FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Formal Semantics of Programming Languages

Florian Zuleger

SS 2014

The language *While*

$E \in \mathit{Arith} ::= x \mid n \mid E + E \mid E * E \mid \ldots$

$B \in \mathit{Bool} ::= \texttt{true} \mid \texttt{false} \mid E = E \mid E \leq E$
$\mid B \wedge B \mid \neg B$

$C \in \mathit{Com} ::= x := E \mid \texttt{if } B \texttt{ then } C \texttt{ else } C \mid C \texttt{;} C$
$\mid \texttt{skip} \mid \texttt{while } B \texttt{ do } C$

x is taken from some set of variables *Var*

Example Program

```
y := 1;
while ¬ (x=1) do
  y := y * x;
  x := x – 1
```

Questions:

- What is the behavior of this program?
- How is this program executed?

Evaluating Expressions

Value of expressions depend on current values of the variables, e.g., the value of

- x + y -1

depends on current values of variable x and y

Values stored at variables change as programs are executed

States

- A **state** s is a function from the variables to the integer, i.e.,
  $s \in \text{Var} \to \mathbb{Z}$.
- The **updated** state $s[x \mapsto n]$ is defined by

$$s[x \mapsto n](y) = \begin{cases} n & \text{if } y = x \\ s(y) & \text{if } y \neq x \end{cases}$$

- Behavior of commands is relative to a state.
- The state changes as the execution of a command proceeds.
- Complete execution of a command transforms an initial state into a final state.

Arithmetic Expressions With States

We add states to our earlier definition of denotational semantic:

$[\![x]\!]\, s = s(x)$,

$[\![\mathbf{n}]\!]\, s = n$, using $n = \text{number}(\mathbf{n})$

$[\![E_1 + E_2]\!]\, s = [\![E_1]\!]\, s + [\![E_2]\!]\, s$,

$[\![E_1 * E_2]\!]\, s = [\![E_1]\!]\, s * [\![E_2]\!]\, s$,

…

The denotational semantics of Boolean expressions can be defined similarly…

Big-step Semantics of *While*

Judgements:

$\langle C,s \rangle \Downarrow s'$

Meaning:

Command C with **initial state** s evaluates to **final state** s' when executing C on s.

Big-step Semantics of *While*

$$\text{B-ASS}\ \frac{}{\langle x := E, s\rangle \Downarrow s[x \mapsto [\![E]\!]\, s]}$$

$$\text{B-SKIP}\ \frac{}{\langle \texttt{skip}, s\rangle \Downarrow s}$$

$$\text{B-SEQ}\ \frac{\langle C_1, s\rangle \Downarrow s_1 \quad \langle C_2, s_1\rangle \Downarrow s'}{\langle C_1 ; C_2, s\rangle \Downarrow s'}$$

$$\text{B-IF.T}\ \frac{\langle C_1, s\rangle \Downarrow s'}{\langle \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2, s\rangle \Downarrow s'}\ [\![B]\!]\, s = \texttt{true}$$

$$\text{B-IF.F}\ \frac{\langle C_2, s\rangle \Downarrow s'}{\langle \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2, s\rangle \Downarrow s'}\ [\![B]\!]\, s = \texttt{false}$$

$$\text{B-WHILE.F}\ \frac{}{\langle \texttt{while}\ B\ \texttt{do}\ C, s\rangle \Downarrow s}\ [\![B]\!]\, s = \texttt{false}$$

$$\text{B-WHILE.T}\ \frac{\langle C, s\rangle \Downarrow s_1 \qquad \langle \texttt{while}\ B\ \texttt{do}\ C, s_1\rangle \Downarrow s'}{\langle \texttt{while}\ B\ \texttt{do}\ C, s\rangle \Downarrow s'}\ [\![B]\!]\, s = \texttt{true}$$

Assignments

Evaluate the command (x := E) relative to state s?

Intuition:

(1) evaluate E relative to state s to some value n = $[\![E]\!]\ s$

(2) update state s with new value n for variable x

Inference Rule:

$$\text{B-ASS}\ \frac{}{\langle x := E, s\rangle \Downarrow s[x \mapsto [\![E]\!]\ s]}$$

We could also have used big-step or small-step semantics for expressions!

The Skip Command

Evaluate command `skip` relative to state s?

Intuition:

(1) nothing to do

Inference Rule:

$$\text{B-SKIP}\quad \frac{}{\langle \texttt{skip}, s\rangle \Downarrow s}$$

Sequential Composition

Evaluate command C_1 ; C_2 relative to state s?

<u>Intuition:</u>

(1) evaluate C_1 relative to state s, to get new state s_1

(2) then evaluate C_2 relative to new state s_1

Rule:

$$\text{B-SEQ} \quad \frac{\langle C_1, s\rangle \Downarrow s_1 \qquad \langle C_2, s_1\rangle \Downarrow s'}{\langle C_1 ; C_2, s\rangle \Downarrow s'}$$

If Commands

Evaluate command `if B then` C_1 `else` C_2 relative to state s?

<u>Intuition:</u>

(1) first evaluate B to some Boolean value b
(2) if b equals true evaluate C_1 relative to state s
(3) if b equals false evaluate C_2 relative to state s

Rules:

$$\text{B-IF.T}\quad \frac{\langle C_1, s\rangle \Downarrow s'}{\langle \texttt{if B then } C_1 \texttt{ else } C_2, s\rangle \Downarrow s'} \quad [\![B]\!]\, s = \texttt{true}$$

$$\text{B-IF.F}\quad \frac{\langle C_2, s\rangle \Downarrow s'}{\langle \texttt{if B then } C_1 \texttt{ else } C_2, s\rangle \Downarrow s'} \quad [\![B]\!]\, s = \texttt{false}$$

While Commands

Evaluate command `while B do C` relative to state s?

Intuition:

(1) first evaluate B to some Boolean value b
(2) if b equals false nothing to be done
(3) if b equals true evaluate C with state s to get new state s_1
(4) then evaluate original `while B do C` relative to s_1

Rules:

$$\text{B-WHILE.F}\ \frac{}{\langle \texttt{while B do C}, s\rangle \Downarrow s}\ [\![\texttt{B}]\!]\, s = \texttt{false}$$

$$\text{B-WHILE.T}\ \frac{\langle \texttt{C}, s\rangle \Downarrow s_1 \quad \langle \texttt{while B do C}, s_1\rangle \Downarrow s'}{\langle \texttt{while B do C}, s\rangle \Downarrow s'}\ [\![\texttt{B}]\!]\, s = \texttt{true}$$

Example

Factorial: $y := 1; \texttt{while} \neg (x=1) \texttt{ do } (y := y * x; x := x - 1)$

We assume $s_1(x) = 2$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\langle y := 1, s_1\rangle \Downarrow s_2}\,\text{B-ASS} \quad \dfrac{\dfrac{\dfrac{}{\langle y := y * x, s_2\rangle \Downarrow s_3}\,\text{B-ASS} \quad \dfrac{}{\langle x := x - 1, s_3\rangle \Downarrow s_4}\,\text{B-ASS}}{\langle y := y * x; x := x - 1, s_2\rangle \Downarrow s_4}\,\text{B-SEQ} \quad \dfrac{}{\langle \texttt{while} \neg (x=1) \texttt{ do } (y := y * x; x := x - 1), s_4\rangle \Downarrow s_4}\,\text{B-WHILE.F}}{\langle \texttt{while} \neg (x=1) \texttt{ do } (y := y * x; x := x - 1), s_2\rangle \Downarrow s_4}\,\text{B-WHILE.T} \quad [\![\neg(x=1)]\!]\, s = \texttt{true}}{\langle y := 1; \texttt{while} \neg (x=1) \texttt{ do } (y := y * x; x := x - 1), s_1\rangle \Downarrow s_4}\,\text{B-SEQ}$$

$[\![\neg(x=1)]\!]\, s_5 = \texttt{false}$

Non-termination

Let C be the command

$$\texttt{while}\ \neg\,(x=1)\ \texttt{do}\ (y := y * x;\ x := x - 1).$$

We consider $s = \{x \mapsto -1,\ y \mapsto 1\}$.

Is there any s' such that $\langle C,s\rangle \Downarrow s'$?

Non-termination

Proof by contradiction:

Assume there is an s with s(x) $\leq$ 0 and we can prove $\langle C,s\rangle \Downarrow s'$ for some s'.

Let s be a state such that $\langle C,s\rangle \Downarrow s'$ can be proven with the *fewest* number of rule applications, say k rule applications.

By definition of the *While* the last step in deriving $\langle C,s\rangle \Downarrow s'$ was by the rule:

$$\frac{\langle C,s\rangle \Downarrow s_1 \qquad \langle \texttt{while} \neg (x=1) \texttt{ do } (y := y * x;\ x := x - 1), s_1\rangle \Downarrow s'}{\langle \texttt{while} \neg (x=1) \texttt{ do } (y := y * x;\ x := x - 1), s\rangle \Downarrow s'}\ \text{B-WHILE.T}$$

We see $\langle C,s_1\rangle \Downarrow s'$ in the hypothesis, that means, $\langle C,s_1\rangle \Downarrow s'$ can be proved by fewer than k rule applications! Contradiction.

Properties of Big-step Semantics

Termination (aka Normalisation):

For every state s and every command C there exists some state s' such that $\langle C,s\rangle \Downarrow s'$. **NO**

Determinacy:

If $\langle C,s\rangle \Downarrow s_1$ and $\langle C,s\rangle \Downarrow s_2$ then $s_1 = s_2$. **YES**

The Meaning of Commands

$[\![\text{-}]\!]_B$: *Com* $\rightarrow$ *States* $\rightharpoonup$ *States*

$[\![C]\!]_B$ transforms an initial state s into a final state.

Definition:

$$[\![C]\!]_B(s) = \begin{cases} s' & \text{if } \langle C,s\rangle \Downarrow s' \\ \perp & \text{otherwise} \end{cases}$$

Determinacy ensures this is proper definition.
$\perp$ stands for 'undefined'.

Example

Let C be the command

y := 1; `while` ¬ (x=1) `do` (y := y * x; x := x – 1).

What is $[\![C]\!]_B$?

$[\![C]\!]_B(s)$ is a state, where

$$[\![C]\!]_B(s) = \begin{cases} \{x \mapsto 1 \,,\, y \mapsto s(x)!\} & \text{if } s(x) \geq 1 \\ \bot & \text{otherwise} \end{cases}$$

Proof? (Mathematical induction on s(x))

Semantic Equivalence

Two Commands C_1 and C_2 are **semantically equivalent**, iff $[\![C_1]\!]_B = [\![C_2]\!]_B$.

Lemma

$[\![\texttt{while}\ B\ \texttt{do}\ C]\!]_B =$

$[\![\texttt{if}\ B\ \texttt{then}\ (C;\texttt{while}\ B\ \texttt{do}\ C)\ \texttt{else skip}]\!]_B$.

Proof

We prove split the proof into

$\langle \texttt{while}\ B\ \texttt{do}\ C, s\rangle \Downarrow s'$ implies
$\langle \texttt{if}\ B\ \texttt{then}\ (C;\texttt{while}\ B\ \texttt{do}\ C)\ \texttt{else skip}, s\rangle \Downarrow s'$

and

$\langle \texttt{if}\ B\ \texttt{then}\ (C;\texttt{while}\ B\ \texttt{do}\ C)\ \texttt{else skip}, s\rangle \Downarrow s'$ implies
$\langle \texttt{while}\ B\ \texttt{do}\ C, s\rangle \Downarrow s'$.

$\langle \texttt{while}\ B\ \texttt{do}\ C, s\rangle \Downarrow s'$ implies $\langle \texttt{if}\ B\ \texttt{then}\ (C;\texttt{while}\ B\ \texttt{do}\ C)\ \texttt{else skip}, s\rangle \Downarrow s'$

Assume $\langle \texttt{while}\ B\ \texttt{do}\ C, s\rangle \Downarrow s'$ and $[\![B]\!]\, s = \texttt{true}$.

By B-WHILE.T there is a proof for $\langle \texttt{while}\ B\ \texttt{do}\ C, s\rangle \Downarrow s'$, where T_1 is a proof for $\langle C, s\rangle \Downarrow s_1$ and T_2 is a proof for $\langle \texttt{while}\ B\ \texttt{do}\ C, s_1\rangle \Downarrow s'$:

$$\text{B-WHILE.T}\ \frac{T_1 \qquad T_2}{\langle \texttt{while}\ B\ \texttt{do}\ C, s\rangle \Downarrow s'}$$

We construct a new proof

$$\text{B-IF.T}\ \frac{\text{B-SEQ}\ \dfrac{T_1 \qquad T_2}{\langle C\ ;\ \texttt{while}\ B\ \texttt{do}\ C, s\rangle \Downarrow s'}}{\langle \texttt{if}\ B\ \texttt{then}\ (C;\texttt{while}\ B\ \texttt{do}\ C)\ \texttt{else skip}, s\rangle \Downarrow s'}$$

$\langle$`while B do C`,s$\rangle \Downarrow$ s' implies $\langle$`if B then (C;while B do C) else skip`,s$\rangle \Downarrow$ s'

Assume $\langle$`while B do C`,s$\rangle \Downarrow$ s' and $[\![$B$]\!]$ s = `false`.

By B-WHILE.F we have:

$$\text{B-WHILE.F}\ \frac{}{\langle \texttt{while B do C},s\rangle \Downarrow s}$$

We construct a new proof:

$$\text{B-IF.F}\ \frac{\text{B-SKIP}\ \dfrac{}{\langle \texttt{skip},s\rangle \Downarrow s}}{\langle \texttt{if B then (C;while B do C) else skip},s\rangle \Downarrow s'}$$

$\langle$`if B then (C;while B do C) else skip`,s$\rangle$ $\Downarrow$ s' implies $\langle$`while B do C`,s$\rangle$ $\Downarrow$ s'.

Assume $\langle$`if B then (C;while B do C) else skip`,s$\rangle$ $\Downarrow$ s' and $[\![B]\!]$ s = `true`.

This means there is a proof:

$$\text{B-IF.T}\ \dfrac{\text{B-SEQ}\ \dfrac{T_1 \qquad T_2}{\langle C\ ;\ \texttt{while}\ B\ \texttt{do}\ C, s\rangle \Downarrow s'}}{\langle \texttt{if}\ B\ \texttt{then}\ (C;\texttt{while}\ B\ \texttt{do}\ C)\ \texttt{else skip}, s\rangle \Downarrow s'}$$

We construct a new proof:

$$\text{B-WHILE.T}\ \dfrac{T_1 \qquad T_2}{\langle \texttt{while}\ B\ \texttt{do}\ C, s\rangle \Downarrow s'}$$

$\langle$`if B then (C;while B do C) else skip`,s$\rangle$ $\Downarrow$ s' implies $\langle$`while B do C`,s$\rangle$ $\Downarrow$ s'.

Assume $\langle$`if B then (C;while B do C) else skip`,s$\rangle$ $\Downarrow$ s' and $[\![$B$]\!]$ s = `false`.

This means there is a proof:

$$\text{B-IF.F}\ \dfrac{\text{B-SKIP}\ \dfrac{}{\langle \texttt{skip},s\rangle \Downarrow s'}}{\langle \texttt{if B then (C;while B do C) else skip},s\rangle \Downarrow s'}$$

By the B-SKIP rule we must have s=s'!

We construct a new proof:

$$\text{B-WHILE.F}\ \dfrac{}{\langle \texttt{while B do C},s\rangle \Downarrow s}$$

Small-step Semantics of *While*

Judgements:

$\langle C,s \rangle \rightarrow \langle C',s' \rangle$ or $\langle C,s \rangle \rightarrow s'$

Meaning:

Starting from state s when executing command C one step of computation leads to state s' with command C' remaining to be executed or simply to state s' when there is no command C remaining.

What is a step?

Depends..

What is in a step?

Decision:

- Ignore how expressions and Booleans are evaluated
- One step consists of
  - state update
  - or branching decision

We want to concentrate on the execution of commands!

Idea: We consider states s as **terminal** configurations.

We will ensure that there is no rule s $\rightarrow$ … .

Small-step Semantics of *While*

S-ASS
$$\frac{}{\langle x := E, s\rangle \to s[x \mapsto [\![E]\!]\, s]}$$

S-SKIP
$$\frac{}{\langle \texttt{skip}, s\rangle \to s}$$

S-SEQ.STEP
$$\frac{\langle C_1, s\rangle \to \langle C_1', s'\rangle}{\langle C_1 ; C_2, s\rangle \to \langle C_1' ; C_2, s'\rangle}$$

S-SEQ.FINAL
$$\frac{\langle C_1, s\rangle \to s'}{\langle C_1 ; C_2, s\rangle \to \langle C_2, s'\rangle}$$

S-IF.T
$$\frac{[\![B]\!]\, s = \texttt{true}}{\langle \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, s\rangle \to \langle C_1, s\rangle}$$

S-IF.F
$$\frac{[\![B]\!]\, s = \texttt{false}}{\langle \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, s\rangle \to \langle C_2, s\rangle}$$

S-WHILE.F
$$\frac{[\![B]\!]\, s = \texttt{false}}{\langle \texttt{while } B \texttt{ do } C, s\rangle \to s}$$

S-WHILE.T
$$\frac{[\![B]\!]\, s = \texttt{true}}{\langle \texttt{while } B \texttt{ do } C, s\rangle \to \langle C ; \texttt{while } B \texttt{ do } C, s\rangle}$$

Assignments

Execute the command (x := E) relative to state s?

Intuition:

(1) evaluate E relative to state s to some value n = $[\![E]\!]\ s$

(2) update state s with new value n for variable x

Inference Rule:

$$\text{S-ASS}\quad \frac{}{\langle x := E, s\rangle \rightarrow s[x \mapsto [\![E]\!]\ s]}$$

One step suffices for entire execution - ignoring the evaluation of E.

The Skip Command

Execute command `skip` relative to state s?

Intuition:

- `skip` has no effect on the state s, i.e., the final state equals the initial state

Rule:

$$\text{S-SKIP}\ \frac{}{\langle \texttt{skip}, s\rangle \rightarrow s}$$

Sequential Composition

Execute command C_1 ; C_2 relative to state s?

Intuition:

(1) execute one step of C_1 relative to state s

(2) if has terminated start executing C_2

Rules:

$$\text{S-SEQ.STEP}\quad \frac{\langle C_1, s\rangle \rightarrow \langle C_1', s'\rangle}{\langle C_1 ; C_2, s\rangle \rightarrow \langle C_1' ; C_2, s'\rangle}$$

$$\text{S-SEQ.FINAL}\quad \frac{\langle C_1, s\rangle \rightarrow s'}{\langle C_1 ; C_2, s\rangle \rightarrow \langle C_2, s'\rangle}$$

If Commands

Execute command `if` B `then` C_1 `else` C_2 relative to state s?

<u>Intuition:</u>

(1) first evaluate B to some Boolean value b
(2) if b equals true execute C_1 relative to state s
(3) if b equals false execute C_2 relative to state s

Rules:

$$\text{S-IF.T}\quad \frac{}{\langle \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2, s\rangle \rightarrow \langle C_1, s\rangle}\ [\![B]\!]\, s = \texttt{true}$$

$$\text{S-IF.F}\quad \frac{}{\langle \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2, s\rangle \rightarrow \langle C_2, s\rangle}\ [\![B]\!]\, s = \texttt{false}$$

While Commands

Execute command `while B do C` relative to state s?

Intuition:

(1) first evaluate B to some Boolean value b

(2) if b equals false then terminate

(3) if b equals true create a copy of C to be executed before `while B do C`

Rules:

$$\text{S-WHILE.F}\ \frac{}{\langle \texttt{while B do C}, s\rangle \rightarrow s}\ [\![\texttt{B}]\!]\, s = \texttt{false}$$

$$\text{S-WHILE.T}\ \frac{}{\langle \texttt{while B do C}, s\rangle \rightarrow \langle \texttt{C;while B do C}, s\rangle}\ [\![\texttt{B}]\!]\, s = \texttt{true}$$

Alternative: While Commands

Execute command `while B do C` relative to state s?

Intuition:

- combination of (`if B then` C_1 `else` C_2) and sequential composition

Rules:

S-WHILE

$$\frac{}{\langle \texttt{while B do C}, s\rangle \rightarrow \langle \texttt{if B then C;while B do C else skip}, s\rangle}$$

Example

Factorial: y := 1; `while` ¬ (x=1) do (y := y * x; x := x – 1)

We assume $s_1(x) = 2$.

Small-step vs Big-step Semantics

$$\text{S-SEQ.STEP}\quad \frac{\langle C_1,s\rangle \rightarrow \langle C_1',s'\rangle}{\langle C_1 ; C_2,s\rangle \rightarrow \langle C_1' ; C_2,s'\rangle}$$

$$\text{S-SEQ.FINAL}\quad \frac{\langle C_1,s\rangle \rightarrow s'}{\langle C_1 ; C_2,s\rangle \rightarrow \langle C_2,s'\rangle}$$

$$\text{B-SEQ}\quad \frac{\langle C_1,s\rangle \Downarrow s_1 \qquad \langle C_2,s_1\rangle \Downarrow s'}{\langle C_1 ; C_2,s\rangle \Downarrow s'}$$

Small-step semantics requires additional bookkeeping!

Running Commands

To run command C from state s:

Find state s' such that $\langle C,s \rangle \rightarrow^* s'$.

Configurations s are **terminal**.

Note that there is no rule such that $s \rightarrow \ldots$.

Running commands: Infinite Loops

Let C be command `while true do skip`.

$\langle$C,s$\rangle \rightarrow \langle$`skip`;C, s$\rangle \rightarrow \langle$C,s$\rangle \rightarrow \langle$`skip`;C,s$\rangle \rightarrow$ …

There is no state s' such that $\langle$C,s$\rangle \rightarrow^*$ s'. (Proof?)

<u>Progress property:</u>

Congurations s are terminal.

For every configuration $\langle$C,s$\rangle$ there is either a step $\langle$C,s$\rangle \rightarrow \langle$C',s'$\rangle$ or a step $\langle$C,s$\rangle \rightarrow$ s'.

Properties of Small-step Semantics

Lemma:

If $\langle C_1;C_2,s\rangle \rightarrow^k s'$ then there exists a state s'' and natural numbers k_1 and k_2 such that $\langle C_1;s\rangle \rightarrow^{k_1} s''$ and $\langle C_2,s''\rangle \rightarrow^{k_2} s'$ where $k_1 + k_2 = k$.

Lemma:

If $\langle C_1,s\rangle \rightarrow^k s'$ then $\langle C_1;C_2,s\rangle \rightarrow^k \langle C_2,s'\rangle$.

Determinacy:

If $\langle C,s\rangle \rightarrow^* s_1$ and $\langle C,s\rangle \rightarrow^* s_2$ then $s_1 = s_2$.

The Meaning of Commands

$[\![\text{-}]\!]_S$: *Com* $\rightarrow$ *States* $\rightharpoonup$ *States*

$[\![C]\!]_S$ transforms an initial state s into a final (aka terminal) state

Definition:

$$[\![C]\!]_S(s) = \begin{cases} s' & \text{if } \langle C,s\rangle \rightarrow^* s' \\ \perp & \text{otherwise} \end{cases}$$

Determinacy ensures this is proper definition.

$\perp$ stands for 'undefined'.

An Equivalence Result

Theorem

For all commands C we have $[\![C]\!]_B = [\![C]\!]_S$.

Proof

We split the proof into the two lemmas for the cases

$$\langle C,s\rangle \to^* s' \text{ implies } \langle C,s\rangle \Downarrow s'$$

and

$$\langle C,s\rangle \Downarrow s' \quad \text{implies } \langle C,s\rangle \to^* s'.$$

$\langle C,s\rangle \Downarrow s'$ implies $\langle C,s\rangle \rightarrow^* s'$

The proof proceeds by rule induction on $\langle C,s\rangle \Downarrow s'$.

Case B-ASS:

We assume $\langle x := E,s\rangle \Downarrow s[x \mapsto [\![E]\!] s]$.

From S-ASS we get $\langle x := E,s\rangle \rightarrow s[x \mapsto [\![E]\!] s]$.

Case B-SKIP: similar

$\langle C,s\rangle \Downarrow s'$ implies $\langle C,s\rangle \rightarrow^* s'$

Case B-SEQ:

We assume $\langle C_1 ; C_2,s\rangle \Downarrow s'$ has been derived from $\langle C_1,s\rangle \Downarrow s''$ and $\langle C_2,s''\rangle \Downarrow s'$.

The induction hypothesis can be applied to both premises $\langle C_1,s\rangle \Downarrow s''$ and $\langle C_2,s''\rangle \Downarrow s'$.

This gives us $\langle C_1,s\rangle \rightarrow^* s''$ and $\langle C_2,s''\rangle \rightarrow^* s'$.

According to the lemma on slide 38 we have

$$\langle C_1 ; C_2,s\rangle \rightarrow^* \langle C_2,s''\rangle.$$

Thus $\langle C_1 ; C_2,s\rangle \rightarrow^* s'$.

$\langle C,s\rangle \Downarrow s'$ implies $\langle C,s\rangle \rightarrow^* s'$

Case B-IF.T:

We assume $\langle \texttt{if B then } C_1 \texttt{ else } C_2,s\rangle \Downarrow s'$ has been derived from $\langle C_1,s\rangle \Downarrow s'$ and $[\![B]\!]\, s = \texttt{true}$.

The induction hypothesis can be applied to the premise $\langle C_1,s\rangle \Downarrow s'$.

This gives us $\langle C_1,s\rangle \rightarrow^* s'$.

From S-ASS we get $\langle \texttt{if B then } C_1 \texttt{ else } C_2,s\rangle \rightarrow \langle C_1,s\rangle$.

Thus $\langle \texttt{if B then } C_1 \texttt{ else } C_2,s\rangle \rightarrow^* s'$.

Case B-IF.F: Analogous.

$\langle C,s\rangle \Downarrow s'$ implies $\langle C,s\rangle \rightarrow^* s'$

Case B-WHILE.T:

We assume $\langle \texttt{while}\ B\ \texttt{do}\ C,s\rangle \Downarrow s'$ has been derived from $\langle C,s\rangle \Downarrow s''$, $\langle \texttt{while}\ B\ \texttt{do}\ C,s''\rangle \Downarrow s'$ and $[\![B]\!]\, s = \texttt{true}$.

The induction hypothesis can be applied to both premises $\langle C,s\rangle \Downarrow s''$ and $\langle \texttt{while}\ B\ \texttt{do}\ C,s''\rangle \Downarrow s'$.

This gives us $\langle C,s\rangle \rightarrow^* s''$ and $\langle \texttt{while}\ B\ \texttt{do}\ C,s''\rangle \rightarrow^* s'$.

According to the lemma on slide 38 we have

$$\langle C\ ;\ \texttt{while}\ B\ \texttt{do}\ C,s\rangle \rightarrow^* \langle \texttt{while}\ B\ \texttt{do}\ C,s''\rangle.$$

From S-WHILE.T we get

$$\langle \texttt{while}\ B\ \texttt{do}\ C,s'\rangle \rightarrow \langle C\ ;\ \texttt{while}\ B\ \texttt{do}\ C,s\rangle.$$

Together we get $\langle \texttt{while}\ B\ \texttt{do}\ C,s''\rangle \rightarrow^* s'$.

Case B-WHILE.F: Straightforward.

$\langle C,s \rangle \rightarrow^* s'$ implies $\langle C,s \rangle \Downarrow s'$

The proof proceeds by induction on the length of the derivation sequence $\langle C,s \rangle \rightarrow^k s'$, that is by induction on k.

Induction start (k=0): vacuously true!

Induction step: We assume the lemma holds for all $k' \leq k$. We proceed by case distinction on the first step of $\langle C,s \rangle \rightarrow^{k+1} s'$.

Cases S-ASS, S-SKIP: Straightforward.

$\langle C,s\rangle \rightarrow^* s'$ implies $\langle C,s\rangle \Downarrow s'$

Cases S-SEQ.STEP, S-SEQ.FINAL:

In both cases we assume that $\langle C_1 ; C_2,s\rangle \rightarrow^{k+1} s'$.

According to the lemma on slide 38 we have $\langle C_1,s\rangle \rightarrow^{k_1} s''$ and $\langle C_2,s''\rangle \rightarrow^{k_2} s'$ for same state s'' and some numbers k_1 and k_2 with $k_1 + k_2 = k+1$.

Because of $k_1 \leq k$ and $k_2 \leq k$ the induction hypothesis can be applied to both derivation sequences $\langle C_1,s\rangle \rightarrow^{k_1} s''$ and $\langle C_2,s''\rangle \rightarrow^{k_2} s'$.

This gives us $\langle C_1,s\rangle \Downarrow s''$ and $\langle C_2,s''\rangle \Downarrow s'$.

From B-SEQ we get $\langle C_1 ; C_2,s\rangle \Downarrow s'$.

$\langle C,s\rangle \rightarrow^* s'$ implies $\langle C,s\rangle \Downarrow s'$

Case S-IF.T:

We assume $[\![B]\!]\, s = \texttt{true}$ and
$\langle \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, s\rangle \rightarrow \langle C_1,s\rangle \rightarrow^k s'$.

The induction hypothesis can be applied to the derivation sequence $\langle C_1,s\rangle \rightarrow^k s'$.

This gives us $\langle C_1,s\rangle \Downarrow s'$.

From B-IF.T we get $\langle \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2,s\rangle \Downarrow s'$.

Case S-IF.F: Analogous.

$\langle C,s\rangle \rightarrow^* s'$ implies $\langle C,s\rangle \Downarrow s'$

Case S-WHILE.T:

We assume $[\![B]\!]\, s = \texttt{true}$ and
$\langle \texttt{while}\ B\ \texttt{do}\ C, s'\rangle \rightarrow \langle C\ ;\ \texttt{while}\ B\ \texttt{do}\ C, s\rangle \rightarrow^k s'$.

The induction hypothesis can be applied to the derivation sequence $\langle C\ ;\ \texttt{while}\ B\ \texttt{do}\ C, s\rangle \rightarrow^k s'$.

This gives us $\langle C\ ;\ \texttt{while}\ B\ \texttt{do}\ C, s\rangle \Downarrow s'$.

From B-SEQ we get $\langle C, s\rangle \Downarrow s''$ and
$\langle \texttt{while}\ B\ \texttt{do}\ C, s''\rangle \Downarrow s'$ for some state s''.

From B-WHILE.T we get $\langle \texttt{while}\ B\ \texttt{do}\ C, s\rangle \Downarrow s'$.

Case S-WHILE.F: Straightforward.

Mathematical Induction

To prove a property P(n) for every natural number n:

- Base Case:
  - prove P(0) is true using known facts
- Induction Case:
  - assume the **inductive hypothesis**: P(k) is true
  - prove P(k+1) is true using known facts and the inductive hypothesis

Example: $\sum_{i=1..n} (2i - 1) = n^2$

Strong Mathematical Induction

To prove a property P(n) for every natural number n:

- Consider some number k .
- Assume the **inductive hypothesis**: P(k') is true for every number k' < k
- prove P(k) is true using known facts and the inductive hypothesis

Note: There is no distinction between the base case and the induction case!

Example: Every number is a product of prime numbers.

Structural Induction

Consider some inductively defined structure given by **axioms** and **constructors**, e.g.,

Tree ::= Leaf | Branch(Tree,Tree).

To prove a property P(T) for every tree T:

- Base case (T = Leaf):
  - prove P(Leaf) is true using known facts
- Induction case (T = Branch(T_1,T_2)):
  - assume the **inductive hypothesis**: P(T_1) and P(T_2) are true
  - prove P(T) is true using known facts and the inductive hypothesis

Example

leaves(Leaf) = 1

leaves(Branch(T_1,T_2)) = leaves(T_1) + leaves(T_2)

branches(Leaf) = 0

branches(Branch(T_1,T_2)) = branches (T_1) + branches (T_2) + 1

leaves(T) = branches(T) + 1 for every Tree T.

Rule Induction for Deductive Systems

Assume some universe of elements U.

We define a **deductive system** D by

Axioms: an element a of U

Rules:

$$\frac{h_1 \qquad \ldots \qquad h_n}{c}$$

where h_1 ,…, h_n,c are elements of U.

Let D(U) be the set of elements that can be derived from the axioms and rules.

Rule Induction for Deductive Systems

To prove a property P(t) for every $t \in D(U)$:

- Base Case (for every axiom):
  - prove P(a) using known facts
- Induction Case (for every rule):
  - assume the **inductive hypothesis**: $P(h_i)$ is true for every hypothesis h_i
  - prove P(c) is true using known facts and the inductive hypothesis