Concurrency Theory

Winter Semester 2019/20

Lecture 11: Trace Equivalence

Joost-Pieter Katoen and Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-19-20/ct/

Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

Outline of Lecture 11

Introduction

- When using process algebras like CCS, an important approach is to model both the specification and implementation as CCS processes, say *Spec* and *Impl*.

Introduction

- When using process algebras like CCS, an important approach is to model both the specification and implementation as CCS processes, say *Spec* and *Impl*.
- This gives rise to the natural question: when are two CCS processes behaving the same?

Introduction

- When using process algebras like CCS, an important approach is to model both the specification and implementation as CCS processes, say *Spec* and *Impl*.
- 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

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

Specification

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

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:

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

Outline of Lecture 11

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

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: $\mathit{Spec}_0 \equiv \ldots \equiv \mathit{Spec}_n \equiv \mathit{Impl}$ implies that $\mathit{Spec}_0 \equiv \mathit{Impl}$

Definition 11.1 (Equivalence)

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: $\mathit{Spec}_0 \equiv \ldots \equiv \mathit{Spec}_n \equiv \mathit{Impl}$ implies that $\mathit{Spec}_0 \equiv \mathit{Impl}$

Definition 11.1 (Equivalence)

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

Isomorphism: An Example Behavioural Equivalence

Definition 11.2 (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 11.2 (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

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

It follows immediately that $\equiv_{iso}$ is an equivalence. (Why?)

Isomorphism: An Example Behavioural Equivalence

Definition 11.2 (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. (Why?)

Example 11.3 (Abelian monoid laws for $+$ and $\|$)

For all CCS processes $P, Q \in Prc$,

1. $LTS(P + Q) \equiv_{iso} LTS(Q + P)$, $LTS(P \parallel Q) \equiv_{iso} LTS(Q \parallel P)$
2. $LTS((P + Q) + R) \equiv_{iso} LTS(P + (Q + R))$, $LTS((P \parallel Q) \parallel R) \equiv_{iso} LTS(P \parallel (Q \parallel R))$
3. $LTS(P + \text{nil}) \equiv_{iso} LTS(P \parallel \text{nil}) \equiv_{iso} LTS(P)$

Isomorphism II

Assumption

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

Isomorphism II

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 11

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:

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

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:

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

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:

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

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:

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

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:

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

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:
$$LTS(P) \equiv_{iso} LTS(Q) \implies P \equiv Q.$$
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:
$$P \equiv Q \implies Tr(P) = Tr(Q).$$
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., equivalent process can either both deadlock, or both cannot.[1]

[1] Later, we will generalise this to a set of properties that can be expressed in a logic.

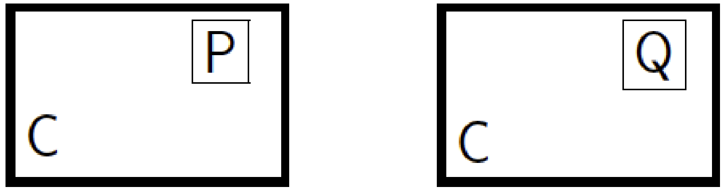
Requirements on Behavioural Equivalences

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:
$$LTS(P) \equiv_{iso} LTS(Q) \implies P \equiv Q.$$
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:
$$P \equiv Q \implies Tr(P) = Tr(Q).$$
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., equivalent process can either both deadlock, or both cannot.[1]
5. Optional: the coarsest possible equivalence: there should be no less discriminating equivalence satisfying all these requirements.

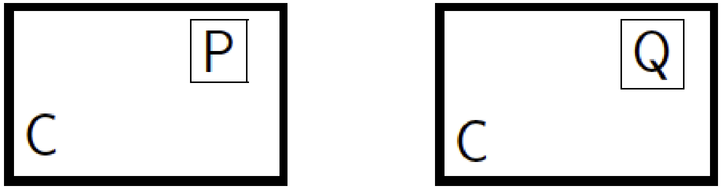
[1] Later, we will generalise this to a set of properties that can be expressed in a logic.

What is a Congruence?



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

What is a Congruence?

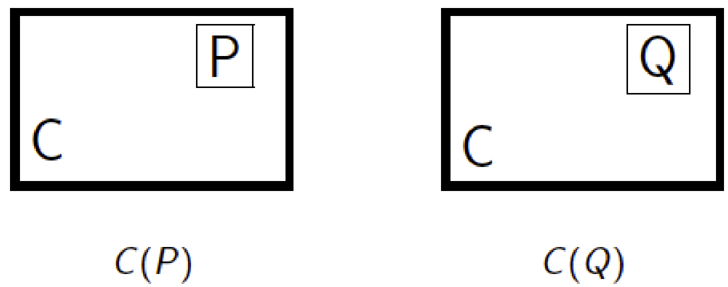


$C(P)$ $C(Q)$

CCS contexts informally

A **CCS context** is a CCS process fragment with a "hole" in it (examples on the board).

What is a Congruence?



CCS contexts informally

A CCS context is a CCS process fragment with a “hole” in it (examples on the board).

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

Example 11.4 (Congruence)

Let $a \equiv b$ for $a, b \in \mathbb{Z}$ whenever $a \textbf{ mod } k = b \textbf{ mod } k$, for some $k \in \mathbb{N}_+$.
Equivalence relation $\equiv$ is a congruence for addition and multiplication.

The Importance of Congruences

CCS congruences informally

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

Example 11.4 (Congruence)

Let $a \equiv b$ for $a, b \in \mathbb{Z}$ whenever $a \textbf{ mod } k = b \textbf{ mod } k$, for some $k \in \mathbb{N}_+$.
Equivalence relation $\equiv$ is a congruence for addition and multiplication.

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

1. Model-based development through refinement: replacing an abstract model *Spec* by a more detailed model *Impl*
2. Optimisation: replacing an implementation *Impl* by a more efficient implementation *Impl'*.

CCS Congruences Formally

Definition 11.5 (CCS congruence)

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

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

CCS Congruences Formally

Definition 11.5 (CCS congruence)

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

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

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

Deadlocks

Definition 11.6 (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.

Requirements on Behavioural Equivalences

Deadlocks

Definition 11.6 (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.

Example 11.7

$P = a.b.\text{nil} + a.\text{nil}$ 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 possible.

Deadlocks

Definition 11.6 (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.

Example 11.7

$P = a.b.\text{nil} + a.\text{nil}$ 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 possible.

Definition 11.8 (Deadlock sensitivity)

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

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

Outline of Lecture 11

Trace Equivalence

Trace language (Definition 3.2)

The trace language of $P \in Prc$ is defined by:

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

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Trace Equivalence

Trace language (Definition 3.2)

The trace language of $P \in Prc$ is defined by:

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

Trace equivalence (Definition 3.2)

$P, Q \in Prc$ are called trace equivalent iff $Tr(P) = Tr(Q)$.

Trace Equivalence

Trace language (Definition 3.2)

The trace language of $P \in Prc$ is defined by:

$$Tr(P) := \{w \in Act^* \mid \exists P' \in Prc.\, P \xrightarrow{w} P'\}.$$

Trace equivalence (Definition 3.2)

$P, Q \in Prc$ are called trace equivalent iff $Tr(P) = Tr(Q)$.

Trace equivalence is evidently an equivalence relation and is less discriminative than isomorphism.

Trace Equivalence is a Congruence

Theorem 11.9

Trace equivalence is a CCS congruence.

Trace Equivalence is a Congruence

Theorem 11.9

Trace equivalence is a CCS congruence.

Proof.

By structural induction over the syntax of CCS processes.

Trace Equivalence is a Congruence

Theorem 11.9

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 11.9

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 11.9

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 11.9

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 11.9

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.
Exercise: do the proof for $\|$. $\square$

Two Coffee/Tea Machines

Example 11.10

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 11.10

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

$$CTM = coin.\left(\overline{coffee}.CTM + \overline{tea}.CTM\right)$$

$$CTM' = coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM'.$$

Note the difference between the two processes. Nevertheless:

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

Two Coffee/Tea Machines

Example 11.10

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

$$\mathit{CTM} = \mathit{coin}.\left(\overline{\mathit{coffee}}.\mathit{CTM} + \overline{\mathit{tea}}.\mathit{CTM}\right)$$

$$\mathit{CTM'} = \mathit{coin}.\overline{\mathit{coffee}}.\mathit{CTM'} + \mathit{coin}.\overline{\mathit{tea}}.\mathit{CTM'}.$$

Note the difference between the two processes. Nevertheless:

$$\mathit{Tr}(\mathit{CTM}) = \mathit{Tr}(\mathit{CTM'}).$$

Are we satisfied?

Two Coffee/Tea Machines

Example 11.10

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

$$CTM = coin.\left(\overline{coffee}.CTM + \overline{tea}.CTM\right)$$

$$CTM' = coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM'.$$

Note the difference between the two processes. Nevertheless:

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

Are we satisfied? No, as *CTM* and *CTM'* differ in the context:

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

Two Coffee/Tea Machines

Example 11.10

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

$$CTM \;=\; coin.\left(\overline{coffee}.CTM + \overline{tea}.CTM\right)$$

$$CTM' \;=\; coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM'.$$

Note the difference between the two processes. Nevertheless:

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

Are we satisfied? No, as *CTM* and *CTM'* differ in the context:

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

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

Checking Trace Equivalence

Traces by automata

For finite-state P, the trace language $Tr(P)$ of process P is accepted by the (non-deterministic) finite automaton obtained from the LTS of P with initial state P and making all states accepting (final).

Checking Trace Equivalence

Traces by automata

For finite-state P, the trace language $Tr(P)$ of process P is accepted by the (non-deterministic) finite automaton obtained from the LTS of P with initial state P and making all states accepting (final).

Theorem 11.11

Checking trace equivalence of two finite processes is PSPACE-complete.

Checking Trace Equivalence

Traces by automata

For finite-state P, the trace language $Tr(P)$ of process P is accepted by the (non-deterministic) finite automaton obtained from the LTS of P with initial state P and making all states accepting (final).

Theorem 11.11

Checking trace equivalence of two finite processes is PSPACE-complete.

Proof.

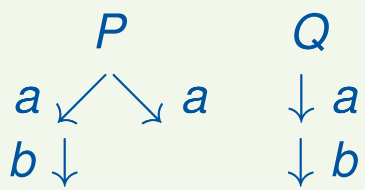
Checking whether $Tr(P) = Tr(Q)$, for finite-state P and Q, boils down to deciding whether their non-deterministic automata accept the same language. As this problem in automata theory is PSPACE-complete, it follows that checking $Tr(P) = Tr(Q)$ is PSPACE-complete. □

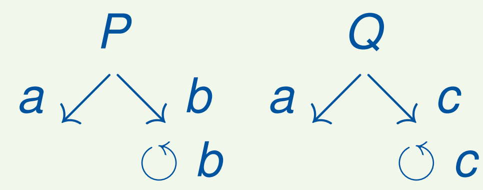
Trace Equivalence Revisited

Traces and Deadlocks

Example 11.12 (Traces and deadlocks)

Traces and deadlocks are independent in the following sense:





same traces
different deadlocks

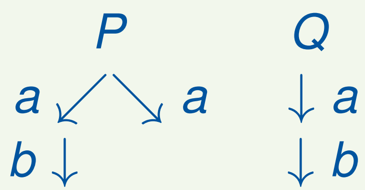
different traces
same deadlocks

Trace Equivalence Revisited

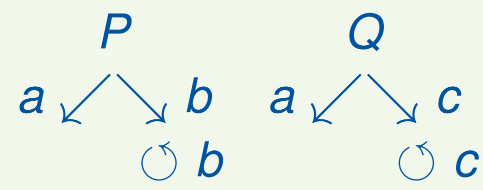
Traces and Deadlocks

Example 11.12 (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).

Outline of Lecture 11

Completed Trace Equivalence

Definition 11.13 (Completed traces)

A completed trace of $P \in Prc$ is a sequence $w \in Act^*$ such that:

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

for some $Q \in Prc$.

Completed Trace Equivalence

Definition 11.13 (Completed traces)

A completed trace of $P \in Prc$ is a sequence $w \in Act^*$ such that:

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

for some $Q \in Prc$.

The completed traces of process P may be seen as capturing its deadlock behaviour, as they are precisely the action sequences that can lead to a process from which no transition is possible (i.e., is a deadlock).

Completed Trace Equivalence

Definition 11.13 (Completed traces)

A completed trace of $P \in Prc$ is a sequence $w \in Act^*$ such that:

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

for some $Q \in Prc$.

The completed traces of process P may be seen as capturing its deadlock behaviour, as they are precisely the action sequences that can lead to a process from which no transition is possible (i.e., is a deadlock).

Exercise

Check whether completed trace equivalence is a congruence for restriction.

Further Variations of Trace Equivalence

Definition 11.14 (Ready trace equivalence) (Baeten et al.)

A sequence $A_0\alpha_0 A_1\alpha_1 \ldots \alpha_n A_n$ with $A_i \subseteq Act$ and $\alpha_i \in Act$ ($i \in \mathbb{N}$) is a ready trace of process P if $P = P_0 \xrightarrow{\alpha_0} P_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} P_n$ such that $A_i = \{\alpha \in Act \mid P_i \xrightarrow{\alpha}\}$. Processes P and Q are ready-trace equivalent if they have exactly the same set of ready traces.

Further Variations of Trace Equivalence

Definition 11.14 (Ready trace equivalence) (Baeten et al.)

A sequence $A_0\alpha_0A_1\alpha_1\ldots\alpha_nA_n$ with $A_i \subseteq Act$ and $\alpha_i \in Act$ ($i \in \mathbb{N}$) is a ready trace of process P if $P = P_0 \xrightarrow{\alpha_0} P_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} P_n$ such that $A_i = \{\alpha \in Act \mid P_i \xrightarrow{\alpha}\}$. Processes P and Q are ready-trace equivalent if they have exactly the same set of ready traces.

Definition 11.15 (Failure trace equivalence) (Reed and Roscoe)

A sequence $A_0\alpha_0A_1\alpha_1\ldots\alpha_nA_n$ with $A_i \subseteq Act$ and $\alpha_i \in Act$ ($i \in \mathbb{N}$) is a failure trace of process P if $P = P_0 \xrightarrow{\alpha_0} P_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} P_n$ such that $A_i \cap \{\alpha \in Act \mid P_i \xrightarrow{\alpha}\} = \emptyset$. Processes P and Q are failure-trace equivalent if they have exactly the same set of failure traces.

Further Variations of Trace Equivalence

Definition 11.14 (Ready trace equivalence) (Baeten et al.)

A sequence $A_0\alpha_0A_1\alpha_1\ldots\alpha_nA_n$ with $A_i \subseteq Act$ and $\alpha_i \in Act$ ($i \in \mathbb{N}$) is a ready trace of process P if $P = P_0 \xrightarrow{\alpha_0} P_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} P_n$ such that $A_i = \{\alpha \in Act \mid P_i \xrightarrow{\alpha}\}$. Processes P and Q are ready-trace equivalent if they have exactly the same set of ready traces.

Definition 11.15 (Failure trace equivalence) (Reed and Roscoe)

A sequence $A_0\alpha_0A_1\alpha_1\ldots\alpha_nA_n$ with $A_i \subseteq Act$ and $\alpha_i \in Act$ ($i \in \mathbb{N}$) is a failure trace of process P if $P = P_0 \xrightarrow{\alpha_0} P_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} P_n$ such that $A_i \cap \{\alpha \in Act \mid P_i \xrightarrow{\alpha}\} = \emptyset$. Processes P and Q are failure-trace equivalent if they have exactly the same set of failure traces.

Example 11.16

$P := a.b + a.c$ and $Q := a.(b + c)$ are

- trace equivalent: $Tr(P) = \{\varepsilon, a, ab, ac\} = Tr(Q)$, but
- not ready equivalent: $\{a\}\, a\, \{b, c\}\, b\, \emptyset \in rTr(Q) \setminus rTr(P)$

Outline of Lecture 11

Summary

1. Behavioural equivalences should be
   i. less distinctive than isomorphism
   ii. more distinctive than trace equivalence
   iii. a CCS congruence
   iv. deadlock sensitive

Summary

1. Behavioural equivalences should be
   i. less distinctive than isomorphism
   ii. more distinctive than trace equivalence
   iii. a CCS congruence
   iv. deadlock sensitive
2. Trace equivalence
   i. equates processes that have the same traces, i.e., action sequences
   ii. is implied by LTS isomorphism
   iii. is a CCS congruence
   iv. is not deadlock sensitive
   v. checking trace equivalence is PSPACE-complete

Summary

1. Behavioural equivalences should be
   i. less distinctive than isomorphism
   ii. more distinctive than trace equivalence
   iii. a CCS congruence
   iv. deadlock sensitive
2. Trace equivalence
   i. equates processes that have the same traces, i.e., action sequences
   ii. is implied by LTS isomorphism
   iii. is a CCS congruence
   iv. is not deadlock sensitive
   v. checking trace equivalence is PSPACE-complete
3. Variations: completed, ready, and failure traces