hsrm Hochschule RheinMain

Programming Language Foundations

06 Semantics

Prof. Dr. David Sabel

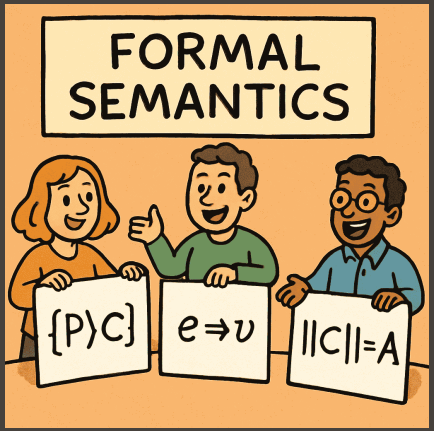
Winter term 2025/26

Last update: January 14, 2026

Image credits: All images are self-drawn or generated with AI from OpenAI at https://ki.hs-rm.de

hsrm

Formal Semantics



Formal Semantics

hsrm

Semantics of a programming language = formally describe the behavior of programs

Applications of the semantics

- reason on correctness of program optimizations
- reason on correctness of program translations
- reason on correctness of program transformations
- verify program correctness
- ...

formal semantics of programming languages is an established and active research field in compute science

Approaches to Program Semantics

hsrm

Main approaches are

- Axiomatic Semantics
- Operational Semantics
- Denotational Semantics

We briefly explain them, before studying details

Axiomatic Semantics

hsrm

- Define the meaning of programs using **logical axioms**
- Deduce properties of programs using **logical inference rules**
- Usually **not all**, but only **selected properties** are considered

Prominent Example: Hoare calculus

hsrm

- Triples $\{P\}\ C\ \{Q\}$ describe the effect of the programs on the environment: if precondition P holds and command C is executed, then postcondition Q holds.
- Exemplary axioms:

$$\frac{}{\{x = n\}\ x := m\ \{x = m\}} \qquad \frac{}{\{\}\ x := 0\ \{x = 0\}}$$

- An exemplary inference rule:

$$\frac{\{P\}\ C_1\ \{Q\}, \qquad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1; C_2\ \{R\}}$$

- An exemplary derivation

$$\frac{\dfrac{}{\{\}\ x := 0\ \{x = 0\}}\ , \qquad \dfrac{}{\{x = 0\}\ x := 10\ \{x = 10\}}}{\{\}\ x := 0; x := 10\ \{x = 10\}}$$

Operational Semantics

hsrm

- Defines **how** a program is **executed**, i.e. describes the program evaluation
- Fomalisms:
  - state transition systems: they describe how states are transformed to eventually reach a final state
    (for instance, finite automata)
  - abstract machines: machine model to evaluate programs (for instance, a universal Turing-machine, RAM-machines, etc.)
  - rewrite systems: describe how programs are rewritten to obtain a value (that's what we mainly did in the lambda calculus and KFPT-languages)

Further classification:

- **small-step semantics**: evaluation requires severals steps,
  all of them are fine-grained steps
- **big-step semantics**: evaluation in few or often only one step

Denotational Semantics

hsrm

- Program is mapped to a mathematical object (the denotation of the program)
- Wide-spread approach for denotational semantics is using domains = partially ordered sets.
- Allows to use mathematics in the domain
- Very elegant but often complicated
- Example notation $[\![P]\!]\sigma$ is the denotation of program P starting in state σ

Further Approaches to Program Semantics

hsrm

- Contextual Semantics: contextual equivalence as an equality notion for programs

  Semantics of a program = Equivalence class of the program.

- Transformational Semantics: Transform the program in a program of another language and use the semantics of the target language.

  Examples:

  - removal of syntactic sugar
  - expressing recursive supercombinators with the fixpoint operator
  - Church-encoding of data
  - ...

Compositionality

hsrm

- a desired property of every semantic description:

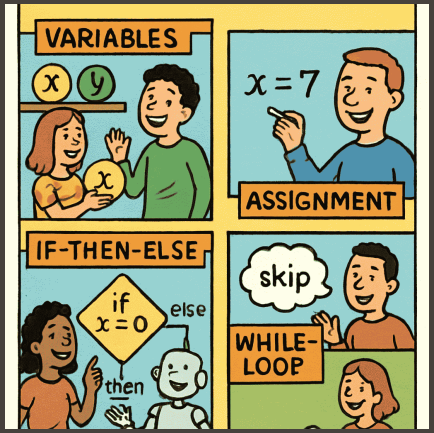
  the semantics of a **whole program** can be computed by computing the semantics of the **subprograms** and then **joining** them

- E.g. if $\langle \cdot \rangle$ computes the semantics of arithmetic expressions:

  $\langle s + t \rangle = \langle s \rangle + \langle t \rangle$ should hold

hsrm

The IMP Language



The IMP Language

hsrm

Definition

Arithmetic Expressions

$$\mathbf{AExp} ::= n \mid V \mid \mathbf{AExp} + \mathbf{AExp} \mid \mathbf{AExp} - \mathbf{AExp} \mid \mathbf{AExp} * \mathbf{AExp}$$

Boolean Expressions

$$\begin{aligned}\mathbf{BExp} ::= {} & \texttt{True} \mid \texttt{False} \mid \mathbf{AExp} = \mathbf{AExp} \mid \mathbf{AExp} \leq \mathbf{AExp} \\ & \mid \neg \mathbf{BExp} \mid \mathbf{BExp} \vee \mathbf{BExp} \mid \mathbf{Bexp} \wedge \mathbf{BExp}\end{aligned}$$

IMP-Programs

$$\begin{aligned}\mathbf{Cmd} ::= {} & \texttt{skip} \mid V := \mathbf{AExp} \mid \mathbf{Cmd}; \mathbf{Cmd} \\ & \mid \texttt{if } \mathbf{BExp} \texttt{ then } \mathbf{Cmd} \texttt{ else } \mathbf{Cmd} \texttt{ fi} \mid \texttt{while } \mathbf{BExp} \texttt{ do } \mathbf{Cmd} \texttt{ od}\end{aligned}$$

where

- V generates storage locations $\in Loc$
- n, m represent arbitrary integers

Examples

hsrm

$y := 2; z := 4; x := y + z$

- assigns 2 to storage location y
- assigns 4 to storage location z
- assigns 6 to storage location x

$x := 1; y := 100;$ `while` $1 \leq y$ `do` $x := x * y; y := y - 1$ `od`

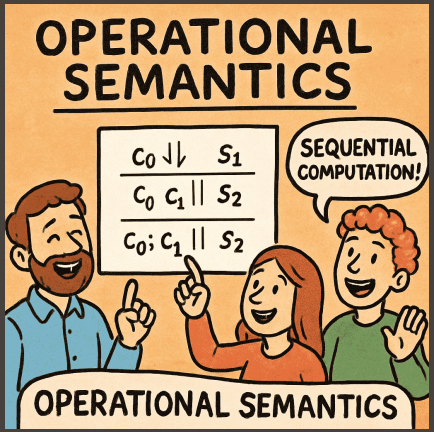
- computes $100!$

$s := 0; i := 100;$ `while` $1 \leq i$ `do` $s := s + i * i; i := i - 1$ `od`

- computes the sum $\sum_{i=0}^{100} i^2$.

hsrm

Operational Semantics



Operational Semantics: States

hsrm

- A state is a partial function $\sigma : Loc \to \mathbb{Z}$ such that $Dom(\sigma)$ is finite.
- Storage locations store numbers, but no boolean values
- Accessing not initialized storage locations is treated as runtime error
- Let Σ be the set of all states, i.e.

$$\Sigma = \{\sigma \mid \sigma : Loc \to \mathbb{Z} \wedge Dom(\sigma) \text{ is finite}\}$$

- For $\sigma \in \Sigma$ and $x \in Loc$, $\sigma(x) \in \mathbb{Z}$ is the value of storage location x, or if σ is not defined for x, $\sigma(x) = \bot$

A Big-Step Semantics

hsrm

Definition (Evaluation Relation)

A configuration $\langle s, \sigma \rangle$ consists of a command, arithmetic expression, or boolean expression s and a state $\sigma \in \Sigma$.
We use the evaluation relation $\downarrow$ for all three kinds of configurations:

- For arithmetic expression a: $\langle a, \sigma \rangle \downarrow n$ if a evaluates to number $n \in \mathbb{Z}$ in state σ.
- For boolean expression b: $\langle b, \sigma \rangle \downarrow v \in \{\texttt{True}, \texttt{False}\}$ if b evaluates to v in state σ.
- For command c: $\langle c, \sigma \rangle \downarrow \sigma'$ if c changes the state σ to state σ'.

Note:

- $\downarrow$ is a relation and not necessarily a function
- If it is a function, then the programming language is **deterministic**
- Sometimes, $\downarrow$ **must** be a relation:
  e.g., if the language can generate random numbers

Derivation Rules

hsrm

Axioms and derivation rules for the big-step semantics are written as

$$\frac{\text{premises}}{\text{conclusion}}$$

For an axiom, the premises are empty

Rules for Evaluation of Arithmetic Expressions

hsrm

The rules for evaluation of arithmetic expressions are:

(AxNum) $\dfrac{}{\langle n, \sigma \rangle \downarrow n}$

(AxLoc) $\dfrac{}{\langle x, \sigma \rangle \downarrow \sigma(x)}$ if $\sigma(x)$ is defined

(Sum) $\dfrac{\langle a_1, \sigma \rangle \downarrow n_1 \quad \langle a_2, \sigma \rangle \downarrow n_2}{\langle a_1 + a_2, \sigma \rangle \downarrow n'}$ if $n' = n_1 + n_2$

(Prod) $\dfrac{\langle a_1, \sigma \rangle \downarrow n_1 \quad \langle a_2, \sigma \rangle \downarrow n_2}{\langle a_1 * a_2, \sigma \rangle \downarrow n'}$ if $n' = n_1 \cdot n_2$

(Diff) $\dfrac{\langle a_1, \sigma \rangle \downarrow n_1 \quad \langle a_2, \sigma \rangle \downarrow n_2}{\langle a_1 - a_2, \sigma \rangle \downarrow n'}$ if $n' = n_1 - n_2$

Rules for Evaluation of Boolean Expressions

hsrm

$$\text{(AxT)}\ \frac{}{\langle \texttt{True}, \sigma\rangle \downarrow \texttt{True}}$$

$$\text{(AxF)}\ \frac{}{\langle \texttt{False}, \sigma\rangle \downarrow \texttt{False}}$$

$$\text{(Leq)}\ \frac{\langle a_1, \sigma\rangle \downarrow n \quad \langle a_2, \sigma\rangle \downarrow m}{\langle a_1 \leq a_2, \sigma\rangle \downarrow \texttt{True}}\ \text{if } n \leq m$$

$$\text{(AndT)}\ \frac{\langle b_1, \sigma\rangle \downarrow \texttt{True} \quad \langle b_2, \sigma\rangle \downarrow \texttt{True}}{\langle b_1 \wedge b_2, \sigma\rangle \downarrow \texttt{True}}$$

$$\text{(Eq)}\ \frac{\langle a_1, \sigma\rangle \downarrow n \quad \langle a_2, \sigma\rangle \downarrow m}{\langle a_1 = a_2, \sigma\rangle \downarrow \texttt{True}}\ \text{if } n = m$$

$$\text{(NEq)}\ \frac{\langle a_1, \sigma\rangle \downarrow n \quad \langle a_2, \sigma\rangle \downarrow m}{\langle a_1 = a_2, \sigma\rangle \downarrow \texttt{False}}\ \text{if } n \neq m$$

$$\text{(NLeq)}\ \frac{\langle a_1, \sigma\rangle \downarrow n \quad \langle a_2, \sigma\rangle \downarrow m}{\langle a_1 \leq a_2, \sigma\rangle \downarrow \texttt{False}}\ \text{if } n > m$$

$$\text{(AndF1)}\ \frac{\langle b_1, \sigma\rangle \downarrow \texttt{False}}{\langle b_1 \wedge b_2, \sigma\rangle \downarrow \texttt{False}}$$

Rules for Evaluation of Boolean Expressions (cont'ed)

hsrm

$$\text{(AndF2)}\ \frac{\langle b_1, \sigma\rangle \downarrow \texttt{True} \quad \langle b_2, \sigma\rangle \downarrow \texttt{False}}{\langle b_1 \wedge b_2, \sigma\rangle \downarrow \texttt{False}} \qquad \text{(OrF)}\ \frac{\langle b_1, \sigma\rangle \downarrow \texttt{False} \quad \langle b_2, \sigma\rangle \downarrow \texttt{False}}{\langle b_1 \vee b_2, \sigma\rangle \downarrow \texttt{False}}$$

$$\text{(OrT1)}\ \frac{\langle b_1, \sigma\rangle \downarrow \texttt{True}}{\langle b_1 \vee b_2, \sigma\rangle \downarrow \texttt{True}} \qquad \text{(OrT2)}\ \frac{\langle b_1, \sigma\rangle \downarrow \texttt{False} \quad \langle b_2, \sigma\rangle \downarrow \texttt{True}}{\langle b_1 \vee b_2, \sigma\rangle \downarrow \texttt{True}}$$

$$\text{(Not1)}\ \frac{\langle b, \sigma\rangle \downarrow \texttt{False}}{\langle \neg b, \sigma\rangle \downarrow \texttt{True}} \qquad \text{(Not2)}\ \frac{\langle b, \sigma\rangle \downarrow \texttt{True}}{\langle \neg b, \sigma\rangle \downarrow \texttt{False}}$$

conjunction and disjunction are evaluated "sequentially", i.e. $\langle \texttt{True} \vee b, \sigma\rangle \downarrow \texttt{True}$ and $\langle \texttt{False} \wedge b, \sigma\rangle \downarrow \texttt{False}$ for every b, in particular when b is undefined.

Example

For $\sigma = \{x \mapsto 10, y \mapsto 7, z \mapsto 8\}$, we can built the derivation tree for the arithmetic expression $x \leq y + 4 \vee \texttt{True}$ as follows

$$\dfrac{\dfrac{\dfrac{}{\langle x, \sigma\rangle \downarrow 10}\,\text{AxNum} \quad \dfrac{\dfrac{}{\langle y, \sigma\rangle \downarrow 7}\,\text{AxLoc} \quad \dfrac{}{\langle 4, \sigma\rangle \downarrow 4}\,\text{AxNum}}{\langle y + 4, \sigma\rangle \downarrow 11}\,\text{Sum}\ \text{if } 11 = 7 + 4}{\langle x \leq y + 4, \sigma\rangle \downarrow \texttt{True}}\,\text{Leq}\ \text{if } 10 \leq 11}{\langle x \leq y + 4 \vee \texttt{True}, \sigma\rangle \downarrow \texttt{True}}\,\text{OrT1}$$

The construction is done bottom-up, until the top of the tree consists of axioms and thus no more premises have to be shown.

Remarks

- The semantics does not prescribe an exact order of evaluation
- E.g, in $a_1 + a_2$, the semantics does not fix the order of evaluating a_1 and a_2.
- This is a typical characteristics of a big-step semantics – it leaves some freedom in the implementation.

This could be changed, by replacing rule (Sum) by:

$$\frac{\langle a_1, \sigma\rangle \downarrow n \quad \langle n + a_2, \sigma\rangle \downarrow m}{\langle a_1 + a_2, \sigma\rangle \downarrow m} \qquad \frac{\langle a_2, \sigma\rangle \downarrow n}{\langle m + a_2, \sigma\rangle \downarrow n'} \text{ if } n' = m + n$$

Rules for Evaluation of Commands

hsrm

The rules for evaluation of commands have side-effects, i.e. they modify the state σ. We write $\sigma[m/x]$ for the state σ where the value of x is changed to m, i.e.

$$\sigma[m/x](y) = \begin{cases} \sigma(x) & \text{if } y \neq x \\ m & \text{if } y = x \end{cases}$$

Rules for Evaluation of Commands (Cont'd)

hsrm

$$\text{(AxSkip)}\ \frac{}{\langle \texttt{skip}, \sigma\rangle \downarrow \sigma} \qquad \text{(Asgn)}\ \frac{\langle a, \sigma\rangle \downarrow m}{\langle x := a, \sigma\rangle \downarrow \sigma[m/x]} \qquad \text{(Seq)}\ \frac{\langle c_1, \sigma\rangle \downarrow \sigma' \quad \langle c_2, \sigma'\rangle \downarrow \sigma''}{\langle c_1; c_2, \sigma\rangle \downarrow \sigma''}$$

$$\text{(IfT)}\ \frac{\langle b, \sigma\rangle \downarrow \texttt{True} \quad \langle c_1, \sigma\rangle \downarrow \sigma'}{\langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{fi}, \sigma\rangle \downarrow \sigma'} \qquad \text{(IfF)}\ \frac{\langle b, \sigma\rangle \downarrow \texttt{False} \quad \langle c_2, \sigma\rangle \downarrow \sigma'}{\langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{fi}, \sigma\rangle \downarrow \sigma'}$$

$$\text{(WhileF)}\ \frac{\langle b, \sigma\rangle \downarrow \texttt{False}}{\langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}, \sigma\rangle \downarrow \sigma} \qquad \text{(WhileT)}\ \frac{\langle b, \sigma\rangle \downarrow \texttt{True} \qquad \langle c, \sigma\rangle \downarrow \sigma' \quad \langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}, \sigma'\rangle \downarrow \sigma''}{\langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}, \sigma\rangle \downarrow \sigma''}$$

Examples

hsrm

Evaluation of $c = x := 1; y := 2$ in state $\{x \mapsto 2\}$:

Examples

hsrm

Evaluation of $c = x := 1; y := 2$ in state $\{x \mapsto 2\}$:

$$\text{(Seq)}\ \frac{\langle x := 1, \{x \mapsto 2\}\rangle \downarrow \sigma' \quad \langle y := 2, \sigma'\rangle \downarrow \sigma''}{\langle x := 1; y := 2, \{x \mapsto 2\}\rangle \downarrow \sigma''}$$

Examples

Evaluation of $c = x := 1; y := 2$ in state $\{x \mapsto 2\}$:

$$\text{(Seq)}\ \frac{\text{(Asgn)}\ \dfrac{\langle 1, \{x \mapsto 2\}\rangle \downarrow m}{\langle x := 1, \{x \mapsto 2\}\rangle \downarrow \{x \mapsto 2\}[m/x]} \qquad \langle y := 2, \{x \mapsto 2\}[m/x]\rangle \downarrow \sigma''}{\langle x := 1; y := 2, \{x \mapsto 2\}\rangle \downarrow \sigma''}$$

Examples

hsrm

Evaluation of $c = x := 1; y := 2$ in state $\{x \mapsto 2\}$:

$$\dfrac{\dfrac{\dfrac{}{\langle 1, \{x \mapsto 2\}\rangle \downarrow 1}\,(\text{AxNum})}{\langle x := 1, \{x \mapsto 2\}\rangle \downarrow \{x \mapsto 1\}}\,(\text{Asgn}) \qquad \langle y := 2, \{x \mapsto 1\}\rangle \downarrow \sigma''}{\langle x := 1; y := 2, \{x \mapsto 2\}\rangle \downarrow \sigma''}\,(\text{Seq})$$

Examples

Evaluation of $c = x := 1; y := 2$ in state $\{x \mapsto 2\}$:

$$\dfrac{\dfrac{\dfrac{}{\langle 1, \{x \mapsto 2\}\rangle \downarrow 1}\,(\text{AxNum})}{\langle x := 1, \{x \mapsto 2\}\rangle \downarrow \{x \mapsto 1\}}\,(\text{Asgn}) \qquad \dfrac{\langle 2, \{x \mapsto 1\}\rangle \downarrow m}{\langle y := 2, \{x \mapsto 1\}\rangle \downarrow \{x \mapsto 1\}[m/y]}\,(\text{Asgn})}{\langle x := 1; y := 2, \{x \mapsto 2\}\rangle \downarrow \sigma''}\,(\text{Seq})$$

Examples

hsrm

Evaluation of $c = x := 1; y := 2$ in state $\{x \mapsto 2\}$:

$$\dfrac{\dfrac{\dfrac{}{\langle 1, \{x \mapsto 2\}\rangle \downarrow 1}\,(\text{AxNum})}{\langle x := 1, \{x \mapsto 2\}\rangle \downarrow \{x \mapsto 1\}}\,(\text{Asgn}) \qquad \dfrac{\dfrac{}{\langle 2, \{x \mapsto 1\}\rangle \downarrow 2}\,(\text{AxNum})}{\langle y := 2, \{x \mapsto 1\}\rangle \downarrow \{x \mapsto 1, y \mapsto 2\}}\,(\text{Asgn})}{\langle x := 1; y := 2, \{x \mapsto 2\}\rangle \downarrow \sigma''}\,(\text{Seq})$$

Examples

hsrm

Evaluation of $c = x := 1; y := 2$ in state $\{x \mapsto 2\}$:

$$\dfrac{\dfrac{\dfrac{}{\langle 1, \{x \mapsto 2\}\rangle \downarrow 1}\,(\text{AxNum})}{\langle x := 1, \{x \mapsto 2\}\rangle \downarrow \{x \mapsto 1\}}\,(\text{Asgn}) \qquad \dfrac{\dfrac{}{\langle 2, \{x \mapsto 1\}\rangle \downarrow 2}\,(\text{AxNum})}{\langle y := 2, \{x \mapsto 1\}\rangle \downarrow \{x \mapsto 1, y \mapsto 2\}}\,(\text{Asgn})}{\langle x := 1; y := 2, \{x \mapsto 2\}\rangle \downarrow \sigma''}\,(\text{Seq})$$

Examples

Evaluation of $c = x := 1; y := 2$ in state $\{x \mapsto 2\}$:

$$\dfrac{\dfrac{\dfrac{}{\langle 1, \{x \mapsto 2\}\rangle \downarrow 1}\,(\text{AxNum})}{\langle x := 1, \{x \mapsto 2\}\rangle \downarrow \{x \mapsto 1\}}\,(\text{Asgn}) \qquad \dfrac{\dfrac{}{\langle 2, \{x \mapsto 1\}\rangle \downarrow 2}\,(\text{AxNum})}{\langle y := 2, \{x \mapsto 1\}\rangle \downarrow \{x \mapsto 1, y \mapsto 2\}}\,(\text{Asgn})}{\langle x := 1; y := 2, \{x \mapsto 2\}\rangle \downarrow \{x \mapsto 1, y \mapsto 2\}}\,(\text{Seq})$$

Examples (2)

hsrm

Evaluation of

$$\texttt{while } \neg(x \leq 1) \texttt{ do } y := y + 1; x := x - 1 \texttt{ od in state } \sigma = \{x \mapsto 2, y \mapsto 0\}$$

Examples (2)

hsrm

Evaluation of

$$\texttt{while}\ \neg(x \leq 1)\ \texttt{do}\ y := y + 1; x := x - 1\ \texttt{od}\ \text{in state}\ \sigma = \{x \mapsto 2, y \mapsto 0\}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\langle x, \sigma\rangle \downarrow 2}\,(\text{AxLoc}) \quad \dfrac{}{\langle 1, \sigma\rangle \downarrow 1}\,(\text{AxNum})}{\langle x \leq 1, \sigma\rangle \downarrow \texttt{False}}\,(\text{NLeq})}{\langle \neg(x \leq 1), \sigma\rangle \downarrow \texttt{True}}\,(\text{Not1}) \quad \dfrac{\dfrac{\dfrac{}{\langle y, \sigma\rangle \downarrow 0}\,(\text{AxLoc}) \quad \dfrac{}{\langle 1, \sigma\rangle \downarrow 1}\,(\text{AxNum})}{\langle y := y + 1, \sigma\rangle \downarrow \sigma_2}\,(\text{Sum}) \quad \dfrac{\dfrac{}{\langle x, \sigma_2\rangle \downarrow 2}\,(\text{AxLoc}) \quad \dfrac{}{\langle 1, \sigma_2\rangle \downarrow 1}\,(\text{AxNum})}{\langle x := x - 1, \sigma_2\rangle \downarrow \sigma_1}\,(\text{Diff})}{\langle y := y + 1; x := x - 1, \sigma\rangle \downarrow \sigma_1}\,(\text{Asgn})}{\langle \texttt{while}\ \neg(x \leq 1)\ \texttt{do}\ y := y + 1; x := x - 1\ \texttt{od}, \sigma\rangle \downarrow \sigma_1}\,(\text{WhileT}) \quad \dfrac{\dfrac{\dfrac{}{\langle x, \sigma_1\rangle \downarrow 1}\,(\text{AxLoc}) \quad \dfrac{}{\langle 1, \sigma_1\rangle \downarrow 1}\,(\text{AxNum})}{\langle (x \leq 1), \sigma_1\rangle \downarrow \texttt{True}}\,(\text{Leq})}{\langle \neg(x \leq 1), \sigma_1\rangle \downarrow \texttt{False}}\,(\text{Not2})}{\langle \texttt{while}\ \neg(x \leq 1)\ \texttt{do}\ y := y + 1;\ x := x - 1\ \texttt{od}, \sigma_1\rangle \downarrow \sigma_1}\,(\text{WhileF})$$

where $\sigma_1 = \{x \mapsto 1, y \mapsto 1\}$ and $\sigma_2 = \{x \mapsto 2, y \mapsto 1\}$

Examples (3)

hsrm

Let σ be an arbitrary state. Try to derive $\langle \texttt{while True do skip od} \rangle \downarrow \sigma'$ for some σ'

Examples (3)

Let σ be an arbitrary state. Try to derive $\langle \texttt{while True do skip od} \rangle \downarrow \sigma'$ for some σ'

$$\text{(WhileT)}\ \frac{\text{(AxT)}\ \dfrac{}{\langle \texttt{True}, \sigma \rangle \downarrow \texttt{True}} \qquad \text{(AxSkip)}\ \dfrac{}{\langle \texttt{skip}, \sigma \rangle \downarrow \sigma} \qquad \dfrac{\vdots}{\langle \texttt{while True do skip od}, \sigma \rangle \downarrow \sigma'}}{\langle \texttt{while True do skip od}, \sigma \rangle \downarrow \sigma'}$$

Examples (3)

Let σ be an arbitrary state. Try to derive $\langle \texttt{while True do skip od} \rangle \downarrow \sigma'$ for some σ'

$$\dfrac{\dfrac{}{\langle \texttt{True}, \sigma \rangle \downarrow \texttt{True}}\ (\text{AxT}) \qquad \dfrac{}{\langle \texttt{skip}, \sigma \rangle \downarrow \sigma}\ (\text{AxSkip}) \qquad \dfrac{\vdots}{\langle \texttt{while True do skip od}, \sigma \rangle \downarrow \sigma'}}{\langle \texttt{while True do skip od}, \sigma \rangle \downarrow \sigma'}\ (\text{WhileT})$$

Examples (3)

Let σ be an arbitrary state. Try to derive $\langle \texttt{while True do skip od} \rangle \downarrow \sigma'$ for some σ'

$$\dfrac{\dfrac{}{\langle \texttt{True}, \sigma \rangle \downarrow \texttt{True}}\ (\text{AxT}) \qquad \dfrac{}{\langle \texttt{skip}, \sigma \rangle \downarrow \sigma}\ (\text{AxSkip}) \qquad \dfrac{\vdots}{\langle \texttt{while True do skip od}, \sigma \rangle \downarrow \sigma'}}{\langle \texttt{while True do skip od}, \sigma \rangle \downarrow \sigma'}\ (\text{WhileT})$$

➽ Derivation is impossible

Evaluation is Deterministic

hsrm

- Goal: show if $\langle c, \sigma \rangle \downarrow \sigma'$ and $\langle c, \sigma \rangle \downarrow \sigma''$, then $\sigma' = \sigma''$.
- Three parts: arithmetic expressions, boolean expressions, programs

Lemma

Let a be an arithmetic expression and $\sigma \in \Sigma$. If $\langle a, \sigma \rangle \downarrow n$ and $\langle a, \sigma \rangle \downarrow n'$ then $n = n'$.

Lemma

Let b be a boolean expression, $\sigma \in \Sigma$ and $\langle b, \sigma \rangle \downarrow v_1$ and $\langle b, \sigma \rangle \downarrow v_2$ then $v_1 = v_2$

Both lemmas can be shown by structural induction (on a or on b).

Evaluation is Deterministic (Cont'd)

hsrm

Proposition

Let c be a command, $\sigma \in \Sigma$ and $\langle c, \sigma \rangle \downarrow \sigma_1$ and $\langle c, \sigma \rangle \downarrow \sigma_2$ then $\sigma_1 = \sigma_2$

Proof.

Evaluation is Deterministic (Cont'd)

hsrm

Proposition

Let c be a command, $\sigma \in \Sigma$ and $\langle c, \sigma \rangle \downarrow \sigma_1$ and $\langle c, \sigma \rangle \downarrow \sigma_2$ then $\sigma_1 = \sigma_2$

Proof. Note: Induction on c does not work (consider rule (WhileT)!

Evaluation is Deterministic (Cont'd)

hsrm

Proposition

Let c be a command, $\sigma \in \Sigma$ and $\langle c, \sigma \rangle \downarrow \sigma_1$ and $\langle c, \sigma \rangle \downarrow \sigma_2$ then $\sigma_1 = \sigma_2$

Proof. By induction on the derivation of $\langle c, \sigma \rangle \downarrow \sigma_1$.

Evaluation is Deterministic (Cont'd)

hsrm

Proposition

Let c be a command, $\sigma \in \Sigma$ and $\langle c, \sigma \rangle \downarrow \sigma_1$ and $\langle c, \sigma \rangle \downarrow \sigma_2$ then $\sigma_1 = \sigma_2$

Proof. By induction on the derivation of $\langle c, \sigma \rangle \downarrow \sigma_1$.

- Base case: 1 step. This must be (AxSkip) and $c = \texttt{skip}$. Then the proof is easy.

Evaluation is Deterministic (Cont'd)

hsrm

Proposition

Let c be a command, $\sigma \in \Sigma$ and $\langle c, \sigma \rangle \downarrow \sigma_1$ and $\langle c, \sigma \rangle \downarrow \sigma_2$ then $\sigma_1 = \sigma_2$

Proof. By induction on the derivation of $\langle c, \sigma \rangle \downarrow \sigma_1$.

- Base case: 1 step. This must be (AxSkip) and $c = \texttt{skip}$. Then the proof is easy.
- Step: Consider the last rule applied in the derivation of $\langle c, \sigma \rangle \downarrow \sigma_1$.

Evaluation is Deterministic (Cont'd)

hsrm

Proposition

Let c be a command, $\sigma \in \Sigma$ and $\langle c, \sigma \rangle \downarrow \sigma_1$ and $\langle c, \sigma \rangle \downarrow \sigma_2$ then $\sigma_1 = \sigma_2$

Proof. By induction on the derivation of $\langle c, \sigma \rangle \downarrow \sigma_1$.

- Base case: 1 step. This must be (AxSkip) and $c = \texttt{skip}$. Then the proof is easy.
- Step: Consider the last rule applied in the derivation of $\langle c, \sigma \rangle \downarrow \sigma_1$.
  - Case (Seq). Then $c = c_1; c_2$, $\langle c_1, \sigma \rangle \downarrow \sigma'$, and $\langle c_2, \sigma' \rangle \downarrow \sigma_1$ for some σ'.
    For $\langle c, \sigma \rangle \downarrow \sigma_2$, the rule must also be (Seq), i.e. $\langle c_1, \sigma \rangle \downarrow \sigma''$ and $\langle c_2, \sigma'' \rangle \downarrow \sigma_2$.
    Apply IH to the subderivations: This shows $\sigma' = \sigma''$ and $\sigma_1 = \sigma_2$.

Evaluation is Deterministic (Cont'd)

hsrm

Proposition

Let c be a command, $\sigma \in \Sigma$ and $\langle c, \sigma\rangle \downarrow \sigma_1$ and $\langle c, \sigma\rangle \downarrow \sigma_2$ then $\sigma_1 = \sigma_2$

Proof. By induction on the derivation of $\langle c, \sigma\rangle \downarrow \sigma_1$.

- Base case: 1 step. This must be (AxSkip) and $c = \texttt{skip}$. Then the proof is easy.
- Step: Consider the last rule applied in the derivation of $\langle c, \sigma\rangle \downarrow \sigma_1$.
  - Case (Seq). Then $c = c_1; c_2$, $\langle c_1, \sigma\rangle \downarrow \sigma'$, and $\langle c_2, \sigma'\rangle \downarrow \sigma_1$ for some σ'.
    For $\langle c, \sigma\rangle \downarrow \sigma_2$, the rule must also be (Seq), i.e. $\langle c_1, \sigma\rangle \downarrow \sigma''$ and $\langle c_2, \sigma''\rangle \downarrow \sigma_2$.
    Apply IH to the subderivations: This shows $\sigma' = \sigma''$ and $\sigma_1 = \sigma_2$.
  - Case (WhileT): Then $\langle b, \sigma\rangle \downarrow \texttt{True}$, $\langle c', \sigma\rangle \downarrow \sigma'$ and $\langle \texttt{while } b \texttt{ do } c' \texttt{ od}, \sigma'\rangle \downarrow \sigma_1$.
    The previous lemma shows that $\langle b, \sigma\rangle \downarrow \texttt{False}$ is impossible and thus for $\langle c, \sigma\rangle \downarrow \sigma_2$ also rule (WhileT) was used, i.e. $\langle c, \sigma\rangle \downarrow \sigma''$ and $\langle \texttt{while } b \texttt{ do } c' \texttt{ od}, \sigma''\rangle \downarrow \sigma_2$ for some σ''.
    The IH shows that $\sigma' = \sigma''$ and thus also $\sigma_1 = \sigma_2$.

Evaluation is Deterministic (Cont'd)

hsrm

Proposition

Let c be a command, $\sigma \in \Sigma$ and $\langle c, \sigma \rangle \downarrow \sigma_1$ and $\langle c, \sigma \rangle \downarrow \sigma_2$ then $\sigma_1 = \sigma_2$

Proof. By induction on the derivation of $\langle c, \sigma \rangle \downarrow \sigma_1$.

- Base case: 1 step. This must be (AxSkip) and $c = \texttt{skip}$. Then the proof is easy.
- Step: Consider the last rule applied in the derivation of $\langle c, \sigma \rangle \downarrow \sigma_1$.
  - Case (Seq). Then $c = c_1; c_2$, $\langle c_1, \sigma \rangle \downarrow \sigma'$, and $\langle c_2, \sigma' \rangle \downarrow \sigma_1$ for some σ'.
    For $\langle c, \sigma \rangle \downarrow \sigma_2$, the rule must also be (Seq), i.e. $\langle c_1, \sigma \rangle \downarrow \sigma''$ and $\langle c_2, \sigma'' \rangle \downarrow \sigma_2$.
    Apply IH to the subderivations: This shows $\sigma' = \sigma''$ and $\sigma_1 = \sigma_2$.
  - Case (WhileT): Then $\langle b, \sigma \rangle \downarrow \texttt{True}$, $\langle c', \sigma \rangle \downarrow \sigma'$ and $\langle \texttt{while } b \texttt{ do } c' \texttt{ od}, \sigma' \rangle \downarrow \sigma_1$.
    The previous lemma shows that $\langle b, \sigma \rangle \downarrow \texttt{False}$ is impossible and thus for $\langle c, \sigma \rangle \downarrow \sigma_2$ also rule (WhileT) was used, i.e. $\langle c, \sigma \rangle \downarrow \sigma''$ and $\langle \texttt{while } b \texttt{ do } c' \texttt{ od}, \sigma'' \rangle \downarrow \sigma_2$ for some σ''.
    The IH shows that $\sigma' = \sigma''$ and thus also $\sigma_1 = \sigma_2$.
  - Cases (IfT), (IfF), (WhileF): Similar. □

Equivalence of IMP-Programs

hsrm

Since evaluation is deterministic, semantics of a program is a partial function on states:

Definition

Let c be a command, then $[\![c]\!]_{eval} : \Sigma \to \Sigma$ is the partial function such that

$$[\![c]\!]_{eval}\sigma = \sigma' \text{ iff } \langle c, \sigma\rangle \downarrow \sigma'$$

There are programs such that $[\![c]\!]_{eval}$ is undefined for all states.
One such program is `while True do skip od`.

Definition (Equivalence $\sim$ on Programs)

The relation $\sim$ is defined as $c_1 \sim c_2$ iff for all $\sigma \in \Sigma : [\![c_1]\!]_{eval}\sigma = [\![c_2]\!]_{eval}\sigma$

Note: $\sim$ means that the input-output behavior is the same.

Example

hsrm

Lemma

The equivalence

$$(\texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}) \sim (\texttt{if}\ b\ \texttt{then}\ c; \texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}\ \texttt{else}\ \texttt{skip}\ \texttt{fi})$$

holds.

This can be shown by a case distinction:

- $\langle b, \sigma \rangle \downarrow \texttt{False}$
- $\langle b, \sigma \rangle \downarrow \texttt{True}$
- $\langle b, \sigma \rangle \downarrow v$ does not hold for any v

Remark

hsrm

- Let $\sim_{init}$ be like $\sim$, but with the difference, that all variables are initialized with value 0 (i.e. $\sigma(x) = 0$, if σ does not define a value for x).
- $\sim \neq \sim_{init}$:

  `if` b `then skip else skip fi` $\sim_{init}$ `skip` holds for every boolean expression b

  `if` b `then skip else skip fi` $\not\sim$ `skip` does not hold for all b:

  e.g. if b is $x = x$ and $x \notin \sigma$

A Small-Step-Semantics of IMP

hsrm

- similar to the reduction relations in the lambda-calculus
- rewrite pairs of programs and state (i.e. configurations) until a successful configuration is obtained
- defined by reduction rules and reduction contexts
- alternative definition with labeling to fix the strategy

A Small-Step-Semantics of IMP

hsrm

Reduction rules $\rightarrow$ operate on configurations $\langle t, \sigma \rangle$

$(skip)$ $\langle \texttt{skip}; c, \sigma \rangle \rightarrow \langle c, \sigma \rangle$

$(asgn)$ $\langle x := m, \sigma \rangle \rightarrow \langle \texttt{skip}, \sigma[m/x] \rangle$ if $m \in \mathbb{Z}$

(ifT) $\langle \texttt{if True then } c_1 \texttt{ else } c_2 \texttt{ fi}, \sigma \rangle \rightarrow \langle c_1, \sigma \rangle$

(ifF) $\langle \texttt{if False then } c_1 \texttt{ else } c_2 \texttt{ fi}, \sigma \rangle \rightarrow \langle c_2, \sigma \rangle$

$(while)$ $\langle \texttt{while } b \texttt{ do } c \texttt{ od}, \sigma \rangle$
$\rightarrow \langle \texttt{if } b \texttt{ then } c; \texttt{while } b \texttt{ do } c \texttt{ od else skip fi}, \sigma \rangle$

(sum) $\langle n + m, \sigma \rangle \rightarrow \langle n', \sigma \rangle$ if $n' = n + m$

$(prod)$ $\langle n * m, \sigma \rangle \rightarrow \langle n', \sigma \rangle$ if $n' = n \cdot m$

$(diff)$ $\langle n - m, \sigma \rangle \rightarrow \langle n', \sigma \rangle$ if $n' = n - m$

(loc) $\langle x, \sigma \rangle \rightarrow \langle n, \sigma \rangle$ if $\sigma(x) = n$

$(leqT)$ $\langle n \leq m, \sigma \rangle \rightarrow \langle \texttt{True}, \sigma \rangle$ if $n \leq m$

$(leqF)$ $\langle n \leq m, \sigma \rangle \rightarrow \langle \texttt{False}, \sigma \rangle$ if $n > m$

(eqT) $\langle n = n, \sigma \rangle \rightarrow \langle \texttt{True}, \sigma \rangle$ if $n = m$

(eqF) $\langle n = m, \sigma \rangle \rightarrow \langle \texttt{False}, \sigma \rangle$ if $n \neq m$

(orT) $\langle \texttt{True} \vee b, \sigma \rangle \rightarrow \langle \texttt{True}, \sigma \rangle$

(orF) $\langle \texttt{False} \vee v, \sigma \rangle \rightarrow \langle v, \sigma \rangle$
if $v \in \{\texttt{True}, \texttt{False}\}$

$(andF)$ $\langle \texttt{False} \wedge b, \sigma \rangle \rightarrow \langle \texttt{False}, \sigma \rangle$

$(andT)$ $\langle \texttt{True} \wedge v, \sigma \rangle \rightarrow \langle v, \sigma \rangle$
if $v \in \{\texttt{True}, \texttt{False}\}$

$(notT)$ $\langle \neg \texttt{True}, \sigma \rangle \rightarrow \langle \texttt{False}, \sigma \rangle$

$(notF)$ $\langle \neg \texttt{False}, \sigma \rangle \rightarrow \langle \texttt{True}, \sigma \rangle$

Reduction Contexts

hsrm

Three classes of reduction contexts

$$
\begin{aligned}
R_A &::= [\cdot] \mid R_A + a \mid R_A * a \mid R_A - a \mid n + R_A \mid n * R_A \mid n - R_A \\
R_B &::= [\cdot] \mid R_B \vee b \mid R_B \wedge b \mid \texttt{False} \vee R_B \mid \texttt{True} \wedge R_B \mid \neg R_B \\
&\quad \mid R_A \leq a \mid n \leq R_A \mid R_A = a \mid n = R_A \\
R_C &::= [\cdot] \mid R_C; c \mid \texttt{if } R_B \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ fi} \mid x := R_A
\end{aligned}
$$

Definition

Reduction relation $\xrightarrow{eval}$:
If $\langle s, \sigma\rangle \to \langle s', \sigma'\rangle$ then for every R_C-context: $\langle R_C[s], \sigma\rangle \xrightarrow{eval} \langle R_C[s'], \sigma'\rangle$.

We also write $\xrightarrow{eval,rule}$ where $rule$ is the name of the used rule.
Note: `while` is not missing in R_C, since rule $(while)$ rewrites the whole `while`

Alternative Definition with Labeling Algorithm

hsrm

For command c, start with $c^\star$ and exhaustively apply the shifting rules:

$$
\begin{array}{lcl}
(c_1; c_2)^\star & \Rightarrow & (c_1^\star; c_2) \\
(X := a)^\star & \Rightarrow & (X := a^\star) \\
\texttt{if } b \texttt{ then } c \texttt{ else } c' \texttt{ fi}^\star & \Rightarrow & \texttt{if } b^\star \texttt{ then } c \texttt{ else } c' \texttt{ fi} \\
(a_1 \oplus a_2)^\star & \Rightarrow & (a_1^\star \oplus a_2) \text{ if } \oplus \in \{+, -, *, =, \leq\} \\
(n^\star \oplus a) & \Rightarrow & (n \oplus a^\star) \text{ if } n \in \mathbb{Z} \text{ and } \oplus \in \{+, -, *, =, \leq\} \\
(b_1 \vee b_2)^\star & \Rightarrow & (b_1^\star \vee b_2) \\
(b_1 \wedge b_2)^\star & \Rightarrow & (b_1^\star \wedge b_2) \\
(\neg\ b)^\star & \Rightarrow & (\neg\ b^\star) \\
(\texttt{False}^\star \vee b) & \Rightarrow & (\texttt{False} \vee b^\star) \\
(\texttt{True}^\star \wedge b) & \Rightarrow & (\texttt{True} \wedge b^\star)
\end{array}
$$

Reduction Rules with Label

$$\langle C[\texttt{skip}^\star; c], \sigma\rangle \xrightarrow{eval,skip} \langle C[c], \sigma\rangle$$

$$\langle C[x := m^\star], \sigma\rangle \xrightarrow{eval,asgn} \langle C[\texttt{skip}], \sigma[m/x]\rangle \text{ if } m \in \mathbb{Z}$$

$$\langle C[\texttt{if True}^\star \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ fi}], \sigma\rangle \xrightarrow{eval,ifT} \langle C[c_1], \sigma\rangle$$

$$\langle C[\texttt{if False}^\star \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ fi}], \sigma\rangle \xrightarrow{eval,ifF} \langle C[c_2], \sigma\rangle$$

$$\langle C[\texttt{while } b \texttt{ do } c \texttt{ od}], \sigma\rangle^\star \xrightarrow{eval,while} \langle C[\texttt{if } b \texttt{ then } c; \texttt{while } b \texttt{ do } c \texttt{ od else skip fi}], \sigma\rangle$$

$$\langle C[n + m^\star], \sigma\rangle \xrightarrow{eval,sum} \langle C[n'], \sigma\rangle \text{ if } n' = n + m$$

$$\langle C[n * m^\star], \sigma\rangle \xrightarrow{eval,prod} \langle C[n'], \sigma\rangle \text{ if } n' = n \cdot m$$

$$\langle C[n - m^\star], \sigma\rangle \xrightarrow{eval,diff} \langle C[n'], \sigma\rangle \text{ if } n' = n - m$$

$$\langle C[x^\star], \sigma\rangle \xrightarrow{eval,loc} \langle C[n], \sigma\rangle \text{ if } \sigma(x) = n$$

$$\langle C[n \leq m^\star], \sigma\rangle \xrightarrow{eval,leqT} \langle C[\texttt{True}], \sigma\rangle \text{ if } n \leq m$$

$$\langle C[n \leq m^\star], \sigma\rangle \xrightarrow{eval,leqF} \langle C[\texttt{False}], \sigma\rangle \text{ if } n > m$$

$$\langle C[n = n^\star], \sigma\rangle \xrightarrow{eval,eqT} \langle C[\texttt{True}], \sigma\rangle \text{ if } n = m$$

$$\langle C[n = m^\star], \sigma\rangle \xrightarrow{eval,eqF} \langle C[\texttt{False}], \sigma\rangle \text{ if } n \neq m$$

$$\langle C[\texttt{True}^\star \vee b], \sigma\rangle \xrightarrow{eval,orT} \langle C[\texttt{True}], \sigma\rangle$$

$$\langle C[\texttt{False} \vee v^\star], \sigma\rangle \xrightarrow{eval,orF} \langle C[v], \sigma\rangle \text{ if } v \in \{\texttt{True}, \texttt{False}\}$$

$$\langle C[\texttt{False}^\star \wedge b], \sigma\rangle \xrightarrow{eval,andF} \langle C[\texttt{False}], \sigma\rangle$$

$$\langle C[\texttt{True} \wedge v^\star], \sigma\rangle \xrightarrow{eval,andT} \langle C[v], \sigma\rangle \text{ if } v \in \{\texttt{True}, \texttt{False}\}$$

$$\langle C[\neg \texttt{True}^\star], \sigma\rangle \xrightarrow{eval,notT} \langle C[\texttt{False}], \sigma\rangle$$

$$\langle C[\neg \texttt{False}^\star], \sigma\rangle \xrightarrow{eval,notF} \langle C[\texttt{True}], \sigma\rangle$$

Remarks and Notations

hsrm

- Definitions with reduction contexts and with labeling algorithm are the same
- We write $\xrightarrow{eval,n}$ for n $\xrightarrow{eval}$-steps, $\xrightarrow{eval,+}$ for the transitive closure and $\xrightarrow{eval,*}$ for the reflexive-transitive closure of $\xrightarrow{eval}$
- Small-step evaluation successfully stops if the configuration $\langle \texttt{skip}, \sigma \rangle$ for some $\sigma \in \Sigma$ is reached.
- For command c and environment σ, we write $\langle c, \sigma \rangle \downarrow_{eval} \sigma'$ iff $\langle c, \sigma \rangle \xrightarrow{eval,*} \langle \texttt{skip}, \sigma' \rangle$.
- There are stuck configurations: E.g. $\langle R_C[x], \sigma \rangle$ where $\sigma(x)$ is undefined.

By inspecting all syntactic cases one can verify:

Lemma

The reduction relation $\xrightarrow{eval}$ deterministic, i.e. if $\langle c, \sigma \rangle \xrightarrow{eval} \langle c', \sigma' \rangle$ and $\langle c, \sigma \rangle \xrightarrow{eval} \langle c'', \sigma'' \rangle$, then $c' = c''$ and $\sigma' = \sigma''$.

Example

$$
\begin{aligned}
& \langle (\texttt{while}\ \neg(x \leq 1)\ \texttt{do}\ x := x - 1\ \texttt{od}, \{x \mapsto 3\}\rangle \\
\xrightarrow{eval,while} & \langle \texttt{if}\ \neg(x \leq 1)\ \texttt{then}\ x := x - 1; \texttt{while}\ \neg(x \leq 1)\ \texttt{do}\ x := x - 1\ \texttt{od}\ \texttt{else skip fi}, \{x \mapsto 3\}\rangle \\
\xrightarrow{eval,loc} & \langle \texttt{if}\ \neg(3 \leq 1)\ \texttt{then}\ x := x - 1; \texttt{while}\ \neg(x \leq 1)\ \texttt{do}\ x := x - 1\ \texttt{od}\ \texttt{else skip fi}, \{x \mapsto 3\}\rangle \\
\xrightarrow{eval,leqF} & \langle \texttt{if}\ \neg\texttt{False}\ \texttt{then}\ x := x - 1; \texttt{while}\ \neg(x \leq 1)\ \texttt{do}\ x := x - 1\ \texttt{od}\ \texttt{else skip fi}, \{x \mapsto 3\}\rangle \\
\xrightarrow{eval,notF} & \langle \texttt{if True then}\ x := x - 1; \texttt{while}\ \neg(x \leq 1)\ \texttt{do}\ x := x - 1\ \texttt{od}\ \texttt{else skip fi}, \{x \mapsto 3\}\rangle \\
\xrightarrow{eval,ifT} & \langle x := x - 1; \texttt{while}\ \neg(x \leq 1)\ \texttt{do}\ x := x - 1\ \texttt{od}, \{x \mapsto 3\}\rangle \\
\xrightarrow{eval,loc} & \langle x := 3 - 1; \texttt{while}\ \neg(x \leq 1)\ \texttt{do}\ x := x - 1\ \texttt{od}, \{x \mapsto 3\}\rangle \\
\xrightarrow{eval,diff} & \langle x := 2; \texttt{while}\ \neg(x \leq 1)\ \texttt{do}\ x := x - 1\ \texttt{od}, \{x \mapsto 3\}\rangle \\
\xrightarrow{eval,asgn} & \langle \texttt{skip}; \texttt{while}\ \neg(x \leq 1)\ \texttt{do}\ x := x - 1\ \texttt{od}, \{x \mapsto 2\}\rangle \\
\xrightarrow{eval,skip} & \langle \texttt{while}\ \neg(x \leq 1)\ \texttt{do}\ x := x - 1\ \texttt{od}, \{x \mapsto 2\}\rangle \\
\xrightarrow{eval,while} & \langle \texttt{if}\ \neg(x \leq 1)\ \texttt{then}\ x := x - 1; \texttt{while}\ \neg(x \leq 1)\ \texttt{do}\ x := x - 1\ \texttt{od}\ \texttt{else skip fi}, \{x \mapsto 2\}\rangle \\
\xrightarrow{eval,loc} & \langle \texttt{if}\ \neg(2 \leq 1)\ \texttt{then}\ x := x - 1; \texttt{while}\ \neg(x \leq 1)\ \texttt{do}\ x := x - 1\ \texttt{od}\ \texttt{else skip fi}, \{x \mapsto 2\}\rangle
\end{aligned}
$$

Example (Cont'd)

hsrm

$$
\begin{aligned}
&\xrightarrow{eval,leqF} \langle \texttt{if } \neg\texttt{False then } x := x-1; \texttt{while } \neg(x \leq 1) \texttt{ do } x := x-1 \texttt{ od else skip fi}, \{x \mapsto 2\}\rangle \\
&\xrightarrow{eval,notF} \langle \texttt{if True then } x := x-1; \texttt{while } \neg(x \leq 1) \texttt{ do } x := x-1 \texttt{ od else skip fi}, \{x \mapsto 2\}\rangle \\
&\xrightarrow{eval,ifT} \langle x := x-1; \texttt{while } \neg(x \leq 1) \texttt{ do } x := x-1 \texttt{ od}, \{x \mapsto 2\}\rangle \\
&\xrightarrow{eval,loc} \langle x := 2-1; \texttt{while } \neg(x \leq 1) \texttt{ do } x := x-1 \texttt{ od}, \{x \mapsto 2\}\rangle \\
&\xrightarrow{eval,diff} \langle x := 1; \texttt{while } \neg(x \leq 1) \texttt{ do } x := x-1 \texttt{ od}, \{x \mapsto 2\}\rangle \\
&\xrightarrow{eval,asgn} \langle \texttt{skip}; \texttt{while } \neg(x \leq 1) \texttt{ do } x := x-1 \texttt{ od}, \{x \mapsto 1\}\rangle \\
&\xrightarrow{eval,skip} \langle \texttt{while } \neg(x \leq 1) \texttt{ do } x := x-1 \texttt{ od}], \{x \mapsto 1\}\rangle \\
&\xrightarrow{eval,while} \langle \texttt{if } \neg(x \leq 1) \texttt{ then } x := x-1; \texttt{while } \neg(x \leq 1) \texttt{ do } x := x-1 \texttt{ od else skip fi}, \{x \mapsto 1\}\rangle \\
&\xrightarrow{eval,loc} \langle \texttt{if } \neg(1 \leq 1) \texttt{ then } x := x-1; \texttt{while } \neg(x \leq 1) \texttt{ do } x := x-1 \texttt{ od else skip fi}, \{x \mapsto 1\}\rangle \\
&\xrightarrow{eval,leqT} \langle \texttt{if } \neg\texttt{True then } x := x-1; \texttt{while } \neg(x \leq 1) \texttt{ do } x := x-1 \texttt{ od else skip fi}, \{x \mapsto 1\}\rangle \\
&\xrightarrow{eval,notT} \langle \texttt{if False then } x := x-1; \texttt{while } \neg(x \leq 1) \texttt{ do } x := x-1 \texttt{ od else skip fi}, \{x \mapsto 1\}\rangle \\
&\xrightarrow{eval,ifF} \langle \texttt{skip}, \{x \mapsto 1\}\rangle
\end{aligned}
$$

Equivalence of Big-Step and Reduction Semantics

hsrm

Lemma

Let a be an arithmetic expression and σ be a state. Then $\langle a, \sigma\rangle \downarrow m$ iff $\langle a, \sigma\rangle \xrightarrow{n} \langle m, \sigma\rangle$ by applying the reduction rules.

Proof.
The "if"-direction can be shown by induction on the derivation tree for $\langle a, \sigma\rangle \downarrow m$
The "only-if"-direction can be shown by induction on the number n of steps.

Lemma

Let b be a boolean expression and σ be a state, $v \in \{\texttt{true}, \texttt{false}\}$. Then $\langle b, \sigma\rangle \downarrow v$ iff $\langle b, \sigma\rangle \xrightarrow{n} \langle v, \sigma\rangle$ by applying the reduction rules.

Proof.
The "if"-direction can be shown by induction on the derivation tree for $\langle b, \sigma\rangle \downarrow m$.
the "only-if"-direction can be shown by induction on the number n of steps.

Equivalence of Big-Step and Reduction Semantics (2)

hsrm

Proposition

For IMP-commands c and states $\sigma \in \Sigma$: $\langle c, \sigma\rangle \downarrow_{eval} \sigma'$ iff $\langle c, \sigma\rangle \downarrow \sigma'$

Proof. We only show one direction:

$$\langle c, \sigma\rangle \xrightarrow{eval,n} \langle \texttt{skip}, \sigma'\rangle \implies \langle c, \sigma\rangle \downarrow \sigma'$$

By induction on the number n of steps.

Base case: $n = 0$. Then $c = \texttt{skip}$, $\sigma' = \sigma$, and (AxSkip) shows the claim.

Step: $n > 0$ and $\langle c, \sigma\rangle \xrightarrow{eval} \langle c_1, \sigma_1\rangle \xrightarrow{eval,n-1} \langle \texttt{skip}, \sigma'\rangle$.

The induction hypothesis shows that $\langle c_1, \sigma_1\rangle \downarrow \sigma'$.

Now all cases of the first reduction step (and c, c_1, σ, σ_1) have to be considered.

Equivalence of Big-Step and Reduction Semantics (3)

hsrm

- An $\xrightarrow{eval,skip}$-step, $c = \texttt{skip}; c_1$, and $\sigma = \sigma_1$. Then $\dfrac{\overset{\text{(AxSkip)}}{\langle \texttt{skip} \rangle \downarrow \sigma} \quad \langle c_1, \sigma \rangle \downarrow \sigma'}{\langle \texttt{skip}; c_1, \sigma \rangle \downarrow \sigma'}\ \text{(Seq)}$
- An $\xrightarrow{eval,asgn}$-step. Two cases:
  - $c = x := m$, $c_1 = \texttt{skip}$ and $\sigma_1 = \sigma[m/x] = \sigma'$. Then $\dfrac{\overset{\text{(AxNum)}}{\langle m, \sigma \rangle \downarrow m}}{\langle x := m, \sigma \rangle \downarrow \sigma[m/x]}\ \text{(Asgn)}$
  - $c = x := m; c'$, $c_1 = \texttt{skip}; c'$, and $\sigma_1 = \sigma[m/x]$. Then $\langle \texttt{skip}; c', \sigma_1 \rangle \xrightarrow{eval,skip} \langle c', \sigma_1 \rangle$, and the induction hypothesis can also be applied to $\langle c', \sigma_1 \rangle \xrightarrow{eval,n-2} \langle \texttt{skip}, \sigma' \rangle$ showing $\langle c', \sigma_1 \rangle \downarrow \sigma'$. Then $\dfrac{\dfrac{\overset{\text{(AxNum)}}{\langle m, \sigma \rangle \downarrow m}}{\langle x := m, \sigma \rangle \downarrow \sigma_1}\ \text{(Asgn)} \quad \langle c', \sigma_1 \rangle \downarrow \sigma'}{\langle x := m; c_1, \sigma \rangle \downarrow \sigma'}\ \text{(Seq)}$
- $\xrightarrow{eval,ifT}$- or $\xrightarrow{eval,ifF}$-step: similar to the previous one.

Equivalence of Big-Step and Reduction Semantics (4)

hsrm

- $\xrightarrow{eval, while}$-step. Two cases, we only consider one:
  - $c = \texttt{while}\ b\ \texttt{do}\ c'\ \texttt{od}$, $c_1 = \texttt{if}\ b\ \texttt{then}\ c'; \texttt{while}\ b\ \texttt{do}\ c'\ \texttt{od}\ \texttt{else}\ \texttt{skip}\ \texttt{fi}$, and $\sigma_1 = \sigma$.

The reduction semantics will evaluate b until it is a boolean value.

Two subcases:

- b evaluates to `False`. Then

  $\langle \texttt{if}\ b\ \texttt{then}\ c'; \texttt{while}\ b\ \texttt{do}\ c'\ \texttt{od}\ \texttt{else}\ \texttt{skip}\ \texttt{fi}, \sigma\rangle$
  $\xrightarrow{eval, *} \langle \texttt{if False then}\ c'; \texttt{while}\ b\ \texttt{do}\ c'\ \texttt{od}\ \texttt{else}\ \texttt{skip}\ \texttt{fi}, \sigma\rangle$
  $\xrightarrow{eval, ifF} (\texttt{skip}, \sigma_2)$.

  Then also $\langle b, \sigma\rangle \xrightarrow{*} \langle \texttt{False}, \sigma\rangle$ and by the previous lemmas $\langle b, \sigma\rangle \downarrow \langle \texttt{False}, \sigma\rangle$.

  This shows

$$\text{(WhileF)}\ \frac{\langle b, \sigma\rangle \downarrow \texttt{False}}{\langle \texttt{while}\ b\ \texttt{do}\ c'\ \texttt{od}, \sigma\rangle \downarrow \sigma}$$

Equivalence of Big-Step and Reduction Semantics (5)

hsrm

- b evaluates to `True`. Then
  $\langle \texttt{if } b \texttt{ then } c'; \texttt{while } b \texttt{ do } c' \texttt{ od else skip fi}, \sigma\rangle$
  $\xrightarrow{eval,*} \langle \texttt{if True then } c'; \texttt{while } b \texttt{ do } c' \texttt{ od else skip fi}, \sigma\rangle$
  $\xrightarrow{eval,ifT} (c'; \texttt{while } b \texttt{ do } c' \texttt{ od}, \sigma)$
  Then also $\langle b, \sigma\rangle \xrightarrow{*} \langle \texttt{True}, \sigma\rangle$ and by the previous lemmas $\langle b, \sigma\rangle \downarrow \langle \texttt{False}, \sigma\rangle$.
  By the IH: $\langle c'; \texttt{while } b \texttt{ do } c' \texttt{ od}, \sigma\rangle \downarrow \sigma'$ and there exists a derivation tree:

$$\text{(Seq)}\ \frac{\langle c', \sigma\rangle \downarrow \sigma_2 \quad \langle \texttt{while } b \texttt{ do } c' \texttt{ od}, \sigma_2\rangle \downarrow \sigma'}{\langle c'; \texttt{while } b \texttt{ do } c' \texttt{ od}, \sigma\rangle \downarrow \sigma'}$$

  Thus $\langle c', \sigma\rangle \downarrow \sigma_2$ and $\langle \texttt{while } b \texttt{ do } c' \texttt{ od}, \sigma_2\rangle \downarrow \sigma'$ must hold
  Putting everything together:

$$\text{(WhileT)}\ \frac{\langle b, \sigma\rangle \downarrow \texttt{True} \quad \langle c', \sigma\rangle \downarrow \sigma_2 \quad \langle \texttt{while } b \texttt{ do } c' \texttt{ od}, \sigma_2\rangle \downarrow \sigma'}{\langle \texttt{while } b \texttt{ do } c' \texttt{ od}, \sigma\rangle \downarrow \sigma'}$$

Equivalence of Big-Step and Reduction Semantics (6)

hsrm

All other cases: the reduction step operates on a boolean or an arithmetic expression.
We only consider the case $c = R_c[\texttt{if } b \texttt{ then } c' \texttt{ else } c'' \texttt{ fi}]$.
Then $\langle c, \sigma\rangle \xrightarrow{eval,k} \langle R_c[\texttt{if } v \texttt{ then } c' \texttt{ else } c'' \texttt{ fi}], \sigma\rangle \xrightarrow{eval,n-k} \langle \texttt{skip}, \sigma'\rangle$ with $v \in \{\texttt{True}, \texttt{False}\}$ and $k \geq 1$.
Then also $\langle b, \sigma\rangle \xrightarrow{k} \langle v, \sigma\rangle$, and the previous lemmas show $\langle b, \sigma\rangle \downarrow v$.
We only consider the case $v = \texttt{True}$: Then

$$\langle R_c[\texttt{if } v \texttt{ then } c' \texttt{ else } c'' \texttt{ fi}], \sigma\rangle \xrightarrow{eval} \langle R_c[c'], \sigma\rangle \xrightarrow{eval,n-k-1} \langle \texttt{skip}, \sigma'\rangle$$

Since $k > 0$ the IH applied to $\langle R_c[c'], \sigma\rangle \xrightarrow{eval,n-k-1} \langle \texttt{skip}, \sigma'\rangle$ shows $\langle R_c[c'], \sigma\rangle \downarrow \sigma'$.
...

Equivalence of Big-Step and Reduction Semantics (7)

hsrm

...

If $R_c = [\cdot]$, then this shows

$$\text{(IfT)}\ \frac{\langle b, \sigma\rangle \downarrow \texttt{True} \quad \langle c', \sigma\rangle \downarrow \sigma'}{\langle \texttt{if}\ b\ \texttt{then}\ c'\ \texttt{else}\ c''\ \texttt{fi}, \sigma\rangle \downarrow \sigma'}$$

If $R_c = [\cdot]; c_0$ then $\langle R_c[c'], \sigma\rangle \downarrow \sigma'$ implies $\langle c', \sigma\rangle \downarrow \sigma_0$ and $\langle c_0, \sigma_0\rangle \downarrow \sigma'$ for some σ_0.

$$\text{(Seq)}\ \frac{\text{(IfT)}\ \dfrac{\langle b, \sigma\rangle \downarrow \texttt{True} \quad \langle c', \sigma\rangle \downarrow \sigma_0}{\langle \texttt{if}\ b\ \texttt{then}\ c'\ \texttt{else}\ c''\ \texttt{fi}, \sigma\rangle \downarrow \sigma_0} \qquad \langle c_0, \sigma_0\rangle \downarrow \sigma'}{\langle \texttt{if}\ b\ \texttt{then}\ c'\ \texttt{else}\ c''\ \texttt{fi}; c_0, \sigma\rangle \downarrow \sigma'}$$

All other cases are similar.

Sketch of Turing Completeness of IMP

hsrm

- Turing completeness can be shown by simulating a Turing machine with an IMP-program
- Proof in Schoening's book: While-programs and Goto-programs compute the same functions, and Goto-programs are shown to be Turing complete
- We give a sketch of a direct proof

Sketch of Turing Completeness of IMP(Cont'd)

hsrm

- Turing machine configuration wqw': Encode w, w' and state q by numbers to a base large enough to capture the tape alphabet. Then recode as integers
- The three numbers are stored in locations $x_w, x_{w'}, x_q$ of the IMP-program.
- Operations of a Turing machine (i.e. replacing the current symbol and moving the read-/write-head) are operations on the numbers (implemented using division with reminders, subtraction, addition, and multiplication.)

Sketch of Turing Completeness of IMP(Cont'd)

hsrm

Assume that $\Gamma = \{a_1, \ldots, a_n\}$, $Q = \{q_1, \ldots, q_m\}$ and F are final states of the TM. State transition of the TM is simulated by a single while-loop, written in pseudo-code:

```
while decode(x_q) ∉ F do
 if decode(x_q) = q_1 ∧ decode(x_w)) = a_1 v then adjust x_q, x_w, x'_w for δ(q_1, a_1) else
 if decode(x_q) = q_1 ∧ decode(x_w)) = a_2 v then adjust x_q, x_w, x'_w for δ(q_1, a_2) else
 ...
 if decode(x_q) = q_m ∧ decode(x_w)) = a_n v then adjust x_q, x_w, x'_w for δ(q_m, a_n) else
 skip
fi...fi
od
```

Abstract Machine Semantics of IMP

hsrm

- operational semantics as an abstract machine
- abstract means independent from real hardware
- usually easy to implement on real hardware
- we define an abstract machine for IMP

States of the IMP Machine

hsrm

The state of the IMP machine is a triple (E, T, S) with

- Environment E: maps storage locations to numbers
- Task T: a command or an (arithmetic or boolean) expression
- Stack S: Contains numbers, booleans, commands, etc.
  Notation:
  - $s_1; s_2; \ldots; s_n$ = stack with n-elements, where s_1 is on the top
  - $s_1; S$ = stack with top element s_1 and S is the remaining stack.
  - $[]$ = empty stack.

Start state for program c: $(\emptyset, c, [])$.

Start state for program c in environment E: $(E, c, [])$.

Final state = any state of the form $(E, \texttt{skip}, [])$

Stack Entries

hsrm

- commands c
- branches $[T : c_1, F : c_2]$ to continue the evaluation of a conditional or a while loop
- $x :=$ means that x has to be updated in the environment
- $(\oplus t)$ means that the current task evaluates the left argument of operator $\oplus \in \{+, -, *, =, \leq, \wedge, \vee\}$ where t is the right argument
- $\neg$ to negate the result of the current task
- $(n \oplus)$ means that the right argument of $\oplus \in \{+, -, *, =, \leq\}$ is currently evaluated

Transition Relation $\rightsquigarrow$ of the IMP Machine (1)

hsrm

$$
\begin{array}{lcl}
(E, (c_1; c_2), S) & \rightsquigarrow & (E, c_1, c_2; S) \\
(E, x := a, S) & \rightsquigarrow & (E, a, x :=; S) \\
(E, n, x :=; S) & \rightsquigarrow & (E[n/x], \texttt{skip}, S) \\
(E, x, S) & \rightsquigarrow & (E, n, S) \text{ if } E(x) = n \\
(E, \texttt{while } b \texttt{ do } c \texttt{ od}, S) & \rightsquigarrow & (E, b, [T : c; \texttt{while } b \texttt{ do } c \texttt{ od}, F : \texttt{skip}]; S) \\
(E, \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ fi}, S) & \rightsquigarrow & (E, b, [T : c_1, F : c_2]; S) \\
(E, \texttt{skip}, c; S) & \rightsquigarrow & (E, c, S) \\
(E, \texttt{True}, [T : c_1, F : c_2]; S) & \rightsquigarrow & (E, c_1, S) \\
(E, \texttt{False}, [T : c_1, F : c_2]; S) & \rightsquigarrow & (E, c_2, S)
\end{array}
$$

Transition Relation $\rightsquigarrow$ of the IMP Machine (2)

hsrm

$$
\begin{array}{lcl}
(E, a_1 + a_2, S) & \rightsquigarrow & (E, a_1, (+a_2); S) \\
(E, n, (+a); S) & \rightsquigarrow & (E, a, (n+); S) \\
(E, m, (n+); S) & \rightsquigarrow & (E, m', S) \text{ if } m' = n + m \\
(E, a_1 - a_2, S) & \rightsquigarrow & (E, a_1, (-a_2); S) \\
(E, n, (-a); S) & \rightsquigarrow & (E, a, (n-); S) \\
(E, m, (n-); S) & \rightsquigarrow & (E, m', S) \text{ if } m' = n - m \\
(E, a_1 * a_2, S) & \rightsquigarrow & (E, a_1, (*a_2); S) \\
(E, n, (*a); S) & \rightsquigarrow & (E, a, (n*); S) \\
(E, m, (n*); S) & \rightsquigarrow & (E, m', S) \text{ if } m' = n \cdot m
\end{array}
$$

Transition Relation $\rightsquigarrow$ of the IMP Machine (3)

hsrm

$$
\begin{array}{lcl}
(E, b_1 \wedge b_2, S) & \rightsquigarrow & (E, b_2, (\wedge\ b_2); S) \\
(E, \texttt{True}, (\wedge\ b_2); S) & \rightsquigarrow & (E, b_2, S) \\
(E, \texttt{False}, (\wedge\ b_2); S) & \rightsquigarrow & (E, \texttt{False}, S) \\
(E, b_1 \vee b_2, S) & \rightsquigarrow & (E, b_2, (\vee\ b_2); S) \\
(E, \texttt{True}, (\vee\ b_2); S) & \rightsquigarrow & (E, \texttt{True}, S) \\
(E, \texttt{False}, (\vee\ b_2); S) & \rightsquigarrow & (E, b_2, S) \\
(E, \neg b, S) & \rightsquigarrow & (E, b, \neg; S) \\
(E, \texttt{True}, \neg; S) & \rightsquigarrow & (E, \texttt{False}, S) \\
(E, \texttt{False}, \neg; S) & \rightsquigarrow & (E, \texttt{True}, S)
\end{array}
$$

Transition Relation $\rightsquigarrow$ of the IMP Machine (4)

hsrm

$$
\begin{array}{lcl}
(E, a_1 = a_2, S) & \rightsquigarrow & (E, a_1, (= a_2); S) \\
(E, n, (= a); S) & \rightsquigarrow & (E, a, (n =); S) \\
(E, m, (n =); S) & \rightsquigarrow & (E, \texttt{True}, S) \text{ if } m = n \\
(E, m, (n =); S) & \rightsquigarrow & (E, \texttt{False}, S) \text{ if } m \neq n \\
(E, a_1 \leq a_2, S) & \rightsquigarrow & (E, a_1, (\leq a_2); S) \\
(E, n, (\leq a); S) & \rightsquigarrow & (E, a, (n \leq); S) \\
(E, m, (n \leq); S) & \rightsquigarrow & (E, \texttt{True}, S) \text{ if } n \leq m \\
(E, m, (n \leq); S) & \rightsquigarrow & (E, \texttt{False}, S) \text{ if } n > m
\end{array}
$$

Example

$$(\emptyset, x := 2; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, [])$$

Example

$$(\emptyset, x := 2; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, [])$$

$$\rightsquigarrow \quad (\emptyset, x := 2, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od})$$

Example

$$
\begin{aligned}
& (\emptyset, x := 2; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
\rightsquigarrow \quad & (\emptyset, x := 2, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow \quad & (\emptyset, 2, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od})
\end{aligned}
$$

Example

$$
\begin{aligned}
& (\emptyset, x := 2; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
\rightsquigarrow \quad & (\emptyset, x := 2, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow \quad & (\emptyset, 2, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow \quad & (\{x \mapsto 2\}, \texttt{skip}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od})
\end{aligned}
$$

Example

$$
\begin{aligned}
& (\emptyset, x := 2; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
\rightsquigarrow \quad & (\emptyset, x := 2, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow \quad & (\emptyset, 2, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow \quad & (\{x \mapsto 2\}, \texttt{skip}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow \quad & (\{x \mapsto 2\}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, [])
\end{aligned}
$$

Example

$$
\begin{aligned}
& (\emptyset, x := 2; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
\rightsquigarrow \quad & (\emptyset, x := 2, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow \quad & (\emptyset, 2, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow \quad & (\{x \mapsto 2\}, \texttt{skip}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow \quad & (\{x \mapsto 2\}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
\rightsquigarrow \quad & (\{x \mapsto 2\}, 2 \leq x, [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}])
\end{aligned}
$$

Example

hsrm

$$
\begin{aligned}
& (\emptyset, x := 2; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
\rightsquigarrow \quad & (\emptyset, x := 2, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow \quad & (\emptyset, 2, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow \quad & (\{x \mapsto 2\}, \texttt{skip}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow \quad & (\{x \mapsto 2\}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
\rightsquigarrow \quad & (\{x \mapsto 2\}, 2 \leq x, [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
\rightsquigarrow \quad & (\{x \mapsto 2\}, 2, \leq x; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}])
\end{aligned}
$$

Example

$$
\begin{aligned}
& (\emptyset, x := 2; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
\rightsquigarrow\ & (\emptyset, x := 2, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow\ & (\emptyset, 2, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, \texttt{skip}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, 2 \leq x, [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, 2, \leq x; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, x, 2 \leq; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}])
\end{aligned}
$$

Example

$$
\begin{aligned}
& (\emptyset, x := 2; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
\rightsquigarrow\ & (\emptyset, x := 2, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow\ & (\emptyset, 2, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, \texttt{skip}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, 2 \leq x, [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, 2, \leq x; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, x, 2 \leq; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, 2, 2 \leq; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}])
\end{aligned}
$$

Example

$$
\begin{aligned}
& (\emptyset, x := 2; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
\rightsquigarrow\ & (\emptyset, x := 2, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow\ & (\emptyset, 2, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, \texttt{skip}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, 2 \leq x, [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, 2, \leq x; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, x, 2 \leq; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, 2, 2 \leq; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, \texttt{True}; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}])
\end{aligned}
$$

Example

$$
\begin{aligned}
& (\emptyset, x := 2; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
\rightsquigarrow\ & (\emptyset, x := 2, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow\ & (\emptyset, 2, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, \texttt{skip}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, 2 \leq x, [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, 2, \leq x; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, x, 2 \leq; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, 2, 2 \leq; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, \texttt{True}; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
\rightsquigarrow\ & (\{x \mapsto 2\}, x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od})
\end{aligned}
$$

Example (Cont'd')

hsrm

$$\rightsquigarrow \quad (\{x \mapsto 2\}, x - 1, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od})$$

Example (Cont'd')

$$\rightsquigarrow \quad (\{x \mapsto 2\}, x - 1, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od})$$
$$\rightsquigarrow \quad (\{x \mapsto 2\}, x, -1; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od})$$

Example (Cont'd')

$$\rightsquigarrow \quad (\{x \mapsto 2\}, x - 1, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od})$$
$$\rightsquigarrow \quad (\{x \mapsto 2\}, x, -1; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od})$$
$$\rightsquigarrow \quad (\{x \mapsto 2\}, 2, -1; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od})$$

Example (Cont'd')

$$
\begin{aligned}
&\rightsquigarrow \quad (\{x \mapsto 2\}, x - 1, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, x, -1; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 2, -1; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 1, 2-; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od})
\end{aligned}
$$

Example (Cont'd')

$$
\begin{aligned}
&\rightsquigarrow \quad (\{x \mapsto 2\}, x - 1, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, x, -1; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 2, -1; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 1, 2-; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 1, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od})
\end{aligned}
$$

Example (Cont'd')

$$
\begin{aligned}
&\rightsquigarrow \quad (\{x \mapsto 2\}, x - 1, x := ; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, x, -1; x := ; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 2, -1; x := ; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 1, 2-; x := ; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 1, x := ; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 1\}, \texttt{skip}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od})
\end{aligned}
$$

Example (Cont'd')

hsrm

$$
\begin{aligned}
&\rightsquigarrow \quad (\{x \mapsto 2\}, x - 1, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, x, -1; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 2, -1; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 1, 2-; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 1, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 1\}, \texttt{skip}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 1\}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, [])
\end{aligned}
$$

Example (Cont'd')

hsrm

$$
\begin{aligned}
&\rightsquigarrow \quad (\{x \mapsto 2\}, x - 1, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, x, -1; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 2, -1; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 1, 2-; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 1, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 1\}, \texttt{skip}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 1\}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
&\rightsquigarrow \quad (\{x \mapsto 1\}, 2 \leq x, [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}])
\end{aligned}
$$

Example (Cont'd')

hsrm

$$
\begin{aligned}
&\rightsquigarrow \quad (\{x \mapsto 2\}, x - 1, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, x, -1; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 2, -1; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 1, 2-; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 1, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 1\}, \texttt{skip}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 1\}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
&\rightsquigarrow \quad (\{x \mapsto 1\}, 2 \leq x, [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
&\rightsquigarrow \quad (\{x \mapsto 1\}, 2, \leq x; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}])
\end{aligned}
$$

Example (Cont'd')

hsrm

$$
\begin{aligned}
&\rightsquigarrow \quad (\{x \mapsto 2\}, x - 1, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, x, -1; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 2, -1; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 1, 2-; x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 2\}, 1, x :=; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 1\}, \texttt{skip}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}) \\
&\rightsquigarrow \quad (\{x \mapsto 1\}, \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, []) \\
&\rightsquigarrow \quad (\{x \mapsto 1\}, 2 \leq x, [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
&\rightsquigarrow \quad (\{x \mapsto 1\}, 2, \leq x; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}]) \\
&\rightsquigarrow \quad (\{x \mapsto 1\}, x, 2 \leq; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}])
\end{aligned}
$$

Example (Cont'd')

hsrm

$$
\begin{aligned}
&\rightsquigarrow\ (\{x \mapsto 2\}, x-1, x :=; \texttt{while}\ 2 \leq x\ \texttt{do}\ x := x-1\ \texttt{od})\\
&\rightsquigarrow\ (\{x \mapsto 2\}, x, -1; x :=; \texttt{while}\ 2 \leq x\ \texttt{do}\ x := x-1\ \texttt{od})\\
&\rightsquigarrow\ (\{x \mapsto 2\}, 2, -1; x :=; \texttt{while}\ 2 \leq x\ \texttt{do}\ x := x-1\ \texttt{od})\\
&\rightsquigarrow\ (\{x \mapsto 2\}, 1, 2-; x :=; \texttt{while}\ 2 \leq x\ \texttt{do}\ x := x-1\ \texttt{od})\\
&\rightsquigarrow\ (\{x \mapsto 2\}, 1, x :=; \texttt{while}\ 2 \leq x\ \texttt{do}\ x := x-1\ \texttt{od})\\
&\rightsquigarrow\ (\{x \mapsto 1\}, \texttt{skip}, \texttt{while}\ 2 \leq x\ \texttt{do}\ x := x-1\ \texttt{od})\\
&\rightsquigarrow\ (\{x \mapsto 1\}, \texttt{while}\ 2 \leq x\ \texttt{do}\ x := x-1\ \texttt{od}, [])\\
&\rightsquigarrow\ (\{x \mapsto 1\}, 2 \leq x, [T : x := x-1; \texttt{while}\ 2 \leq x\ \texttt{do}\ x := x-1\ \texttt{od}, F : \texttt{skip}])\\
&\rightsquigarrow\ (\{x \mapsto 1\}, 2, \leq x; [T : x := x-1; \texttt{while}\ 2 \leq x\ \texttt{do}\ x := x-1\ \texttt{od}, F : \texttt{skip}])\\
&\rightsquigarrow\ (\{x \mapsto 1\}, x, 2 \leq; [T : x := x-1; \texttt{while}\ 2 \leq x\ \texttt{do}\ x := x-1\ \texttt{od}, F : \texttt{skip}])\\
&\rightsquigarrow\ (\{x \mapsto 1\}, 1, 2 \leq; [T : x := x-1; \texttt{while}\ 2 \leq x\ \texttt{do}\ x := x-1\ \texttt{od}, F : \texttt{skip}])
\end{aligned}
$$

Example (Cont'd')

$$\rightsquigarrow \quad (\{x \mapsto 1\}, \texttt{False}; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}])$$

Example (Cont'd')

$$\rightsquigarrow \quad (\{x \mapsto 1\}, \texttt{False}; [T : x := x - 1; \texttt{while } 2 \leq x \texttt{ do } x := x - 1 \texttt{ od}, F : \texttt{skip}])$$

$$\rightsquigarrow \quad (\{x \mapsto 1\}, \texttt{skip}, [])$$

Machine Evaluation: Equivalence of other Semantics

hsrm

Definition

Let $\overset{*}{\rightsquigarrow}$ be the reflexive-transitive closure of $\rightsquigarrow$ and let $\overset{n}{\rightsquigarrow}$ be n steps of $\rightsquigarrow$.
For a program c and an environment σ, we write

$$\langle c, \sigma\rangle \downarrow_{absm} \sigma' \text{ iff } (\sigma, c, \emptyset) \overset{*}{\rightsquigarrow} (\sigma', \texttt{skip}, [])$$

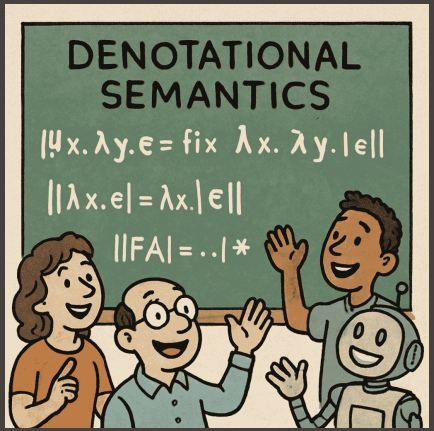
Theorem

The abstract machine is equivalent to the big-step semantics and to the reduction semantics, i.e. $\langle c, \sigma\rangle \downarrow_{absm} \sigma$ iff $\langle c, \sigma\rangle \downarrow_{eval} \sigma$ and $\langle c, \sigma\rangle \downarrow_{absm} \sigma$ iff $\langle c, \sigma\rangle \downarrow \sigma$ and

Proof Sketch. It suffices to show the claim for the reduction semantics. It is straight-forward by an induction on the number of $\xrightarrow{eval}$-steps for one direction, and another induction on the number of $\rightsquigarrow$-steps for the other direction.

hsrm*

Denotational Semantics



Denotational Semantics

hsrm

- Goal: define a denotational semantics for IMP
- Idea of denotational semantics:

  map each program construct to a mathematical object

- Since the meaning of an arithmetic or boolean expression or command depends on the state, the mathematical objects are

  relations between states and values.

- Since evaluation in IMP is deterministic, these relations are (partial) functions
- Partiality is necessary, since programs may loop, etc.

Denotation

hsrm

- We write $\mathcal{A}, \mathcal{B}$, and $\mathcal{C}$ for the denotation of arithmetic expressions, boolean expressions, and commands.
- The syntactic argument is written in $[\![\cdot]\!]$ brackets

This means, we write:

- for arithmetic expression a, $\mathcal{A}[\![a]\!] : \Sigma \to \mathbb{Z}$
- for boolean expression b, $\mathcal{B}[\![b]\!] : \Sigma \to \{\texttt{True}, \texttt{False}\}$
- for command c, $\mathcal{C}[\![c]\!] : \Sigma \to \Sigma$

where the images are partial functions.

To describe partial functions, we use λ-notation and write $\lambda\sigma \in \Sigma.e$ to explicitly note that σ must be state.

Partial Functions

hsrm

- A partial function $f : M \to N$ is not necessarily defined for all elements of M (we write $f(x) = \perp$ if f is not defined for $x \in M$.)
- The domain of partial function f, is denoted as $Dom(f)$ ($Dom(f) = \{x \in M \mid f(x) \neq \perp\}$)
- Function f with $Dom(f) = \emptyset$ is never defined (f is called the empty function, and written as $\emptyset$)

Denotational Semantics of Arithmetic Expressions

hsrm

Definition

$$
\begin{array}{lcl}
\mathcal{A}[\![n]\!] & := & \lambda\sigma \in \Sigma.n, \text{ if } n \in \mathbb{Z} \\
\mathcal{A}[\![x]\!] & := & \lambda\sigma \in \Sigma.\sigma(x) \text{ if } x \in Loc \\
\mathcal{A}[\![a_1 + a_2]\!] & := & \lambda\sigma \in \Sigma.(\mathcal{A}[\![a_1]\!]\sigma) + (\mathcal{A}[\![a_2]\!]\sigma) \\
\mathcal{A}[\![a_1 - a_2]\!] & := & \lambda\sigma \in \Sigma.(\mathcal{A}[\![a_1]\!]\sigma) - (\mathcal{A}[\![a_2]\!]\sigma) \\
\mathcal{A}[\![a_1 * a_2]\!] & := & \lambda\sigma \in \Sigma.(\mathcal{A}[\![a_1]\!]\sigma) \cdot (\mathcal{A}[\![a_2]\!]\sigma)
\end{array}
$$

Remarks:

- If $\sigma(x)$ is not defined, then $\sigma \notin Dom(\lambda\sigma \in \Sigma.\sigma(x))$.
- Numbers n, operators $+, -$ have a different meaning on the lhs and the rhs of $:=$
  - on the left hand side, they are syntax of IMP
  - on the right hand side, they are integers and mathematical operations
- $\mathcal{A}[\![\cdot]\!]$ is also called a semantic function. The domains are arithmetic IMP expressions, the co-domain are sets of partial functions from states to integers

Denotational Semantics of Boolean Expressions

hsrm

Definition

$$
\begin{array}{lcl}
\mathcal{B}[\![\texttt{True}]\!] & := & \lambda\sigma \in \Sigma.\texttt{True} \\
\mathcal{B}[\![\texttt{False}]\!] & := & \lambda\sigma \in \Sigma.\texttt{False} \\
\mathcal{B}[\![a_1 = a_2]\!] & := & \lambda\sigma \in \Sigma.\mathcal{A}[\![a_1]\!]\sigma = \mathcal{A}[\![a_2]\!]\sigma \\
\mathcal{B}[\![a_1 \leq a_2]\!] & := & \lambda\sigma \in \Sigma.\mathcal{A}[\![a_1]\!]\sigma \leq \mathcal{A}[\![a_2]\!]\sigma \\
\mathcal{B}[\![\neg b]\!] & := & \lambda\sigma \in \Sigma.\neg(\mathcal{B}[\![b]\!]\sigma) \\
\mathcal{B}[\![b_1 \vee b_2]\!] & := & \lambda\sigma \in \Sigma.(\mathcal{B}[\![b_1]\!]\sigma) \vee (\mathcal{B}[\![b_2]\!]\sigma) \\
\mathcal{B}[\![b_1 \wedge b_2]\!] & := & \lambda\sigma \in \Sigma.(\mathcal{B}[\![b_1]\!]\sigma) \wedge (\mathcal{B}[\![b_2]\!]\sigma)
\end{array}
$$

Again $\texttt{True}, \texttt{False}, =, \leq, \vee, \wedge, \neg$ on the lhs and rhs of $:=$ have a different meaning

Denotational Semantics of Commands

hsrm

For the denotational semantics of commands, we introduce a helper function

$$\texttt{cond} : (\Sigma \to \{\texttt{True}, \texttt{False}\}) \to (\Sigma \to \Sigma) \to (\Sigma \to \Sigma) \to \Sigma \to \Sigma$$

defined as

$$(\texttt{cond}\ f\ g_1\ g_2)\ \sigma = \begin{cases} g_1\ \sigma, & \text{if } f\ \sigma = \texttt{True} \\ g_2\ \sigma, & \text{if } f\ \sigma = \texttt{False} \end{cases}$$

This is like an if-then-else on the semantic level.

We also define the identity:

$$\texttt{id}_\Sigma := \lambda\sigma \in \Sigma.\sigma$$

Denotation of Commands (without `while`)

hsrm

Definition

$$
\begin{array}{lcl}
\mathcal{C}[\![\texttt{skip}]\!] & := & \texttt{id}_\Sigma \\
\mathcal{C}[\![x := a]\!] & := & \lambda\sigma \in \Sigma.\sigma[(\mathcal{A}[\![a]\!]\sigma)/x] \\
\mathcal{C}[\![c_0; c_1]\!] & := & \mathcal{C}[\![c_1]\!] \circ \mathcal{C}[\![c_0]\!] \\
 & = & \lambda\sigma \in \Sigma.(\mathcal{C}[\![c_1]\!])(\mathcal{C}[\![c_0]\!]\sigma) \\
\mathcal{C}[\![\texttt{if } b \texttt{ then } c_0 \texttt{ else } c_1 \texttt{ fi}]\!] & := & \lambda\sigma \in \Sigma.(\texttt{cond } (\mathcal{B}[\![b]\!]) \ (\mathcal{C}[\![c_0]\!]) \ (\mathcal{C}[\![c_1]\!])) \ \sigma
\end{array}
$$

Note that $\sigma \notin Dom(\mathcal{C}[\![x := a]\!])$ if $(\mathcal{A}[\![a]\!]\sigma)$ is undefined.

Denotation of While

hsrm

Defining the denotation of `while` is not straight-forward

Denotation of While

hsrm

Defining the denotation of `while` is not straight-forward

A first approach is to use the equivalence

$$\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}\ \sim\ \texttt{if}\ b\ \texttt{then}\ c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}\ \texttt{else skip fi}$$

This results in

$$\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!] := \lambda\sigma \in \Sigma.(\texttt{cond}\ (\mathcal{B}[\![b]\!])\ (\mathcal{C}[\![c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!])\ \texttt{id}_\Sigma)\ \sigma$$

which can be simplified to

$$\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!] := \texttt{cond}\ (\mathcal{B}[\![b]\!])\ (\mathcal{C}[\![c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!])\ \texttt{id}_\Sigma$$

Denotation of While

hsrm

Defining the denotation of `while` is not straight-forward

A first approach is to use the equivalence

$$\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}\ \sim\ \texttt{if}\ b\ \texttt{then}\ c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}\ \texttt{else skip fi}$$

This results in

$$\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!] := \lambda\sigma \in \Sigma.(\texttt{cond}\ (\mathcal{B}[\![b]\!])\ (\mathcal{C}[\![c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!])\ \texttt{id}_\Sigma)\ \sigma$$

which can be simplified to

$$\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!] := \texttt{cond}\ (\mathcal{B}[\![b]\!])\ (\mathcal{C}[\![c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!])\ \texttt{id}_\Sigma$$

Computing the denotation for the sequence $c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}$:

$$\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!] := \texttt{cond}\ (\mathcal{B}[\![b]\!])\ ((\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!]) \circ (\mathcal{C}[\![c_0]\!]))\ \texttt{id}_\Sigma$$

Denotation of While

hsrm

Defining the denotation of `while` is not straight-forward

A first approach is to use the equivalence

$$\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}\ \sim\ \texttt{if}\ b\ \texttt{then}\ c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}\ \texttt{else skip fi}$$

This results in

$$\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!] := \lambda\sigma \in \Sigma.(\texttt{cond}\ (\mathcal{B}[\![b]\!])\ (\mathcal{C}[\![c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!])\ \texttt{id}_\Sigma)\ \sigma$$

which can be simplified to

$$\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!] := \texttt{cond}\ (\mathcal{B}[\![b]\!])\ (\mathcal{C}[\![c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!])\ \texttt{id}_\Sigma$$

Computing the denotation for the sequence $c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}$:

$$\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!] := \texttt{cond}\ (\mathcal{B}[\![b]\!])\ ((\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!]) \circ (\mathcal{C}[\![c_0]\!]))\ \texttt{id}_\Sigma$$

➽ the lhs $\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!]$ of the defining equation occurs in the rhs.

Denotation of While

hsrm

Defining the denotation of `while` is not straight-forward

A first approach is to use the equivalence

$$\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}\ \sim\ \texttt{if}\ b\ \texttt{then}\ c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}\ \texttt{else skip fi}$$

This results in

$$\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!] := \lambda\sigma \in \Sigma.(\texttt{cond}\ (\mathcal{B}[\![b]\!])\ (\mathcal{C}[\![c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!])\ \texttt{id}_\Sigma)\ \sigma$$

which can be simplified to

$$\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!] := \texttt{cond}\ (\mathcal{B}[\![b]\!])\ (\mathcal{C}[\![c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!])\ \texttt{id}_\Sigma$$

Computing the denotation for the sequence $c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}$:

$$\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!] := \texttt{cond}\ (\mathcal{B}[\![b]\!])\ ((\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!]) \circ (\mathcal{C}[\![c_0]\!]))\ \texttt{id}_\Sigma$$

➽ the lhs $\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!]$ of the defining equation occurs in the rhs.

➽ This is a circular description and not a well-formed definition!

Denotation of While (Cont'd)

hsrm

Use the "circular description" to find the definition:

$$\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!] = \texttt{cond}\ (\mathcal{B}[\![b]\!])\ ((\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!]) \circ (\mathcal{C}[\![c_0]\!]))\ \texttt{id}_\Sigma$$

Denotation of While (Cont'd)

hsrm

Use the "circular description" to find the definition:

$$\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!] = \texttt{cond}\ (\mathcal{B}[\![b]\!])\ ((\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!]) \circ (\mathcal{C}[\![c_0]\!]))\ \texttt{id}_\Sigma$$

Let $\varphi = \mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!]$, then we can write

$$\varphi = \texttt{cond}\ (\mathcal{B}[\![b]\!])\ (\varphi \circ (\mathcal{C}[\![c_0]\!]))\ \texttt{id}_\Sigma$$

Denotation of While (Cont'd)

hsrm

Use the "circular description" to find the definition:

$$\mathcal{C}[\![\texttt{while } b \texttt{ do } c_0 \texttt{ od}]\!] = \texttt{cond } (\mathcal{B}[\![b]\!])\ ((\mathcal{C}[\![\texttt{while } b \texttt{ do } c_0 \texttt{ od}]\!]) \circ (\mathcal{C}[\![c_0]\!]))\ \texttt{id}_\Sigma$$

Let $\varphi = \mathcal{C}[\![\texttt{while } b \texttt{ do } c_0 \texttt{ od}]\!]$, then we can write

$$\varphi = \texttt{cond } (\mathcal{B}[\![b]\!])\ (\varphi \circ (\mathcal{C}[\![c_0]\!]))\ \texttt{id}_\Sigma$$

Let Γ be the function that does the computation on φ on the rhs:

$$\Gamma = \lambda u \in (\Sigma \to \Sigma).\texttt{cond } (\mathcal{B}[\![b]\!])\ (u \circ (\mathcal{C}[\![c_0]\!]))\ \texttt{id}_\Sigma$$

Note that $\Gamma : (\Sigma \to \Sigma) \to (\Sigma \to \Sigma)$ takes a function of type $\Sigma \to \Sigma$ and returns a function of type $\Sigma \to \Sigma$.

Denotation of While (Cont'd)

hsrm

Use the "circular description" to find the definition:

$$\mathcal{C}[\![\texttt{while } b \texttt{ do } c_0 \texttt{ od}]\!] = \texttt{cond } (\mathcal{B}[\![b]\!])\ ((\mathcal{C}[\![\texttt{while } b \texttt{ do } c_0 \texttt{ od}]\!]) \circ (\mathcal{C}[\![c_0]\!]))\ \texttt{id}_\Sigma$$

Let $\varphi = \mathcal{C}[\![\texttt{while } b \texttt{ do } c_0 \texttt{ od}]\!]$, then we can write

$$\varphi = \texttt{cond } (\mathcal{B}[\![b]\!])\ (\varphi \circ (\mathcal{C}[\![c_0]\!]))\ \texttt{id}_\Sigma$$

Let Γ be the function that does the computation on φ on the rhs:

$$\Gamma = \lambda u \in (\Sigma \to \Sigma).\texttt{cond } (\mathcal{B}[\![b]\!])\ (u \circ (\mathcal{C}[\![c_0]\!]))\ \texttt{id}_\Sigma$$

Note that $\Gamma : (\Sigma \to \Sigma) \to (\Sigma \to \Sigma)$ takes a function of type $\Sigma \to \Sigma$ and returns a function of type $\Sigma \to \Sigma$.

Using Γ, the equation becomes $\boxed{\varphi = \Gamma(\varphi)}$

Denotation of While (Cont'd)

hsrm

Use the "circular description" to find the definition:

$$\mathcal{C}[\![\texttt{while } b \texttt{ do } c_0 \texttt{ od}]\!] = \texttt{cond } (\mathcal{B}[\![b]\!])\ ((\mathcal{C}[\![\texttt{while } b \texttt{ do } c_0 \texttt{ od}]\!]) \circ (\mathcal{C}[\![c_0]\!]))\ \texttt{id}_\Sigma$$

Let $\varphi = \mathcal{C}[\![\texttt{while } b \texttt{ do } c_0 \texttt{ od}]\!]$, then we can write

$$\varphi = \texttt{cond } (\mathcal{B}[\![b]\!])\ (\varphi \circ (\mathcal{C}[\![c_0]\!]))\ \texttt{id}_\Sigma$$

Let Γ be the function that does the computation on φ on the rhs:

$$\Gamma = \lambda u \in (\Sigma \to \Sigma).\texttt{cond } (\mathcal{B}[\![b]\!])\ (u \circ (\mathcal{C}[\![c_0]\!]))\ \texttt{id}_\Sigma$$

Note that $\Gamma : (\Sigma \to \Sigma) \to (\Sigma \to \Sigma)$ takes a function of type $\Sigma \to \Sigma$ and returns a function of type $\Sigma \to \Sigma$.

Using Γ, the equation becomes $\boxed{\varphi = \Gamma(\varphi)}$

Hence, φ is a **fixpoint** of Γ – the denotation of $\mathcal{C}[\![\texttt{while } b \texttt{ do } c_0 \texttt{ od}]\!]$ is a fixpoint of Γ!

Denotation of While (Cont'd)

hsrm

We use the least fixpoint of Γ, and omit the proof that it **exists** and that it **can always be constructed** (see literature)

Let us write $\texttt{Fix}(\Gamma)$ for least fixpoint of Γ.

Definition (Denotation of `while`)

$$\mathcal{C}[\![\texttt{while } b \texttt{ do } c_0 \texttt{ od}]\!] := \texttt{Fix}(\Gamma)$$

$$\text{where } \Gamma := \lambda u : \Sigma \to \Sigma.\texttt{cond } (\mathcal{B}[\![b]\!]) \ (u \circ (\mathcal{C}[\![c_0]\!])) \ \texttt{id}_\Sigma$$

But: How can we compute the fixpoint?

Computing the Fixpoint

hsrm

An idea to compute the least fixpoint:

- compute the partial functions $F_n[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!]$

  that represent the denotation of $\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}$

  - where only n iterations are allowed
  - for states σ that require more than n iterations, F_n is undefined

Computing the Fixpoint

hsrm

An idea to compute the least fixpoint:

- compute the partial functions $F_n[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!]$

  that represent the denotation of $\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}$

  - where only n iterations are allowed
  - for states σ that require more than n iterations, F_n is undefined
- the denotation $\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!]$ is the union of all functions $F_n[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!]$

Computing the Fixpoint (Cont'd)

hsrm

- For $n = 0$: function $F_0[\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}]\!]$ is only defined for states σ where no iteration of the loop is necessary, i.e. states σ with $\mathcal{B}[\![b]\!]\sigma = \texttt{False}$
  This shows

$$F_0[\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}]\!] = \{\sigma \mapsto \sigma \mid \mathcal{B}[\![b]\!](\sigma) = \texttt{False}\}$$

Computing the Fixpoint (Cont'd)

hsrm

- For $n = 0$: function $F_0[\![\texttt{while } b \texttt{ do } c \texttt{ od}]\!]$ is only defined for states σ where no iteration of the loop is necessary, i.e. states σ with $\mathcal{B}[\![b]\!]\sigma = \texttt{False}$
  This shows

$$F_0[\![\texttt{while } b \texttt{ do } c \texttt{ od}]\!] = \{\sigma \mapsto \sigma \mid \mathcal{B}[\![b]\!](\sigma) = \texttt{False}\}$$

- For $n = 1$: function $F_1[\![\texttt{while } b \texttt{ do } c \texttt{ od}]\!]$ is defined for states σ where at most 1 iteration of the loop is necessary:

$$\begin{aligned} F_1[\![\texttt{while } b \texttt{ do } c \texttt{ od}]\!] = \quad & \{\sigma \mapsto \sigma \mid \mathcal{B}[\![b]\!](\sigma) = \texttt{False}\} \\ \cup \; & \{\sigma \mapsto \sigma' \mid \mathcal{B}[\![b]\!](\sigma) = \texttt{True}, \mathcal{C}[\![c]\!](\sigma) = \sigma' \text{ and } \mathcal{B}[\![b]\!](\sigma') = \texttt{False}\} \end{aligned}$$

Computing the Fixpoint (Cont'd)

hsrm

- general case, $n \geq 1$:

$$
\begin{aligned}
F_n[\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}]\!] = \quad & F_{n-1}[\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}]\!] \\
& \cup \{\sigma \mapsto \sigma' \mid F_{n-1}[\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}]\!](\sigma) = \sigma', \mathcal{B}[\![b]\!](\sigma') = \texttt{True}, \\
& \qquad \mathcal{C}[\![c]\!](\sigma') = \sigma'', \text{ and } \mathcal{B}[\![b]\!](\sigma'') = \texttt{False}\}
\end{aligned}
$$

Computing the Fixpoint (Cont'd)

hsrm

- general case, $n \geq 1$:

$$
\begin{aligned}
F_n[\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}]\!] = \quad & F_{n-1}[\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}]\!] \\
& \cup \{\sigma \mapsto \sigma' \mid F_{n-1}[\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}]\!](\sigma) = \sigma', \mathcal{B}[\![b]\!](\sigma') = \texttt{True}, \\
& \qquad \mathcal{C}[\![c]\!](\sigma') = \sigma'', \text{ and } \mathcal{B}[\![b]\!](\sigma'') = \texttt{False}\}
\end{aligned}
$$

Since

$$F_i[\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}]\!] \subseteq F_{i+1}[\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}]\!]$$

for all $i \in \mathbb{N}_0$, the infinite union can be built:

$$\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!] = \bigcup_{n \in \mathbb{N}_0} F_n[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!]$$

It can be shown that this union is a fixpoint of Γ and that it is the least fixpoint.

Computing the Fixpoint: Iterative Construction

hsrm

Approach:

- start with the "smallest" function and then iteratively apply Γ and union all results
- the "smallest" function is the empty function $\emptyset$ which is undefined for all states.
- Write φ_i for the i-fold application of Γ to $\emptyset$, i.e.

$$\varphi_0 = \emptyset \text{ and } \varphi_i = \Gamma(\varphi_{i-1}) \text{ for } i > 0.$$

- Then

$$\texttt{Fix}(\Gamma) = \bigcup_{i \in \mathbf{N}_0} \varphi_i$$

- This union can be built, since the chain $\varphi_0 \subseteq \varphi_1 \subseteq \varphi_2 \subseteq \ldots$ is increasing w.r.t. $\subseteq$.

Example

hsrm

We compute the denotation of `while` $x = 0$ `do skip od`:

$$\mathcal{C}[\![\texttt{while } x = 0 \texttt{ do skip od}]\!] = \texttt{Fix}(\Gamma) \text{ where}$$

$$\begin{aligned}\Gamma &:= \lambda u \in \Sigma \to \Sigma.(\texttt{cond } (\mathcal{B}[\![x = 0]\!])\ (u \circ \texttt{id})\ \texttt{id}) \\ &= \lambda u \in \Sigma \to \Sigma.(\texttt{cond } (\lambda\sigma \in \Sigma.\sigma(x) = 0)\ u\ \texttt{id})\end{aligned}$$

We compute $\varphi_0, \varphi_1, \ldots$.

Example

hsrm

We compute the denotation of `while` $x = 0$ `do skip od`:

$$\mathcal{C}[\![\texttt{while } x = 0 \texttt{ do skip od}]\!] = \texttt{Fix}(\Gamma) \text{ where}$$

$$\begin{aligned}\Gamma &:= \lambda u \in \Sigma \to \Sigma.(\texttt{cond } (\mathcal{B}[\![x = 0]\!])\ (u \circ \texttt{id})\ \texttt{id}) \\ &= \lambda u \in \Sigma \to \Sigma.(\texttt{cond } (\lambda\sigma \in \Sigma.\sigma(x) = 0)\ u\ \texttt{id})\end{aligned}$$

We compute $\varphi_0, \varphi_1, \ldots$.

- $\varphi_0 = \emptyset$

Example

hsrm

We compute the denotation of `while` $x = 0$ `do skip od`:

$$\mathcal{C}[\![\texttt{while } x = 0 \texttt{ do skip od}]\!] = \texttt{Fix}(\Gamma) \text{ where}$$

$$\begin{aligned}\Gamma &:= \lambda u \in \Sigma \to \Sigma.(\texttt{cond } (\mathcal{B}[\![x = 0]\!])\ (u \circ \texttt{id})\ \texttt{id}) \\ &= \lambda u \in \Sigma \to \Sigma.(\texttt{cond } (\lambda\sigma \in \Sigma.\sigma(x) = 0)\ u\ \texttt{id})\end{aligned}$$

We compute $\varphi_0, \varphi_1, \ldots$.

- $\varphi_0 = \emptyset$
- $\varphi_1 = \Gamma(\varphi_0) = \texttt{cond } (\lambda\sigma \in \Sigma.\sigma(x) = 0)\ \emptyset\ \texttt{id}$.

Example

hsrm

We compute the denotation of `while` $x = 0$ `do skip od`:

$$\mathcal{C}[\![\texttt{while } x = 0 \texttt{ do skip od}]\!] = \texttt{Fix}(\Gamma) \text{ where}$$

$$\begin{aligned}\Gamma &:= \lambda u \in \Sigma \to \Sigma.(\texttt{cond } (\mathcal{B}[\![x = 0]\!])\ (u \circ \texttt{id})\ \texttt{id}) \\ &= \lambda u \in \Sigma \to \Sigma.(\texttt{cond } (\lambda\sigma \in \Sigma.\sigma(x) = 0)\ u\ \texttt{id})\end{aligned}$$

We compute $\varphi_0, \varphi_1, \ldots$.

- $\varphi_0 = \emptyset$
- $\varphi_1 = \Gamma(\varphi_0) = \texttt{cond } (\lambda\sigma \in \Sigma.\sigma(x) = 0)\ \emptyset\ \texttt{id}$.
  This can be expressed as $\varphi_1 = \{\sigma \mapsto \sigma \mid x \in Dom(\sigma) \text{ and } \sigma(x) \neq 0\}$

Example

hsrm

We compute the denotation of `while` $x = 0$ `do skip od`:

$$\mathcal{C}[\![\texttt{while } x = 0 \texttt{ do skip od}]\!] = \texttt{Fix}(\Gamma) \text{ where}$$

$$\begin{aligned}\Gamma &:= \lambda u \in \Sigma \to \Sigma.(\texttt{cond } (\mathcal{B}[\![x = 0]\!])\ (u \circ \texttt{id})\ \texttt{id}) \\ &= \lambda u \in \Sigma \to \Sigma.(\texttt{cond } (\lambda\sigma \in \Sigma.\sigma(x) = 0)\ u\ \texttt{id})\end{aligned}$$

We compute $\varphi_0, \varphi_1, \ldots$.

- $\varphi_0 = \emptyset$
- $\varphi_1 = \Gamma(\varphi_0) = \texttt{cond } (\lambda\sigma \in \Sigma.\sigma(x) = 0)\ \emptyset\ \texttt{id}$.
  This can be expressed as $\varphi_1 = \{\sigma \mapsto \sigma \mid x \in Dom(\sigma) \text{ and } \sigma(x) \neq 0\}$
- $\varphi_2 = \texttt{cond } (\lambda\sigma \in \Sigma.\sigma(x) = 0)\ \varphi_1\ \texttt{id}$
  If $\sigma(x) \neq 0$, then it is `id` and otherwise it is φ_1. This can be expressed as

$$\varphi_2 = \{\sigma \mapsto \sigma \mid x \in Dom(\sigma) \text{ and } \sigma(x) \neq 0\} \cup \{\sigma \mapsto \varphi_1\sigma \mid \sigma(x) = 0\}$$

Example

We compute the denotation of `while` $x = 0$ `do skip od`:

$$\mathcal{C}[\![\texttt{while } x = 0 \texttt{ do skip od}]\!] = \texttt{Fix}(\Gamma) \text{ where}$$

$$\begin{aligned}\Gamma &:= \lambda u \in \Sigma \to \Sigma.(\texttt{cond } (\mathcal{B}[\![x = 0]\!])\ (u \circ \texttt{id})\ \texttt{id}) \\ &= \lambda u \in \Sigma \to \Sigma.(\texttt{cond } (\lambda\sigma \in \Sigma.\sigma(x) = 0)\ u\ \texttt{id})\end{aligned}$$

We compute $\varphi_0, \varphi_1, \ldots$.

- $\varphi_0 = \emptyset$
- $\varphi_1 = \Gamma(\varphi_0) = \texttt{cond } (\lambda\sigma \in \Sigma.\sigma(x) = 0)\ \emptyset\ \texttt{id}$.
  This can be expressed as $\varphi_1 = \{\sigma \mapsto \sigma \mid x \in Dom(\sigma) \text{ and } \sigma(x) \neq 0\}$
- $\varphi_2 = \texttt{cond } (\lambda\sigma \in \Sigma.\sigma(x) = 0)\ \varphi_1\ \texttt{id}$
  If $\sigma(x) \neq 0$, then it is `id` and otherwise it is φ_1. This can be expressed as

  $$\varphi_2 = \{\sigma \mapsto \sigma \mid x \in Dom(\sigma) \text{ and } \sigma(x) \neq 0\} \cup \{\sigma \mapsto \varphi_1\sigma \mid \sigma(x) = 0\}$$

  But $\{\sigma \mapsto \varphi_1\sigma \mid \sigma(x) = 0\} = \emptyset$, since $\varphi_1\sigma$ is undefined for $\sigma(x) = 0$.
  This shows $\varphi_2 = \varphi_1$.

Example (Cont'd)

hsrm

- Since $\varphi_2 = \varphi_1$, we also have $\varphi_i = \varphi_1$ for all $i \geq 1$
- thus $\texttt{Fix}(\Gamma) = \varphi_1 = \{\sigma \mapsto \sigma \mid x \in Dom(\sigma) \text{ and } \sigma(x) \neq 0\}$.
- matches the intuition that the program terminates (with unchanged state), if x is defined and $x \neq 0$ holds in the initial state

A Second Example

Our goal is to compute:

$$\mathcal{C}[\![\texttt{while } 1 \leq X \texttt{ do } Y := Y * 2; X := X - 1 \texttt{ od}]\!]$$

We make some subcalculuations:

- $\mathcal{B}[\![1 \leq X]\!]$
- $\mathcal{C}[\![Y := Y * 2; X : X - 1]\!]$

A Second Example (2)

$$
\begin{aligned}
\mathcal{B}[\![1 \leq X]\!] &= \lambda\sigma \in \Sigma.\mathcal{A}[\![1]\!]\sigma \leq \mathcal{A}[\![X]\!]\sigma \\
&= \lambda\sigma \in \Sigma.(\lambda\sigma \in \Sigma.1)\sigma \leq (\lambda\sigma \in \Sigma.\sigma(X))\sigma \\
&= \lambda\sigma \in \Sigma.1 \leq \sigma(X)
\end{aligned}
$$

A Second Example (3)

hsrm

$$
\begin{aligned}
& \mathcal{C}[\![Y := Y * 2; X : X - 1]\!] \\
= \; & \lambda\sigma \in \Sigma.\mathcal{C}[\![X : X - 1]\!](\mathcal{C}[\![Y := Y * 2]\!]\sigma) \\
= \; & \lambda\sigma \in \Sigma.\,(\lambda\sigma \in \Sigma.\sigma[\mathcal{A}[\![X - 1]\!]\sigma/X]((\lambda\sigma \in \Sigma.\sigma[\mathcal{A}[\![Y * 2]\!]\sigma/Y])\sigma)) \\
= \; & \lambda\sigma \in \Sigma.\,(\lambda\sigma \in \Sigma.\sigma[\mathcal{A}[\![X - 1]\!]\sigma/X](\sigma[\mathcal{A}[\![Y * 2]\!]\sigma/Y])) \\
= \; & \lambda\sigma \in \Sigma.\,(\lambda\sigma \in \Sigma.\sigma[\sigma(X) - 1/X](\sigma[\sigma(Y) * 2/Y])) \\
= \; & \lambda\sigma \in \Sigma.\sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X]
\end{aligned}
$$

A Second Example (4)

hsrm

$$\mathcal{C}[\![\texttt{while } 1 \leq X \texttt{ do } Y := Y * 2; X := X - 1 \texttt{ od}]\!] = \texttt{Fix}(\Gamma)$$

with

$$\begin{aligned}
\Gamma &= \lambda u \in \Sigma \to \Sigma.\texttt{cond } \mathcal{B}[\![1 \leq X]\!]\ (u \circ \mathcal{C}[\![Y := Y * 2; X : X - 1]\!])\ \texttt{id} \\
&= \lambda u \in \Sigma \to \Sigma.\texttt{cond } (\lambda\sigma \in \Sigma.1 \leq \sigma(X)) \\
&\qquad\qquad (u \circ (\lambda\sigma \in \Sigma.\sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X])) \\
&\qquad\qquad \texttt{id} \\
&= \lambda u \in \Sigma \to \Sigma.\texttt{cond } (\lambda\sigma \in \Sigma.1 \leq \sigma(X)) \\
&\qquad\qquad (\lambda\sigma \in \Sigma.u(\sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X])) \\
&\qquad\qquad \texttt{id}
\end{aligned}$$

$\texttt{Fix}(\Gamma) = \bigcup \varphi_i$ with $\varphi_i = \varphi^i(\emptyset)$

A Second Example (5)

hsrm

$$
\begin{aligned}
\Gamma &= \lambda u \in \Sigma \to \Sigma.\texttt{cond}\ (\lambda\sigma \in \Sigma.1 \leq \sigma(X)) \\
&\qquad\qquad (\lambda\sigma \in \Sigma.u(\sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X])) \\
&\qquad\qquad \texttt{id} \\
\varphi_0 &= \emptyset = \lambda\sigma \in \Sigma.\bot \\
\varphi_1 &= \Gamma(\varphi_0) = \Gamma(\emptyset) \\
&= \texttt{cond}\ (\lambda\sigma \in \Sigma.1 \leq \sigma(X)) \\
&\qquad (\lambda\sigma \in \Sigma.(\lambda\sigma \in \Sigma.\bot)\ (\sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X])) \\
&\qquad \texttt{id} \\
&= \texttt{cond}\ (\lambda\sigma \in \Sigma.1 \leq \sigma(X))\ (\lambda\sigma \in \Sigma.\bot)\ \texttt{id} \\
&= \texttt{cond}\ (\lambda\sigma \in \Sigma.1 \leq \sigma(X))\ \emptyset\ \texttt{id} \\
&= \{\sigma \to \sigma \mid 1 > \sigma(X)\}
\end{aligned}
$$

A Second Example (6)

hsrm

$\varphi_2 = \Gamma(\varphi_1)$

A Second Example (6)

hsrm

$$
\begin{aligned}
\varphi_2 &= \Gamma(\varphi_1) \\
&= \texttt{cond}\ (\lambda\sigma \in \Sigma.1 \leq \sigma(X)) \\
&\qquad (\lambda\sigma \in \Sigma.(\texttt{cond}\ (\lambda\sigma \in \Sigma.1 \leq \sigma(X))\ \emptyset\ \texttt{id})\ (\sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X])) \\
&\qquad \texttt{id} \\
&= \{\sigma \to \sigma \mid 1 > \sigma(X)\} \cup \{\sigma \to \sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X] \mid 1 \leq \sigma(X) \wedge 1 > \sigma(X) - 1\} \\
&= \{\sigma \to \sigma \mid 1 > \sigma(X)\} \cup \{\sigma \to \sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X] \mid 1 \leq \sigma(X) \wedge 2 > \sigma(X)\}
\end{aligned}
$$

A Second Example (6)

$$
\begin{aligned}
\varphi_2 &= \Gamma(\varphi_1) \\
&= \texttt{cond}\ (\lambda\sigma \in \Sigma.1 \leq \sigma(X)) \\
&\qquad (\lambda\sigma \in \Sigma.(\texttt{cond}\ (\lambda\sigma \in \Sigma.1 \leq \sigma(X))\ \emptyset\ \texttt{id})\ (\sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X])) \\
&\qquad \texttt{id} \\
&= \{\sigma \to \sigma \mid 1 > \sigma(X)\} \cup \{\sigma \to \sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X] \mid 1 \leq \sigma(X) \wedge 1 > \sigma(X) - 1\} \\
&= \{\sigma \to \sigma \mid 1 > \sigma(X)\} \cup \{\sigma \to \sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X] \mid 1 \leq \sigma(X) \wedge 2 > \sigma(X)\}
\end{aligned}
$$

$$
\begin{aligned}
\varphi_3 &= \Gamma(\varphi_2) \\
&= \texttt{cond}\ (\lambda\sigma \in \Sigma.1 \leq \sigma(X)) \\
&\qquad (\lambda\sigma \in \Sigma.(\varphi_2\ (\sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X])) \\
&\qquad \texttt{id} \\
&= \{\sigma \to \sigma \mid 1 > \sigma(X)\} \cup \{\sigma \to \sigma[\sigma(Y) * 2 * 2/Y, \sigma(X) - 1 - 1/X] \mid 1 \leq \sigma(X) \wedge 1 > \sigma(X) - 1 - 1\} \\
&= \{\sigma \to \sigma \mid 1 > \sigma(X)\} \cup \{\sigma \to \sigma[\sigma(Y) * 2^2/Y, \sigma(X) - 2/X] \mid 1 \leq \sigma(X) \wedge 3 > \sigma(X)\}
\end{aligned}
$$

A Second Example (6)

hsrm

$$
\begin{aligned}
\varphi_2 &= \Gamma(\varphi_1) \\
&= \texttt{cond}\ (\lambda\sigma \in \Sigma.1 \leq \sigma(X)) \\
&\qquad (\lambda\sigma \in \Sigma.(\texttt{cond}\ (\lambda\sigma \in \Sigma.1 \leq \sigma(X))\ \emptyset\ \texttt{id})\ (\sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X])) \\
&\qquad \texttt{id} \\
&= \{\sigma \to \sigma \mid 1 > \sigma(X)\} \cup \{\sigma \to \sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X] \mid 1 \leq \sigma(X) \wedge 1 > \sigma(X) - 1\} \\
&= \{\sigma \to \sigma \mid 1 > \sigma(X)\} \cup \{\sigma \to \sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X] \mid 1 \leq \sigma(X) \wedge 2 > \sigma(X)\}
\end{aligned}
$$

$$
\begin{aligned}
\varphi_3 &= \Gamma(\varphi_2) \\
&= \texttt{cond}\ (\lambda\sigma \in \Sigma.1 \leq \sigma(X)) \\
&\qquad (\lambda\sigma \in \Sigma.(\varphi_2\ (\sigma[\sigma(Y) * 2/Y, \sigma(X) - 1/X])) \\
&\qquad \texttt{id} \\
&= \{\sigma \to \sigma \mid 1 > \sigma(X)\} \cup \{\sigma \to \sigma[\sigma(Y) * 2 * 2/Y, \sigma(X) - 1 - 1/X] \mid 1 \leq \sigma(X) \wedge 1 > \sigma(X) - 1 - 1\} \\
&= \{\sigma \to \sigma \mid 1 > \sigma(X)\} \cup \{\sigma \to \sigma[\sigma(Y) * 2^2/Y, \sigma(X) - 2/X] \mid 1 \leq \sigma(X) \wedge 3 > \sigma(X)\}
\end{aligned}
$$

We could proceed with $\varphi_4, \varphi_5, \ldots$ but this will not stop.

A Second Example (7)

hsrm

Solution: guess the loop-invariant:

$$\varphi_n = \{\sigma \to \sigma \mid 1 > \sigma(X)\} \cup \{\sigma \to \sigma[\sigma(Y) * 2^{\sigma(X)}/Y, 0/X] \mid 1 \leq \sigma(X) \text{ and } n > \sigma(X)\}$$

Then prove the invariant by induction on n (We skip this)

A Second Example (7)

hsrm

Solution: guess the loop-invariant:

$$\varphi_n = \{\sigma \to \sigma \mid 1 > \sigma(X)\} \cup \{\sigma \to \sigma[\sigma(Y) * 2^{\sigma(X)}/Y, 0/X] \mid 1 \leq \sigma(X) \text{ and } n > \sigma(X)\}$$

Then prove the invariant by induction on n (We skip this)

$\mathcal{C}[\![\texttt{while } 1 \leq X \texttt{ do } Y := Y * 2; X := X - 1 \texttt{ od}]\!] = \bigcup \varphi_n = f$ with

$$f\sigma = \begin{cases} \sigma, & \text{if } \sigma(X) \leq 1 \\ \sigma[Y * 2^{\sigma(X)}/Y, 0/X], & \text{if } \sigma(X) > 1 \\ \bot, & \text{if } \sigma(X) = \bot \text{ or } \sigma(Y) = \bot \end{cases}$$

Equivalence: Big-Step & Denotational Semantics

hsrm

Lemma 6.4.1

For all arithmetic expressions a and states $\sigma \in \Sigma$: $\mathcal{A}[\![a]\!]\sigma = n$ iff $\langle a, \sigma\rangle \downarrow n$

Proof. We use structural induction on a.

- Base case $a = n$: Then $\mathcal{A}[\![n]\!]\sigma = n$ and $\langle n, \sigma\rangle \downarrow n$ by axiom (AxNum).
- Base case $a = x$: Then $\mathcal{A}[\![x]\!]\sigma = \sigma(x)$ and $\langle x, \sigma\rangle \downarrow \sigma(x)$ by axiom (AxLoc).
- Step case $a = a_1 + a_2$:

$$
\begin{aligned}
&\mathcal{A}[\![a_1 + a_2]\!]\sigma = n = (\mathcal{A}[\![a_1]\!]\sigma) + (\mathcal{A}[\![a_2]\!]\sigma) \\
&\text{iff } \mathcal{A}[\![a_1]\!]\sigma = n_1 \text{ and } \mathcal{A}[\![a_2]\!]\sigma \text{ and } n = n_1 + n_2 \\
&\text{iff } \langle a_1, \sigma\rangle \downarrow n_1 \text{ and } \langle a_2, \sigma\rangle \downarrow n_2 && \text{(by the IH)} \\
&\text{iff } \langle a_1 + a_2, \sigma\rangle \downarrow n && \text{(rule (Sum))}
\end{aligned}
$$

- The cases $a = a_1 - a_2$ and $a = a_1 * a_2$ are analogous. □

Equivalence: Big-Step & Denotational Semantics (2)

hsrm

Lemma 6.4.2

For all boolean expressions b, states $\sigma \in \Sigma$, and $v \in \{\texttt{True}, \texttt{False}\}$:

$$\mathcal{B}[\![b]\!]\sigma = v \text{ iff } \langle a, \sigma \rangle \downarrow v$$

Proof (Sketch).

- The proof is by structural induction on b.
- It is similar to the previous proof.
- It uses Lemma 6.4.1 if b requires the value of an arithmetic expression.

Equivalence: Big-Step & Denotational Semantics (3)

hsrm

Theorem

For all commands c of IMP and $\sigma \in \Sigma$: $\mathcal{C}[\![c]\!]\sigma = \sigma'$ iff $\langle c, \sigma\rangle \downarrow \sigma'$

Proof.

- The "only-if" direction can be proved by induction on the derivation tree for $\langle c, \sigma\rangle \downarrow \sigma'$ (we omit the details)
- We show the "if"-direction by structural induction on c
- Base cases:
  - $\mathcal{C}[\![\texttt{skip}]\!]\sigma = \sigma$ and $\langle \texttt{skip}, \sigma\rangle \downarrow \sigma$.
  - $\mathcal{C}[\![x := a]\!]\sigma$: Assume $\mathcal{A}[\![a]\!]\sigma = n$. Then $\mathcal{C}[\![x := a]\!]\sigma = \sigma[n/x]$, and Lemma 6.4.1 shows $\langle a, \sigma\rangle \downarrow n$ and thus $\langle x := a, \sigma\rangle \downarrow \sigma[n/x]$ by (Asgn).

Equivalence: Big-Step & Denotational Semantics (4)

hsrm

Step cases:

- $\mathcal{C}[\![c_0; c_1]\!]\sigma = \mathcal{C}[\![c_1]\!](\mathcal{C}[\![c_0]\!]\sigma) = \sigma'$. Let $\sigma'' = \mathcal{C}[\![c_0]\!]\sigma$. The IH shows

  $$\mathcal{C}[\![c_0]\!]\sigma = \sigma'' \text{ implies } \langle c_0, \sigma\rangle \downarrow \sigma'' \text{ and } \mathcal{C}[\![c_1]\!]\sigma'' = \sigma' \text{ implies } \langle c_1, \sigma''\rangle \downarrow \sigma'.$$

  Now rule (Seq) shows $\langle c_0; c_1, \sigma\rangle \downarrow \sigma'$.

- $\mathcal{C}[\![\texttt{if } b \texttt{ then } c_0 \texttt{ else } c_1 \texttt{ fi}]\!]\sigma = (\texttt{cond } \mathcal{B}[\![b]\!]\ \mathcal{C}[\![c_0]\!]\ \mathcal{C}[\![c_1]\!])\sigma = \sigma'$.
  By Lemma 6.4.2 and $v \in \{\texttt{True}, \texttt{False}\}$: If $\mathcal{B}[\![b]\!]\sigma = v$, then $\langle b, \sigma\rangle \downarrow v$.
  Let $\sigma' = \begin{cases} \mathcal{C}[\![c_0]\!], & \text{if } \mathcal{B}[\![b]\!]\sigma = \texttt{True} \\ \mathcal{C}[\![c_1]\!], & \text{if } \mathcal{B}[\![b]\!]\sigma = \texttt{False} \end{cases}$
  The induction hypothesis shows $\langle c_0, \sigma\rangle \downarrow \sigma'$ or $\langle c_1, \sigma\rangle \downarrow \sigma'$ resp.
  Now rule (IfT) or (IfF), resp. can be applied, showing
  $\langle \texttt{if } b \texttt{ then } c_0 \texttt{ else } c_1 \texttt{ fi}, \sigma\rangle \downarrow \sigma'$.

Equivalence: Big-Step & Denotational Semantics (5)

hsrm

- $\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}]\!]\sigma = \texttt{Fix}(\Gamma)\sigma = \sigma'$ where

$$\Gamma = \lambda u \in (\Sigma \to \Sigma).\texttt{cond}\ \mathcal{B}[\![b]\!]\ (u \circ \mathcal{C}[\![c_0]\!])\ \texttt{id}$$

Then

$$\begin{aligned}\Gamma(\varphi) = &\{\sigma_1 \mapsto \sigma_1 \mid \mathcal{B}[\![b]\!]\sigma_1 = \texttt{False}\} \\ &\cup \{\sigma_1 \mapsto \sigma_2 \mid \mathcal{B}[\![b]\!]\sigma_1 = \texttt{True and}\ (\sigma_1 \mapsto \sigma_2) \in \varphi \circ \mathcal{C}[\![c_0]\!]\}\end{aligned}$$

Let $\varphi_n := \Gamma^n(\emptyset)$. Then

$$\begin{aligned}\varphi_{n+1} = &\{\sigma_1 \mapsto \sigma_1 \mid \mathcal{B}[\![b]\!]\sigma_1 = \texttt{False}\} \\ &\cup \{\sigma_1 \mapsto \sigma_2 \mid \mathcal{B}[\![b]\!]\sigma_1 = \texttt{True and}\ (\sigma_1 \mapsto \sigma_2) \in \varphi_n \circ \mathcal{C}[\![c_0]\!]\}\end{aligned}$$

By induction on n, we show

$$(\sigma_1 \mapsto \sigma_2) \in \varphi_n \implies \langle \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}, \sigma_1\rangle \downarrow \sigma_2$$

Equivalence: Big-Step & Denotational Semantics (6)

hsrm

By induction on n, we show that $(\sigma_1 \mapsto \sigma_2) \in \varphi_n \implies \langle \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}, \sigma_1\rangle \downarrow \sigma_2$.

- $n = 0$: $\varphi_0 = \emptyset$. The lhs of the implication is false and the implication is true.
- $n > 0$: Let the claim hold for n and let $(\sigma_1 \mapsto \sigma_2) \in \varphi_{n+1}$.
  - If $(\mathcal{B}[\![b]\!]\sigma_1) = \texttt{False}$ and $\sigma_2 = \sigma_1$, then Lemma 6.4.2 shows $\langle b, \sigma_1\rangle \downarrow \texttt{False}$.
    Rule (WhileF) shows $\langle \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}, \sigma_1\rangle \downarrow \sigma_1$.
  - If $\mathcal{B}[\![b]\!]\sigma_1 = \texttt{True}$, then Lemma 6.4.2 shows $\langle b, \sigma_1\rangle \downarrow \texttt{True}$.
    Since $\sigma_1 \mapsto \sigma_2 \in \varphi_{n+1}$, there exists σ_3 with $\mathcal{C}[\![c_0]\!]\sigma_1 = \sigma_3$ and $(\sigma_3 \mapsto \sigma_2) \in \varphi_n$.
    By the outer IH we get $\langle c_0, \sigma_1\rangle \downarrow \sigma_3$.
    By the inner IH we have $\langle \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}, \sigma_3\rangle \downarrow \sigma_2$.
    Now rule (WhileT) shows $\langle \texttt{while}\ b\ \texttt{do}\ (c_0; \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od})\ \texttt{od}, \sigma_1\rangle \downarrow \sigma_2$.

Since $\texttt{Fix}(\Gamma) = \bigcup \varphi_n$, we have: $(\sigma \mapsto \sigma') \in \texttt{Fix}(\Gamma) \implies (\sigma \mapsto \sigma') \in \varphi_n$ for some n and thus $\langle \texttt{while}\ b\ \texttt{do}\ c_0\ \texttt{od}, \sigma\rangle \downarrow \sigma'$. □

Example

hsrm

We consider the loop (`while True do skip od`) and all semantics that we have defined.

Big-Step Semantics

hsrm

Big-step operational semantics cannot derive any σ' such that $\langle \texttt{while True do skip od}, \sigma \rangle \downarrow \sigma'$ for any σ.

Small-Step semantics

hsrm

The small-step operational semantics has an infinite sequence of reduction steps:

$$
\begin{aligned}
& \langle \texttt{while True do skip od}, \sigma \rangle \\
\xrightarrow{eval, while} \; & \langle \texttt{if True then while True do skip od else skip fi}, \sigma \rangle \\
\xrightarrow{eval, ifT} \; & \langle \texttt{while True do skip od}, \sigma \rangle \\
\xrightarrow{eval, while} \; & \ldots
\end{aligned}
$$

Abstract Machine Semantics

hsrm

The abstract machine has infinite sequence of transitions:

$$
\begin{aligned}
& (E, \texttt{while True do skip od}, []) \\
\rightsquigarrow\ & (E, \texttt{True}, [T : \texttt{skip}; \texttt{while True do skip od}, F : \texttt{skip}]) \\
\rightsquigarrow\ & (E, \texttt{skip}; \texttt{while True do skip od}, []) \\
\rightsquigarrow\ & (E, \texttt{skip}, \texttt{while True do skip od}) \\
\rightsquigarrow\ & (E, \texttt{while True do skip od}, []) \\
\rightsquigarrow\ & \ldots
\end{aligned}
$$

Denotational Semantics

hsrm

The denotational semantics is $\mathcal{C}[\![\texttt{while True do skip od}]\!] := \texttt{Fix}(\Gamma)$ where $\Gamma := \lambda u \in (\Sigma \to \Sigma).\texttt{cond}\ (\lambda\sigma.\texttt{True})\ (u \circ \texttt{id})\ \texttt{id}$

For computing the fixpoint, we compute $\varphi_0, \varphi_1, \ldots$:

- $\varphi_0 = \emptyset$
- $\varphi_1 = \Gamma(\varphi_0) = \texttt{cond}\ (\lambda\sigma.\texttt{True})\ (\emptyset \circ \texttt{id})\ \texttt{id} = \texttt{cond}\ (\lambda\sigma.\texttt{True})\ \emptyset\ \texttt{id} = \emptyset$ and thus $\varphi_1 = \varphi_0$

Since $\varphi_1 = \varphi_0$, this shows $\varphi_i = \varphi_0 = \emptyset$ and thus $\texttt{Fix}(\Gamma) = \emptyset$. Thus the denotation is the partial function that is undefined for every state σ.

Conclusion

hsrm

- Different concepts and formalisms for semantics
- Operational semantics and denotational semantics
- Different styles of operational semantics
- Equivalence of the denotational and the operational semantics
- Outlook: advanced concepts for non-determinism or parallelism