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/`

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

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...)

$\Rightarrow$ All registered participants are admitted

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...)

$\Rightarrow$ All registered participants are admitted

- Information of inactive registered students
- Exam dates will be announced in beginning of 2018

Outline of Lecture 16

Motivation

- Essential question: what is the meaning of

$$c_1 \parallel c_2$$

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

Motivation

- Essential question: what is the meaning of

$$c_1 \parallel c_2$$

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

- Easy to answer when state spaces are disjoint:

$$\langle \mathtt{x} := 1 \parallel \mathtt{y} := 2, \sigma \rangle \to \sigma[\mathtt{x} \mapsto 1, \mathtt{y} \mapsto 2]$$

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

Motivation

- Essential question: what is the meaning of

$$c_1 \parallel c_2$$

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

- Easy to answer when state spaces are disjoint:

$$\langle \texttt{x := 1} \parallel \texttt{y := 2}, \sigma\rangle \to \sigma[\texttt{x} \mapsto 1, \texttt{y} \mapsto 2]$$

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

- But what if variables are shared?

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

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

Motivation

- Essential question: what is the meaning of

$$c_1 \parallel c_2$$

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

- Easy to answer when state spaces are disjoint:

$$\langle \texttt{x := 1} \parallel \texttt{y := 2}, \sigma\rangle \to \sigma[\texttt{x} \mapsto 1, \texttt{y} \mapsto 2]$$

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

- But what if variables are shared?

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

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

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

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\texttt{x := 0;}$$
$$(\texttt{x := x + 1} \parallel \texttt{x := x + 2})$$

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\texttt{x := 0;}$$
$$(\texttt{x := x + 1} \parallel \texttt{x := x + 2})$$

- At first glance: `x` is assigned 3

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\texttt{x := 0;}$$
$$(\texttt{x := x + 1} \parallel \texttt{x := x + 2})$$

- At first glance: `x` is assigned 3
- But: both parallel components could read `x` before it is written

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

```
x := 0;
(x := x + 1 || x := x + 2)
```

value of x: 0

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written

Software Modeling
and Verification Chair
RWTH AACHEN UNIVERSITY

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\begin{array}{c} \texttt{x := 0;} \\ (\texttt{x := x + 1} \parallel \texttt{x := x + 2}) \end{array}$$

1

value of $\texttt{x}$: 0

- At first glance: $\texttt{x}$ is assigned 3
- But: both parallel components could read $\texttt{x}$ before it is written

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\begin{array}{c} \texttt{x := 0;} \\ (\underbrace{\texttt{x := x + 1}}_{1} \parallel \underbrace{\texttt{x := x + 2}}_{2}) \end{array}$$

value of x: 0

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\begin{array}{c} \texttt{x := 0;} \\ (\underbrace{\texttt{x := x + 1}}_{1} \parallel \underbrace{\texttt{x := x + 2}}_{2}) \end{array}$$

value of $\texttt{x}$: 1

- At first glance: $\texttt{x}$ is assigned 3
- But: both parallel components could read $\texttt{x}$ before it is written

Software Modeling and Verification Chair

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\texttt{x := 0;}$$
$$(\texttt{x := x + 1} \parallel \texttt{x := x + 2}) \qquad \text{value of } \texttt{x}\text{: } 2$$

2

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2,

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\texttt{x := 0;}$$
$$(\texttt{x := x + 1} \parallel \texttt{x := x + 2}) \qquad \text{value of } \texttt{x}\text{: } 0$$

- At first glance: `x` is assigned 3
- But: both parallel components could read `x` before it is written
- Thus: `x` is assigned 2,

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\begin{array}{c}\texttt{x := 0;}\\ (\texttt{x := x + 1} \parallel \texttt{x := x + 2})\end{array} \qquad \text{value of } \texttt{x}\text{: } 0$$

1

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2,

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\texttt{x := 0;}$$
$$(\underbrace{\texttt{x := x + 1}}_{1} \parallel \underbrace{\texttt{x := x + 2}}_{2}) \qquad \text{value of } \texttt{x}\text{: } 0$$

- At first glance: `x` is assigned 3
- But: both parallel components could read `x` before it is written
- Thus: `x` is assigned 2,

Software Modeling and Verification Chair RWTH AACHEN UNIVERSITY

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\texttt{x := 0;}$$
$$(\underset{1}{\texttt{x := x + 1}} \parallel \underset{2}{\texttt{x := x + 2}}) \qquad \text{value of } \texttt{x}\text{: } 2$$

- At first glance: `x` is assigned 3
- But: both parallel components could read `x` before it is written
- Thus: `x` is assigned 2,

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

```
x := 0;
(x := x + 1 || x := x + 2)
1
```

value of x: 1

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2, 1,

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

```
x := 0;
(x := x + 1 || x := x + 2)
```

value of x: 0

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2, 1,

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\texttt{x := 0;}$$
$$(\texttt{x := x + 1} \parallel \texttt{x := x + 2})$$
$$2$$

value of $\texttt{x}$: 0

- At first glance: $\texttt{x}$ is assigned 3
- But: both parallel components could read $\texttt{x}$ before it is written
- Thus: $\texttt{x}$ is assigned 2, 1,

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\begin{array}{c} \texttt{x := 0;} \\ (\texttt{x := x + 1} \parallel \texttt{x := x + 2}) \\ \qquad\qquad\quad 2 \end{array}$$

value of x: 2

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2, 1,

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\texttt{x := 0;}$$
$$(\underbrace{\texttt{x := x + 1}}_{3} \parallel \texttt{x := x + 2}) \qquad \text{value of } \texttt{x}: 2$$

- At first glance: `x` is assigned 3
- But: both parallel components could read `x` before it is written
- Thus: `x` is assigned 2, 1,

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\texttt{x := 0;}$$
$$(\texttt{x := x + 1} \parallel \texttt{x := x + 2}) \qquad \text{value of } \texttt{x}\text{: 3}$$
$$3$$

- 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

Non-Atomic Assignments

Observation: parallelism introduces new phenomena

Example 16.1

$$\texttt{x := 0;}$$
$$(\texttt{x := x + 1} \parallel \texttt{x := x + 2})$$

- 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
  $\Rightarrow$ only possible outcome: 3

Introduction

Parallelism and Interaction

The problem arises due to the combination of

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

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.

Introduction

Reactive Systems

- Thus: “classical” model for sequential systems

  $$\text{System} : \text{Input} \to \text{Output}$$

  (transformational systems) is not adequate
- Missing: aspect of interaction

Reactive Systems

- Thus: "classical" model for sequential systems

$$\text{System} : \text{Input} \rightarrow \text{Output}$$

  (transformational systems) is not adequate
- Missing: aspect of interaction
- Rather: reactive systems which interact with environment and among themselves

Reactive Systems

- Thus: "classical" model for sequential systems

$$\text{System} : \text{Input} \to \text{Output}$$

  (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)

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)

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)

Outline of Lecture 16

Introduction

Nondeterminism

Shared-Variables Communication

The NdWHILE Language

Definition 16.2 (Syntax of NdWHILE)

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

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

Big-Step Semantics

Definition 16.3 (Big-step execution relation for NdWHILE)

For $c \in Cmd$ and $\sigma, \sigma' \in \Sigma$, the execution relation $\langle c, \sigma \rangle \to \sigma'$ is defined by:

$$\text{(skip)}\ \frac{}{\langle \texttt{skip}, \sigma \rangle \to \sigma} \qquad \text{(asgn)}\ \frac{\langle a, \sigma \rangle \to z}{\langle x := a, \sigma \rangle \to \sigma[x \mapsto z]}$$

$$\text{(seq)}\ \frac{\langle c_1, \sigma \rangle \to \sigma' \quad \langle c_2, \sigma' \rangle \to \sigma''}{\langle c_1 ; c_2, \sigma \rangle \to \sigma''} \qquad \text{(if-t)}\ \frac{\langle b, \sigma \rangle \to \text{true} \quad \langle c_1, \sigma \rangle \to \sigma'}{\langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}, \sigma \rangle \to \sigma'}$$

$$\text{(if-f)}\ \frac{\langle b, \sigma \rangle \to \text{false} \quad \langle c_2, \sigma \rangle \to \sigma'}{\langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}, \sigma \rangle \to \sigma'} \qquad \text{(wh-f)}\ \frac{\langle b, \sigma \rangle \to \text{false}}{\langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}, \sigma \rangle \to \sigma}$$

$$\text{(wh-t)}\ \frac{\langle b, \sigma \rangle \to \text{true} \quad \langle c, \sigma \rangle \to \sigma' \quad \langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}, \sigma' \rangle \to \sigma''}{\langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}, \sigma \rangle \to \sigma''}$$

$$\text{(alt-1)}\ \frac{\langle c_1, \sigma \rangle \to \sigma'}{\langle c_1 \ \square\ c_2, \sigma \rangle \to \sigma'} \qquad \text{(alt-2)}\ \frac{\langle c_2, \sigma \rangle \to \sigma'}{\langle c_1 \ \square\ c_2, \sigma \rangle \to \sigma'}$$

Small-Step Semantics I

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

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

  - only initial and final states
  - works for constructs whose computations are pure evaluation, without side-effects, no exceptions, and “usually” terminating

Small-Step Semantics 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:

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

  - specifies order of computation steps (which can be significant)
  - required for interleaving, exception handling, and concurrency

Small-Step Semantics I

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

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

  - 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:

$$\langle c, \sigma \rangle \to_1 \langle c_1, \sigma_1 \rangle \to_1 \langle c_2, \sigma_2 \rangle \to_1 \ldots \to_1 \sigma_n$$

  - 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 $\langle c, \sigma \rangle \in Cmd \times \Sigma$ 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 $\langle \downarrow \texttt{ ; x := 0}, \sigma \rangle$ as $\langle \texttt{x := 0}, \sigma \rangle$

Small-Step Semantics II

Definition 16.4 (Small-step execution relation for NdWHILE)

The small-step execution relation, $\to_1 \subseteq (Cmd \times \Sigma) \times (Cmd \times \Sigma)$, is defined by the following rules:

$$\frac{}{\langle \texttt{skip}, \sigma\rangle \to_1 \langle \downarrow, \sigma\rangle} \qquad \frac{\langle a, \sigma\rangle \to z}{\langle x := a, \sigma\rangle \to_1 \langle \downarrow, \sigma[x \mapsto z]\rangle}$$

$$\frac{\langle c_1, \sigma\rangle \to_1 \langle c_1', \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \to_1 \langle c_1'; c_2, \sigma'\rangle} \qquad \frac{\langle b, \sigma\rangle \to \mathsf{true}}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ end}, \sigma\rangle \to_1 \langle c_1, \sigma\rangle}$$

$$\frac{\langle b, \sigma\rangle \to \mathsf{false}}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ end}, \sigma\rangle \to_1 \langle c_2, \sigma\rangle} \qquad \frac{\langle b, \sigma\rangle \to \mathsf{false}}{\langle \texttt{while } b \texttt{ do } c \texttt{ end}, \sigma\rangle \to_1 \langle \downarrow, \sigma\rangle}$$

$$\frac{\langle b, \sigma\rangle \to \mathsf{true}}{\langle \texttt{while } b \texttt{ do } c \texttt{ end}, \sigma\rangle \to_1 \langle c; \texttt{while } b \texttt{ do } c \texttt{ end}, \sigma\rangle}$$

$$\frac{}{\langle c_1 \,\Box\, c_2, \sigma\rangle \to_1 \langle c_1, \sigma\rangle} \qquad \frac{}{\langle c_1 \,\Box\, c_2, \sigma\rangle \to_1 \langle c_2, \sigma\rangle}$$

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 $c \in Cmd$ and $\sigma, \sigma' \in \Sigma$:

$$\langle c, \sigma \rangle \to \sigma' \iff \langle c, \sigma \rangle \to_1^+ \langle \downarrow, \sigma' \rangle$$

Small-Step Semantics III

Remarks:

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

$$\langle c, \sigma \rangle \to \sigma' \iff \langle c, \sigma \rangle \to_1^+ \langle \downarrow, \sigma' \rangle$$

- Alternative (equivalent) formalisation of choice:

$$\frac{\langle c_1, \sigma \rangle \to_1 \langle c_1', \sigma' \rangle}{\langle c_1 \;\square\; c_2, \sigma \rangle \to_1 \langle c_1', \sigma' \rangle} \qquad \frac{\langle c_2, \sigma \rangle \to_1 \langle c_2', \sigma' \rangle}{\langle c_1 \;\square\; c_2, \sigma \rangle \to_1 \langle c_2', \sigma' \rangle}$$

Outline of Lecture 16

Introduction

Nondeterminism

Shared-Variables Communication

The ParWHILE Language

Definition 16.5 (Syntax of ParWHILE)

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

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

$\Rightarrow$ Reduction of parallelism to nondeterminism + sequential execution (similar to multitasking on sequential computers)

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

$\Rightarrow$ 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 $\downarrow ; c = \downarrow \| c = c \| \downarrow = c$
  - Thus: read $\langle \downarrow \texttt{ ; x := 0} \| \downarrow, \sigma \rangle$ as $\langle \texttt{x := 0}, \sigma \rangle$

Semantics of ParWHILE II

Definition 16.6 (Small-step execution relation for ParWHILE)

The small-step execution relation, $\to_1 \subseteq (Cmd \times \Sigma) \times (Cmd \times \Sigma)$, is defined by the following rules:

$$\frac{}{\langle \texttt{skip}, \sigma \rangle \to_1 \langle \downarrow, \sigma \rangle} \qquad \frac{\langle a, \sigma \rangle \to z}{\langle x := a, \sigma \rangle \to_1 \langle \downarrow, \sigma[x \mapsto z] \rangle}$$

$$\frac{\langle c_1, \sigma \rangle \to_1 \langle c_1', \sigma' \rangle}{\langle c_1; c_2, \sigma \rangle \to_1 \langle c_1'; c_2, \sigma' \rangle} \qquad \frac{\langle b, \sigma \rangle \to \mathsf{true}}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ end}, \sigma \rangle \to_1 \langle c_1, \sigma \rangle}$$

$$\frac{\langle b, \sigma \rangle \to \mathsf{false}}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ end}, \sigma \rangle \to_1 \langle c_2, \sigma \rangle} \qquad \frac{\langle b, \sigma \rangle \to \mathsf{false}}{\langle \texttt{while } b \texttt{ do } c \texttt{ end}, \sigma \rangle \to_1 \langle \downarrow, \sigma \rangle}$$

$$\frac{\langle b, \sigma \rangle \to \mathsf{true}}{\langle \texttt{while } b \texttt{ do } c \texttt{ end}, \sigma \rangle \to_1 \langle c; \texttt{while } b \texttt{ do } c \texttt{ end}, \sigma \rangle}$$

$$\frac{\langle c_1, \sigma \rangle \to_1 \langle c_1', \sigma' \rangle}{\langle c_1 \parallel c_2, \sigma \rangle \to_1 \langle c_1' \parallel c_2, \sigma' \rangle} \qquad \frac{\langle c_2, \sigma \rangle \to_1 \langle c_2', \sigma' \rangle}{\langle c_1 \parallel c_2, \sigma \rangle \to_1 \langle c_1 \parallel c_2', \sigma' \rangle}$$

Semantics of ParWHILE III

Example 16.7

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

Semantics of ParWHILE III

Example 16.7

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

$$\langle c, \sigma\rangle \to_1 \langle x := 2; \texttt{if } x = 1 \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ end}, \sigma[x \mapsto 1]\rangle$$

$$\text{since} \qquad \frac{\dfrac{\dfrac{}{\langle 1, \sigma\rangle \to 1}}{\langle x := 1, \sigma\rangle \to_1 \langle \downarrow, \sigma[x \mapsto 1]\rangle}}{\langle x := 1 \parallel x := 2, \sigma\rangle \to_1 \langle \downarrow \parallel x := 2, \sigma[x \mapsto 1]\rangle}$$

Semantics of ParWHILE III

Example 16.7

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

$$\langle c, \sigma\rangle \to_1 \langle x := 2; \texttt{if } x = 1 \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ end}, \sigma[x \mapsto 1]\rangle$$

since
$$\frac{\dfrac{\overline{\langle 1, \sigma\rangle \to 1}}{\langle x := 1, \sigma\rangle \to_1 \langle \downarrow, \sigma[x \mapsto 1]\rangle}}{\langle x := 1 \parallel x := 2, \sigma\rangle \to_1 \langle \downarrow \parallel x := 2, \sigma[x \mapsto 1]\rangle}$$

$$\to_1 \langle \texttt{if } x = 1 \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ end}, \sigma[x \mapsto 2]\rangle$$

since
$$\frac{\overline{\langle 2, \sigma\rangle \to 2}}{\langle x := 2, \sigma\rangle \to_1 \langle \downarrow, \sigma[x \mapsto 2]\rangle}$$

Semantics of ParWHILE III

Example 16.7

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

$$\langle c, \sigma\rangle \to_1 \langle x := 2; \texttt{if } x = 1 \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ end}, \sigma[x \mapsto 1]\rangle$$

since $$\frac{\dfrac{\dfrac{}{\langle 1, \sigma\rangle \to 1}}{\langle x := 1, \sigma\rangle \to_1 \langle \downarrow, \sigma[x \mapsto 1]\rangle}}{\langle x := 1 \parallel x := 2, \sigma\rangle \to_1 \langle \downarrow \parallel x := 2, \sigma[x \mapsto 1]\rangle}$$

$$\to_1 \langle \texttt{if } x = 1 \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ end}, \sigma[x \mapsto 2]\rangle$$

since $$\frac{\dfrac{}{\langle 2, \sigma\rangle \to 2}}{\langle x := 2, \sigma\rangle \to_1 \langle \downarrow, \sigma[x \mapsto 2]\rangle}$$

$$\to_1 \langle c_2, \sigma[x \mapsto 2]\rangle$$

since $$\frac{\dfrac{}{\langle x, \sigma[x \mapsto 2]\rangle \to 2} \quad \dfrac{}{\langle 1, \sigma[x \mapsto 2]\rangle \to 1}}{\langle x = 1, \sigma[x \mapsto 2]\rangle \to \text{false}}$$

Semantics of ParWHILE III

Example 16.7

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

$$\langle c, \sigma \rangle \to_1 \langle x := 2; \texttt{if}\ x = 1\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}, \sigma[x \mapsto 1] \rangle$$

$$\text{since} \quad \dfrac{\dfrac{\overline{\langle 1, \sigma \rangle \to 1}}{\langle x := 1, \sigma \rangle \to_1 \langle \downarrow, \sigma[x \mapsto 1] \rangle}}{\langle x := 1 \parallel x := 2, \sigma \rangle \to_1 \langle \downarrow \parallel x := 2, \sigma[x \mapsto 1] \rangle}$$

$$\to_1 \langle \texttt{if}\ x = 1\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}, \sigma[x \mapsto 2] \rangle$$

$$\text{since} \quad \dfrac{\overline{\langle 2, \sigma \rangle \to 2}}{\langle x := 2, \sigma \rangle \to_1 \langle \downarrow, \sigma[x \mapsto 2] \rangle}$$

$$\to_1 \langle c_2, \sigma[x \mapsto 2] \rangle$$

$$\text{since} \quad \dfrac{\overline{\langle x, \sigma[x \mapsto 2] \rangle \to 2} \qquad \overline{\langle 1, \sigma[x \mapsto 2] \rangle \to 1}}{\langle x = 1, \sigma[x \mapsto 2] \rangle \to \text{false}}$$

Analogously: $\langle c, \sigma \rangle \to_1^3 \langle c_1, \sigma[x \mapsto 1] \rangle$