Concurrency Theory

Trace equivalence

Joost-Pieter Katoen and Thomas Noll

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

http://www-i2.informatik.rwth-aachen.de/i2/ct13

December 4, 2013

RWTH AACHEN UNIVERSITY

Overview

Overview

In using process algebra like CCS, an important approach is to model the specification and implementation as CCS processes, *Spec* and *Impl*, say.

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 equivalence have emerged.

Behavioural equivalence

Implementation

$$CM = coin.\overline{coffee}.CM$$
$$CS = \overline{pub}.\overline{coin}.coffee.CS$$
$$Uni = (CM \,||\, CS) \setminus \{\, coin, coffee \,\}$$

Specification

$Spec = \overline{pub}.Spec$

Question

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

$$Spec \;\equiv\; Uni?$$

Overview

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

The binary relation $\equiv \subseteq S \times S$ over the 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 $s, t, u, \in S$.

Isomorphism: an example behavioural equivalence

Isomorphism

The LTSs $TS_1 = (S_1, Act_1, \rightarrow_1)$ and $TS_2 = (S_2, Act_2, \rightarrow_2)$ are isomorphic, denoted $TS_1 \equiv_{iso} TS_2$, if there exists a bijection $f : S_1 \rightarrow S_2$ 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?

It follows $P + Q \equiv_{iso} Q + P$. The same applies to $P||Q$ and $Q||P$, as well as $P + \text{nil}$ and P. Also $(P + Q) + R \equiv_{iso} P + (Q + R)$, and similar for $||$.

Caveat

But: isomorphism is not very distinctive, e.g., $X = a.X$ and $Y = a.a.Y$ are distinguished, although both can (only) execute infinitely many a-actions and thus should be considered equivalent.

Isomorphism

Assumption

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

This means that $P + Q$ and $Q + P$ will not be distinguished. The same applies to $P||Q$ and $Q||P$, as well as $P + \text{nil}$ and P. But also $(P + Q) + R$ and $P + (Q + R)$, as well as $(P||Q)||R$ and $P||(Q||R)$ are not distinguished.

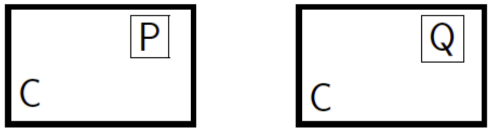
Overview

The wish-list for behavioural equivalences

1. Deadlock preservation: equivalent processes should have the same deadlock behaviour, i.e., equivalent process can either both deadlock, or both cannot.[1]
2. Less distinguishable than isomorphism: an equivalence should distinguish less processes than isomorphism does, i.e. $\equiv$ should be coarser than isomorphism.
3. Congruence property: the equivalence must be substitutive with respect to all CCS operators.
4. More distinguishable than trace equivalence: an equivalence should distinguish more processes than trace equivalence does, i.e. $\equiv$ should be finer than trace equivalence.
5. Optional: the coarsest possible equivalence: there should be no less discriminating equivalence satisfying all these requirements.

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

What is a congruence?



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

Context

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

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

The importance of a congruence

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

Example

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. Replacing an abstract model *Spec* by a more detailed one *Impl*.
2. Replacing a large (concrete) model *Impl* by a smaller (more abstract) model *Spec*.
3. Congruences admit a quotient structure with equivalence classes as elements.

CCS congruence

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:

$$\begin{array}{rcll} \alpha.P & \equiv & \alpha.Q & \text{for every } \alpha \in Act \\ P + R & \equiv & Q + R & \text{for every } R \in Prc \\ P \| R & \equiv & Q \| R & \text{for every } R \in Prc \\ P \backslash L & \equiv & Q \backslash L & \text{for every } L \subseteq A \\ P[f] & \equiv & Q[f] & \text{for every } f : Act \rightarrow Act^1 \end{array}$$

[1] satisfying $f(\tau) = \tau$ and $f(\overline{a}) = \overline{f(a)}$, i.e., f is a renaming function.

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

Deadlocks

Deadlock

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

$P = a.b.\text{nil} + a.\text{nil}$ has an a-deadlock, whereas $Q = a.b.\text{nil}$ has not.

Such properties are important, as it can be crucial that a certain communication is eventually possible.

Deadlock sensitivity

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

$$P \equiv Q \text{ implies } (\forall w.\, P \text{ has a } w\text{-deadlock iff } Q \text{ has a } w\text{-deadlock}).$$

Overview

Trace equivalence

Trace language

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

$P, Q \in Prc$ 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

Trace equivalence is a CCS congruence.

Proof.

By structural induction over the syntax of CCS processes. For $+$ this goes 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. As $P + R$ and $R + P$ are isomorphic, and we consider processes modulo isomorphism, this concludes the proof for $+$. For the other CCS constructs, the proof goes along similar lines. Exercise: do the proof for $||$. □

Two coffee machines

Consider the coffee 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.

It follows: $Tr(CTM) = Tr(CTM')$.

Are we satisfied? No! As *CTM* and *CTM'* <u>differ</u> in the context:

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

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

Checking trace equivalence

Traces by automata

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

Theorem

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

Proof.

Checking whether $Tr(P) = Tr(Q)$, for finite P and Q, thus boils down to check 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: summarizing

1. Trace equivalence equates processes that have the same traces, i.e., action sequences.
2. Trace equivalence is a CCS congruence
3. Trace equivalence trivially implies trace equivalence
4. Trace equivalence is not deadlock sensitive.
5. Checking trace equivalence is PSPACE-complete

Traces and deadlocks

Remark

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).

Overview

Completed trace equivalence

Completed trace

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

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

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 could lead to a process from which no transition is possible (i.e., is a deadlock).

Exercise

Check that $C(CTM)$ and $C(CTM')$ have the same completed traces.

Exercise

Check whether completed trace equivalence is a congruence for restriction.

Variations of trace equivalence

Ready trace equivalence **[Baeten et al.]**

A sequence $A_0\alpha_1 A_1\alpha_1 \ldots \alpha_n A_n$ 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.

Failure trace equivalence **[Reed and Roscoe]**

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

$\alpha.P + \alpha.Q$ and $\alpha.P + \alpha.Q + \alpha.(P + Q)$ are failure trace equivalent for every $P, Q \in Prc$ and $\alpha \in Act$.

Overview

Summary

1. Behavioural equivalences should be:
   - 1.1 deadlock sensitive
   - 1.2 a congruence (for CCS)
   - 1.3 more discriminative than trace equivalence
2. Trace equivalence
   - 2.1 equates processes that have the same traces, i.e., action sequences.
   - 2.2 is a CCS congruence
   - 2.3 is not deadlock sensitive.
   - 2.4 checking trace equivalence is PSPACE-complete
3. Variations: completed, ready, and failure traces.