# Semantics and Verification of Software

## Lecture 20: Nondeterminism and Parallelism V (Wrap-Up)


Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

noll@cs.rwth-aachen.de


http://www-i2.informatik.rwth-aachen.de/i2/svsw13/

Summer Semester 2013

# Online Registration for Seminars and Practical Courses (Praktika) in Winter Term 2013/14

## Who?

Students of:
- Master Courses
- Bachelor Informatik (~~Pro~~Seminar!)

## Where?

www.graphics.rwth-aachen.de/apse

## When?

05.07.2013 - 17.07.2013

# Outline

# Semantics of CCS I

## Definition (Semantics of CCS)

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

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

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

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

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

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

# Definition of Strong Bisimulation

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

## Theorem

*$\sim$ is an equivalence relation.*

## Proof.

omitted $\square$

# Outline

# The Problem

We now show that the word problem for strong bisimulation

**Problem (Word problem for strong bisimulation)**

Given: $P, Q \in Prc$

Question: $P \sim Q$?

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

(in general it is undecidable).

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 20.1 (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* $\Pi := (\Pi \setminus \{B\}) \cup \{B_1, \ldots, B_k\}$
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 20.2**

Binary semaphore (on the board)

# Outline

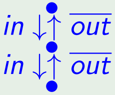
**Observation:** requirement of exact matching sometimes too strong

## Example 20.3

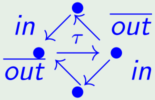
Sequential and parallel two-place buffer:

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

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




$\not\sim$



# Definition of Weak Bisimulation I

**Idea:** abstract from silent actions

## Definition 20.4

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

$$P \overset{w}{\Longrightarrow} Q \iff P \, (\overset{\tau}{\longrightarrow})^* \overset{\alpha_1}{\longrightarrow} (\overset{\tau}{\longrightarrow})^* \ldots (\overset{\tau}{\longrightarrow})^* \overset{\alpha_n}{\longrightarrow} (\overset{\tau}{\longrightarrow})^* \, Q$$

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

# Definition of Weak Bisimulation II

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

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

**Example 20.5**

Sequential and parallel two-place buffer (on the board)

# Outline

# Summary: Nondeterminism and Concurrency

- Requires precise formal description of parallelism and interaction
- Classical "Input $\rightarrow$ Output" view not sufficient (non-terminating/reactive behaviour)
- Parallelism = nondeterminism + sequential execution (interleaving)
  - alternative approach: "true" concurrency (Petri nets, event structures, ...)
- Interaction:
  - shared variables (ParWHILE)
  - value-passing channels (CSP)
  - synchronous handshaking (CCS)
- Requires new notions of program/process equivalence (bisimulation)

# Outline

# Semantics of Functional Languages I

- Program = list of function definitions
- Simplest setting: first-order function definitions of the form

$$f(x_1, \ldots, x_n) = t$$

  - function name $f$
  - formal parameters $x_1, \ldots, x_n$
  - term $t$ over (base and defined) function calls and $x_1, \ldots, x_n$
- Operational semantics (only function calls)
  - call-by-value case:

$$\frac{t_1 \to z_1 \quad \ldots \quad t_n \to z_n \quad t[x_1 \mapsto z_1, \ldots, x_n \mapsto z_n] \to z}{f(t_1, \ldots, t_n) \to z}$$

  - call-by-name case:

$$\frac{t[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n] \to z}{f(t_1, \ldots, t_n) \to z}$$

# Semantics of Functional Languages II

- Denotational semantics
  - program = equation system (for functions)
  - induces call-by-value and call-by-name functional
  - monotonic and continuous w.r.t. graph inclusion
  - semantics := least fixpoint (Tarski/Knaster Theorem)
  - coincides with operational semantics
- Extensions: higher-order types, data types, ...
- see [Winskel 1996, Sct. 9] and Functional Programming course [Giesl]

# Outline

# Miscellaneous

- Remaining lectures:
  - Wed 17 July: recap?
  - Thu 18 July: exercise class
- Oral exams:
  - Mon 22 July – Fri 26 July
  - Thu 15 August – Wed 21 August
  - Wed 4 September – Fri 11 October

  Just drop me a mail!
- Teaching in Winter 2013/14:
  - Course Introduction to Model Checking [Katoen]
  - Course Concurrency Theory [Katoen/Noll]
  - Seminar Trends in Computer-Aided Verification [Katoen/Noll/NN]