



Semantics and Verification of Software

Winter Semester 2017/18

Lecture 16: Nondeterminism and Parallelism I
(Shared-Variables Communication)

Thomas Noll

Software Modeling and Verification Group

RWTH Aachen University

<http://moves.rwth-aachen.de/teaching/ws-1718/sv-sw/>

Oral Exam in Semantics and Verification

- Exam regulations require **admission criteria** (“ \geq 50% of points in exercises”) to be fixed in module specification
 - **Not** specified for *Semantics and Verification* (or any other MSc lecture in CS...)
- ⇒ **All registered participants are admitted**
- Information of **inactive registered students**
 - **Exam dates** will be announced in beginning of 2018

Motivation

- Essential question: what is the meaning of

$$c_1 \parallel c_2$$

(parallel execution of $c_1, c_2 \in \text{Cmd}$)?

- Easy to answer when state spaces are disjoint:

$$\langle x := 1 \parallel y := 2, \sigma \rangle \rightarrow \sigma[x \mapsto 1, y \mapsto 2]$$

(no interaction \Rightarrow corresponds to sequential execution in any order)

- But what if variables are shared?

$$(x := 1 \parallel x := 2); \text{if } x = 1 \text{ then } c_1 \text{ else } c_2 \text{ end}$$

(runs c_1 or c_2 depending on execution order of initial assignments)

- Even more involved for non-atomic assignments...

Introduction

Non-Atomic Assignments

Observation: **parallelism** introduces new phenomena

Example 16.1

$$\left(\begin{array}{c} x := 0; \\ x := x + 1 \\ 13 \end{array} \parallel \begin{array}{c} x := x + 2 \\ 2 \end{array} \right) \quad \text{value of } x: 0123$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2, 1, or 3
- If **exclusive (write) access** to shared memory and **atomic execution** of assignments guaranteed
⇒ only possible outcome: 3

Introduction

Parallelism and Interaction

The problem arises due to the combination of

- **parallelism** and
- **interaction** (here: via shared memory)

Conclusion

When defining the semantics of parallel systems, the precise description of the mechanisms of both **parallelism** and **interaction** is crucially important.

Reactive Systems

- Thus: “classical” model for sequential systems

System : Input \rightarrow Output

(**transformational systems**) is not adequate

- Missing: aspect of **interaction**
- Rather: **reactive systems** which interact with environment and among themselves
- Main interest: not terminating computations but **infinite behaviour**
(system maintains ongoing interaction with environment)
- Examples:
 - operating systems
 - embedded systems controlling mechanical or electrical devices
(planes, cars, home appliances, ...)
 - power plants, production lines, ...

Introduction

Overview

Here: study of parallelism in connection with two different kinds of interaction

1. Shared-variables communication (ParWHILE)
2. Channel communication (CSP)

Essential principle:

- Reduction of parallelism to **nondeterminism + sequential execution**
(similar to multitasking on sequential computers)

Preparatory step:

- Semantic description of **nondeterminism** (NdWHILE)

Nondeterminism

The NdWHILE Language

Definition 16.2 (Syntax of NdWHILE)

$$\begin{aligned} a &::= z \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \in AExp \\ b &::= t \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in BExp \\ c &::= \text{skip} \mid x := a \mid c_1 ; c_2 \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end} \mid \text{while } b \text{ do } c \text{ end} \mid \\ &\quad c_1 \square c_2 \in Cmd \end{aligned}$$

Here, $c_1 \square c_2$ stands for the **nondeterministic** choice between statements c_1 and c_2 .

Nondeterminism

Big-Step Semantics

Definition 16.3 (Big-step execution relation for NdWHILE)

For $c \in \text{Cmd}$ and $\sigma, \sigma' \in \Sigma$, the **execution relation** $\langle c, \sigma \rangle \rightarrow \sigma'$ is defined by:

$$\begin{array}{c} \text{(skip)} \frac{}{\langle \text{skip}, \sigma \rangle \rightarrow \sigma} \\ \text{(seq)} \frac{\langle c_1, \sigma \rangle \rightarrow \sigma' \quad \langle c_2, \sigma' \rangle \rightarrow \sigma''}{\langle c_1; c_2, \sigma \rangle \rightarrow \sigma''} \\ \text{(if-f)} \frac{\langle b, \sigma \rangle \rightarrow \text{false} \quad \langle c_2, \sigma \rangle \rightarrow \sigma'}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, \sigma \rangle \rightarrow \sigma'} \\ \text{(if-t)} \frac{\langle b, \sigma \rangle \rightarrow \text{true} \quad \langle c_1, \sigma \rangle \rightarrow \sigma'}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, \sigma \rangle \rightarrow \sigma'} \\ \text{(wh-f)} \frac{\langle b, \sigma \rangle \rightarrow \text{false}}{\langle \text{while } b \text{ do } c \text{ end}, \sigma \rangle \rightarrow \sigma} \\ \text{(wh-t)} \frac{\langle b, \sigma \rangle \rightarrow \text{true} \quad \langle c, \sigma \rangle \rightarrow \sigma' \quad \langle \text{while } b \text{ do } c \text{ end}, \sigma' \rangle \rightarrow \sigma''}{\langle \text{while } b \text{ do } c \text{ end}, \sigma \rangle \rightarrow \sigma''} \\ \text{(alt-1)} \frac{\langle c_1, \sigma \rangle \rightarrow \sigma'}{\langle c_1 \square c_2, \sigma \rangle \rightarrow \sigma'} \\ \text{(alt-2)} \frac{\langle c_2, \sigma \rangle \rightarrow \sigma'}{\langle c_1 \square c_2, \sigma \rangle \rightarrow \sigma'} \end{array}$$

Small-Step Semantics I

- Computations in **big-step style** do not involve any intermediate configurations:

$$\langle c, \sigma \rangle \rightarrow \sigma'$$

- only initial and final states
- works for constructs whose computations are pure evaluation, without side-effects, no exceptions, and “usually” terminating
- **Small-step semantics** introduce explicit representation of intermediate configurations:

$$\langle c, \sigma \rangle \rightarrow_1 \langle c_1, \sigma_1 \rangle \rightarrow_1 \langle c_2, \sigma_2 \rangle \rightarrow_1 \dots \rightarrow_1 \sigma_n$$

- specifies order of computation steps (which can be significant)
- required for interleaving, exception handling, and concurrency
- To minimise number of rules: uniform treatment of configurations of the form $\langle c, \sigma \rangle \in \text{Cmd} \times \Sigma$ and $\sigma \in \Sigma$:
 - σ interpreted as $\langle \downarrow, \sigma \rangle$ with “**terminated**” command \downarrow
 - \downarrow satisfies $\downarrow ; c = c \parallel \downarrow = \downarrow \parallel c = c$
 - thus: read $\langle \downarrow ; x := 0, \sigma \rangle$ as $\langle x := 0, \sigma \rangle$

Nondeterminism

Small-Step Semantics II

Definition 16.4 (Small-step execution relation for NdWHILE)

The **small-step execution relation**, $\rightarrow_1 \subseteq (Cmd \times \Sigma) \times (Cmd \times \Sigma)$, is defined by the following rules:

$$\begin{array}{c} \frac{}{\langle \text{skip}, \sigma \rangle \rightarrow_1 \langle \downarrow, \sigma \rangle} \\ \frac{\langle c_1, \sigma \rangle \rightarrow_1 \langle c'_1, \sigma' \rangle}{\langle c_1; c_2, \sigma \rangle \rightarrow_1 \langle c'_1; c_2, \sigma' \rangle} \\ \frac{\langle b, \sigma \rangle \rightarrow \text{false}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, \sigma \rangle \rightarrow_1 \langle c_2, \sigma \rangle} \\ \frac{\langle b, \sigma \rangle \rightarrow \text{true}}{\langle \text{while } b \text{ do } c \text{ end}, \sigma \rangle \rightarrow_1 \langle c; \text{while } b \text{ do } c \text{ end}, \sigma \rangle} \\ \frac{}{\langle c_1 \square c_2, \sigma \rangle \rightarrow_1 \langle c_1, \sigma \rangle} \end{array} \quad \begin{array}{c} \frac{\langle a, \sigma \rangle \rightarrow z}{\langle x := a, \sigma \rangle \rightarrow_1 \langle \downarrow, \sigma[x \mapsto z] \rangle} \\ \frac{\langle b, \sigma \rangle \rightarrow \text{true}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, \sigma \rangle \rightarrow_1 \langle c_1, \sigma \rangle} \\ \frac{\langle b, \sigma \rangle \rightarrow \text{false}}{\langle \text{while } b \text{ do } c \text{ end}, \sigma \rangle \rightarrow_1 \langle \downarrow, \sigma \rangle} \\ \frac{}{\langle c_1 \square c_2, \sigma \rangle \rightarrow_1 \langle c_2, \sigma \rangle} \end{array}$$

Note: atomic execution of assignments

Small-Step Semantics III

Remarks:

- Possible to show: big-step and small-step semantics are **equivalent**, i.e., for all $c \in \text{Cmd}$ and $\sigma, \sigma' \in \Sigma$:

$$\langle c, \sigma \rangle \rightarrow \sigma' \iff \langle c, \sigma \rangle \rightarrow_1^+ \langle \downarrow, \sigma' \rangle$$

- Alternative (equivalent) formalisation of choice:

$$\frac{\langle c_1, \sigma \rangle \rightarrow_1 \langle c'_1, \sigma' \rangle}{\langle c_1 \square c_2, \sigma \rangle \rightarrow_1 \langle c'_1, \sigma' \rangle} \qquad \frac{\langle c_2, \sigma \rangle \rightarrow_1 \langle c'_2, \sigma' \rangle}{\langle c_1 \square c_2, \sigma \rangle \rightarrow_1 \langle c'_2, \sigma' \rangle}$$

The ParWHILE Language

Definition 16.5 (Syntax of ParWHILE)

$$\begin{aligned} a &::= z \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \in AExp \\ b &::= t \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in BExp \\ c &::= \text{skip} \mid x := a \mid c_1 ; c_2 \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end} \mid \text{while } b \text{ do } c \text{ end} \mid \\ &\quad c_1 \parallel c_2 \in Cmd \end{aligned}$$

Semantics of ParWHILE I

- Approach for defining semantics:
 - assignments are executed **atomically**
 - parallelism is modeled by **interleaving**, i.e., the actions of parallel statements are merged
- ⇒ Reduction of parallelism to **nondeterminism + sequential execution**
(similar to multitasking on sequential computers)
- Requires **small-step execution relation** for statements (cf. Definition 16.4)
- Again: **“terminated” command** \downarrow
 - \downarrow additionally satisfies $\downarrow ; c = \downarrow \parallel c = c \parallel \downarrow = c$
 - Thus: read $\langle \downarrow ; x := 0 \parallel \downarrow, \sigma \rangle$ as $\langle x := 0, \sigma \rangle$

Semantics of ParWHILE II

Definition 16.6 (Small-step execution relation for ParWHILE)

The **small-step execution relation**, $\rightarrow_1 \subseteq (Cmd \times \Sigma) \times (Cmd \times \Sigma)$, is defined by the following rules:

$$\begin{array}{c} \frac{}{\langle \text{skip}, \sigma \rangle \rightarrow_1 \langle \downarrow, \sigma \rangle} \qquad \frac{\langle a, \sigma \rangle \rightarrow z}{\langle x := a, \sigma \rangle \rightarrow_1 \langle \downarrow, \sigma[x \mapsto z] \rangle} \\ \frac{\langle c_1, \sigma \rangle \rightarrow_1 \langle c'_1, \sigma' \rangle}{\langle c_1; c_2, \sigma \rangle \rightarrow_1 \langle c'_1; c_2, \sigma' \rangle} \qquad \frac{\langle b, \sigma \rangle \rightarrow \text{true}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, \sigma \rangle \rightarrow_1 \langle c_1, \sigma \rangle} \\ \frac{\langle b, \sigma \rangle \rightarrow \text{false}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, \sigma \rangle \rightarrow_1 \langle c_2, \sigma \rangle} \qquad \frac{\langle b, \sigma \rangle \rightarrow \text{false}}{\langle \text{while } b \text{ do } c \text{ end}, \sigma \rangle \rightarrow_1 \langle \downarrow, \sigma \rangle} \\ \frac{\langle b, \sigma \rangle \rightarrow \text{true}}{\langle \text{while } b \text{ do } c \text{ end}, \sigma \rangle \rightarrow_1 \langle c; \text{while } b \text{ do } c \text{ end}, \sigma \rangle} \\ \frac{\langle c_1, \sigma \rangle \rightarrow_1 \langle c'_1, \sigma' \rangle}{\langle c_1 \parallel c_2, \sigma \rangle \rightarrow_1 \langle c'_1 \parallel c_2, \sigma' \rangle} \qquad \frac{\langle c_2, \sigma \rangle \rightarrow_1 \langle c'_2, \sigma' \rangle}{\langle c_1 \parallel c_2, \sigma \rangle \rightarrow_1 \langle c_1 \parallel c'_2, \sigma' \rangle} \end{array}$$

Semantics of ParWHILE III

Example 16.7

Let $c := (x := 1 \parallel x := 2); \text{if } x = 1 \text{ then } c_1 \text{ else } c_2 \text{ end}$ and $\sigma \in \Sigma$.

$$\begin{aligned} & \langle c, \sigma \rangle \rightarrow_1 \langle x := 2; \text{if } x = 1 \text{ then } c_1 \text{ else } c_2 \text{ end}, \sigma[x \mapsto 1] \rangle \\ & \qquad \qquad \qquad \frac{\langle 1, \sigma \rangle \rightarrow 1}{\langle x := 1, \sigma \rangle \rightarrow_1 \langle \downarrow, \sigma[x \mapsto 1] \rangle} \\ & \text{since} \qquad \frac{\langle x := 1, \sigma \rangle \rightarrow_1 \langle \downarrow, \sigma[x \mapsto 1] \rangle}{\langle x := 1 \parallel x := 2, \sigma \rangle \rightarrow_1 \langle \downarrow \parallel x := 2, \sigma[x \mapsto 1] \rangle} \\ & \rightarrow_1 \langle \text{if } x = 1 \text{ then } c_1 \text{ else } c_2 \text{ end}, \sigma[x \mapsto 2] \rangle \\ & \qquad \qquad \qquad \frac{\langle 2, \sigma \rangle \rightarrow 2}{\langle x := 2, \sigma \rangle \rightarrow_1 \langle \downarrow, \sigma[x \mapsto 2] \rangle} \\ & \text{since} \qquad \frac{\langle x := 2, \sigma \rangle \rightarrow_1 \langle \downarrow, \sigma[x \mapsto 2] \rangle}{\langle c_2, \sigma[x \mapsto 2] \rangle} \\ & \rightarrow_1 \langle c_2, \sigma[x \mapsto 2] \rangle \\ & \qquad \qquad \qquad \frac{\langle x, \sigma[x \mapsto 2] \rangle \rightarrow 2 \quad \langle 1, \sigma[x \mapsto 2] \rangle \rightarrow 1}{\langle x = 1, \sigma[x \mapsto 2] \rangle \rightarrow \text{false}} \\ & \text{since} \qquad \frac{\langle x, \sigma[x \mapsto 2] \rangle \rightarrow 2 \quad \langle 1, \sigma[x \mapsto 2] \rangle \rightarrow 1}{\langle x = 1, \sigma[x \mapsto 2] \rangle \rightarrow \text{false}} \end{aligned}$$

Analogously: $\langle c, \sigma \rangle \rightarrow_1^3 \langle c_1, \sigma[x \mapsto 1] \rangle$