Modeling Concurrent and Probabilistic Systems

Lecture 9: Decidability of Observation Congruence

Joost-Pieter Katoen Thomas Noll

Software Modeling and Verification Group
RWTH Aachen University
noll@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/mcps07/

Winter Semester 2007/08

Outline

Repetition: Definition of Obs. 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 $Q \stackrel{\alpha}{\Longrightarrow} Q'$ and $P' \approx Q'$
2. $Q \xrightarrow{\alpha} Q' \implies$ ex. $P' \in Prc$ such that $P \stackrel{\alpha}{\Longrightarrow} P'$ and $P' \approx Q'$

Remark: $\simeq$ differs from $\approx$ only in the use of $\stackrel{\alpha}{\Longrightarrow}$ rather than $\stackrel{\hat{\alpha}}{\Longrightarrow}$, 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 $P' \simeq Q'$).

Repetition: Properties of Obs. Congruence

Properties

1. $$\begin{aligned} & LTS(P) = LTS(Q) \\ \Longrightarrow\ & P \sim Q \\ \Longrightarrow\ & P \simeq Q \\ \Longrightarrow\ & P \approx Q \\ \Longrightarrow\ & \hat{Tr}(P) = \hat{Tr}(Q) \end{aligned}$$
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 $P, Q \in Prc$,
$$P \approx Q \iff P \simeq Q \text{ or } P \simeq \tau.Q \text{ or } \tau.P \simeq Q$$

Outline

The Problem

We now show that the word problem for observation equivalence

Problem

Given: $P, Q \in Prc$

Question: $P \simeq Q$?

is decidable for finite–state processes (i.e., for those with $|S(P)|, |S(Q)| < \infty$ where $S(P) := \{P' \in Prc \mid P \longrightarrow^* P'\}$)

(in general it is undecidable).

Since the definition of $\simeq$ directly relies on $\approx$ (cf. Definitions 7.6 and 8.2), we first extend the partitioning algorithm from $\sim$ (Theorem 6.2) to $\approx$.

The Problem

We now show that the word problem for observation equivalence

Problem

Given: $P, Q \in Prc$

Question: $P \simeq Q$?

is decidable for finite–state processes (i.e., for those with $|S(P)|, |S(Q)| < \infty$ where $S(P) := \{P' \in Prc \mid P \longrightarrow^* P'\}$)

(in general it is undecidable).

Since the definition of $\simeq$ directly relies on $\approx$ (cf. Definitions 7.6 and 8.2), we first extend the partitioning algorithm from $\sim$ (Theorem 6.2) to $\approx$.

The Partitioning Algorithm I

Theorem 9.1 (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*
$$\alpha(P) := \{C \in \Pi \mid ex.\ P' \in C \text{ with } P \xrightarrow{\alpha} P'\}$$
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 $P, Q \in S$*,*
$$P \sim Q \iff ex.\ B \in \hat{\Pi} \text{ with } P, Q \in B$$

The Partitioning Algorithm I

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*
$$\alpha^*(P) := \{C \in \Pi \mid \textit{ex. } P' \in C \textit{ with } P \stackrel{\hat{\alpha}}{\Longrightarrow} P'\}$$
be the set of P*'s* α*–successor blocks*
4. *Partition* $B = \sum_{i=1}^{k} B_i$ *such that*
$$P, Q \in B_i \iff \alpha^*(P) = \alpha^*(Q) \textit{ for every } \alpha \in Act$$
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 $P, Q \in S$*,*
$$P \approx Q \iff \textit{ex. } B \in \hat{\Pi} \textit{ with } P, Q \in B$$

The Partitioning Algorithm II

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:

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

Proof.

similar to Theorem 6.2 ($\sim$–partitioning algorithm) □

The Partitioning Algorithm II

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:

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

Proof.

similar to Theorem 6.2 ($\sim$–partitioning algorithm) □

Decidability of Observation Congruence

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 $P, Q \in S$,

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

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

Proof.

omitted □

Example 9.3

on the board

Decidability of Observation Congruence

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 $P, Q \in S$,

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

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

Proof.

omitted □

Example 9.3

on the board

Decidability of Observation Congruence

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 $P, Q \in S$,

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

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

Proof.

omitted $\square$

Example 9.3

on the board

Decidability of Observation Congruence

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 $P, Q \in S$,

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

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

Proof.

omitted $\square$

Example 9.3

on the board

Outline

1. Repetition: Observation Congruence
2. Decidability of Observation Congruence
3. The Alternating Bit Protocol

The Setting

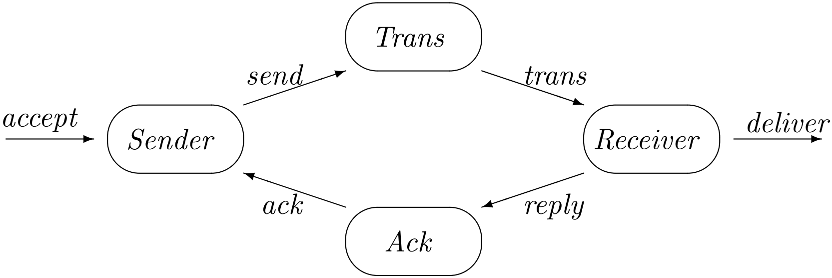
Goal: design of a communication protocol which guarantees reliable data transfer over unreliable channels

Overview:

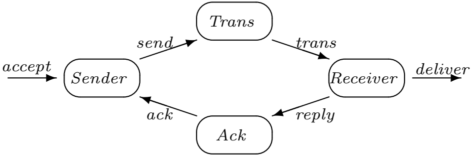
The Setting

Goal: design of a communication protocol which guarantees reliable data transfer over unreliable channels

Overview:

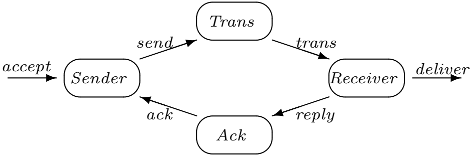


Working Principle



- *Sender* transfers data (from a given finite set D) to *Receiver* using channel *Trans*
- *Receiver* confirms reception via *Ack*
- Properties of channels:
  - unidirectional data transfer
  - capacity: one message
    ($\Longrightarrow$ sequential, i.e., respects order of messages)
  - detection of transmission errors
    (loss/duplication/corruption of messages)
  - errors reported to target process

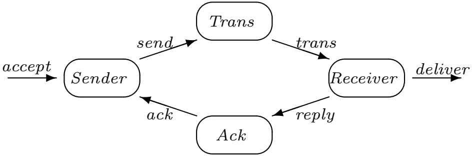
Idea: use redundancy (additional control bit) to ensure safeness of data transfer



- *Sender* transfers data (from a given finite set D) to *Receiver* using channel *Trans*
- *Receiver* confirms reception via *Ack*
- Properties of channels:
  - unidirectional data transfer
  - capacity: one message
    ($\Longrightarrow$ sequential, i.e., respects order of messages)
  - detection of transmission errors
    (loss/duplication/corruption of messages)
  - errors reported to target process

Idea: use redundancy (additional control bit) to ensure safeness of data transfer

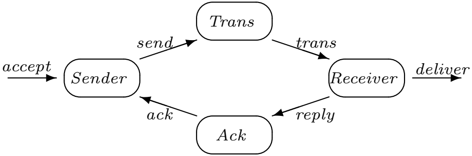
Working Principle



- *Sender* transfers data (from a given finite set D) to *Receiver* using channel *Trans*
- *Receiver* confirms reception via *Ack*
- Properties of channels:
  - unidirectional data transfer
  - capacity: one message
    ($\Longrightarrow$ sequential, i.e., respects order of messages)
  - detection of transmission errors
    (loss/duplication/corruption of messages)
  - errors reported to target process

Idea: use redundancy (additional control bit) to ensure safeness of data transfer

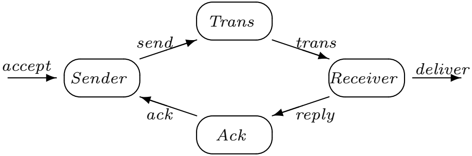
Working Principle



- *Sender* transfers data (from a given finite set D) to *Receiver* using channel *Trans*
- *Receiver* confirms reception via *Ack*
- Properties of channels:
  - unidirectional data transfer
  - capacity: one message
    ($\Longrightarrow$ sequential, i.e., respects order of messages)
  - detection of transmission errors
    (loss/duplication/corruption of messages)
  - errors reported to target process

Idea: use redundancy (additional control bit) to ensure safeness of data transfer

Modelling of Channels



- *Trans* transmits frames of the following form:

$$F := \{db \mid d \in D, b \in \{0,1\}\} \qquad \text{(finite)}$$

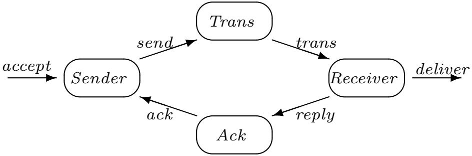
It detects transmission errors and reports it to *Receiver*:

$$\mathit{Trans} = \sum_{f \in F} \mathit{send}_f.(\underbrace{\overline{\mathit{trans}_f}.\mathit{Trans}}_{\text{successful}} + \underbrace{\overline{\mathit{trans}_\perp}.\mathit{Trans}}_{\text{error}})$$

- *Ack* behaves like *Trans* but transmits only control bits:

$$\mathit{Ack} = \sum_{b \in \{0,1\}} \mathit{reply}_b.(\underbrace{\overline{\mathit{ack}_b}.\mathit{Ack}}_{\text{successful}} + \underbrace{\overline{\mathit{ack}_\perp}.\mathit{Ack}}_{\text{error}})$$

Modelling of Channels



- *Trans* transmits frames of the following form:

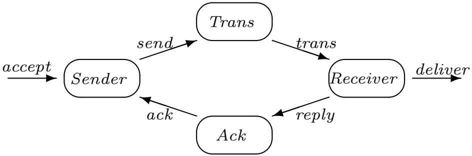
$$F := \{db \mid d \in D, b \in \{0,1\}\} \qquad \text{(finite)}$$

It detects transmission errors and reports it to *Receiver*:

$$\mathit{Trans} = \sum_{f \in F} \mathit{send}_f.(\underbrace{\overline{\mathit{trans}_f}.\mathit{Trans}}_{\text{successful}} + \underbrace{\overline{\mathit{trans}_\perp}.\mathit{Trans}}_{\text{error}})$$

- *Ack* behaves like *Trans* but transmits only control bits:

$$\mathit{Ack} = \sum_{b \in \{0,1\}} \mathit{reply}_b.(\underbrace{\overline{\mathit{ack}_b}.\mathit{Ack}}_{\text{successful}} + \underbrace{\overline{\mathit{ack}_\perp}.\mathit{Ack}}_{\text{error}})$$



Under the above side conditions, give CCS implementations of *Sender* and *Receiver* such that the overall system works correctly, i.e., behaves like a one–element buffer:

$$
\begin{aligned}
Buffer(\overrightarrow{accept}, \overrightarrow{deliver}) &= \sum_{d \in D} accept_d.Buffer_d(\overrightarrow{accept}, \overrightarrow{deliver}) \\
Buffer_d(\overrightarrow{accept}, \overrightarrow{deliver}) &= \overline{deliver_d}.Buffer(\overrightarrow{accept}, \overrightarrow{deliver})
\end{aligned}
$$

where

$$
\begin{aligned}
\overrightarrow{accept} &:= (accept_{d_1}, \ldots, accept_{d_n}) \\
\text{and } \overrightarrow{deliver} &:= (deliver_{d_1}, \ldots, deliver_{d_n}) \\
\text{for } D &= \{d_1, \ldots, d_n\}
\end{aligned}
$$