

# Modeling Concurrent and Probabilistic Systems

## Lecture 7: Decidability of Strong Bisimulation & Definition of Strong Simulation

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/mcps09/>

Summer Semester 2009

- 1 Repetition: Strong Bisimulation
- 2 Decidability of Strong Bisimulation
- 3 Summary: Properties of Strong Bisimulation
- 4 Strong Simulation

# Repetition: Definition of Strong Bisimulation

## Definition (Strong bisimulation)

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

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

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

## Theorem

$\sim$  is an equivalence relation.

## Proof.

on the board



# Congruence Property of Strong Bisimulation

## Definition (CCS congruence)

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

$$\alpha.P \cong \alpha.Q$$

$$P + R \cong Q + R$$

$$R + P \cong R + Q$$

$$P \parallel R \cong Q \parallel R$$

$$R \parallel P \cong R \parallel Q$$

$$\text{new } a P \cong \text{new } a Q$$

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

## Theorem 7.3

$\sim$  is a CCS congruence.

## Proof.

on the board □

# Deadlock Sensitivity of Strong Bisimulation

## Definition (Deadlock)

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

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

## Theorem 7.4

$\sim$  is deadlock sensitive.

Proof.

on the board □

- 1 Repetition: Strong Bisimulation
- 2 Decidability of Strong Bisimulation
- 3 Summary: Properties of Strong Bisimulation
- 4 Strong Simulation

# The Problem

We now show that the **word problem for strong bisimulation**

Problem (Word problem for strong bisimulation)

Given:  $P, Q \in Proc$

Question:  $P \sim Q?$

is **decidable for finite-state processes** (i.e., for those with  $|S(P)|, |S(Q)| < \infty$  where  $S(P) := \{P' \in Proc \mid P \longrightarrow P'\}$ )  
(in general it is undecidable – see 4th ex. sheet).

To this aim we give an algorithm which **iteratively partitions** the state set of an LTS such that the single blocks correspond to the  $\sim$ -equivalence classes.

# The Problem

We now show that the **word problem for strong bisimulation**

Problem (Word problem for strong bisimulation)

Given:  $P, Q \in Proc$

Question:  $P \sim Q?$

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

(in general it is undecidable – see 4th ex. sheet).

To this aim we give an algorithm which **iteratively partitions** the state set of an LTS such that the single blocks correspond to the  $\sim$ -equivalence classes.

# The Partitioning Algorithm I

## Theorem 7.1 (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
$$\alpha(P) := \{C \in \Pi \mid \text{ex. } P' \in C \text{ with } P \xrightarrow{\alpha} P'\}$$
be the set of  $P$ 's  $\alpha$ -successor blocks
  - 4 Partition  $B = \bigcup_{i=1}^k B_i$  such that
$$P, Q \in B_i \iff \alpha(P) = \alpha(Q) \text{ for every } \alpha \in Act$$
  - 5 Let  $\Pi := (\Pi \setminus \{B\}) \cup \{B_1, \dots, B_k\}$
  - 6 Continue with (2) until  $\Pi$  becomes stable

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

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

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

# The Partitioning Algorithm I

## Theorem 7.1 (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

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

be the set of  $P$ 's  $\alpha$ -successor blocks

④ Partition  $B = \bigcup_{i=1}^k B_i$  such that

$$P, Q \in B_i \iff \alpha(P) = \alpha(Q) \text{ for every } \alpha \in Act$$

⑤ Let  $\Pi := (\Pi \setminus \{B\}) \cup \{B_1, \dots, B_k\}$

⑥ Continue with (2) until  $\Pi$  becomes stable

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

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

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

# The Partitioning Algorithm I

## Theorem 7.1 (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
    - 4  $\alpha(P) := \{C \in \Pi \mid \text{ex. } P' \in C \text{ with } P \xrightarrow{\alpha} P'\}$   
be the set of  $P$ 's  $\alpha$ -successor blocks
    - 5 Partition  $B = \bigcup_{i=1}^k B_i$  such that
$$P, Q \in B_i \iff \alpha(P) = \alpha(Q) \text{ for every } \alpha \in Act$$
    - 6 Let  $\Pi := (\Pi \setminus \{B\}) \cup \{B_1, \dots, B_k\}$
    - 7 Continue with (2) until  $\Pi$  becomes stable

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

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

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

# The Partitioning Algorithm I

## Theorem 7.1 (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
$$\alpha(P) := \{C \in \Pi \mid \text{ex. } P' \in C \text{ with } P \xrightarrow{\alpha} P'\}$$
be the set of  $P$ 's  $\alpha$ -successor blocks
  - 4 Partition  $B = \bigcup_{i=1}^k B_i$  such that
$$P, Q \in B_i \iff \alpha(P) = \alpha(Q) \text{ for every } \alpha \in Act$$
  - 5 Let  $\Pi := (\Pi \setminus \{B\}) \cup \{B_1, \dots, B_k\}$
  - 6 Continue with (2) until  $\Pi$  becomes stable

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

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

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

# The Partitioning Algorithm I

## Theorem 7.1 (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
$$\alpha(P) := \{C \in \Pi \mid \text{ex. } P' \in C \text{ with } P \xrightarrow{\alpha} P'\}$$
be the set of  $P$ 's  $\alpha$ -successor blocks
  - 4 Partition  $B = \bigcup_{i=1}^k B_i$  such that
$$P, Q \in B_i \iff \alpha(P) = \alpha(Q) \text{ for every } \alpha \in Act$$
  - 5 Let  $\Pi := (\Pi \setminus \{B\}) \cup \{B_1, \dots, B_k\}$
  - 6 Continue with (2) until  $\Pi$  becomes stable

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

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

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

# The Partitioning Algorithm I

## Theorem 7.1 (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
$$\alpha(P) := \{C \in \Pi \mid \text{ex. } P' \in C \text{ with } P \xrightarrow{\alpha} P'\}$$
be the set of  $P$ 's  $\alpha$ -successor blocks
  - 4 Partition  $B = \bigcup_{i=1}^k B_i$  such that
$$P, Q \in B_i \iff \alpha(P) = \alpha(Q) \text{ for every } \alpha \in Act$$
  - 5 Let  $\Pi := (\Pi \setminus \{B\}) \cup \{B_1, \dots, B_k\}$
  - 6 Continue with (2) until  $\Pi$  becomes stable

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

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

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

# The Partitioning Algorithm I

## Theorem 7.1 (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
$$\alpha(P) := \{C \in \Pi \mid \text{ex. } P' \in C \text{ with } P \xrightarrow{\alpha} P'\}$$
be the set of  $P$ 's  $\alpha$ -successor blocks
  - 4 Partition  $B = \bigcup_{i=1}^k B_i$  such that
$$P, Q \in B_i \iff \alpha(P) = \alpha(Q) \text{ for every } \alpha \in Act$$
  - 5 Let  $\Pi := (\Pi \setminus \{B\}) \cup \{B_1, \dots, B_k\}$
  - 6 Continue with (2) until  $\Pi$  becomes stable

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

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

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

# The Partitioning Algorithm I

## Theorem 7.1 (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
$$\alpha(P) := \{C \in \Pi \mid \text{ex. } P' \in C \text{ with } P \xrightarrow{\alpha} P'\}$$
be the set of  $P$ 's  $\alpha$ -successor blocks
  - 4 Partition  $B = \bigcup_{i=1}^k B_i$  such that
$$P, Q \in B_i \iff \alpha(P) = \alpha(Q) \text{ for every } \alpha \in Act$$
  - 5 Let  $\Pi := (\Pi \setminus \{B\}) \cup \{B_1, \dots, B_k\}$
  - 6 Continue with (2) until  $\Pi$  becomes stable

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

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

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

# The Partitioning Algorithm I

## Theorem 7.1 (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
$$\alpha(P) := \{C \in \Pi \mid \text{ex. } P' \in C \text{ with } P \xrightarrow{\alpha} P'\}$$
be the set of  $P$ 's  $\alpha$ -successor blocks
  - 4 Partition  $B = \bigcup_{i=1}^k B_i$  such that
$$P, Q \in B_i \iff \alpha(P) = \alpha(Q) \text{ for every } \alpha \in Act$$
  - 5 Let  $\Pi := (\Pi \setminus \{B\}) \cup \{B_1, \dots, B_k\}$
  - 6 Continue with (2) until  $\Pi$  becomes stable

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

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

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

# The Partitioning Algorithm II

**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  $Act_1 = Act_2$ )

Example 7.2

Binary semaphore (on the board)

Proof.

(Theorem 7.1; on the board) □

**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  $Act_1 = Act_2$ )

## Example 7.2

Binary semaphore (on the board)

Proof.

(Theorem 7.1; on the board) □

**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  $Act_1 = Act_2$ )

## Example 7.2

Binary semaphore (on the board)

Proof.

(Theorem 7.1; on the board) □

- 1 Repetition: Strong Bisimulation
- 2 Decidability of Strong Bisimulation
- 3 Summary: Properties of Strong Bisimulation**
- 4 Strong Simulation

## Properties of strong bisimulation

- 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
- 6  $\sim$  is decidable for finite-state processes

- 1 Repetition: Strong Bisimulation
- 2 Decidability of Strong Bisimulation
- 3 Summary: Properties of Strong Bisimulation
- 4 Strong Simulation

# Strong Simulation

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

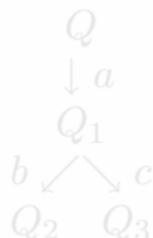
## Definition 7.3 (Strong simulation)

A relation  $\rho \subseteq Proc \times Proc$  is called a **strong simulation** if, whenever  $P \rho Q$  and  $P \xrightarrow{\alpha} P'$ , there exists  $Q' \in Proc$  such that  $Q \xrightarrow{\alpha} Q'$  and  $P' \rho Q'$ .

We say that  $Q$  **strongly simulates**  $P$  if there exists a strong simulation  $\rho$  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 7.4



$Q$  strongly simulates  $P$ ,  
but not vice versa

# Strong Simulation

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

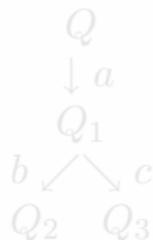
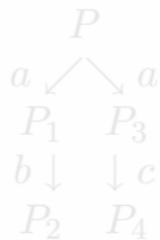
## Definition 7.3 (Strong simulation)

A relation  $\rho \subseteq Proc \times Proc$  is called a **strong simulation** if, whenever  $P \rho Q$  and  $P \xrightarrow{\alpha} P'$ , there exists  $Q' \in Proc$  such that  $Q \xrightarrow{\alpha} Q'$  and  $P' \rho Q'$ .

We say that  $Q$  **strongly simulates**  $P$  if there exists a strong simulation  $\rho$  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 7.4



$Q$  strongly simulates  $P$ ,  
but not vice versa

# Strong Simulation

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

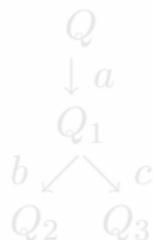
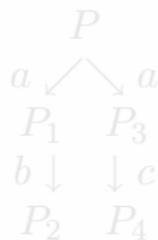
## Definition 7.3 (Strong simulation)

A relation  $\rho \subseteq Proc \times Proc$  is called a **strong simulation** if, whenever  $P \rho Q$  and  $P \xrightarrow{\alpha} P'$ , there exists  $Q' \in Proc$  such that  $Q \xrightarrow{\alpha} Q'$  and  $P' \rho Q'$ .

We say that  $Q$  **strongly simulates**  $P$  if there exists a strong simulation  $\rho$  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 7.4



$Q$  strongly simulates  $P$ ,  
but not vice versa

# Strong Simulation

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

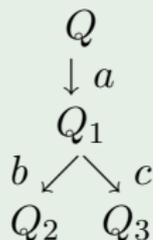
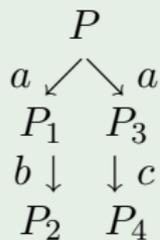
## Definition 7.3 (Strong simulation)

A relation  $\rho \subseteq Proc \times Proc$  is called a **strong simulation** if, whenever  $P \rho Q$  and  $P \xrightarrow{\alpha} P'$ , there exists  $Q' \in Proc$  such that  $Q \xrightarrow{\alpha} Q'$  and  $P' \rho Q'$ .

We say that  $Q$  **strongly simulates**  $P$  if there exists a strong simulation  $\rho$  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 7.4



$Q$  strongly simulates  $P$ ,  
but not vice versa

# Strong Simulation and Bisimulation

## Corollary 7.5

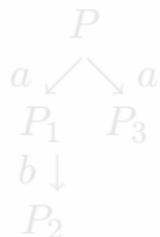
If  $P \sim Q$ , then  $Q$  strongly simulates  $P$ , and  $P$  strongly simulates  $Q$ .

Proof.

A strong bisimulation  $\rho \subseteq \text{Proc} \times \text{Proc}$  for  $P \sim Q$  is a strong simulation for both directions.  $\square$

**Caveat:** the converse does generally not hold!

## Example 7.6



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

# Strong Simulation and Bisimulation

## Corollary 7.5

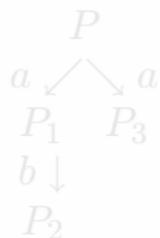
If  $P \sim Q$ , then  $Q$  strongly simulates  $P$ , and  $P$  strongly simulates  $Q$ .

## Proof.

A strong bisimulation  $\rho \subseteq \text{Proc} \times \text{Proc}$  for  $P \sim Q$  is a strong simulation for both directions. □

**Caveat:** the converse does generally not hold!

## Example 7.6



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

# Strong Simulation and Bisimulation

## Corollary 7.5

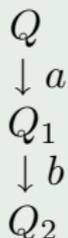
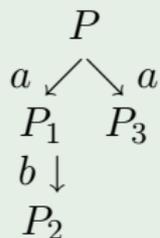
If  $P \sim Q$ , then  $Q$  strongly simulates  $P$ , and  $P$  strongly simulates  $Q$ .

## Proof.

A strong bisimulation  $\rho \subseteq \text{Proc} \times \text{Proc}$  for  $P \sim Q$  is a strong simulation for both directions.  $\square$

**Caveat:** the converse does generally not hold!

## Example 7.6



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