

Definition (Deadlock; repetition)

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

An equivalence relation $\cong \subseteq \mathit{Prc} \times \mathit{Prc}$ is called deadlock sensitive if for every $P \cong Q$ such that P has a w-deadlock, Q also has a w-deadlock.

Theorem 5.4

$\sim$ *is deadlock sensitive.*

Summary

Definition (Strong bisimulation)

A relation $\rho \subseteq Prc \times Prc$ is called a strong bisimulation if $P \rho Q$ implies, for every $\alpha \in Act$,

1. $P \xrightarrow{\alpha} P' \implies$ ex. $Q' \in Prc$ such that $Q \xrightarrow{\alpha} Q'$ and $P' \rho Q'$
2. $Q \xrightarrow{\alpha} Q' \implies$ ex. $P' \in Prc$ such that $P \xrightarrow{\alpha} P'$ and $P' \rho Q'$

$P, Q \in Prc$ are called strongly bisimilar (notation: $P \sim Q$) if there exists a strong bisimulation ρ such that $P \rho Q$.

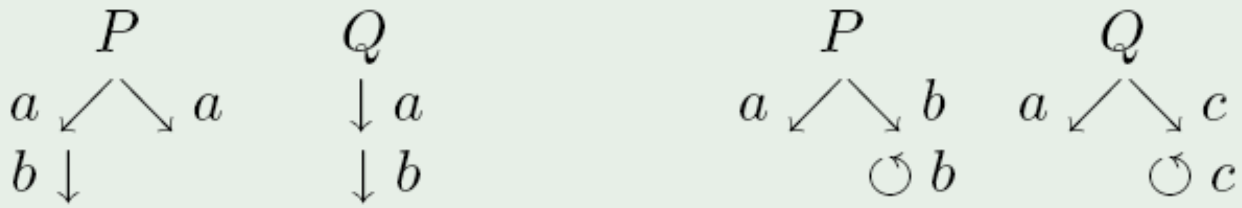
Theorem

1. $\sim$ *is an equivalence relation*
2. $LTS(P) = LTS(Q) \implies P \sim Q$
3. $P \sim Q \implies Tr(P) = Tr(Q)$
4. $\sim$ *is a CCS congruence*
5. $\sim$ *is deadlock sensitive*

Traces and Deadlocks

Remark: traces and deadlocks are independent in the following sense

Example 6.1



same traces
different deadlocks

different traces
same deadlocks

But: if all traces are finite, then processes with identical deadlocks are trace equivalent (since every trace is a prefix of some deadlock)

Computing Equivalences

Problem

Given: $P, Q \in Prc$

Question: $P \sim Q$?

- Basic Algorithm:
  - Paige, Tarjan: Three partition refinement algoriths, SIAM J. Computing, 16, 1987.
- Multiple variants and refinements, in particular wrt stochastic models
  - P. Buchholz. Exact and ordinary lumpability in finite Markov chains. Journal of Applied Probability, 31:59–75, 1994.
  - S. Derisavi, H. Hermanns, and W. H. Sanders. Optimal State-Space Lumping in Markov Chains, Information Proc. Letters, 87, 6, 2003

Remark: if states from two disjoint LTSs $(S_1, Act_1, \longrightarrow_1)$ and $(S_2, Act_2, \longrightarrow_2)$ (where $S_1 \cap S_2 = \emptyset$) are to be compared, their union $(S_1 \cup S_2, Act_1 \cup Act_2, \longrightarrow_1 \cup \longrightarrow_2)$ is chosen as input (here usually $Act_1 = Act_2$)

Partition Refinement Algorithm

Theorem 6.2 (Partitioning algorithm for $\sim$)

Input: *LTS* $(S, Act, \longrightarrow)$ *(S finite)*

Procedure:

1. *Start with initial partition* $\Pi := \{S\}$
2. *Let* $B \in \Pi$ *be a block and* $\alpha \in Act$ *an action*
3. *For every* $P \in B$, *let*

   $$\alpha(P) := \{C \in \Pi \mid \textit{ex. } P' \in C \textit{ with } P \xrightarrow{\alpha} P'\}$$

   be the set of P's α-successor blocks
4. *Partition* $B = \bigcup_{i=1}^{k} B_i$ *such that*

   $$P, Q \in B_i \iff \alpha(P) = \alpha(Q) \textit{ for every } \alpha \in Act$$
5. *Let* $\Pi := (\Pi \setminus \{B\}) \cup \{B_1, \ldots, B_k\}$
6. *Continue with (2) until* Π *is stable*

Output: *Partition* $\hat{\Pi}$ *of* S

Then, for every $P, Q \in S$,

$$P \sim Q \iff \textit{ex. } B \in \hat{\Pi} \textit{ with } P, Q \in B$$

Strong Simulation

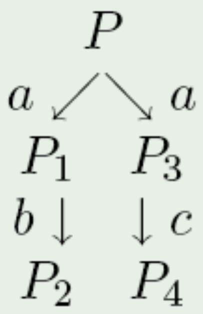
Observation: sometimes, the concept of strong bisimulation is too strong (example: extending a system by new features)

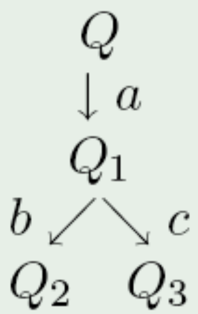
Definition 7.1 (Strong simulation)

A relation $\rho \subseteq Prc \times Prc$ is called a strong simulation if, whenever $P \rho Q$ and $P \xrightarrow{\alpha} P'$, there exists $Q' \in Prc$ such that $Q \xrightarrow{\alpha} Q'$ and $P' \rho Q'$.
We say that Q strongly simulates P if there exists a strong simulation ρ such that $P \rho Q$.

Thus: if Q strongly simulates P, then whatever transition path P takes, Q can match it by a path which retains all of P's options.

Example 7.2





Q strongly simulates P, but not vice versa

Strong Simulation and Bisimulation

Corollary 7.3

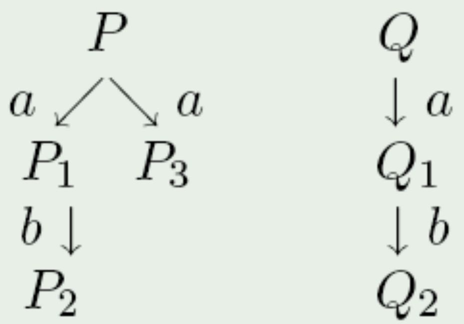
If $P \sim Q$, then Q strongly simulates P, and P strongly simulates Q.

Proof.

A strong bisimulation $\rho \subseteq Prc \times Prc$ for $P \sim Q$ is a strong simulation for both directions. $\square$

Caveat: the converse does generally not hold!

Example 7.4



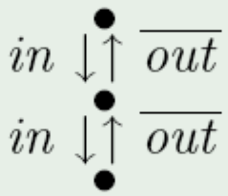
Q simulates P and vice versa,
but $P \not\sim Q$

Strong Bisimulation is not an ideal solution!

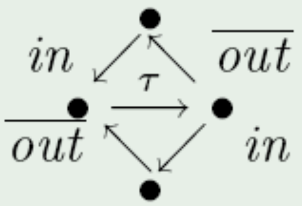
Example 7.5

Sequential and parallel two-place buffer:

$$\begin{aligned} B_0(in, out) &= in.B_1(in, out) \\ B_1(in, out) &= \overline{out}.B_0(in, out) + \\ &\quad\; in.B_2(in, out) \\ B_2(in, out) &= \overline{out}.B_1(in, out) \end{aligned} \qquad \begin{aligned} B_{\|}(in, out) &= \mathsf{new}\ com\ (B(in, com) \parallel \\ &\qquad B(com, out)) \\ B(in, out) &= in.\overline{out}.B(in, out) \end{aligned}$$



$\not\sim$



Weak Bisimulation

Idea: abstract from silent actions

Definition 7.6

- Given $w \in Act^*$, $\hat{w} \in (N \cup \overline{N})^*$ denotes the sequence of non-τ-actions in w (in particular, $\hat{\tau^n} = \varepsilon$ for every $n \in \mathbb{N}$).
- For $w = \alpha_1 \ldots \alpha_n \in Act^*$ and $P, Q \in Prc$, we let

$$P \overset{w}{\Longrightarrow} Q \iff P \left(\xrightarrow{\tau}\right)^* \xrightarrow{\alpha_1} \left(\xrightarrow{\tau}\right)^* \ldots \left(\xrightarrow{\tau}\right)^* \xrightarrow{\alpha_n} \left(\xrightarrow{\tau}\right)^* Q$$

  (and hence: $\overset{\varepsilon}{\Longrightarrow} = \left(\xrightarrow{\tau}\right)^*$).
- A relation $\rho \subseteq Prc \times Prc$ is called a weak bisimulation if $P \rho Q$ implies, for every $\alpha \in Act$,
  1. $P \xrightarrow{\alpha} P' \implies$ ex. $Q' \in Prc$ such that $Q \overset{\hat{\alpha}}{\Longrightarrow} Q'$ and $P' \rho Q'$
  2. $Q \xrightarrow{\alpha} Q' \implies$ ex. $P' \in Prc$ such that $P \overset{\hat{\alpha}}{\Longrightarrow} P'$ and $P' \rho Q'$
- $P, Q \in Prc$ are called weakly bisimilar (notation: $P \approx Q$) if there exists a weak bisimulation ρ such that $P \rho Q$.

Weak Bisimulation

Remark: each of the two clauses in the definition of weak bisimulation subsumes two cases:

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

Weak Bisimulation

Theorem 7.8

$\approx$ is an equivalence relation.

Proof.

in analogy to the corresponding proof for $\sim$ (Theorem 4.2)

In particular, the following characterization is still valid:

$$\approx = \bigcup \{\rho \mid \rho \text{ weak bisimulation}\},$$

i.e., $\approx$ is again itself a weak bisimulation. ☐

Weak Bisimulation

Moreover Definition 7.6 implies that every strong bisimulation is also a weak one (since, for every $\alpha \in Act$, $\xrightarrow{\alpha} \subseteq \overset{\hat{\alpha}}{\Longrightarrow}$). This yields the desired connection to LTS equivalence: for every $P, Q \in Prc$,

$$LTS(P) = LTS(Q) \implies P \sim Q \implies P \approx Q.$$

Furthermore trace equivalence is implied if the definition is adapted:

$$P \approx Q \implies \hat{Tr}(P) = \hat{Tr}(Q)$$

where $\hat{Tr}(P) := \{\hat{w} \mid w \in Tr(P)\} \subseteq (N \cup \overline{N})^*$.

Weak Bisimulation

Lemma 7.9

For every $P \in Prc$,

$$P \approx \tau.P$$

Proof.

We show that

$$\rho := \{(P, \tau.P)\} \cup id_{Prc}$$

is a weak bisimulation:

1. let $P \xrightarrow{\alpha} P'$
   $\implies \tau.P \xrightarrow{\tau} P \xrightarrow{\alpha} P'$
   $\implies \tau.P \overset{\hat{\alpha}}{\Longrightarrow} P'$ with $P' \rho P'$ (since $id_{Prc} \subseteq \rho$)
2. the only transition of $\tau.P$ is $\tau.P \xrightarrow{\tau} P$;
   it is simulated by $P \overset{\varepsilon}{\Longrightarrow} P$ with $P \rho P$

☐

Weak Bisimulation

Using Lemma 7.9, however, we can show that $\approx$ is not a congruence:

It is true that $b.\mathsf{nil} \approx \tau.b.\mathsf{nil}$ (Theorem 7.8, Lemma 7.9)
but $a.\mathsf{nil} + b.\mathsf{nil} \not\approx a.\mathsf{nil} + \tau.b.\mathsf{nil}$ (Example 7.7(b))

The other operators are uncritical, i.e., weak bisimilarity is preserved under prefixing, parallel composition, and restriction.

Weak Bisimulation

Also deadlock sensitivity is guaranteed if τ-actions are appropriately handled:

Theorem 7.10

Let $P, Q \in Prc$ such that $P \approx Q$. Then, for every $w \in (N \cup \overline{N})^*$,

$$P \overset{w}{\Longrightarrow} \not\longrightarrow \quad \Longleftrightarrow \quad Q \overset{w}{\Longrightarrow} \not\longrightarrow .$$

Proof.

analogously to Theorem 5.4 (induction on $|w|$) ☐

Properties of Weak Bisimulation

Properties

1. $P \sim Q \implies P \approx Q$
2. $\approx$ is an equivalence relation
3. $LTS(P) = LTS(Q) \implies P \approx Q$
4. $P \approx Q \implies \hat{Tr}(P) = \hat{Tr}(Q)$
5. $\approx$ is (non-τ) deadlock sensitive
6. For every $P \in Prc$, $P \approx \tau.P$
7. $\approx$ is not a congruence:

   It is true that $b.\mathsf{nil} \approx \tau.b.\mathsf{nil}$

   but $a.\mathsf{nil} + b.\mathsf{nil} \not\approx a.\mathsf{nil} + \tau.b.\mathsf{nil}$

Properties of Weak Bisimulation

Lemma 8.1

For every $P, Q, R \in Prc$,

1. $P + Q \approx Q + P$
2. $P + (Q + R) \approx (P + Q) + R$
3. $P + \mathsf{nil} \approx P$
4. $P \parallel Q \approx Q \parallel P$
5. $P \parallel (Q \parallel R) \approx (P \parallel Q) \parallel R$
6. $P \parallel \mathsf{nil} \approx P$

Observation Congruence

Goal: introduce an equivalence which has most of the desirable properties of $\approx$ and which is preserved under all CCS operators

Definition 8.2

$P, Q \in Prc$ are called observationally congruent (notation: $P \simeq Q$) if, for every $\alpha \in Act$,

① $P \xrightarrow{\alpha} P' \implies$ ex. $Q' \in Prc$ such that $Q \overset{\alpha}{\Longrightarrow} Q'$ and $P' \approx Q'$

② $Q \xrightarrow{\alpha} Q' \implies$ ex. $P' \in Prc$ such that $P \overset{\alpha}{\Longrightarrow} P'$ and $P' \approx Q'$

Remark: $\simeq$ differs from $\approx$ only in the use of $\overset{\alpha}{\Longrightarrow}$ rather than $\overset{\hat{\alpha}}{\Longrightarrow}$, i.e., it requires τ-actions from P or Q to be simulated by at least one τ-step in the other process. This only applies to the first step; the successors just have to satisfy $P' \approx Q'$ (and not $P' \simeq Q'$).

Observation Congruence

Example 8.3

1. Sequential and parallel two-place buffer:

$$\begin{tikzcd}
P_1 \arrow[d, shift right, "in"'] \\
P_2 \arrow[u, shift right, "\overline{out}"'] \arrow[d, shift right, "in"'] \\
P_3 \arrow[u, shift right, "\overline{out}"']
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
& Q_1 \arrow[dl, "in"'] & \\
Q_2 \arrow[rr, "\tau"] & & Q_3 \arrow[ul, "\overline{out}"'] \arrow[dl, "in"] \\
& Q_4 \arrow[ul, "\overline{out}"] &
\end{tikzcd}$$

   $P_1 \simeq Q_1$ since $P_1 \approx Q_1$ (cf. Example 7.7) and neither P_1 nor Q_1 has initial τ-steps

2. $\tau.a.\mathsf{nil} \not\simeq a.\mathsf{nil}$
   (since $\tau.a.\mathsf{nil} \xrightarrow{\tau}$ but $a.\mathsf{nil} \not\xrightarrow{\tau}$)

3. $a.\tau.\mathsf{nil} \simeq a.\mathsf{nil}$
   (since $\tau.\mathsf{nil} \approx \mathsf{nil}$)

Observation Congruence

Corollary 8.4

For every $P, Q \in Prc$,

1. $P \sim Q \implies P \simeq Q$
2. $P \simeq Q \implies P \approx Q$

Proof.

1. since $\xrightarrow{\alpha} \subseteq \overset{\alpha}{\Longrightarrow}$ and $\sim \subseteq \approx$
2. since $\overset{\alpha}{\Longrightarrow} \subseteq \overset{\hat{\alpha}}{\Longrightarrow}$

□

Remark: this implies that

- processes with identical LTSs are $\simeq$-equivalent,
- $\simeq$-equivalent processes are (non-τ) trace equivalent, and
- $\simeq$ is (non-τ) deadlock sensitive.

Observation Congruence

Theorem 8.5

For every $P, Q \in Prc$,

$$P \simeq Q \iff P + R \approx Q + R \textit{ for every } R \in Prc.$$

Remark: $\simeq$ is therefore the largest congruence contained in $\approx$

Theorem 8.6

$\simeq$ *is an equivalence relation.*

Theorem 8.7

$\simeq$ *is a CCS congruence.*

Theorem 8.8

For every $P, Q \in Prc$,

$$P \approx Q \iff P \simeq Q \textit{ or } P \simeq \tau.Q \textit{ or } \tau.P \simeq Q.$$

Observation Congruence

Goal: introduce an equivalence which has most of the desirable properties of $\approx$ and which is preserved under all CCS operators

Definition

$P, Q \in Prc$ are called observationally congruent (notation: $P \simeq Q$) if, for every $\alpha \in Act$,

1. $P \xrightarrow{\alpha} P' \implies$ ex. $Q' \in Prc$ such that $Q \overset{\alpha}{\Longrightarrow} Q'$ and $P' \approx Q'$
2. $Q \xrightarrow{\alpha} Q' \implies$ ex. $P' \in Prc$ such that $P \overset{\alpha}{\Longrightarrow} P'$ and $P' \approx Q'$

Remark: $\simeq$ differs from $\approx$ only in the use of $\overset{\alpha}{\Longrightarrow}$ rather than $\overset{\hat{\alpha}}{\Longrightarrow}$, i.e., it requires τ-actions from P or Q to be simulated by at least one τ-step in the other process. This only applies to the first step; the successors just have to satisfy $P' \approx Q'$ (and not $P' \simeq Q'$).

Observation Congruence

Properties

1. $$\begin{aligned} & LTS(P) = LTS(Q) \\ \implies & P \sim Q \\ \implies & P \simeq Q \\ \implies & P \approx Q \\ \implies & \hat{Tr}(P) = \hat{Tr}(Q) \end{aligned}$$
2. $\simeq$ is an equivalence relation
3. $\simeq$ is (non-τ) deadlock sensitive
4. $\simeq$ is a CCS congruence
5. For every $P, Q \in Prc$,
   $$P \simeq Q \iff P + R \approx Q + R \text{ for every } R \in Prc$$
6. For every $P, Q \in Prc$,
   $$P \approx Q \iff P \simeq Q \text{ or } P \simeq \tau.Q \text{ or } \tau.P \simeq Q$$

Observation Congruence

Theorem 9.1 (Partitioning algorithm for $\approx$)

Input: *LTS* $(S, Act, \longrightarrow)$ *(S finite)*

Procedure:

1. *Start with initial partition* $\Pi := \{S\}$
2. *Let* $B \in \Pi$ *be a block and* $\alpha \in Act$ *an action*
3. *For every* $P \in B$*, let*

   $$\alpha^*(P) := \{C \in \Pi \mid ex.\ P' \in C \text{ with } P \overset{\hat{\alpha}}{\Longrightarrow} P'\}$$

   be the set of P's α–successor blocks
4. *Partition* $B = \sum_{i=1}^{k} B_i$ *such that*

   $$P, Q \in B_i \iff \alpha^*(P) = \alpha^*(Q) \text{ for every } \alpha \in Act$$
5. *Let* $\Pi := (\Pi \setminus \{B\}) \cup \{B_1, \ldots, B_k\}$
6. *Continue with (2) until* Π *is stable*

Output: *Partition* $\hat{\Pi}$ *of* S

Then, for every $P, Q \in S$,

$$P \approx Q \iff ex.\ B \in \hat{\Pi} \text{ with } P, Q \in B$$

Observation Congruence

Remarks:

1. Since S is finite, $\alpha^*(P)$ is effectively computable in step (3) of the algorithm.
2. The $\approx$–partitioning algorithm can be interpreted as the application of the $\sim$–partitioning algorithm to an appropriately modified LTS:

$$\begin{array}{ll} & \text{Theorem 9.1 for } (S, Act, \longrightarrow) \\ \hat{=} & \text{Theorem 6.2 for } (S, Act, \longrightarrow') \\ & \text{where } \longrightarrow' := \bigcup_{\alpha \in Act} \xrightarrow{\alpha}{}' \text{ with } \xrightarrow{\alpha}{}' := \overset{\hat{\alpha}}{\Longrightarrow} \end{array}$$

Since the definition of $\simeq$ requires the weak bisimilarity of the intermediate states after the first step, Theorem 9.1 yields the decidability of $\simeq$:

Theorem 9.2 (Decidability of $\simeq$)

Let $(S, Act, \longrightarrow)$ *and* $\hat{\Pi}$ *as in Theorem 9.1. Then, for every* $P, Q \in S$,

$$P \simeq Q \iff \alpha^+(P) = \alpha^+(Q) \textit{ for every } \alpha \in Act$$

where $\alpha^+(P) := \{C \in \hat{\Pi} \mid \textit{ex. } P' \in C \textit{ with } P \overset{\alpha}{\Longrightarrow} P'\}$.

Summary

- Origin of Process Algebras:

  Calculus of Communicating Systems (CCS)
- Trace Equivalence
  - Insensitive to deadlocks!
- Bisimulation
  - Strong Bisimulation:

    too restrictive to be used for an equivalence between an abstract specification and a detailed implementation model,

    we need to abstract from internal operations
  - Weak Bisimulation:

    no congruence wrt to choice, problem is an initial Tau step
- Observational Congruence
  - Compromise between strong and week bisimulation
  - Yields congruence wrt CCS operations
- Equivalence classes can be determined with algorithms based on partition refinement