Semantics and Verification of Software

Summer Semester 2015

Lecture 17: Axiomatic Semantics of WHILE V (Correctness Properties for Execution Time)

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

http://moves.rwth-aachen.de/teaching/ss-15/sv-sw/

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Online Registration for Seminars and Practical Courses (Praktika) in Winter Term 2015/16

Who?

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

Where?

www.graphics.rwth-aachen.de/apse

When?

25.06.2015 - 08.07.2015


SOMMERFEST
2015
26. Juni
Informatikzentrum
Karriere
13 30 Firmenkontaktmesse
Science Tunnel
Ort: Foyer und Korridor Hauptbau
Kluge Köpfe
15 30 Festveranstaltung
Absolventenfeier
Ort: Aula 2 Hauptbau
Coole Party
19 30 Eröffnung des Buffets
20 30–02 00 Party mit Live-Band und DJ
Ort: Foyer E2 und Parkplatz

bonding CyberDay
STUDENTENINITIATIVE E.V.

Workshops

ProSiebenSat.1
IT-Sicherheit

Accenture
Consulting: Networks Analytics
(Anmeldung unter www.bonding.de/CyberDay)

8 Fachvorträge

Podiumsdiskussion

„Digitalisierung der Industrie -
Chancen und Risiken“

bonding CyberDay 2015

KOSTENLOS
von Studenten
für Studenten

SuperC - Mittwoch 8. Juli 2015
Digitalisierung der Industrie -
von eCommerce bis zur Hybrid Cloud

www.bonding.de/CyberDay

bonding – erlebe, was du werden kannst.

Outline of Lecture 17

Recap: Communicating Sequential Processes

Fairness in CSP

Correctness Properties for Execution Time

Operational Semantics with Exact Execution Times

Timed Correctness Properties

Syntax of CSP

Definition (Syntax of CSP)

The syntax of CSP is given by

$$\begin{aligned}
a &::= z \mid x \mid a_1{+}a_2 \mid a_1{-}a_2 \mid a_1{*}a_2 \in AExp \\
b &::= t \mid a_1{=}a_2 \mid a_1{>}a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in BExp \\
c &::= \texttt{skip} \mid x := a \mid \alpha?x \mid \alpha!a \mid \\
&\qquad c_1; c_2 \mid \texttt{if}\ gc\ \texttt{fi} \mid \texttt{do}\ gc\ \texttt{od} \mid c_1 \parallel c_2 \in Cmd \\
gc &::= b \to c \mid b \wedge \alpha?x \to c \mid b \wedge \alpha!a \to c \mid gc_1 \mathbin{\Box} gc_2 \in GCmd
\end{aligned}$$

- In $c_1 \parallel c_2$, commands c_1 and c_2 must not use common variables (only local store)
- Guarded command $gc_1 \mathbin{\Box} gc_2$ represents an alternative
- In $b \to c$, b acts as a guard that enables the execution of c only if evaluated to true
- $b \wedge \alpha?x \to c$ and $b \wedge \alpha!a \to c$ additionally require the respective I/O operation to be enabled
- If none of its alternatives is enabled, a guarded command gc fails (configuration fail)
- `if` nondeterministically picks an enabled alternative
- A `do` loop is iterated until its body fails

Semantics of CSP I

Definition (Semantics of CSP – Commands (*Cmd*))

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

$$\frac{}{\langle \alpha?x, \sigma\rangle \xrightarrow{\alpha?z} \langle \downarrow, \sigma[x \mapsto z]\rangle} \qquad \frac{\langle a, \sigma\rangle \to z}{\langle \alpha!a, \sigma\rangle \xrightarrow{\alpha!z} \langle \downarrow, \sigma\rangle}$$

$$\frac{\langle c_1, \sigma\rangle \xrightarrow{\lambda} \langle c_1', \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \xrightarrow{\lambda} \langle c_1'; c_2, \sigma'\rangle} \qquad \frac{\langle gc, \sigma\rangle \xrightarrow{\lambda} \langle c, \sigma'\rangle}{\langle \mathtt{if}\ gc\ \mathtt{fi}, \sigma\rangle \xrightarrow{\lambda} \langle c, \sigma'\rangle}$$

$$\frac{\langle gc, \sigma\rangle \xrightarrow{\lambda} \langle c, \sigma'\rangle}{\langle \mathtt{do}\ gc\ \mathtt{od}, \sigma\rangle \xrightarrow{\lambda} \langle c; \mathtt{do}\ gc\ \mathtt{od}, \sigma'\rangle} \qquad \frac{\langle gc, \sigma\rangle \to \mathrm{fail}}{\langle \mathtt{do}\ gc\ \mathtt{od}, \sigma\rangle \to \langle \downarrow, \sigma\rangle}$$

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

$$\frac{\langle c_1, \sigma\rangle \xrightarrow{\alpha?z} \langle c_1', \sigma'\rangle \quad \langle c_2, \sigma\rangle \xrightarrow{\alpha!z} \langle c_2', \sigma\rangle}{\langle c_1 \parallel c_2, \sigma\rangle \to \langle c_1' \parallel c_2', \sigma'\rangle} \qquad \frac{\langle c_1, \sigma\rangle \xrightarrow{\alpha!z} \langle c_1', \sigma\rangle \quad \langle c_2, \sigma\rangle \xrightarrow{\alpha?z} \langle c_2', \sigma'\rangle}{\langle c_1 \parallel c_2, \sigma\rangle \to \langle c_1' \parallel c_2', \sigma'\rangle}$$

Semantics of CSP II

Definition (Semantics of CSP – Guarded commands (*GCmd*))

$$\frac{\langle b, \sigma\rangle \to \mathsf{true}}{\langle b \to c, \sigma\rangle \to \langle c, \sigma\rangle} \qquad \frac{\langle b, \sigma\rangle \to \mathsf{false}}{\langle b \to c, \sigma\rangle \to \mathsf{fail}}$$

$$\frac{\langle b, \sigma\rangle \to \mathsf{true}}{\langle b \wedge \alpha?x \to c, \sigma\rangle \xrightarrow{\alpha?z} \langle c, \sigma[x \mapsto z]\rangle} \qquad \frac{\langle b, \sigma\rangle \to \mathsf{false}}{\langle b \wedge \alpha?x \to c, \sigma\rangle \to \mathsf{fail}}$$

$$\frac{\langle b, \sigma\rangle \to \mathsf{true} \quad \langle a, \sigma\rangle \to z}{\langle b \wedge \alpha!a \to c, \sigma\rangle \xrightarrow{\alpha!z} \langle c, \sigma\rangle} \qquad \frac{\langle b, \sigma\rangle \to \mathsf{false}}{\langle b \wedge \alpha!a \to c, \sigma\rangle \to \mathsf{fail}}$$

$$\frac{\langle gc_1, \sigma\rangle \xrightarrow{\lambda} \langle c, \sigma'\rangle}{\langle gc_1 \,\square\, gc_2, \sigma\rangle \xrightarrow{\lambda} \langle c, \sigma'\rangle} \qquad \frac{\langle gc_2, \sigma\rangle \xrightarrow{\lambda} \langle c, \sigma'\rangle}{\langle gc_1 \,\square\, gc_2, \sigma\rangle \xrightarrow{\lambda} \langle c, \sigma'\rangle}$$

$$\frac{\langle gc_1, \sigma\rangle \to \mathsf{fail} \quad \langle gc_2, \sigma\rangle \to \mathsf{fail}}{\langle gc_1 \,\square\, gc_2, \sigma\rangle \to \mathsf{fail}}$$

Recap: Communicating Sequential Processes

CSP Examples

Example

(on the board)

1. $\mathtt{do}\ (\mathsf{true} \wedge \alpha?x \to \beta!x)\ \mathtt{od}$
   describes a process that repeatedly receives a value along α and forwards it along β (i.e., a one-place buffer)
2. $\mathtt{do}\ \mathsf{true} \wedge \alpha?x \to \beta!x\ \mathtt{od} \parallel \mathtt{do}\ \mathsf{true} \wedge \beta?y \to \gamma!y\ \mathtt{od}$
   specifies a two-place buffer that receives along α and sends along γ (using β for internal communication)
3. Nondeterministic choice between input channels:
   i. $\mathtt{if}\ (\mathsf{true} \wedge \alpha?x \to c_1 \,\square\, \mathsf{true} \wedge \beta?y \to c_2)\ \mathtt{fi}$
   ii. $\mathtt{if}\ (\mathsf{true} \to (\alpha?x; c_1) \,\square\, \mathsf{true} \to (\beta?y; c_2))\ \mathtt{fi}$

   Expected: progress whenever environment provides data on α or β
   i. correct
   ii. incorrect (can deadlock)

Outline of Lecture 17

Recap: Communicating Sequential Processes

Fairness in CSP

Correctness Properties for Execution Time

Operational Semantics with Exact Execution Times

Timed Correctness Properties

Fairness in CSP

Fairness I

- Informally: unfair behaviour excludes processes from being executed
- Here: consider parallel composition of $n \geq 1$ sequential programs with executions of the form $\kappa_0 \to \kappa_1 \to \kappa_2 \to \ldots$ where $\kappa_j = \langle c_1^{(j)} \parallel \ldots \parallel c_n^{(j)}, \sigma_j \rangle$ and, for some $1 \leq i \leq n$ and $k_0 \in \mathbb{N}$, $c_i^{(k)} = c_i^{(k_0)}$ for all $k \geq k_0$
- But: only unfair if c_i not enabled

Fairness I

- Informally: unfair behaviour excludes processes from being executed
- Here: consider parallel composition of $n \geq 1$ sequential programs with executions of the form $\kappa_0 \to \kappa_1 \to \kappa_2 \to \ldots$ where $\kappa_j = \langle c_1^{(j)} \parallel \ldots \parallel c_n^{(j)}, \sigma_j \rangle$ and, for some $1 \leq i \leq n$ and $k_0 \in \mathbb{N}$, $c_i^{(k)} = c_i^{(k_0)}$ for all $k \geq k_0$
- But: only unfair if c_i not enabled

Definition (Enabledness)

c_i is enabled in configuration $\kappa = \langle c_1 \parallel \ldots \parallel c_n, \sigma \rangle$ if there exists $\kappa' = \langle c_1' \parallel \ldots \parallel c_n', \sigma' \rangle$ with $\kappa \to \kappa'$ and $c_i' \neq c_i$.

Fairness I

- Informally: unfair behaviour excludes processes from being executed
- Here: consider parallel composition of $n \geq 1$ sequential programs with executions of the form $\kappa_0 \to \kappa_1 \to \kappa_2 \to \ldots$ where $\kappa_j = \langle c_1^{(j)} \parallel \ldots \parallel c_n^{(j)}, \sigma_j \rangle$ and, for some $1 \leq i \leq n$ and $k_0 \in \mathbb{N}$, $c_i^{(k)} = c_i^{(k_0)}$ for all $k \geq k_0$
- But: only unfair if c_i not enabled

Definition (Enabledness)

c_i is enabled in configuration $\kappa = \langle c_1 \parallel \ldots \parallel c_n, \sigma \rangle$ if there exists $\kappa' = \langle c_1' \parallel \ldots \parallel c_n', \sigma' \rangle$ with $\kappa \to \kappa'$ and $c_i' \neq c_i$.

Example

1. $x := 0$ enabled in $\langle x := 0 \parallel y := 1, \sigma \rangle$ (actually always enabled)
2. $\alpha?x$ enabled in $\langle \alpha?x \parallel \alpha!0, \sigma \rangle$
3. $\alpha?x$ not enabled in $\langle \alpha?x \parallel \beta!1, \sigma \rangle$

Fairness II

Definition (Fairness)

An execution $\kappa_0 \to \kappa_1 \to \kappa_2 \to \ldots$ where $\kappa_j = \langle c_1^{(j)} \parallel \ldots \parallel c_n^{(j)}, \sigma_j \rangle$ and, for some $1 \leq i \leq n$ and $k_0 \in \mathbb{N}$, $c_i^{(k)} = c_i^{(k_0)}$ for all $k \geq k_0$ is called

1. strongly unfair if $c_i^{(k)}$ is enabled in κ_k for all $k \geq k_0$
2. weakly unfair if $c_i^{(k)}$ is enabled in κ_k for infinitely many $k \geq k_0$

Fairness III

Example

1. $$\begin{aligned} & \langle \texttt{do true} \to x := x+1 \texttt{ od} \parallel y := y+1, \ldots \rangle \\ \to\ & \langle x := x+1; \texttt{do true} \to x := x+1 \texttt{ od} \parallel y := y+1, \ldots \rangle \\ \to\ & \langle \texttt{do true} \to x := x+1 \texttt{ od} \parallel y := y+1, \ldots \rangle \to \ldots \end{aligned}$$
   is strongly unfair since $y := y+1$ is always enabled

Fairness III

Example

1. $\langle \text{do true} \to x := x + 1 \text{ od} \parallel y := y + 1, \ldots \rangle$
   $\to \langle x := x + 1; \text{do true} \to x := x + 1 \text{ od} \parallel y := y + 1, \ldots \rangle$
   $\to \langle \text{do true} \to x := x + 1 \text{ od} \parallel y := y + 1, \ldots \rangle \to \ldots$
   is strongly unfair since $y := y + 1$ is always enabled
2. $\langle \text{do true} \to x := x + 1 \text{ od} \parallel \alpha!1 \parallel \alpha?y, \ldots \rangle$
   $\to \langle x := x + 1; \text{do true} \to x := x + 1 \text{ od} \parallel \alpha!1 \parallel \alpha?y, \ldots \rangle$
   $\to \langle \text{do true} \to x := x + 1 \text{ od} \parallel \alpha!1 \parallel \alpha?y, \ldots \rangle \to \ldots$
   is strongly unfair since both I/O operations are always enabled

Fairness in CSP

Fairness III

Example

1. $\langle \text{do true} \rightarrow x := x + 1 \text{ od} \parallel y := y + 1, \ldots \rangle$
   $\rightarrow \langle x := x + 1; \text{do true} \rightarrow x := x + 1 \text{ od} \parallel y := y + 1, \ldots \rangle$
   $\rightarrow \langle \text{do true} \rightarrow x := x + 1 \text{ od} \parallel y := y + 1, \ldots \rangle \rightarrow \ldots$
   is strongly unfair since $y := y + 1$ is always enabled
2. $\langle \text{do true} \rightarrow x := x + 1 \text{ od} \parallel \alpha!1 \parallel \alpha?y, \ldots \rangle$
   $\rightarrow \langle x := x + 1; \text{do true} \rightarrow x := x + 1 \text{ od} \parallel \alpha!1 \parallel \alpha?y, \ldots \rangle$
   $\rightarrow \langle \text{do true} \rightarrow x := x + 1 \text{ od} \parallel \alpha!1 \parallel \alpha?y, \ldots \rangle \rightarrow \ldots$
   is strongly unfair since both I/O operations are always enabled
3. $\langle \text{do } \alpha!1 \rightarrow \texttt{skip} \text{ od} \parallel \text{do } \alpha?x \rightarrow \texttt{skip} \text{ od} \parallel \alpha?y, \ldots \rangle$
   $\rightarrow \langle \texttt{skip}; \text{do } \alpha!1 \rightarrow \texttt{skip} \text{ od} \parallel \texttt{skip}; \text{do } \alpha?x \rightarrow \texttt{skip} \text{ od} \parallel \alpha?y, \ldots \rangle$
   $\rightarrow \langle \texttt{skip}; \text{do } \alpha!1 \rightarrow \texttt{skip} \text{ od} \parallel \text{do } \alpha?x \rightarrow \texttt{skip} \text{ od} \parallel \alpha?y, \ldots \rangle$
   $\rightarrow \langle \text{do } \alpha!1 \rightarrow \texttt{skip} \text{ od} \parallel \text{do } \alpha?x \rightarrow \texttt{skip} \text{ od} \parallel \alpha?y, \ldots \rangle \rightarrow \ldots$
   is weakly unfair since $\alpha?y$ is enabled in every third configuration

Outline of Lecture 17

Recap: Communicating Sequential Processes

Fairness in CSP

Correctness Properties for Execution Time

Operational Semantics with Exact Execution Times

Timed Correctness Properties

Correctness Properties for Execution Time

The Approach

- Definition 11.3: proof system for total correctness
- Can be used to show that program terminates bus does not give any information about required resources

Correctness Properties for Execution Time

The Approach

- Definition 11.3: proof system for total correctness
- Can be used to show that program terminates bus does not give any information about required resources
- Goal: extend proof system to give (order of magnitude of) execution time of a statement
- Details in H.R. Nielson, F. Nielson: *Semantics with Applications: An Appetizer*, Springer, Section 10.2

The Approach

- Definition 11.3: proof system for total correctness
- Can be used to show that program terminates bus does not give any information about required resources
- Goal: extend proof system to give (order of magnitude of) execution time of a statement
- Details in H.R. Nielson, F. Nielson: *Semantics with Applications: An Appetizer*, Springer, Section 10.2
- Informal guidelines (idea: each instruction of abstract machine takes one time unit):
  - `skip`: execution time $\mathcal{O}(1)$ (that is, bounded by a constant)
  - assignment: execution time $\mathcal{O}(1)$ (with maximal size of RHS as constant)
  - composition: sum of execution times of constituent statements
  - conditional: maximal execution time of branches
  - iteration: sum over all iterations of execution times of loop body

Correctness Properties for Execution Time

The Approach

- Definition 11.3: proof system for total correctness
- Can be used to show that program terminates bus does not give any information about required resources
- Goal: extend proof system to give (order of magnitude of) execution time of a statement
- Details in H.R. Nielson, F. Nielson: *Semantics with Applications: An Appetizer*, Springer, Section 10.2
- Informal guidelines (idea: each instruction of abstract machine takes one time unit):
  - `skip`: execution time $\mathcal{O}(1)$ (that is, bounded by a constant)
  - assignment: execution time $\mathcal{O}(1)$ (with maximal size of RHS as constant)
  - composition: sum of execution times of constituent statements
  - conditional: maximal execution time of branches
  - iteration: sum over all iterations of execution times of loop body
- Procedure:
  1. Extend evaluation relation for expressions to give exact evaluation times
  2. Extend execution relation for statements to give exact execution times
  3. Extend total correctness proof system to give order of magnitude of execution time of statements

Outline of Lecture 17

Recap: Communicating Sequential Processes

Fairness in CSP

Correctness Properties for Execution Time

Operational Semantics with Exact Execution Times

Timed Correctness Properties

Recap: Translation of Arithmetic Expressions

Definition (Translation of arithmetic expressions (Definition 5.1))

The translation function

$$\mathfrak{T}_a[\![.]\!] : AExp \to Code$$

is given by

$$\begin{aligned}
\mathfrak{T}_a[\![z]\!] &:= \texttt{PUSH}(z) \\
\mathfrak{T}_a[\![x]\!] &:= \texttt{LOAD}(x) \\
\mathfrak{T}_a[\![a_1 + a_2]\!] &:= \mathfrak{T}_a[\![a_1]\!] ; \mathfrak{T}_a[\![a_2]\!] ; \texttt{ADD} \\
\mathfrak{T}_a[\![a_1 - a_2]\!] &:= \mathfrak{T}_a[\![a_1]\!] ; \mathfrak{T}_a[\![a_2]\!] ; \texttt{SUB} \\
\mathfrak{T}_a[\![a_1 * a_2]\!] &:= \mathfrak{T}_a[\![a_1]\!] ; \mathfrak{T}_a[\![a_2]\!] ; \texttt{MULT}
\end{aligned}$$

Timed Evaluation of Arithmetic Expressions

Definition 17.1 (Timed Evaluation of arithmetic expressions (extends Definition 2.2))

Expression a evaluates to $z \in \mathbb{Z}$ in state σ in $\tau \in \mathbb{N}$ steps (notation: $\langle a, \sigma \rangle \xrightarrow{\tau} z$) if this relationship is derivable by means of the following rules:

Axioms:
$$\frac{}{\langle z, \sigma \rangle \xrightarrow{1} z} \qquad \frac{}{\langle x, \sigma \rangle \xrightarrow{1} \sigma(x)}$$

Rules:
$$\frac{\langle a_1, \sigma \rangle \xrightarrow{\tau_1} z_1 \quad \langle a_2, \sigma \rangle \xrightarrow{\tau_2} z_2}{\langle a_1 + a_2, \sigma \rangle \xrightarrow{\tau_1 + \tau_2 + 1} z} \text{ where } z := z_1 + z_2$$

$$\frac{\langle a_1, \sigma \rangle \xrightarrow{\tau_1} z_1 \quad \langle a_2, \sigma \rangle \xrightarrow{\tau_2} z_2}{\langle a_1 - a_2, \sigma \rangle \xrightarrow{\tau_1 + \tau_2 + 1} z} \text{ where } z := z_1 - z_2$$

$$\frac{\langle a_1, \sigma \rangle \xrightarrow{\tau_1} z_1 \quad \langle a_2, \sigma \rangle \xrightarrow{\tau_2} z_2}{\langle a_1 * a_2, \sigma \rangle \xrightarrow{\tau_1 + \tau_2 + 1} z} \text{ where } z := z_1 \cdot z_2$$

Recap: Translation of Boolean Expressions

Definition (Translation of Boolean expressions (Definition 5.3))

The translation function

$$\mathfrak{T}_b[\![.]\!] : BExp \to Code$$

is given by

$$\begin{aligned}
\mathfrak{T}_b[\![\mathsf{true}]\!] &:= \mathtt{PUSH(}\mathsf{true}\mathtt{)}\\
\mathfrak{T}_b[\![\mathsf{false}]\!] &:= \mathtt{PUSH(}\mathsf{false}\mathtt{)}\\
\mathfrak{T}_b[\![a_1{=}a_2]\!] &:= \mathfrak{T}_a[\![a_1]\!]\,\mathtt{;}\,\mathfrak{T}_a[\![a_2]\!]\,\mathtt{;EQ}\\
\mathfrak{T}_b[\![a_1{>}a_2]\!] &:= \mathfrak{T}_a[\![a_1]\!]\,\mathtt{;}\,\mathfrak{T}_a[\![a_2]\!]\,\mathtt{;GT}\\
\mathfrak{T}_b[\![\neg b]\!] &:= \mathfrak{T}_b[\![b]\!]\,\mathtt{;NOT}\\
\mathfrak{T}_b[\![b_1 \wedge b_2]\!] &:= \mathfrak{T}_b[\![b_1]\!]\,\mathtt{;}\,\mathfrak{T}_b[\![b_2]\!]\,\mathtt{;AND}\\
\mathfrak{T}_b[\![b_1 \vee b_2]\!] &:= \mathfrak{T}_b[\![b_1]\!]\,\mathtt{;}\,\mathfrak{T}_b[\![b_2]\!]\,\mathtt{;OR}
\end{aligned}$$

Timed Evaluation of Boolean Expressions

Definition 17.2 (Timed Evaluation of Boolean expressions (extends Definition 2.7))

For $b \in BExp$, $\sigma \in \Sigma$, $\tau \in \mathbb{N}$, and $t \in \mathbb{B}$, the timed evaluation relation $\langle b, \sigma \rangle \xrightarrow{\tau} t$ is defined by:

$$\frac{}{\langle t, \sigma \rangle \xrightarrow{1} t}$$

$$\frac{\langle a_1, \sigma \rangle \xrightarrow{\tau_1} z \quad \langle a_2, \sigma \rangle \xrightarrow{\tau_2} z}{\langle a_1 {=} a_2, \sigma \rangle \xrightarrow{\tau_1+\tau_2+1} \text{true}} \qquad \frac{\langle a_1, \sigma \rangle \xrightarrow{\tau_1} z_1 \quad \langle a_2, \sigma \rangle \xrightarrow{\tau_2} z_2}{\langle a_1 {=} a_2, \sigma \rangle \xrightarrow{\tau_1+\tau_2+1} \text{false}} \text{ if } z_1 \neq z_2$$

$$\frac{\langle a_1, \sigma \rangle \xrightarrow{\tau_1} z_1 \quad \langle a_2, \sigma \rangle \xrightarrow{\tau_2} z_2}{\langle a_1 {>} a_2, \sigma \rangle \xrightarrow{\tau_1+\tau_2+1} \text{true}} \text{ if } z_1 > z_2 \qquad \frac{\langle a_1, \sigma \rangle \xrightarrow{\tau_1} z_1 \quad \langle a_2, \sigma \rangle \xrightarrow{\tau_2} z_2}{\langle a_1 {>} a_2, \sigma \rangle \xrightarrow{\tau_1+\tau_2+1} \text{false}} \text{ if } z_1 \leq z_2$$

$$\frac{\langle b, \sigma \rangle \xrightarrow{\tau} \text{false}}{\langle \neg b, \sigma \rangle \xrightarrow{\tau+1} \text{true}} \qquad \frac{\langle b, \sigma \rangle \xrightarrow{\tau} \text{true}}{\langle \neg b, \sigma \rangle \xrightarrow{\tau+1} \text{false}}$$

$$\frac{\langle b_1, \sigma \rangle \xrightarrow{\tau_1} \text{true} \quad \langle b_2, \sigma \rangle \xrightarrow{\tau_2} \text{true}}{\langle b_1 \wedge b_2, \sigma \rangle \xrightarrow{\tau_1+\tau_2+1} \text{true}} \qquad \frac{\langle b_1, \sigma \rangle \xrightarrow{\tau_1} \text{true} \quad \langle b_2, \sigma \rangle \xrightarrow{\tau_2} \text{false}}{\langle b_1 \wedge b_2, \sigma \rangle \xrightarrow{\tau_1+\tau_2+1} \text{false}}$$

$$\frac{\langle b_1, \sigma \rangle \xrightarrow{\tau_1} \text{false} \quad \langle b_2, \sigma \rangle \xrightarrow{\tau_2} \text{true}}{\langle b_1 \wedge b_2, \sigma \rangle \xrightarrow{\tau_1+\tau_2+1} \text{false}} \qquad \frac{\langle b_1, \sigma \rangle \xrightarrow{\tau_1} \text{false} \quad \langle b_2, \sigma \rangle \xrightarrow{\tau_2} \text{false}}{\langle b_1 \wedge b_2, \sigma \rangle \xrightarrow{\tau_1+\tau_2+1} \text{false}}$$

($\vee$ analogously)

Recap: Translation of Statements

Definition (Translation of statements (Definition 5.4))

The translation function $\mathfrak{T}_c[\![.]\!] : Cmd \to Code$ is given by

$$\begin{aligned}
\mathfrak{T}_c[\![\texttt{skip}]\!] &:= \varepsilon \\
\mathfrak{T}_c[\![x := a]\!] &:= \mathfrak{T}_a[\![a]\!]\,;\texttt{STO}(x) \\
\mathfrak{T}_c[\![c_1\,;c_2]\!] &:= \mathfrak{T}_c[\![c_1]\!]\,;\mathfrak{T}_c[\![c_2]\!] \\
\mathfrak{T}_c[\![\texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}]\!] &:= \mathfrak{T}_b[\![b]\!]\,;\texttt{JMPF}(|\mathfrak{T}_c[\![c_1]\!]| + 2)\,; \\
&\quad\ \mathfrak{T}_c[\![c_1]\!]\,;\texttt{JMP}(|\mathfrak{T}_c[\![c_2]\!]| + 1)\,; \\
&\quad\ \mathfrak{T}_c[\![c_2]\!] \\
\mathfrak{T}_c[\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}]\!] &:= \mathfrak{T}_b[\![b]\!]\,;\texttt{JMPF}(|\mathfrak{T}_c[\![c]\!]| + 2)\,; \\
&\quad\ \mathfrak{T}_c[\![c]\!]\,;\texttt{JMP}(-(|\mathfrak{T}_b[\![b]\!]| + |\mathfrak{T}_c[\![c]\!]| + 1))
\end{aligned}$$

Operational Semantics with Exact Execution Times

Timed Execution of Statements

Definition 17.3 (Timed execution relation for statements (extends Definition 3.2))

For $c \in Cmd$, $\sigma, \sigma' \in \Sigma$, and $\tau \in \mathbb{N}$, the timed execution relation $\langle c, \sigma \rangle \xrightarrow{\tau} \sigma'$ is defined by:

$$\text{(skip)}\ \frac{}{\langle \texttt{skip}, \sigma \rangle \xrightarrow{1} \sigma} \qquad \text{(asgn)}\ \frac{\langle a, \sigma \rangle \xrightarrow{\tau} z}{\langle x := a, \sigma \rangle \xrightarrow{\tau+1} \sigma[x \mapsto z]}$$

$$\text{(seq)}\ \frac{\langle c_1, \sigma \rangle \xrightarrow{\tau_1} \sigma' \quad \langle c_2, \sigma' \rangle \xrightarrow{\tau_2} \sigma''}{\langle c_1 ; c_2, \sigma \rangle \xrightarrow{\tau_1+\tau_2} \sigma''} \qquad \text{(if-t)}\ \frac{\langle b, \sigma \rangle \xrightarrow{\tau} \mathsf{true} \quad \langle c_1, \sigma \rangle \xrightarrow{\tau_1} \sigma'}{\langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}, \sigma \rangle \xrightarrow{\tau+\tau_1+2} \sigma'}$$

$$\text{(wh-f)}\ \frac{\langle b, \sigma \rangle \xrightarrow{\tau} \mathsf{false}}{\langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}, \sigma \rangle \xrightarrow{\tau+1} \sigma} \qquad \text{(if-f)}\ \frac{\langle b, \sigma \rangle \xrightarrow{\tau} \mathsf{false} \quad \langle c_2, \sigma \rangle \xrightarrow{\tau_2} \sigma'}{\langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}, \sigma \rangle \xrightarrow{\tau+\tau_2+1} \sigma'}$$

$$\text{(wh-t)}\ \frac{\langle b, \sigma \rangle \xrightarrow{\tau} \mathsf{true} \quad \langle c, \sigma \rangle \xrightarrow{\tau_1} \sigma' \quad \langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}, \sigma' \rangle \xrightarrow{\tau_2} \sigma''}{\langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}, \sigma \rangle \xrightarrow{\tau+\tau_1+\tau_2+2} \sigma''}$$

Outline of Lecture 17

Recap: Communicating Sequential Processes

Fairness in CSP

Correctness Properties for Execution Time

Operational Semantics with Exact Execution Times

Timed Correctness Properties

Timed Correctness Properties

Recap: Total Correctness Properties

So far: total correctness properties of the form

$$\{A\}\ c\ \{\Downarrow B\}$$

where $c \in Cmd$ and $A, B \in Assn$

Recap: Total Correctness Properties

So far: total correctness properties of the form

$$\{A\}\ c\ \{\Downarrow B\}$$

where $c \in Cmd$ and $A, B \in Assn$

Validity of property $\{A\}\ c\ \{\Downarrow B\}$

For all states $\sigma \in \Sigma$ which satisfy A:
the execution of c in σ terminates and yields a state which satisfies B.

Timed Correctness Properties

Now: timed correctness properties of the form

$$\{A\}\ c\ \{e \Downarrow B\}$$

where $c \in Cmd$, $A, B \in Assn$, and $e \in AExp$

Timed Correctness Properties

Now: timed correctness properties of the form

$$\{A\}\ c\ \{e \Downarrow B\}$$

where $c \in Cmd$, $A, B \in Assn$, and $e \in AExp$

Validity of property $\{A\}\ c\ \{e \Downarrow B\}$

For all states $\sigma \in \Sigma$ which satisfy A: the execution of c in σ terminates in a state satisfying B, and the required execution time is in $\mathcal{O}(e)$

Timed Correctness Properties

Now: timed correctness properties of the form

$$\{A\}\, c\, \{e \Downarrow B\}$$

where $c \in Cmd$, $A, B \in Assn$, and $e \in AExp$

Validity of property $\{A\}\, c\, \{e \Downarrow B\}$

For all states $\sigma \in \Sigma$ which satisfy A: the execution of c in σ terminates in a state satisfying B, and the required execution time is in $\mathcal{O}(e)$

Example 17.4

1. $\{\mathtt{x} = 3\}$ `y:=1; while ¬(x=1) do y:=y*x; x:=x-1 end` $\{1 \Downarrow \mathsf{true}\}$ expresses that for constant input 3, the execution time of the factorial program is bounded by a constant

Timed Correctness Properties

Now: timed correctness properties of the form

$$\{A\}\, c\, \{e \Downarrow B\}$$

where $c \in Cmd$, $A, B \in Assn$, and $e \in AExp$

Validity of property $\{A\}\, c\, \{e \Downarrow B\}$

For all states $\sigma \in \Sigma$ which satisfy A: the execution of c in σ terminates in a state satisfying B, and the required execution time is in $\mathcal{O}(e)$

Example 17.4

1. $\{\mathtt{x} = 3\}\, \mathtt{y:=1;\ while\ \neg(x=1)\ do\ y:=y*x;\ x:=x-1\ end}\, \{1 \Downarrow \mathsf{true}\}$ expresses that for constant input 3, the execution time of the factorial program is bounded by a constant
2. $\{\mathtt{x} > 0\}\, \mathtt{y:=1;\ while\ \neg(x=1)\ do\ y:=y*x;\ x:=x-1\ end}\, \{\mathtt{x} \Downarrow \mathsf{true}\}$ expresses that for positive inputs, the execution time of the factorial program is linear in that value

Semantics of Timed Correctness Properties

Definition 17.5 (Semantics of timed correctness properties (extends Definition 11.1))

Let $A, B \in Assn$, $c \in Cmd$, and $e \in AExp$. Then $\{A\}\, c\, \{e \Downarrow B\}$ is called valid (notation: $\models \{A\}\, c\, \{e \Downarrow B\}$) if there exists $k \in \mathbb{N}$ such that for each $I \in Int$ and each $\sigma \models^I A$, there exist $\sigma' \in \Sigma$ and $\tau \leq k \cdot \mathfrak{A}[\![e]\!]\sigma$ such that $\langle c, \sigma\rangle \xrightarrow{\tau} \sigma'$ and $\sigma' \models^I B$

Note: e is evaluated in initial (rather than final) state

Proving Timed Correctness I

Definition 17.6 (Hoare Logic for timed correctness (extends Definition 11.3))

The Hoare rules for timed correctness are given by (where $i, u \in LVar$)

$$\text{(skip)}\ \frac{}{\{A\}\ \texttt{skip}\ \{1 \Downarrow A\}} \qquad \text{(asgn)}\ \frac{}{\{A[x \mapsto a]\}\ x := a\ \{1 \Downarrow A\}}$$

$$\text{(seq)}\ \frac{\{A \wedge e_2' = u\}\ c_1\ \{e_1 \Downarrow C \wedge e_2 \leq u\} \quad \{C\}\ c_2\ \{e_2 \Downarrow B\}}{\{A\}\ c_1 ; c_2\ \{e_1 + e_2' \Downarrow B\}}$$

$$\text{(if)}\ \frac{\{A \wedge b\}\ c_1\ \{e \Downarrow B\} \quad \{A \wedge \neg b\}\ c_2\ \{e \Downarrow B\}}{\{A\}\ \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}\ \{e \Downarrow B\}}$$

$$\text{(while)}\ \frac{\{i \geq 0 \wedge A(i+1) \wedge e' = u\}\ c\ \{e_0 \Downarrow A(i) \wedge e \leq u\}}{\{\exists i. i \geq 0 \wedge A(i)\}\ \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}\ \{e \Downarrow A(0)\}}$$

where $\models (i \geq 0 \wedge A(i+1)) \Rightarrow (b \wedge e \geq e_0 + e')$ and $\models A(0) \Rightarrow (\neg b \wedge e \geq 1)$

$$\text{(cons)}\ \frac{\models (A \Rightarrow (A' \wedge \exists k \in \mathbb{N}. e' \leq k \cdot e)) \quad \{A'\}\ c\ \{e' \Downarrow B'\} \quad \models (B' \Rightarrow B)}{\{A\}\ c\ \{\Downarrow e\} B}$$

Proving Timed Correctness II

Remarks:

(asgn) Assignment can be executed in constant time as size of expressions bounded by a constant

(seq) e_2 expresses time requirements of c_2 relative to initial state of c_2
$\Rightarrow$ cannot use $e_1 + e_2$ as time bound for $c_1\,;c_2$
$\Rightarrow$ replace e_2 by e_2' such that value of e_2' in initial state of c_1 bounds value of e_2 in initial state of c_2 (= final state of c_1)

(while) e_0/e represents execution time for body/whole loop
$\Rightarrow$ cannot use $e_0 + e$ for total time as e/e_0 refers to state before/after body is executed once
$\Rightarrow$ introduce e' whose evaluation before body bounds e evaluated after body
$\Rightarrow$ $e \geq e_0 + e'$ as e has to bound loop execution time independently of number of iterations (recurrence (in-)equation; cf. examples)