FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Formal Semantics of Programming Languages

Florian Zuleger

SS 2014

The language *While*

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

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

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

x is taken from some set of variables *Var*

Provably Correct Implementation

The formal specification of the semantics of a programming language allows to argue about the correctness of a compiler:

- We define an **abstract machine** (e.g. a **stack-based** intermediate language such as *Java bytecode*).
- We define small-step semantics for this machine.
- We define a **translation** of *While* into assembly code for the abstract machine.
- We prove that code translation and execution on the abstract machine are **semantics preserving** for every command of *While*.

The *Abstract Machine* (*AM*)

inst $\in$ *Inst* ::= PUSH-n | ADD | MULT |
| TRUE | FALSE | EQ | LE | AND| NEG
| FETCH-x | STORE-x | NOOP
| BRANCH(c,c)| LOOP(c,c)

c $\in$ *Code* ::= ϵ | inst:c

AM Configurations

AM has configurations $\langle$c,e,s$\rangle$:

- c is the **code** (sequence of instructions) to be executed,
- e is the **evaluation stack**,
- s is the **storage**.

We have e $\in$ *Stack* = $(\mathbb{Z} \cup \mathbb{T})^*$,
where $\mathbb{T}$ = {true, false}.

For simplicity we assume s $\in$ State.

Thus $\langle$c,e,s$\rangle$ $\in$ *Code* $\times$ *Stack* $\times$ *State*.

Small-step Semantics of *AM*

Judgements:

$\langle c,e,s \rangle \triangleright \langle c',e',s' \rangle$

Meaning:

One step of execution transforms a configuration $\langle c,e,s \rangle$ into $\langle c',e',s' \rangle$.

Terminal configurations:

A configuration is terminal, if its code component is the empty sequence: $\langle \epsilon,e,s \rangle$

Small-step Semantics of *AM*

$$\langle \text{PUSH-n:c,e,s} \rangle \triangleright \langle \text{c,}[\![\text{n}]\!]\text{:e,s} \rangle$$

$$\langle \text{ADD:c,}z_1\text{:}z_2\text{:e,s} \rangle \triangleright \langle \text{c,}(z_1 + z_2)\text{:e,s} \rangle, \text{ if } z_1, z_2 \in \mathbb{Z}$$

$$\langle \text{MULT:c,}z_1\text{:}z_2\text{:e,s} \rangle \triangleright \langle \text{c,}(z_1 * z_2)\text{:e,s} \rangle, \text{ if } z_1, z_2 \in \mathbb{Z}$$

$$\langle \text{TRUE:c,e,s} \rangle \triangleright \langle \text{c,true:e,s} \rangle$$

$$\langle \text{FALSE:c,e,s} \rangle \triangleright \langle \text{c,false:e,s} \rangle$$

$$\langle \text{EQ:c,}z_1\text{:}z_2\text{:e,s} \rangle \triangleright \langle \text{c, }(z_1 = z_2)\text{:e,s} \rangle, \text{ if } z_1, z_2 \in \mathbb{Z}$$

$$\langle \text{LE:c,}z_1\text{:}z_2\text{:e,s} \rangle \triangleright \langle \text{c, }(z_1 \leq z_2)\text{:e,s} \rangle, \text{ if } z_1, z_2 \in \mathbb{Z}$$

$$\langle \text{AND:c,}t_1\text{:}t_2\text{:e,s} \rangle \triangleright \langle \text{c,}(t_1 \wedge t_2)\text{:e,s} \rangle, \text{ if } t_1, t_2 \in \mathbb{T}$$

$$\langle \text{NEG:c,t:e,s} \rangle \triangleright \langle \text{c,}(\neg t)\text{:e,s} \rangle, \text{ if } t \in \mathbb{T}$$

Small-step Semantics of *AM*

$\langle \text{FETCH-x:c,e,s} \rangle \quad \triangleright \quad \langle \text{c,s(x):e,s} \rangle$

$\langle \text{STORE-x:c,z:e,s} \rangle \quad \triangleright \quad \langle \text{c,e,s}[x \mapsto z] \rangle$

$\langle \text{NOOP:c,e,s} \rangle \quad \triangleright \quad \langle \text{c,e,s} \rangle$

$$\langle \text{BRANCH}(c_1,c_2)\text{:c,t:e,s} \rangle \quad \triangleright \quad \begin{cases} \langle c_1\text{:c,e,s} \rangle, & \text{if t=true} \\ \langle c_2\text{:c,e,s} \rangle, & \text{if t=false} \end{cases}$$

$$\langle \text{LOOP}(c_1,c_2)\text{:c,e,s} \rangle \triangleright \langle c_1\text{:BRANCH}(c_2\text{:LOOP}(c_1,c_2), \text{NOOP:c}),\text{e,s} \rangle$$

Example

We assume s(x) = 3.

$\langle$PUSH-1:FETCH-x:ADD:STORE-x,ϵ,s$\rangle$

$\triangleright$ $\langle$FETCH-x:ADD:STORE-x,1,s$\rangle$

$\triangleright$ $\langle$ADD:STORE-x,3:1,s$\rangle$

$\triangleright$ $\langle$STORE-x,4,s$\rangle$

$\triangleright$ $\langle\epsilon,\epsilon,\text{s}[\text{x} \mapsto 4]\rangle$

Non-termination

$\langle$LOOP(TRUE,NOOP),ϵ,s$\rangle$

$\triangleright$ $\langle$TRUE:BRANCH(NOOP:LOOP(TRUE,NOOP),
NOOP), ϵ,s$\rangle$

$\triangleright$ $\langle$BRANCH(NOOP:LOOP(TRUE,NOOP),
NOOP),true,s$\rangle$

$\triangleright$ $\langle$NOOP:LOOP(TRUE,NOOP),true,s$\rangle$

$\triangleright$ $\langle$LOOP(TRUE,NOOP),true,s$\rangle$

$\triangleright$ …

Properties of *AM*

Lemma:

If $\langle c_1{:}c_2,e,s\rangle \rightarrow^k \langle \epsilon,e',s'\rangle$ s' then there exists a configuration $\langle \epsilon,e'',s''\rangle$ and natural numbers k_1 and k_2 such that $\langle c_1,e,s\rangle \rightarrow^{k_1} \langle \epsilon,e'',s''\rangle$ and $\langle c_2,e'',s''\rangle \rightarrow^{k_2} \langle \epsilon,e',s'\rangle$ where $k_1 + k_2 = k$.

Lemma:

If $\langle c,e,s\rangle \rhd^k \langle c',e',s'\rangle$ then $\langle c_1{:}c_2,e_1{:}e_2,s\rangle \rhd^k \langle c'{:}c_2,e'{:}e_2,s'\rangle$

Determinacy:

If $\langle c,e,s\rangle \rhd^* \gamma_1$ and $\langle c,e,s\rangle \rightarrow^* \gamma_2$ then $\gamma_1 = \gamma_2$.

Stuck Configurations

AM has stuck configurations:

- $\langle$ADD,true:10,s$\rangle$
- $\langle$NEG,5,s$\rangle$
- …

These configurations arise because of **type errors**! (We could add error states to the abstract machine configurations…)

The Meaning of Commands

$[\![\text{-}]\!]_{AM}$: *Code* $\rightarrow$ *States* $\rightharpoonup$ *States*

$[\![c]\!]_{AM}$ transforms an initial state s into a final (aka terminal) state

Definition:

$$[\![c]\!]_{AM}(s) = \begin{cases} s' & \text{if } \langle c,\epsilon,s\rangle \rightarrow^* \langle \epsilon,e,s'\rangle \\ \perp & \text{otherwise} \end{cases}$$

Determinacy ensures this is proper definition.

$\perp$ stands for 'undefined'.

Translation of *While* to *AM*

We define three (total) functions that translate *While* commands into *AM* code:

$\mathcal{CA}[\![\ ]\!]$: *Arith* $\rightarrow$ *Code*

$\mathcal{CB}[\![\ ]\!]$: *Bool* $\rightarrow$ *Code*

$\mathcal{CC}[\![\ ]\!]$: *Com* $\rightarrow$ *Code*

We will define these function in a **compositional manner** (i.e. by structural induction).

Translation of Arithmetic Expressions

$$
\begin{array}{lcl}
\mathcal{CA}[\![\mathsf{n}]\!] & = & \mathsf{PUSH\text{-}n} \\
\mathcal{CA}[\![\mathsf{x}]\!] & = & \mathsf{FETCH\text{-}x} \\
\mathcal{CA}[\![\mathsf{E}_1 + \mathsf{E}_2]\!] & = & \mathcal{CA}[\![\mathsf{E}_2]\!]{:}\mathcal{CA}[\![\mathsf{E}_1]\!]{:}\mathsf{ADD} \\
\mathcal{CA}[\![\mathsf{E}_1 * \mathsf{E}_2]\!] & = & \mathcal{CA}[\![\mathsf{E}_2]\!]{:}\mathcal{CA}[\![\mathsf{E}_1]\!]{:}\mathsf{MULT}
\end{array}
$$

Translation of Boolean expressions is defined similarly.

Translation of Commands

$$\mathcal{CC}[\![\texttt{x := E}]\!] = \mathcal{CA}[\![\texttt{E}]\!]\texttt{:STORE-x}$$

$$\mathcal{CC}[\![\texttt{skip}]\!] = \texttt{NOOP}$$

$$\mathcal{CC}[\![\texttt{C}_1\texttt{;C}_2]\!] = \mathcal{CC}[\![\texttt{C}_1]\!]\texttt{:}\mathcal{CC}[\![\texttt{C}_2]\!]$$

$$\mathcal{CC}[\![\texttt{if B then C}_1\texttt{ else C}_2]\!] = \mathcal{CB}[\![\texttt{B}]\!]\texttt{:BRANCH}(\mathcal{CC}[\![\texttt{C}_1]\!],\mathcal{CC}[\![\texttt{C}_2]\!])$$

$$\mathcal{CC}[\![\texttt{while B do C}]\!] = \texttt{LOOP}(\mathcal{CB}[\![\texttt{B}]\!],\mathcal{CC}[\![\texttt{C}]\!])$$

Note that the definition is compositional.

This guarantees the termination of the translation!

Example

$\mathcal{CC}[\![$y := 1; while $\neg$ (x=1) do (y := y * x; x := x – 1)$]\!]$ =

$\mathcal{CC}[\![$y := 1$]\!]$:$\mathcal{CC}[\![$while $\neg$ (x=1) do (y := y * x; x := x – 1)$]\!]$ =

$\mathcal{CA}[\![1]\!]$:STORE-y:LOOP($\mathcal{CB}[\![\neg$(x=1)$]\!]$,$\mathcal{CC}[\![$y := y * x; x := x – 1$]\!]$) =

PUSH-1:STORE-y:LOOP($\mathcal{CB}[\![$(x=1)$]\!]$:NEG,$\mathcal{CC}[\![$y := y * x$]\!]$:
$\mathcal{CC}[\![$x := x – 1$]\!]$) = … =

PUSH-1:STORE-y:LOOP(FETCH-x:PUSH-1:EQ:NEG,FETCH-x:
FETCH-y:MULT:STORE-y:FETCH-x:PUSH-1:SUB:STORE-y)

Correctness of Arithmetic Expressions

We show that first translating an arithmetic expression into code for *AM* and then executing gives the same result as the semantics of *While*.

Lemma

For all arithmetic expressions E we have

$$\langle \mathcal{CA}[\![\mathsf{E}]\!], \epsilon, \mathsf{s} \rangle \rhd^* \langle \epsilon, [\![\mathsf{E}]\!]\, \mathsf{s}, \mathsf{s} \rangle.$$

Furthermore, all intermediate configurations of this computation have non-empty evaluation stacks.

$$\langle \mathcal{CA}[\![\mathsf{E}]\!], \epsilon, \mathsf{s} \rangle \rhd^* \langle \epsilon, [\![\mathsf{E}]\!]\ \mathsf{s}, \mathsf{s} \rangle$$

Proof By structural induction on the expression E.

Base Case: E is a numeral n. We have $\mathcal{CA}[\![\mathsf{E}]\!]$ = PUSH-n and we get $\langle \mathsf{PUSH\text{-}n}, \epsilon, \mathsf{s} \rangle \rhd \langle \epsilon, [\![\mathsf{n}]\!]\ \mathsf{s}, \mathsf{s} \rangle$.

This solves the case.

Base Case: E is a variable x. We have $\mathcal{CA}[\![\mathsf{x}]\!]$ = FETCH-x and we get $\langle \mathsf{FETCH\text{-}x}, \epsilon, \mathsf{s} \rangle \rhd \langle \epsilon, \mathsf{s}(\mathsf{x}), \mathsf{s} \rangle$.

This solves the case.

$\langle \mathcal{CA}[\![\mathsf{E}]\!],\epsilon,\mathsf{s}\rangle \rhd^* \langle \epsilon,[\![\mathsf{E}]\!]\ \mathsf{s},\mathsf{s}\rangle$

Induction Case:

Suppose E is of the form (E_1 + E_2). (case E_1 * E_2 for is analogous)
We have $\mathcal{CA}[\![\mathsf{E}_1 + \mathsf{E}_2]\!] = \mathcal{CA}[\![\mathsf{E}_2]\!]:\mathcal{CA}[\![\mathsf{E}_1]\!]:\mathsf{ADD}$.

From the induction hypothesis we get
$\langle \mathcal{CA}[\![\mathsf{E}_1]\!],\epsilon,\mathsf{s}\rangle \rhd^* \langle \epsilon,[\![\mathsf{E}_1]\!]\ \mathsf{s},\mathsf{s}\rangle$ and $\langle \mathcal{CA}[\![\mathsf{E}_2]\!],\epsilon,\mathsf{s}\rangle \rhd^* \langle \epsilon,[\![\mathsf{E}_2]\!]\ \mathsf{s},\mathsf{s}\rangle$.

In both cases the intermediate configurations have non-empty evaluation stacks.
From lemma on slide 11 we get $\langle \mathcal{CA}[\![\mathsf{E}_2]\!]:\mathcal{CA}[\![\mathsf{E}_1]\!]:\mathsf{ADD},\epsilon,\mathsf{s}\rangle \rhd^*$
$\langle \mathcal{CA}[\![\mathsf{E}_1]\!]:\mathsf{ADD},\ [\![\mathsf{E}_2]\!]\ \mathsf{s},\mathsf{s}\rangle \rhd^* \langle \mathsf{ADD},([\![\mathsf{E}_1]\!]\ \mathsf{s}):([\![\mathsf{E}_2]\!]\ \mathsf{s}),\mathsf{s}\rangle$.

Furthermore we have $\langle \mathsf{ADD},([\![\mathsf{E}_1]\!]\ \mathsf{s}):([\![\mathsf{E}_2]\!]\ \mathsf{s}),\mathsf{s}\rangle \rhd$
$\langle \epsilon,([\![\mathsf{E}_1]\!]\ \mathsf{s} + [\![\mathsf{E}_2]\!]\ \mathsf{s}),\mathsf{s}\rangle$.

Since $[\![\mathsf{E}_1]\!]\ \mathsf{s} + [\![\mathsf{E}_2]\!]\ \mathsf{s} = [\![\mathsf{E}_1 + \mathsf{E}_2]\!]\ \mathsf{s}$ we get the desired result.

An Equivalence Result

Theorem

For all commands C we have $[\![\mathsf{C}]\!]_{\mathsf{B}} = [\![\mathcal{CC}[\![\mathsf{C}]\!]]\!]_{\mathsf{AM}}$.

Proof

We split the proof into the two lemmas for the cases

$\langle \mathcal{CC}[\![\mathsf{C}]\!], \epsilon, \mathsf{s}\rangle \rhd^{\mathsf{k}} \langle \epsilon, \mathsf{e}, \mathsf{s}'\rangle$ implies $\langle \mathsf{C}, \mathsf{s}\rangle \Downarrow \mathsf{s}'$ and $\mathsf{e}=\epsilon$

and

$\langle \mathsf{C}, \mathsf{s}\rangle \Downarrow \mathsf{s}'$ implies $\langle \mathcal{CC}[\![\mathsf{C}]\!], \epsilon, \mathsf{s}\rangle \rhd^{*} \langle \epsilon, \epsilon, \mathsf{s}'\rangle$.

$\langle C,s\rangle \Downarrow s'$ implies $\langle \mathcal{CC}[\![C]\!],\epsilon,s\rangle \rhd^* \langle \epsilon,\epsilon,s'\rangle$

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

Case B-ASS:

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

We have $\mathcal{CC}[\![x := E]\!] = \mathcal{CA}[\![E]\!]$:STORE-x.

The previous lemma gives us $\langle \mathcal{CA}[\![E]\!],\epsilon,s\rangle \rhd^* \langle \epsilon,[\![E]\!]\, s,s\rangle$.

According to the lemma on slide 11 we have

$$\langle \mathcal{CA}[\![E]\!]\text{:STORE-x},\epsilon,s\rangle \rhd^* \langle \text{STORE-x}, [\![E]\!]\, s,s\rangle \rhd \langle \epsilon,\epsilon, s[x \mapsto [\![E]\!]\, s]\rangle.$$

Case B-SKIP: Straightforward.

$\langle C,s\rangle \Downarrow s'$ implies $\langle \mathcal{CC}[\![C]\!],\epsilon,s\rangle \rhd^* \langle \epsilon,\epsilon,s'\rangle$

Case B-SEQ:

We have $\mathcal{CC}[\![C_1 ; C_2]\!] = \mathcal{CC}[\![C_1]\!]:\mathcal{CC}[\![C_2]\!]$.

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 $\langle \mathcal{CC}[\![C_1]\!],\epsilon,s\rangle \rhd^* \langle \epsilon,\epsilon,s''\rangle$ and $\langle \mathcal{CC}[\![C_2]\!],\epsilon,s''\rangle \rhd^* \langle \epsilon,\epsilon,s'\rangle$.

According to the lemma on slide 11 we have $\langle \mathcal{CC}[\![C_1]\!]:\mathcal{CC}[\![C_2]\!],\epsilon,s\rangle \rhd^* \langle \mathcal{CC}[\![C_2]\!],\epsilon,s''\rangle \rhd^* \langle \epsilon,\epsilon,s'\rangle$.

Thus $\langle \mathcal{CC}[\![C]\!],\epsilon,s\rangle \rhd^* \langle \epsilon,\epsilon,s'\rangle$.

$\langle C,s\rangle \Downarrow s'$ implies $\langle \mathcal{CC}[\![C]\!],\epsilon,s\rangle \triangleright^* \langle \epsilon,\epsilon,s'\rangle$

Case B-IF.T:

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

From the induction hypothesis we get $\langle \mathcal{CC}[\![C_1]\!],\epsilon,s\rangle \triangleright^* \langle \epsilon,\epsilon,s'\rangle$.

We have $\mathcal{CC}[\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!] = \mathcal{CB}[\![B]\!]: \text{BRANCH}(\mathcal{CC}[\![C_1]\!],\mathcal{CC}[\![C_2]\!])$.

We get $\langle \mathcal{CB}[\![B]\!]:\text{BRANCH}(\mathcal{CC}[\![C_1]\!],\mathcal{CC}[\![C_2]\!]) ,\epsilon,s'\rangle \triangleright^* \langle \text{BRANCH}(\mathcal{CC}[\![C_1]\!],\mathcal{CC}[\![C_2]\!]) ,[\![B]\!]\, s,s'\rangle$ using a lemma for Boolean expressions (similar to the lemma for arithmetic expresions on slide 18) and the lemma from slide 11.

Finally we have $\langle \text{BRANCH}(\mathcal{CC}[\![C_1]\!],\mathcal{CC}[\![C_2]\!]) ,[\![B]\!]\, s,s'\rangle \triangleright \langle \mathcal{CC}[\![C_1]\!],\epsilon,s\rangle$ from the small-step semantics of *AM*.

Case B-IF.F: Analogous.

$\langle$C,s$\rangle \Downarrow$ s' implies $\langle \mathcal{CC}[\![\text{C}]\!],\epsilon,\text{s}\rangle \triangleright^* \langle\epsilon,\epsilon,\text{s}'\rangle$

Case B-WHILE.T:

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

From the induction hypothesis to $\langle$C,s$\rangle \Downarrow$ s'' we get $\langle \mathcal{CC}[\![\text{C}]\!],\epsilon,\text{s}\rangle \triangleright^* \langle\epsilon,\epsilon,\text{s}''\rangle$ and $\langle \mathcal{CC}[\![\texttt{while B do C}]\!],\epsilon,\text{s}''\rangle \triangleright^* \langle\epsilon,\epsilon,\text{s}'\rangle$.

We have $\mathcal{CC}[\![\texttt{while B do C}]\!]$ =LOOP($\mathcal{CB}[\![\text{B}]\!],\mathcal{CC}[\![\text{C}]\!]$).

We get $\langle$LOOP($\mathcal{CB}[\![\text{B}]\!],\mathcal{CC}[\![\text{C}]\!],\epsilon$,s$\rangle \triangleright$

$\langle \mathcal{CB}[\![\text{B}]\!]$:BRANCH($\mathcal{CC}[\![\text{C}]\!]$:LOOP($\mathcal{CB}[\![\text{B}]\!],\mathcal{CC}[\![\text{C}]\!]$), NOOP), ϵ,s$\rangle \triangleright^*$

$\langle$BRANCH($\mathcal{CC}[\![\text{C}]\!]$:LOOP($\mathcal{CB}[\![\text{B}]\!],\mathcal{CC}[\![\text{C}]\!]$), NOOP),$\mathcal{CB}[\![\text{B}]\!]$ s,s$\rangle \triangleright$

$\langle \mathcal{CC}[\![\text{C}]\!]$:LOOP($\mathcal{CB}[\![\text{B}]\!],\mathcal{CC}[\![\text{C}]\!]$),$\epsilon$,s$\rangle \triangleright^* \langle$LOOP($\mathcal{CB}[\![\text{B}]\!],\mathcal{CC}[\![\text{C}]\!]$),$\epsilon$,s''$\rangle$.

In the second step we have used a for Boolean expressions (similar to the lemma for arithmetic expresions on slide 18). In the last step we have used the lemma from slide 11.

Case B-WHILE.F: Straightforward.

$\langle \mathcal{CC}[\![\mathsf{C}]\!],\epsilon,\mathsf{s}\rangle \triangleright^{\mathsf{k}} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$ implies $\langle \mathsf{C},\mathsf{s}\rangle \Downarrow \mathsf{s}'$ and $\mathsf{e}=\epsilon$

The proof proceeds by induction on the length of the derivation sequence $\langle \mathcal{CC}[\![\mathsf{C}]\!],\epsilon,\mathsf{s}\rangle \triangleright^{\mathsf{k}} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$, that is by induction on k.

Induction hypothesis: We assume the lemma holds for all $0 \leq \mathsf{k}' \leq \mathsf{k}$. We proceed by case distinction on the first step of $\langle \mathcal{CC}[\![\mathsf{C}]\!],\epsilon,\mathsf{s}\rangle \triangleright^{\mathsf{k+1}} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$.

S-SKIP: Straightforward.

$\langle \mathcal{CC}[\![\mathsf{C}]\!],\epsilon,\mathsf{s}\rangle \rhd^{\mathsf{k}} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$ implies $\langle \mathsf{C},\mathsf{s}\rangle \Downarrow \mathsf{s}'$ and $\mathsf{e}=\epsilon$

Case x := E:

We assume $\langle \mathcal{CA}[\![\mathsf{E}]\!]:\mathsf{STORE\text{-}x},\epsilon,\mathsf{s}\rangle \rhd^{\mathsf{k+1}} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$.

According to the lemma on slide 11 there must be a configuration $\langle \epsilon,\mathsf{e}'',\mathsf{s}''\rangle$ and natural numbers k_1 and k_2 such that $\langle \mathcal{CA}[\![\mathsf{E}]\!],\epsilon,\mathsf{s}\rangle \rightarrow^{k_1} \langle \epsilon,\mathsf{e}'',\mathsf{s}''\rangle$ and

$\langle \mathsf{STORE\text{-}x},\mathsf{e}'',\mathsf{s}''\rangle \rightarrow^{k_2} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$ where $k_1 + k_2 = k+1$.

Due to the lemma on slide 18 and due to the determinacy of *AM* we have $\mathsf{e}'' = [\![\mathsf{E}]\!]\ \mathsf{s}$ and $\mathsf{s}'' = \mathsf{s}$.

By the semantics of STORE-x we get

$\mathsf{s}' = \mathsf{s}[\mathsf{x} \mapsto [\![\mathsf{E}]\!]\ \mathsf{s}]$ and $\mathsf{e} = \epsilon$.

From B-ASS we get $\langle \mathsf{x} := \mathsf{E},\mathsf{s}\rangle \Downarrow \mathsf{s}[\mathsf{x} \mapsto [\![\mathsf{E}]\!]\ \mathsf{s}]$.

$\langle \mathcal{CC}[\![\mathsf{C}]\!],\epsilon,\mathsf{s}\rangle \triangleright^{\mathsf{k}} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$ implies $\langle \mathsf{C},\mathsf{s}\rangle \Downarrow \mathsf{s}'$ and $\mathsf{e}=\epsilon$

Case C_1 ; C_2:
We assume $\mathcal{CC}[\![\mathsf{C}_1]\!]:\mathcal{CC}[\![\mathsf{C}_2]\!] \triangleright^{\mathsf{k+1}} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$.

According to the lemma on slide 11 there must be a configuration $\langle \epsilon,\mathsf{e}'',\mathsf{s}''\rangle$ and natural numbers k_1 and k_2 such that $\langle \mathcal{CC}[\![\mathsf{C}_1]\!],\epsilon,\mathsf{s}\rangle \triangleright^{\mathsf{k}_1} \langle \epsilon,\mathsf{e}'',\mathsf{s}''\rangle$ and $\langle \mathcal{CC}[\![\mathsf{C}_2]\!],\mathsf{e}'',\mathsf{s}''\rangle \triangleright^{\mathsf{k}_2} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$ where $\mathsf{k}_1 + \mathsf{k}_2 = \mathsf{k}+1$.

Because of $\mathsf{k}_1 \leq \mathsf{k}$ the induction hypothesis can be applied and we get $\langle \mathsf{C}_1,\mathsf{s}\rangle \Downarrow \mathsf{s}''$ and $\mathsf{e}''=\epsilon$.

Now we can apply the induction hypothesis again because of $\mathsf{k}_2 \leq \mathsf{k}$ and we get $\langle \mathsf{C}_2,\mathsf{s}''\rangle \Downarrow \mathsf{s}'$ and $\mathsf{e}=\epsilon$.

From $\mathrm{B\text{-}SEQ}$ we get $\langle \mathsf{C}_1 ; \mathsf{C}_2,\mathsf{s}\rangle \Downarrow \mathsf{s}'$.

$\langle \mathcal{CC}[\![\mathsf{C}]\!],\epsilon,\mathsf{s}\rangle \rhd^{\mathsf{k}} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$ implies $\langle \mathsf{C},\mathsf{s}\rangle \Downarrow \mathsf{s}'$ and $\mathsf{e}=\epsilon$

Case `if B then` C_1 `else` C_2:
We assume $\langle \mathcal{CB}[\![\mathsf{B}]\!]:\mathsf{BRANCH}(\mathcal{CC}[\![\mathsf{C}_1]\!],\mathcal{CC}[\![\mathsf{C}_2]\!]),\epsilon,\mathsf{s}\rangle \rhd^{\mathsf{k+1}} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$.

According to the lemma on slide 11 there must be a configuration $\langle \epsilon,\mathsf{e}'',\mathsf{s}''\rangle$ and natural numbers k_1 and k_2 such that $\langle \mathcal{CB}[\![\mathsf{B}]\!],\epsilon,\mathsf{s}\rangle \rhd^{\mathsf{k}_1} \langle \epsilon,\mathsf{e}'',\mathsf{s}''\rangle$ and $\langle \mathsf{BRANCH}(\mathcal{CC}[\![\mathsf{C}_1]\!],\mathcal{CC}[\![\mathsf{C}_2]\!]),\mathsf{e}'',\mathsf{s}''\rangle \rhd^{\mathsf{k}_2} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$ where $\mathsf{k}_1 + \mathsf{k}_2 = \mathsf{k}+1$.

Due to a lemma similar to the one on slide 18 and due to the determinacy of *AM* we have $\mathsf{e}'' = [\![\mathsf{B}]\!]\ \mathsf{s}$ and $\mathsf{s}'' = \mathsf{s}$.

From now on we assume $[\![\mathsf{B}]\!]\ \mathsf{s} = \mathsf{true}$.
Thus $\langle \mathsf{BRANCH}(\mathcal{CC}[\![\mathsf{C}_1]\!],\mathcal{CC}[\![\mathsf{C}_2]\!]), [\![\mathsf{B}]\!]\ \mathsf{s},\mathsf{s}\rangle \rhd \langle \mathcal{CC}[\![\mathsf{C}_1]\!],\epsilon,\mathsf{s}\rangle \rhd^{\mathsf{k}_2-1} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$ by determinacy of *AM* .

Because of $\mathsf{k}_2\text{-}1 \leq \mathsf{k}$ the induction hypothesis can be applied and we get $\langle \mathsf{C}_1,\mathsf{s}\rangle \Downarrow \mathsf{s}''$ and $\mathsf{e}'=\epsilon$.
From B-IF.T we get $\langle$ `if B then` C_1 `else` $\mathsf{C}_2,\mathsf{s}\rangle \Downarrow \mathsf{s}'$.

The case $[\![\mathsf{B}]\!]\ \mathsf{s} = \mathsf{false}$ is analogous.

$\langle \mathcal{CC}[\![\mathsf{C}]\!],\epsilon,\mathsf{s}\rangle \triangleright^{\mathsf{k}} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$ implies $\langle \mathsf{C},\mathsf{s}\rangle \Downarrow \mathsf{s}'$ and $\mathsf{e}=\epsilon$

Case `while B do C`:
We assume $\langle \mathsf{LOOP}(\mathcal{CB}[\![\mathsf{B}]\!],\mathcal{CC}[\![\mathsf{C}]\!]),\epsilon,\mathsf{s}\rangle \triangleright^{\mathsf{k+1}} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$.

Using the semantics of AM we have
$\langle \mathsf{LOOP}(\mathcal{CB}[\![\mathsf{B}]\!],\mathcal{CC}[\![\mathsf{C}]\!]),\epsilon,\mathsf{s}\rangle \triangleright$
$\langle \mathcal{CB}[\![\mathsf{B}]\!]:\mathsf{BRANCH}(\mathcal{CC}[\![\mathsf{C}]\!]:\mathsf{LOOP}(\mathcal{CB}[\![\mathsf{B}]\!],\mathcal{CC}[\![\mathsf{C}]\!]),\mathsf{NOOP}),\epsilon,\mathsf{s}\rangle \triangleright^{\mathsf{k}}$
$\langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$.

According to the lemma on slide 11 there must be a configuration $\langle \epsilon,\mathsf{e}'',\mathsf{s}''\rangle$ and natural numbers k_1 and k_2 such that $\langle \mathcal{CB}[\![\mathsf{B}]\!],\epsilon,\mathsf{s}\rangle \triangleright^{k_1} \langle \epsilon,\mathsf{e}'',\mathsf{s}''\rangle$ and
$\langle \mathsf{BRANCH}(\mathcal{CC}[\![\mathsf{C}]\!]:\mathsf{LOOP}(\mathcal{CB}[\![\mathsf{B}]\!],\mathcal{CC}[\![\mathsf{C}]\!]),\mathsf{NOOP}),\mathsf{e}'',\mathsf{s}''\rangle \triangleright^{k_2}$
$\langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$ where $k_1 + k_2 = k+1$.

Due to a lemma similar to the one on slide 18 and due to the determinacy of *AM* we have $\mathsf{e}'' = [\![\mathsf{B}]\!]\,\mathsf{s}$ and $\mathsf{s}'' = \mathsf{s}$.

$\langle \mathcal{CC}[\![\mathsf{C}]\!],\epsilon,\mathsf{s}\rangle \rhd^{\mathsf{k}} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$ implies $\langle \mathsf{C},\mathsf{s}\rangle \Downarrow \mathsf{s}'$ and $\mathsf{e}=\epsilon$

Case $[\![\mathsf{B}]\!]\ \mathsf{s}$ = true:
Thus $\langle \mathsf{BRANCH}(\mathcal{CC}[\![\mathsf{C}]\!]:\mathsf{LOOP}(\mathcal{CB}[\![\mathsf{B}]\!],\mathcal{CC}[\![\mathsf{C}]\!]), \mathsf{NOOP}),[\![\mathsf{B}]\!]\ \mathsf{s}\ ,\mathsf{s}\rangle \rhd \langle \mathcal{CC}[\![\mathsf{C}]\!]:\mathsf{LOOP}(\mathcal{CB}[\![\mathsf{B}]\!],\mathcal{CC}[\![\mathsf{C}]\!]),\epsilon,\mathsf{s}\rangle \rhd^{k_2-1} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$.
Because $\mathcal{CC}[\![\mathsf{C}]\!]:\mathsf{LOOP}(\mathcal{CB}[\![\mathsf{B}]\!],\mathcal{CC}[\![\mathsf{C}]\!]) = \mathcal{CC}[\![\mathsf{C}\ ;\ \mathtt{while}\ \mathsf{B}\ \mathtt{do}\ \mathsf{C}]\!]$
and $k_2-1 \leq k$ we can apply the induction hypothesis and get
$\langle \mathsf{C}\ ;\ \mathtt{while}\ \mathsf{B}\ \mathtt{do}\ \mathsf{C},\mathsf{s}\rangle \Downarrow \mathsf{s}'$ and $\mathsf{e}=\epsilon$.
From B-SEQ we get $\langle \mathsf{C},\mathsf{s}\rangle \Downarrow \mathsf{s}''$ and $\langle \mathtt{while}\ \mathsf{B}\ \mathtt{do}\ \mathsf{C},\mathsf{s}''\rangle \Downarrow \mathsf{s}'$ for some state s''.
From B-WHILE.T we get $\langle \mathtt{while}\ \mathsf{B}\ \mathtt{do}\ \mathsf{C},\mathsf{s}\rangle \Downarrow \mathsf{s}'$.

Case $[\![\mathsf{B}]\!]\ \mathsf{s}$ = false:
We have $\langle \mathsf{BRANCH}(\mathcal{CC}[\![\mathsf{C}]\!]:\mathsf{LOOP}(\mathcal{CB}[\![\mathsf{B}]\!],\mathcal{CC}[\![\mathsf{C}]\!]), \mathsf{NOOP}),[\![\mathsf{B}]\!]\ \mathsf{s}\ ,\mathsf{s}\rangle \rhd \langle \mathsf{NOOP},\epsilon,\mathsf{s}\rangle \rhd^{k_2-1} \langle \epsilon,\mathsf{e},\mathsf{s}'\rangle$ and thus $\mathsf{e}=\epsilon$ and s=s'.
From B-WHILE.F we get $\langle \mathtt{while}\ \mathsf{B}\ \mathtt{do}\ \mathsf{C},\mathsf{s}\rangle \Downarrow \mathsf{s}'$.

Comment on the Proof

- Proof is very similar to the equivalence proof for the small-step and big-step semantics of *While*.
- Clearly we also have $[\![C]\!]_S = [\![\mathcal{CC}[\![C]\!]]\!]_{AM}$ because of this equivalence!

Question: We defined small-step semantics for AM, so why didn't we prove $[\![C]\!]_S = [\![\mathcal{CC}[\![C]\!]]\!]_{AM}$??

Alternative Proof Technique

We define a bisimulation relation $\equiv$ between the configurations of the small-steps semantics of *AM* and *While*:

$$\langle \mathsf{C},\mathsf{s}\rangle \equiv \langle \mathcal{CC}[\![\mathsf{C}]\!],\epsilon,\mathsf{s}\rangle \text{ for all commands C}$$

$$\mathsf{s} \equiv \langle \epsilon,\epsilon,\mathsf{s}\rangle$$

The idea is that only certain configuration in *AM* correspond to configurations of *While*.

Easy Direction

We could try to show that if

$$\gamma_S \equiv \gamma_{AM} \text{ and } \gamma_S \rightarrow \gamma_S'$$

then exists a configuration γ_{AM}' such that

$$\gamma_{AM} \rhd^{\geq 1} \gamma_{AM}' \text{ and } \gamma_S' \equiv \gamma_{AM}'.$$

This guarantees that if $\langle C,s \rangle \rightarrow^* s'$ then $\langle \mathcal{CC}[\![C]\!],\epsilon,s \rangle \rhd^* \langle \epsilon,\epsilon,s' \rangle$.

Difficult Direction

Assume that $\gamma_S \equiv \gamma_{AM}{}^1$ and

$$\gamma_{AM}{}^1 \rhd \gamma_{AM}{}^2 \rhd \ldots \rhd \gamma_{AM}{}^k,$$

where k > 1 and only $\gamma_{AM}{}^1$ and $\gamma_{AM}{}^k$ **have empty evaluation stacks**, i.e., they are of the form $\langle c,\epsilon,s' \rangle$.

We could try to show that there exists a configuration γ_S' such that

$$\gamma_S \rightarrow \gamma_S' \text{ and } \gamma_S' \equiv \gamma_{AM}{}^k.$$

This guarantees that if $\langle \mathcal{CC}[\![C]\!],\epsilon,s \rangle \rhd^* \langle \epsilon,\epsilon,s' \rangle$ then $\langle C,s \rangle \rightarrow^* s'$.

Difficulties

- Difficult direction: relies on the fact, that if AM moves from some configuration with an empty stack to another configuration with an empty stack, this can be imitated by one step of the small-step semantics of While. (Consider for example our assumption that expressions are evaluated in one step).
- The proof relies on the two semantics proceeding in **lock-step**: we need to find configurations in the two derivation sequences that correspond to one another. Often this is not possible and one has to raise the level of abstraction for one of the semantics. This is exactly what happens when the small-step semantics is replaced by the big-step semantics (we do not care about the individual computation steps but only about the result).

Difficulties: Example

The difficult direction goes through when we use the rule:

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

Does it complicate the proof if we use the alternative rules???

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

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